# UNIT 1 RELATIONAL DATABASE DESIGN

**Structure** **Page Nos.**


| | | |
|---|---|---|
| 1.0 | Introduction | 7 |
| 1.1 | Objectives | 7 |
| 1.2 | Features of Good Database Design | 7 |
| 1.3 | Enhanced ER Tools | 8 |
| 1.4 | Functional Dependency: Theory and Normalisation | 13 |
| 1.5 | Multivalued Dependency and Fourth Normal Form | 15 |
| 1.6 | Join Dependencies and Fifth Normal Form/PJNF | 19 |
| 1.7 | Inclusion Dependencies and Template Dependencies | 23 |
| 1.8 | Domain Key Normal Form (DKNF) | 25 |
| 1.9 | Modeling Temporal Data | 26 |
| 1.10 | Summary | 27 |
| 1.11 | Solutions/Answers | 27 |
| 1.12 | Further Readings | 29 |


## 1.0 INTRODUCTION

This unit provides a detailed discussion of some of the tools and theories of good database design. The ER modeling concepts discussed in MCS-023 Block 1 Unit 2 are sufficient for representing many database schemas for database applications. However, the advanced applications of the database technologies are more complex. This resulted in extension of the ER model to EER model. However, to define the concept of EER model you must go through the concepts of E-R model first (in the above mentioned block) as these concepts are interrelated. This unit also discusses the concepts of functional, multi-valued and join dependencies and related normal forms. Some advance dependencies and a brief introduction about temporal data have also been covered in the unit.

## 1.1 OBJECTIVES

After going through this unit, you should be able to:

- define a good database design;
- draw a simple EER diagram;
- describe functional dependency and normalisation;
- explain the multi-valued dependency and 4NF;
- define join dependency and 5NF;
- define inclusion dependencies and Template Dependency, and
- define temporal data.

## 1.2 FEATURES OF GOOD DATABASE DESIGN

A good database design has the following features:

- ***Faithfulness:*** The design and implementation should be faithful to the requirements.
    - The use of *constraints* helps to achieve this feature.
- ***Avoid Redundancy:*** Something is redundant if when hidden from view, you could still figure it out from other data. This value is important because redundancy.

- wastes space and
- leads to inconsistency.

- ***Simplicity:*** Simplicity requires that the design and implementation avoid introducing more elements than are absolutely necessary – Keep it Simple (KIS).
  - This value requires designers to avoid introducing unnecessary intermediate concepts.
- ***Right kind of element:*** Attributes are easier to implement but entity sets and relationships are necessary to ensure that the right kind of element is introduced.

## 1.3 ENHANCED ER TOOLS

In addition to ER modeling concepts the EER model includes:

1) Subclass and Super class
2) Inheritance
3) Specialisation and Generalisation.

To describe the concepts of **subclass *and* super class** first let us revisit the concept of 'entity'. The basic object that an E-R model represents is an entity, which is a "thing" in the real world with an independent existence. An entity may be an object with a physical existence or it may be an object with a conceptual existence. Each entity has attributes (the particular properties that describe it).

For Example, the entity *vehicle* describes the type (that is, the attributes and relationship) of each *vehicle* entity and also refers to the current set of *vehicle* entities in the showroom database. Some times to signify the database application various meaningful sub-groupings of entity is done explicitly. For example, the members of the entity *vehicle* are further meaningfully sub- grouped as: Car, Scooter, truck and so on.

The set of entities in each of the groupings is a subset of the entities that belongs to the entity set *vehicle*. In other words every sub-grouping must be *vehicle*. Therefore, these sub-groupings are called a subclass of the *vehicle* entity type and the *vehicle* itself is called the super class for each of these subclasses.

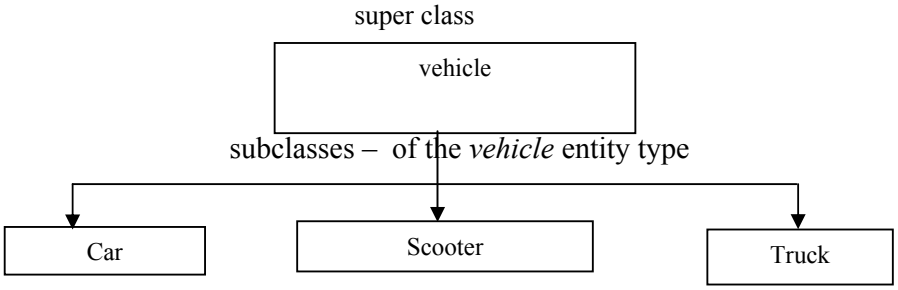



**Figure 1: A class hierarchy**

The relationship between a super class and any of its subclasses is called class/subclass relationship. It is often called an IS-A or relationship because of the way we refer to the concept, we say, "car ***is-a*** vehicle". The member entity of the subclass represents the same real world as the member entity of the super class. If an entity is a member of a subclass, by default it must also become a member of the super class whereas it is not necessary that every entity of the super class must be a member of its subclass. From the discussion above on sub/super classes we can say that an entity that is a member of a subclass inherits all the attributes of

the entity as a member of the super class. Notice that the type of an entity is defined by the attributes it possesses and the relationship types in which it participates; therefore, the entity also inherits all the relationships in which the super class participates. According to **inheritance** the subclass has its own attributes and relationships together with all attributes and relationships it inherits from the super class.

The process of defining the subclasses of an entity type is called **specialisation**, where the entity type is called the super class of the specialisation. The above said specialised set of subclasses are defined on the basis of some common but distinguishing characteristics of the entities in the super class. For example, the set of subclasses (car, scooter, truck) is a specialisation of the super class *vehicle* that distinguished among vehicles entities based on the vehicle type of each entity. We may have several other specialisations of the same entity type based on different common but distinctive characteristics. *Figure 2* shows how we can represent a specialisation with the help of an EER diagram.

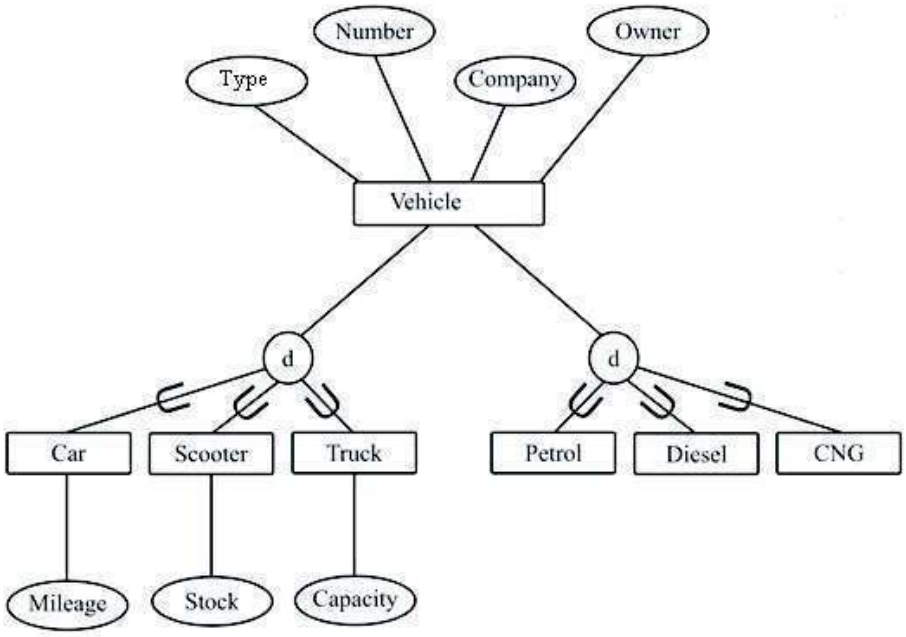



**Figure 2: EER diagram representing specialisation**

The subclasses that define a specialisation are attached by lines to a circle, which is connected further with the super class. The circle connecting the super class with the subclass indicates the direction of the super class/ subclass relationship. The letter 'd' in the circle indicates that all these subclasses are **disjoint** constraints.

Attributes that apply only to entities of a particular subclass – such as mileage of car, stock of scooter and capacity of truck are attached to the rectangle representing that subclass. Notice that an entity that belongs to a subclass represents the same real-world entity as the entity connected to super class, even though the same entity is shown twice – one in the subclass and the other in the super class. A subclass is defined in order to group the entities to which these attributes apply. The members of a subclass may still share the majority of their attributes with the other members of the super class (as shown in *Figure 3*).

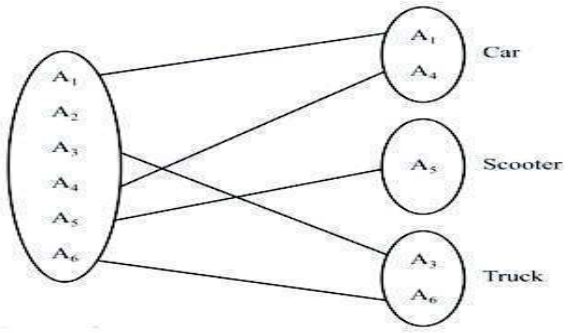



**Figure 3: Sharing of members of the super class vehicle and its subclasses**

Hence the specialisation is a set of subclasses of an entity type, which establishes additional specific attributes with each subclass and also establishes additional specific relationship types between each subclass and other entity types or other subclasses.

**Generalisation** is the reverse process of specialisation; in other words, it is a process of suppressing the differences between several entity types, identifying their common features into a single super class. For example, the entity type CAR and TRUCK can be generalised into entity type VEHICLE. Therefore, CAR and TRUCK can now be subclasses of the super class generalised class VEHICLE.

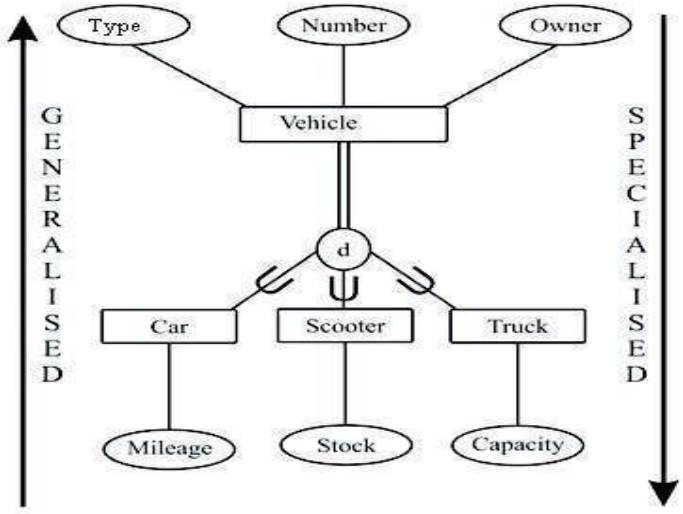



**Figure 4: Generalisation and specialisation**

**Constraints and Characteristics of Specialisation and Generalisation**

A super class may either have a single subclass or many subclasses in specialisation. In case of only one subclass we do not use circle notation to show the relationship of subclass/super class. Sometimes in specialisation, the subclass becomes the member of the super class after satisfying a condition on the value of some attributes of the super class. Such subclasses are called *condition-defined* subclasses or predicate defined subclasses. For example, vehicle entity type has an attribute vehicle "type", as shown in the *Figure 4.*

We can specify the condition of membership for a subclass – car, truck, scooter – by the predicate – vehicle 'type' of the super class vehicle. Therefore, a vehicle object

can be a member of the sub-class, if it satisfies the membership condition for that sub-class. For example, to be a member of sub-class car a vehicle must have the condition vehicle "type = car" as true. A specialisation where all the sub-classes have the membership condition defined on the same attribute of the super class, is called an *attribute-defined* specialisation. The common attribute that defines the condition is called the *defining attribute* of the specialisation. If no condition is specified to determine the membership of subclass in specialisation then it is called user-defined, as in such a case a database user must determine the membership.

Disjointness is also the constraints to a specialisation. It means that an entity can be a member of at most one of the subclasses of the specialisation. In an attribute-defined specialisation the disjointness constraint means that an entity can be a member of a single sub-class only. In the *Figure 4,* the symbol 'd' in circle stands for disjoint.

But if the real world entity is not disjoint their set of entities may overlap; that is an entity may be a member of more than one subclass of the specialisation. This is represented by an (o) in the circle. For example, if we classify cars as luxury cars and cars then they will overlap.

When every entity in the super class must be a member of some subclass in the specialisation it is called total specialisation. But if an entity does not belong to any of the subclasses it is called partial specialisation. The total is represented by a double line.

This is to note that in specialisation and generalisation the deletion of an entity from a super class implies automatic deletion from subclasses belonging to the same; similarly insertion of an entity in a super class implies that the entity is mandatorily inserted in all attribute defined subclass for which the entity satisfies the defining predicate. But in case of total specialisation, insertion of an entity in a super class implies compulsory insertion in at least one of the subclasses of the specialisation.

In some cases, a single class has a similar relationship with more than one class. For example, the sub class 'car' may be owned by two different types of owners: INDIVIDUAL or ORGANISATION. Both these types of owners are different classes thus such a situation can be modeled with the help of a Union.

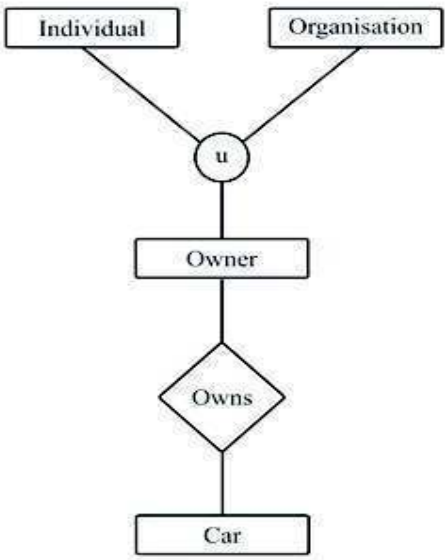



**Figure 5: Union of classes**



**Converting EER diagram to table**

The rules for converting EER diagram – which primarily includes specialisation and generalisation hierarchy are the same as in ER diagram. Let us recapitulate the rules:

- Create a table for each strong entity set.
- Create a table for a weak entity set; however, include the primary key of the strong entity on which it depends in the table.
- Create a table for each binary m : n relationship set having the primary keys of both the participating entities; however, for a binary m : 1 or 1 : m relationship, the primary key on the m side should be used as the primary key. For binary 1:1 relationship set the primary key is chosen from any of the participating relations.
- Composite attribute may sometimes be converted to a separate table.
- For generalisation or specialisation hierarchy a table can be created for higher level and each of the lower level entities. The higher level entity would have the common attributes and each lower level table would have the primary key of higher level entity and the attributes defined at the lower specialised level. However, for a complete disjoint hierarchy no table is made at the higher level, but the tables are made at the lower level including the attributes of higher level.
- For an aggregation, all the entities and relationships of the aggregation are transformed into the table on the basis of the above rules. The relationship that exists between the simple entity and the aggregated entity, have the primary key of the simple entity and the primary key of the relationship of the aggregated entity.

So let us now discuss the process of converting EER diagram into a table. In case of disjoint constrains with total participation. It is advisable to create separate tables for the subclasses. But the only problem in such a case will be to see that referential entity constraints are met suitably.

For example, EER diagram at *Figure 4* can be converted into a table as:

CAR (**Number**, owner, type, mileage)
SCOOTER (**Number**, owner, type, stock)
TRUCK (**Number,** owner, type, capacity)

Please note that referential integrity constraint in this case would require relationship with three tables and thus is more complex.

In case there is no total participation in the *Figure* 4 then there will be some vehicles, which are not car, scooter and truck, so how can we represent these? Also when overlapping constraint is used, then some tuples may get represented in more than one table. Thus, in such cases, it is ideal to create one table for the super class and the primary key and any other attribute of the subclass. For example, assuming total participation does not exist in *Figure* 4, then, a good table design for such a system may be:

VEHICLE (**Number,** owner, type)
CAR (**Number,** mileage)
SCOOTER (**Number,** stock)
TRUCK (**Number,** capacity)



Finally, in the case of union since it represents dissimilar classes we may represent separate tables. For example, both individual and organisation will be modeled to separate tables.

### ☞ Check Your Progress 1

1) What is the use of EER diagram?

……

……

……

2) What are the constraints used in EER diagrams?

……

……

……

3) How is an EER diagram converted into a table?

……

……

……

## 1.4 FUNCTIONAL DEPENDENCY: THEORY AND NORMALISATION

When a single constraint is established between two sets of attributes from the database it is called *functional dependency*. We have briefly discussed this in MCS-023. Let us discuss it in some more detail, especially with respect to formal theory of data dependencies. Let us consider a single universal relation scheme "A". A functional dependency denoted by X → Y, between two sets of attributes X and Y that are subset of universal relation "A" specifies a constraint on the possible tuples that can form a relation state of "A". The constraint is that, for any two tuples t1 and t2 in "A" that have t1(X) = t2 (X), we must also have t1(Y) = t2(Y). It means that, if tuple t1 and t2 have same values for attributes X then X→Y to hold t1 and t2 must have same values for attributes Y.

Thus, FD X→Y means that the values of the Y component of a tuple in "A" depend on or is determined by the values of X component. In other words, the value of Y component is uniquely determined by the value of X component. This is functional dependency from X to Y (**but not Y to X**) that is, Y is functionally dependent on X.

The relation schema "A" determines the function dependency of Y on X (X→Y) when and only when:

1) if two tuples in "A", agree on their X value then
2) they **must** agree on their Y value.

Please note that if X → Y in "A", does not mean Y→ X in "A".

This semantic property of functional dependency explains how the attributes in "A" are related to one another. A FD in "A" must be used to specify constraints on its attributes that must hold at all times.

For example, a FD state, city, place → pin-code should hold for any address in India. It is also possible that certain functional dependencies may cease to exist in the real world if the relationship changes, for example, the FD pin-code→ area-code used to exist as a relationship between postal codes and telephone number codes in India, with the proliferation of mobile telephone, the FD is no longer true.

Consider a relation

**STUDENT-COURSE (enrolno, sname, cname, classlocation, hours)**

We know that the following functional dependencies (we identify these primarily from constraints, there is no thumb rule to do so otherwise) should hold:

- **enrolno →sname** (the enrolment number of a student uniquely determines the student names alternatively, we can say that sname is functionally determined/dependent on enrolment number).

- **classcode → cname, classlocation,** (the value of a class code uniquely determines the class name and class location.

- **enrolno, classcode → Hours** (a combination of enrolment number and class code values uniquely determines the number of hours and students study in the class per week (Hours).

These FDs can be optimised to obtain a minimal set of FDs called the canonical cover. However, these topics are beyond the scope of this course and can be studied by consulting further reading list. You have already studied the functional dependence (FDs) and its use in normalisation till BCNF in MCS-023. However, we will briefly define the normal forms.

**Normalisation**

The first concept of normalisation was proposed by Mr. Codd in 1972. Initially, he alone proposed three normal forms named first, second and third normal form. Later on, with the joint efforts of Boyce and Codd, a stronger definition of 3NF called Boyce-Codd Normal Form (BCNF) was proposed. All the said normal forms are based on the functional dependencies among the attributes of a relation. The normalisation process depends on the assumptions that:

1) a set of functional dependencies is given for each relation, and
2) each relation has a designated primary key.

The normalisation process is based on the two assumptions /information above. Codd takes a relation schema through a series of tests to ascertain whether it satisfies a certain normal form. The process proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as found necessary during analysis. You have already gone through this process in MCS-023.

Later on, fourth normal form (4NF) and a fifth normal form (5NF) were proposed based on the concept of multivalued dependencies and join dependencies respectively. We will discuss these in more detail in the subsequent sections.

Therefore, normalisation is looked upon as a process of analysing the given relation schemas based on their condition (FDs and Primary Keys) to achieve the desirable properties:

- firstly minimizing redundancy, and
- secondly minimizing the insertion, deletion update anomalies.



Thus, the normalisation provides the database designer with:

1) a formal framework for analysing relation schemas.
2) a series of normal form tests that can be normalised to any desired degree.

The degree of normal forms to which a relation schema has been normalised through decomposition confirm the existence of additional properties that the relational schemas should possess. It could include any or both of two properties.

1) the lossless join and non-additive join property, and
2) the dependency preservation property.

Based on performance reasons, relations may be left in a lower normalisation status. It is not mandatory that the database designer must normalise to the highest possible normal form. The process of storing the join of higher normal form relations, as a base relation (which is in a lower normal form) is known as denormalisation).

## 1.5 MULTIVALUED DEPENDENCIES AND FOURTH NORMAL FORM

In database modeling using the E-R Modeling technique, we usually face known difficulties that may arise when an entity has multivalued attributes. In the relational model, if all of the information about such entity is to be represented in one relation, it will be necessary to repeat all the information other than the multivalue attribute value to represent all the information. It will result in multi-tuples about the same instance of the entity in the relation and the relation having a composite key (the entity id and the mutlivalued attribute). This situation becomes much worse if an entity has more than one multivalued attributes and these values are represented in one relation by a number of tuples for each entity instance such that every value of one of the multivalued attributes appears with every value of the second multivalued attribute to maintain consistency. The multivalued dependency relates to this problem when more than one multivalued attributes exist. Let us consider the same through an example relation that represents an entity 'employee'.

*emp (e#, dept, salary, job)*

We have so far considered normalisation based on functional dependencies that apply only to single-valued facts. For example, $e\# \rightarrow dept$ implies only one *dept* value for each value of *e#*. Not all information in a database is single-valued, for example, *job* in an employee relation may be the list of all projects that the employee is currently working on. Although *e#* determines the list of all the projects that an employee is working on, yet, $e\# \rightarrow job$ is not a functional dependency.

The fourth and fifth normal forms deal with multivalued dependencies. Before discussing the 4NF and 5NF we will discuss the following example to illustrate the concept of multivalued dependency.

*programmer (emp_name, projects, languages)*

The above relation includes two multivalued attributes of the entity *programmer* - *projects* and *languages*. There are no functional dependencies.

The attributes *projects* and *languages* are assumed to be independent of each other. If we were to consider *projects* and *languages* as separate entities, we would have two relationships (one between *employees* and *projects* and the other between *employees* and programming *languages*). Both the above relationships are many-to-many relation, in the following sense:

1) one programmer could have several projects, and



2) may know several programming languages, also

3) one project may be obtained by several programmers, and

4) one programming language may be known to many programmers.

The above relation is in 3NF (even in BCNF) with some disadvantages. Suppose a programmer has several projects (Proj_A, Proj_B, Proj_C, etc.) and is proficient in several programming languages, how should this information be represented? There are several possibilities.

| Emp_name | Projects | languages |
|---|---|---|
| DEV | Proj_A | C |
| DEV | Proj_A | JAVA |
| DEV | Proj_A | C++ |
| DEV | Proj_B | C |
| DEV | Proj_B | JAVA |
| DEV | Proj_B | C++ |

| emp_name | Projects | languages |
|---|---|---|
| DEV | Proj_A | NULL |
| DEV | Proj_B | NULL |
| DEV | NULL | C |
| DEV | NULL | JAVA |
| DEV | NULL | C++ |

| emp_name | Projects | languages |
|---|---|---|
| DEV | Proj_A | C |
| DEV | Proj_B | JAVA |
| DEV | NULL | C++ |

Other variations are possible. Please note this is so as there is no relationship between the attributes ‘projects’ and programming ‘languages’. All the said variations have some disadvantages. If the information is repeated, we face the problems of repeated information and anomalies as we did when second or third normal form conditions were violated. Without repetition, difficulties still exist with insertions, deletions and update operations. For example, in the first table we want to insert a new person RAM who has just joined the organisation and is proficient in C and JAVA. However, this information cannot be inserted in the first table as RAM has not been allotted to work on any project. Thus, there is an insertion anomaly in the first table. Similarly, if both Project A and Project B get completed on which DEV was working (so we delete all the tuples in the first table) then the information that DEV is proficient in C, JAVA, and C++ languages will also be lost. This is the deletion anomaly. Finally, please note that the information that DEV is working on Project A is being repeated at least three times. Also the information that DEV is proficient in JAVA is repeated. Thus, there is redundancy of information in the first tables that may lead to inconsistency on updating (update anomaly).

In the second and third tables above, the role of NULL values is confusing. Also the candidate key in the above relations is (emp name, projects, language) and existential integrity requires that no NULLs be specified. These problems may be overcome by decomposing a relation as follows:



| emp_name | Projects |
|---|---|
| DEV | Proj_A |
| DEV | Proj_B |

| emp_name | languages |
|---|---|
| DEV | C |
| DEV | JAVA |
| DEV | C++ |

This decomposition is the concept of 4NF. Functional dependency $A \rightarrow B$ relates one value of $A$ to one value of $B$ while multivalued dependency $A \rightarrow\rightarrow B$ defines a relationship where a set of values of attribute $B$ are determined by a single value of $A$. Multivalued dependencies were developed to provide a basis for decomposition of relations like the one above. Let us define the multivalued dependency formally.

**Multivalued dependency**: *The multivalued dependency $X \rightarrow\rightarrow Y$ is said to hold for a relation R(X, Y, Z) if, for a given set of value (set of values if X is more than one attribute) for attribute X, there is a set of (zero or more) associated values for the set of attributes Y and the Y values depend only on X values and have no dependence on the set of attributes Z.*

Please note that whenever $X \rightarrow\rightarrow Y$ holds, so does $X \rightarrow\rightarrow Z$ since the role of the attributes $Y$ and $Z$ is symmetrical.

In the example given above, if there was some dependence between the attributes *projects* and *language*, for example, the language was related to the projects (perhaps the projects are prepared in a particular language), then the relation would not have MVD and could not be decomposed into two relations as above. However, assuming there is no dependence, *emp_name* $\rightarrow\rightarrow$ *projects and emp_name* $\rightarrow\rightarrow$ *languages* holds.

*Trival MVD:* A MVC X $\rightarrow\rightarrow$Y is called trivial MVD if either Y is a subset of $X$ or $X$ and $Y$ together form the relation R.

The MVD is trivial since it results in no constraints being placed on the relation. If a relation like *emp(eno, edpendent#)* has a relationship between *eno* and *edependent#* in which *eno* uniquely determines the **values** of *edependent#,* the dependence of *edependent#* on *eno* is called a trivial MVD since the relation *emp* cannot be decomposed any further.

Therefore, a relation having non-trivial MVDs must have at least three attributes; two of them multivalued and not dependent on each other. Non-trivial MVDs result in the relation having some constraints on it since all possible combinations of the multivalued attributes are then required to be in the relation.

Let us now define the concept of MVD in a different way. Consider the relation *R(X, Y, Z)* having a multi-valued set of attributes $Y$ associated with a value of $X$. Assume that the attributes $Y$ and $Z$ are independent, and Z is also multi-valued. Now, more formally, $X\rightarrow\rightarrow Y$ is said to hold for *R(X, Y, Z)* if *t1* and *t2* are two tuples in $R$ that have the same values for attributes $X$ *(t1[X] = t2[X])* then $R$ also contains tuples *t3* and *t4* (not necessarily distinct) such that:

*t1X] = t2[X] = t3[X] = t4[X]*
*t3[Y] = t1[Y]* and *t3[Z] = t2[Z]*
*t4[Y] = t2[Y]* and *t4[Z] = t1[Z]*



In other words if *t1* and *t2* are given by:

*t1 = [X, Y1, Z1]*, and

*t2 = [X, Y2, Z2]*

then there must be tuples *t3* and *t4* such that:

*t3 = [X, Y1, Z2]*, and

*t4 = [X, Y2, Z1]*

We are, therefore, insisting that every value of *Y* appears with every value of *Z* to keep the relation instances consistent. In other words, the above conditions insist that *Y* and *Z* are determined by *X* alone and there is no relationship between *Y* and *Z* since *Y* and *Z* appear in every possible pair and hence these pairings present no information and are of no significance. Only if some of these pairings were not present, there would be some significance in the pairings.

**(Note:** If *Z* is single-valued and functionally dependent on *X* then *Z1 = Z2*. If *Z* is multivalue dependent on *X* then *Z1 <> Z2*).

The theory of multivalued dependencies is very similar to that for functional dependencies. Given *D* a set of MVDs, we may find $D^+$, the closure of *D* using a set of axioms. We do not discuss the axioms here. You may refer this topic in further readings.

We have considered an example of *Programmer(Emp name, projects, languages)* and discussed the problems that may arise if the relation is not normalised further. We also saw how the relation could be decomposed into *P1(emp name, projects)* and *P2(emp name, languages)* to overcome these problems. The decomposed relations are in fourth normal form (4NF), which we shall now define.

We now define 4NF. A relation *R* is in 4NF if, whenever a multivalued dependency $X \rightarrow\rightarrow Y$ holds, then either

(a) the dependency is trivial, or

(b) *X* is a candidate key for *R*.

The dependency $X \rightarrow\rightarrow \emptyset$ or $X \rightarrow\rightarrow Y$ in a relation *R (X, Y)* is trivial, since they must hold for all *R (X, Y)*. Similarly, in a trivial MVD *(X, Y)* $\rightarrow\rightarrow Z$ must hold for all relations *R (X, Y, Z)* with only three attributes.

If a relation has more than one multivalued attribute, we should decompose it into fourth normal form using the following rules of decomposition:

For a relation *R(X,Y,Z)*, if it contains two nontrivial MVDs $X \rightarrow\rightarrow Y$ and $X \rightarrow\rightarrow Z$ then decompose the relation into $R_1$ *(X,Y)* and $R_2$ *(X,Z)* or more specifically, if there holds a non-trivial MVD in a relation *R (X,Y,Z)* of the form $X \rightarrow\rightarrow Y$, such that $X \cap Y = \phi$, that is the set of attributes X and Y are disjoint, then R must be decomposed to $R_1(X,Y)$ and $R_2(X,Z)$, where *Z* represents all attributes other than those in *X* and *Y*.

Intuitively *R* is in 4NF if all dependencies are a result of keys. When multivalued dependencies exist, a relation should not contain two or more independent multivalued attributes. The decomposition of a relation to achieve 4NF would normally result in not only reduction of redundancies but also avoidance of anomalies.



# 1.6 JOIN DEPENDENCIES AND 5NF /PJNF

Based on the discussion above, we know that the normal forms require that the given relation *R* if not in the given normal form should be decomposed in two relations to meet the requirements of the normal form. However, in some rare cases, a relation can have problems like redundant information and update anomalies, yet it cannot be decomposed in two relations without loss of information. In such cases, it may be possible to decompose the relation in three or more relations using the 5NF. But when does such a situation arise? Such cases normally happen when a relation has at least three attributes such that all those values are totally independent of each other.

The fifth normal form deals with join-dependencies, which is a generalisation of the MVD. The aim of fifth normal form is to have relations that cannot be decomposed further. A relation in 5NF cannot be constructed from several smaller relations.

*A relation R satisfies* join dependency *($R_1$, $R_2$, ..., $R_n$) if and only if *R* is equal to the join of $R_1$, $R_2$, ..., $R_n$ where $R_i$ are subsets of the set of attributes of *R*.

A relation *R* is in 5NF if for all join dependencies at least one of the following holds:

*(a) ($R_1$, $R_2$, ..., $R_n$) is a trivial join-dependency (that is, one of $R_i$ is R)*

*(b) Every $R_i$ is a candidate key for R.*

An example of 5NF can be provided by the same above example that deals with emp_name, Projects and Programming languages with some modifications:

| emp_name | Projects | Languages |
|---|---|---|
| DEV | Proj_A | C |
| RAM | Proj_A | JAVA |
| DEV | Proj_B | C |
| RAM | Proj_B | C++ |

The relation above assumes that any employee can work on any project and knows any of the three languages. The relation also says that any employee can work on projects Proj_A, Proj_B, Proj_C and may be using a different programming languages in their projects. No employee takes all the projects and no project uses all the programming languages and therefore all three fields are needed to represent the information. Thus, all the three attributes are independent of each other.

The relation above does not have any FDs and MVDs since the attributes emp_name, project and languages are independent; they are related to each other only by the pairings that have significant information in them. For example, DEV is working on Project A using C languague. Thus, the key to the relation is (emp_name, project, languague). The relation is in 4NF, but still suffers from the insertion, deletion, and update anomalies as discussed for the previous form of this relation. However, the relation therefore cannot be decomposed in two relations.

*(emp_name, project), and*

*(emp_name, language)*

Why?

Let us explain this with the help of a definition of join dependency. The decomposition mentioned above will create tables as given below:

Emp_project

| emp_name | Projects |
|---|---|
| DEV | Proj_A |
| RAM | Proj_A |
| DEV | Proj_B |
| RAM | Proj_B |

Emp_language

| Emp_name | languages |
|---|---|
| DEV | C |
| RAM | JAVA |
| RAM | C++ |

On taking join of these relations on emp_name it will produce the following result:

| emp_name | Projects | Languages |
|---|---|---|
| DEV | Proj_A | C |
| RAM | Proj_A | JAVA |
| RAM | Proj_A | C++ |
| DEV | Proj_B | C |
| RAM | Proj_B | JAVA |
| RAM | Proj_B | C++ |

Since the joined table does not match the actual table, we can say that it is a lossy decomposition. Thus, the expected join dependency expression:

*((emp_name, project), (emp_name, language)) does not satisfy the conditions of lossless decomposition. Hence, the decomposed tables are losing some important information.

Can the relation 'Programmer' be decomposed in the following three relations?

*(emp_name, project),*
*(emp_name, language) and*
*(Projects, language)*

Please verify whether this decomposition in lossless or not. The join dependency in this case would be:

*((emp_name, project), (emp_name, language), (project, language))*
and it can be shown that this decomposition is lossless.

**Project-Join Normal Form**

(Reference website: http://codex.cs.yale.edu/avi/db-book/online-dir/c.pdf)
*PJNF* is defined using the concept of the join dependencies. A relation schema *R* having a set *F* of functional, multivalued, and join dependencies, is in PJNF (5 NF), if for all the join dependencies in the closure of F (referred to as *F+*) that are of the form



*(*R1, R2, . . .,Rn*), where each *Ri* ⊆ *R* and *R* = *R*1 ∪ *R*2 ∪ . . . ∪ *Rn*, at least one of the following holds:

- *(*R1, R2, . . .,Rn*) is a trivial join dependency.
- Every *Ri* is a superkey for *R*.

PJNF is also referred to as the Fifth Normal Form (5NF).

Let us first define the concept of PJNF from the viewpoint of the decomposition and then refine it later to a standard form.

**Definition 1:** A JD *[R1, R2, . . . , Rn] over a relation R is trivial if it is satisfied by every relation r(R).

The trivial JDs over R are JDs of the form *[R1, R2, . . . , Rn] where for some i the Ri = R.

**Definition 2:** A JD *[R1, R2, . . ., Rn] applies to a relation scheme R if R = R1 R2 …Rn.

**Definition 3:** Let R be a relation scheme having F as the set of FDs and JDs over R. R will be in project-join normal form (PJNF) if for every JD *[R1, R2, . . ., Rn] which can be derived by F that applies to R, the following holds:

- The JD is trivial, or
- Every Ri is a super key for R.

For a database scheme to be in project-join normal form, every relation *R* in this database scheme should be in project-join normal form with respect to *F*.

Let us explain the above with the help of an example.

Example: Consider a relational scheme R = *A B C D E G* having the set of dependencies F = {*[*A B C D, C D E, B D G*], *[*A B, B C D, A D*], A → *B C D E, B C → A G* }. The *R* as given above is not in PJNF. Why? The two alternate keys to *R* are *A* and *BC*, so please note that the *JD* *[*A B C D, C D E, B D G*], does not satisfy the condition "Every *Ri* is a super key for R" as the two components of this *JD* viz., *C D E* and *B D G*, does not satisfy the condition.

However, if we decompose the *R* as {*R1, R2, R3*}, where *R1 = A B C D, R2 = C D E,* and *R3 = B D G*, then it is in *PJNF* with respect to *F*. Please note that in the example, the *JD* *[*A B, B C D, A D*] is implied by F and applies to *R1*. Whereas, the *FDs* are trivial or have keys as the left side.

The definition of *PJNF* as given above is a weaker than the original definition of *PJNF* given by Fagin. The original definition ensures enforceability of dependencies by satisfying keys, in addition to elimination of redundancy. The final definition is:

**Definition 4:** Let *R* be a relation scheme having *F* as the set of *FDs* and *JDs* over *R*. *R* will be in project-join normal form (*PJNF*) if for every JD *[*R1, R2*, . . ., *Rn*] which can be derived by *F* that applies to *R*, is implied by the key *FDs* of *R*.

The following example demonstrates this definition.

**Example**: Consider a relation scheme *R = A B C* having the set of dependencies as *F* = {A→ B C, C → A B, *[A B, B C] }. Please note that the *R* is not in *PJNF*, although since *A B* and *B C* are the super keys of *R, R* satisfies the earlier definition of *PJNF*. But R does not satisfy the revised definition as given above.

Please note that since every multivalued dependency is also a join dependency, every PJNF schema is also in 4NF. Decomposing a relation scheme using the JDs that cause PJNF violations creates the PJNF scheme. PJNF may also be not dependency preserving.

## ☞ Check Your Progress 2

1) What are Multi-valued Dependencies? When we can say that a constraint X is multi-determined?

……

……

……

2) Decompose the following into 4NF

EMP

| ENAME | PNAME | DNAME |
|---|---|---|
| Dev | X | Sanju |
| Dev | Y | Sainyam |
| Dev | X | Sainyam |
| Dev | Y | Sanju |

……

……

……

3) Does the following relation satisfy MVDs with 4NF? Decompose the following relation into 5NF.

SUPPLY

| SNAME | PARTNAME | PROJNAME |
|---|---|---|
| Dev | Bolt | X |
| Dev | Nut | Y |
| Sanju | Bolt | Y |
| Sainyam | Nut | Z |
| Sanju | Nail | X |
| Sanju | Bolt | X |
| Dev | Bolt | Y |

……

……

……..

4) When is a set of functional dependencies F minimal?

……

……

……

……..

5) Give the proof of transitive rule $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.

……

……

……

……..

# 1.7 INCLUSION DEPENDENCIES AND TEMPLATE DEPENDENCIES

An **inclusion dependency** $R.X < S.Y$ between two sets of attributes – X of a relation schema R, and Y of a relation schema S – is defined as the following constraint:

If r and s are the relation state of R and S respectively at any specific time then:

$$\pi_X(r(R)) \subseteq \pi_Y(s(S))$$

The subset relationship does not necessarily have to be a proper subset. Please note that the sets of attributes on which the inclusion dependency is specified viz. *X* of *R* and *Y* of *S* above, must have the same number of attributes. In addition, the domains for each corresponding pair of attributes in *X* and *Y* should be compatible. The objectives of inclusion Dependencies are to formalise two important types of interrelation constraints that exist between the relations, thus cannot be expressed using FDs and MVDs that are:

1) Referential integrity constraints,

2) Class / subclass relationships.

The common rules for making inferences from defined inclusion dependencies (Inclusion Dependency Inference Rule – IDIR) are:

IDIR1 (reflexivity): $R.X < R.X$
IDIR2 (attribute correspondence): if $R.X < S.Y$

here

X = {*A1, A2, ....., An*} and
Y = {*B1, B2, ....., Bn*} and
*Ai* correspondence to *B*i for $1 \leq i \leq n$

IDIR3 (transitivity): If R.X < S.Y and S.Y < T.Z then R.X < T.Z

**Template Dependencies**[1]

The template dependencies are the more general and natural class of data dependencies that generalizes the concepts of JDs. A template dependency is representation of the statement that a relation is invariant under a certain tableau mapping. Therefore, it resembles a tableau. It consists of a number of hypothesis rows that define certain variables with a special row at the bottom, called the conclusion row. A relation r satisfies a template dependency, if and only if, a valuation (say ρ) that successfully maps the hypothesis rows to tuples in a relation r, finds a map for conclusion row to a tuple in r. (Please note that this is not the complete and formal definition). Let us explain this informal definition with the help of example.

**Example 1**: The Figure 6 shows a template dependency *T* over the scheme *A B C* having specific variables of A (a and a'), B and C. The hypothesis rows are *w1, w2, w3* and *w4.* The row w is the conclusion row. Relation *r* given in Figure 7 does not satisfy *T*, since the valuation ρ that maps hypothesis rows *w1* to *w4* using variable values a = 1, a' = 2, b = 3, b' = 4, c = 5, c' = 6, does not map conclusion row w to any tuple in r. To make the relation in Figure 7 to satisfy the template dependency given in

**(http://www.dbis.informatik.hu-berlin.de/~freytag/Maier/C14.pdf)**



Figure 6, we need to add a tuple that is equivalent to the conclusion row. Thus, we need to add a tuple t5 in figure 1.7 having the values of the variables a, b and c as 1,3,5 respectively. Please note the template dependencies are difficult to check in a large table.

| *T* | (A | B | C) |
|---|---|---|---|
| wl | a | b | c’ |
| w2 | a’ | b | c |
| w3 | a | b’ | c |
| w4 | a’ | b’ | c |
| w | a | b | c |

**Figure 6: A sample template dependency**

| R | (A | B | C) |
|---|---|---|---|
| t1 | l | 3 | 6 |
| t2 | 2 | 3 | 5 |
| t3 | l | 4 | 5 |
| t4 | 2 | 4 | 5 |

**Figure 7: An example relation *r* for checking template dependency**

Please note that although the template dependencies look like tableaux, but they are not exactly the same. We will discuss about this concept in more details later in this section. Let us now define the template dependency formally:

Definition: A template dependency (*TD*) on a relation scheme R is a pair $T = (T,w)$ where $T=\{w1,w2, \ldots . wk\}$ is a set of hypothesis rows on R, and w is a single conclusion row on R. A relation r(R) satisfies *TD T* if for every valuation $\rho$ of T such that $\rho(T) \subseteq r$, $\rho$ can be extended to show that $\rho(w) \in r$. A template dependency is trivial if every relation over R satisfies it.

The template dependencies are written as shown in *Figure 6.* The conclusion row is written at the bottom separated from the hypothesis rows. The variables are written using lowercase alphabets corresponding to possible attribute name. The conclusion row variables are normally have not primed or subscripted. The TDs almost look like tableau mappings turned upside down. A template dependency is different from a tableau in the following two ways:

1) A variable like (a, b, c etc.) in the conclusion row need not appear in any of the hypothesis row.
2) Variables may not be necessarily restricted to a single column.

Let us show both the points above with the help of an example each.

**Example 2:** Consider the TD *T* on scheme A B C in *Figure 8*. It is a valid TD expression; please note that the variable c is not appearing in the hypothesis rows where the variables are c’ and c”. This TD has the variable of conclusion row on A and B in the hypothesis rows, but not on C, therefore, is called A B-partial.

| *T* (A | B | C) |
|---|---|---|
| a’ | b | c” |
| a’ | b | c’ |
| a | b’ | c” |
| a | b | c |

**Figure 8: A sample A B-partial TD*T***

A TD*T* on scheme R where every variable in the conclusion row appears in some hypothesis row is termed as *full*. Consider a TD having wl, w2, . . . , wk as the hypothesis rows and w as the conclusion row, a TD is called S-partial, where S is the set defined as: {S ∈ R | w(S) appears in one of w1, w2, . . . , wk ). The TD is full if the S = R and strictly partial if S ≠ R.

Let us now define second difference, but to do so let us first define few more terms. A TD in which each variable appears in exactly one column is called a *typed* TD, but if some variable appears in multiple columns then it is called an *untyped* TD. The TDs shown in *Figures 6* and *8* are typed.

**Example 3:** *Figure 9* shows an untyped TD*T*. This TD assumes that the domain of A is same as that of domain of B, otherwise such TD will not make any sense.

| *T* (A | B) |
|---|---|
| b | c |
| a | b |
| a | c |

**Figure 9: Untyped TD*T***

Let us now show the relationship of JD and MVD to the TD.

**Example 4:** Consider the MVD A →→ B over the relation scheme A B C is equivalent to the TD *T* in *Figure 10* . TD*T indicates* that if a relation has two tuples tl and t2 that agree on A, it must also have a tuple t3 such that t3 (A B) = t1(A B) and t3 (A C) = t2 (A C), which is just a way of stating that the relation satisfies A→→ B.

| *T* (A | B | C ) |
|---|---|---|
| a | b | c' |
| a | b' | c |
| a | b | c |

**Figure 10: A TD*T* for MVD**

However, please note that not every TD corresponds to a JD. This can be ascertained from the fact that their can an infinite number of different TDs over a given relation scheme, whereas there is only a finite set of JDs over the same scheme. Therefore, some of the TDs must not be equivalent to any JD.

## 1.8 DOMAIN KEY NORMAL FORM (DKNF)

The Domain-Key Normal Form (DKNF) offers a complete solution to avoid the anomalies. Thus, it is an important Normal form. A set of relations that are in DKNF must be free of anomalies. The DKNF is based on the Fagin's theorem that states:

"A relation is in DKNF if every constraint on the relation is a logical consequence of the definitions of keys and domains."

Let us define the key terms used in the definition above – *constraint, key* and *domain* in more detail. These terms were defined as follows:

**Key** can be either the primary keys or the candidate keys.

**Key declaration:** Let *R* be a relation schema with $K \subseteq R$. A key *K* requires that *K* be a superkey for schema *R* such that $K \rightarrow R$. Please note that a key declaration is a functional dependency but not all functional dependencies are key declarations.



**Domain** is the set of definitions of the contents of attributes and any limitations on the kind of data to be stored in the attribute.

**Domain declaration:** Let *A* be an attribute and **dom** be a set of values. The domain declaration stated as $A \subseteq$ **dom** requires that the values of *A* in all the tuples of R be values from **dom**.

**Constraint** is a well defined rule that is to be uphold by any set of legal data of R.
**General constraint:** A *general constraint* is defined as a predicate on the set of all the relations of a given schema. The MVDs, JDs are the examples of general constraints.

A general constraint need not be a functional, multivalued, or join dependency. For example, in the student's enrolment number the first two digit represents year. Assuming all the students are MCA students and the maximum duration of MCA is 6 years, in the year 2006, the valid students will have enrolment number that consists of 00 as the first two digits. Thus, the general constraint for such a case may be: "If the first two digit of *t*[enrolment *number*] is 00, then *t*[marks] are valid."

The constraint suggests that our database design is not in DKNF. To convert this design to DKNF design, we need two schemas as:

*Valid student schema =(enrolment number, subject, marks)*

*Invalid student schema = (enrolment number, subject, marks)*

Please note that the schema of valid account number requires that the enrolment number of the student begin with the 00. The resulting design is in DKNF.

Please note that the constraints that are time-dependent or relate to changes made in data values were excluded from the definition of DKNF. This implies that a time-dependent constraint (or other constraint on changes in value) may exist in a table and may fail to be a logical consequence of the definitions of keys and domains, yet the table may still be in DKNF.

How to convert a relation to DKNF? There is no such direct procedure for converting a table into one or more tables each of which is in DKNF. However, as a matter of practice, the effort to replace an arbitrary table by a set of single-theme tables may covert a set of tables to DKNF.

A result of DKNF is that all insertion and deletion anomalies are removed. DKNF represents an "ultimate" normal form because it allows constraints, rather than dependencies. DKNF allows efficient testing of the constraints. Of course, if a schema is not in DKNF, we may have to perform decomposition, but such decompositions are not always dependency-preserving. Thus, although DKNF is an aim of a database designer, it may not be implemented in a practical design.

## 1.9 MODELING TEMPORAL DATA

Let us first define the concept of a temporal database. Broadly speaking, temporal databases are those database applications that involve some aspect of time. That broad definition puts many database applications that use to record the history of database with respect to the time of updating. Such application where time is a very critical factor may include "medical database systems" where the medical history of a patient is to be recorded along with the timestamp. Similarly, for a railway reservation system the time of booking of trains is important to check, if anybody is booking for the train that s/he cannot board. Also the time of cancellation is important as on this basis refunds are calculated. Another example may be the library information system where the book issue and return system is based on time. Many

such systems follow the concept of time. Temporal data adds complexity in a database application and is sometimes overlooked, thus resulting in loss of valuable information.

In a temporal database system you need to model time keeping the following points in mind:

- You need to define database as a sequence of time based data in chronological order.
- You need to resolve events that happen at the same time.
- A reference point of time may be defined to find time relative to it.
- Sometimes a calendar is used.
- The SQL support for temporal data includes:
    - data types such as Date (dd,mm,yyyy,
    - TIME (hh:mm:ss), TIMESTAMP.
      which specifies a unique sequence number, based on time, to identify sequence of events/activities, INTERVAL (time durations) and PERIOD (period frame reference point).
    - You can also define the concept of valid time for a data entity for example an assignment may be valid till a particular time.

### ☞ Check Your Progress 3

1) Define Inclusion Dependencies.

……

……

……

2) What is the key idea behind DKNF?

……

……

……

## 1.10 SUMMARY

In this unit we studied about several normalisation levels. We first discussed enhanced ER tools like Inheritance, generalisation and specialisation. We then defined additional types of dependencies with normal forms. Multivalued dependencies which arise due to a combination of two or more multivalued attributes in the same relation are used to define 4NF. Join dependencies, which results in lossless multiway decomposition of a relation, led us to the definition of 5NF, which is also known as PJNF. We also defined inclusion dependencies used to specify referential integrity and class/subclass constraints and template dependencies, used to specify arbitrary types of constraints. This unit also included a brief discussion on DKNF. You may refer to the further readings for more details on these topics.

## 1.11 SOLUTIONS/ANSWERS

### Check Your Progress 1

1) The EER diagrams are used to model advanced data model requiring inheritance and specialisation and generalisation.

2) The basic constraints used in EER diagrams are disjointness, overlapping and unions.

3) For disjointness and union constraints the chances are that we create separate tables for the subclasses and no table for super class. For overlapping constraint it is advisable to have a table of super class. For such cases the tables of subclasses will have only those attributes that are not common to super class except for the primary key.

## Check Your Progress 2

1) An MVD is a constraint due to multi-valued attributes. A relational must have at least 3 attributes out of which two should be Multi-valued.

2)

EMP_PROJECTS

| ENAME | PNAME |
|---|---|
| DeV | X |
| Dev | Y |

EMP_DEPENDENTS

| ENAME | DNAME |
|---|---|
| Dev | Sanju |
| Dev | Sainyam |

3)

No,

R1

| SNAME | PARTNAME |
|---|---|
| Dev | Bolt |
| Dev | Nut |
| Sanju | Bolt |
| Sainyam | Nut |
| Sanju | Nail |

R2

| SNAME | PROJNAME |
|---|---|
| Dev | X |
| Dev | Y |
| Sanju | Y |
| Sainyam | Z |
| Sanju | X |

R3

| PARTNAME | PROJNAME |
|---|---|
| Bolt | X |
| Nut | Y |
| Bolt | Y |
| Nut | Z |
| Nail | X |

4) A set of functional dependencies F is minimal if it satisfies the following conditions:

- Every dependency in F has a single attribute for its right-hand side.
- We cannot replace any dependency $X \rightarrow A$ in F with Dependency $Y \rightarrow A$, where $Y$ is a proper subset of $X$ and still have a set of dependencies that is



equivalent to *F*.

- We cannot remove any dependency from F and still have a set of dependencies that is equivalent to F.

5) 1. $X \rightarrow Y$ and 2. $Y \rightarrow Z$ both holds in a relation r. Then for any two tuples t1 and t2 in r such that t1[X] = t2[X], we must have 3. t1[Y] = t2[y], from assumption 1; hence we must also have 4. t1[Z] = t2[Z], from 3 and assumption 2; hence X $\rightarrow$ Z must hold in r.

### Check Your Progress 3

1) An inclusion dependency R.X < S.Y between two sets of attributes – X of a relation schema R, and Y of a relation schema S – is defined as the following constraint:

   If r and s are the relation state of R and S respectively at any specific time then:

   $\prod_X(r(R)) \subseteq \prod_Y(s(S))$

   The subset relationship does not necessarily have to be a proper subset. Please note that the sets of attributes on which the inclusion dependency is specified viz., *X* of *R* and *Y* of *S* above, must have the same number of attributes. In addition, the domains for each corresponding pair of attributes in *X* and *Y* should be compatible.

2) To specify the "ultimate normal form".

## 1.12 FURTHER READINGS


1) *"Fundamentals of Database System"*, Elmasri, Ramez and Navathe Shamkant B., Forth Edition, Pearson Education, India, 2004.

2) *"Database System Concepts"*, Silberschatz A., Korth Henry F., S.Sudarshan, Fifth Edition, McGraw Hill, 2006.

3) Some very useful websites:

   http://www.dbis.informatik.hu-berlin.de/~freytag/Maier/C14.pdf
   http://wwwdb.informatik.uni-rostock.de/Lehre/Vorlesungen/MAIER/C07.pdf
   http://codex.cs.yale.edu/avi/db-book/online-dir/c.pdf
   http://www-staff.it.uts.edu.au/~zhangsc/scpaper/AJITzzqin.pdf

# UNIT 2 DATABASE IMPLEMENTATION AND TOOLS

**Structure** **Page Nos.**


2.0 Introduction 30
2.1 Objectives 30
2.2 Information System and Organisation 30
2.3 Database Design and Implementation 33
2.3.1 The Database Application Life Cycle
2.3.2 Conceptual Database Application Design
2.3.3 Logical and Physical Database Design
2.4 UML Diagrams: An Aid to Database Design Specifications 39
2.4.1 UML Class Diagrams and Database Design
2.4.2 UML Based Design Tools
2.5 Automated Database Design and Implementation Tools 43
2.6 Summary 44
2.7 Solutions/Answers 45


## 2.0 INTRODUCTION

This unit provides a bridge between the database design principles and their implementation in an organisation. This unit first briefly discusses the role of database and information systems in an organisation. During this discussion we will also try to recollect what we have learnt about the development of information base. Thereafter, the steps of database design and the implementation of this design will be discussed. UML has now become one of the important tools in software system modeling and implementation. Databases are generally, an interim part of most systems. Thus, this unit also discusses UML based tools and their relationship to database design and implementation. Finally we will be introducing some of the automated database design tools that automatically implement the databases from the design.

## 2.1 OBJECTIVES

After going through this unit, you should be able to:

- define the role of database in an organisation's information system;
- relate database design process in an organisation to its implementation;
- relate UML diagram to database design and implementation, and
- identify some automatic design and implementation tools.

## 2.2 INFORMATION SYSTEM AND ORGANISATION

For any organisation, information provides the key to success. In the present day world, information technology and management of information resources is very important for the success of an organisation. It is said that an organisation today cannot last for more than 48-72 hours if its information resources have failed. The basis of information in any organisation is the data. The information and the data have the following properties:

- Information is one of the most important corporate resources. Data should be properly controlled and managed for the effective working of the organisation.



- The need of up-to-date data is ever increasing as more and more functions of organisations are being made electronic.
- The complexity of applications is increasing, as complex relationships among the data are needed to be modeled and maintained.
- The long lasting data may be cleaned, consolidated and archived for future decision making.
- Information is very useful for future planning. It provides the major input for supply chain management, enterprise resource planning, financial control and many other managerial functions of an organisation.
- For the day-to-day financial, retail good transactions, a publicly accessible and updatable operational database must be designed and made available for the transactions.
- Electronic selling and purchasing is cutting the cost of business transactions.

Information, which is generated from data, is a very important aspect for survival of an organisation. But, why is the Database Management System important for generating information for an organisation? The following are the major characteristics of database management that make it very valuable for an organisation.

- **Data Independence:** DBMS protects data base application programs from the changes in the logical database design and the physical database structure including access paths and storage structures.
- **Controlled Data Sharing:** DBMS allows the same data to be used for multiple applications with each application having its own **view** of the data. This data of views can also be suitably protected.
- **Simple Operation and Query Mechanism:** SQL provides a very simple style query language across the DBMS. Easy data handling, support for transactions, and security are the other major features of a DBMS. DBMS also has the extended feature of knowledge generation. Thus, DBMS are major tools for an organisation.

Thus, the database is typically a part of the information system. But, what are the activities required in an information system?

(a) Collection of data.
(b) Management of data using DBMS software.
(c) Use of the hardware and storage Media.
(d) Dissemination of the information Resources to the persons who use and manage the data. (Also known as DBA).
(e) The Application software that accesses and updates the data, and the application programmers.

We can easily conclude that the Database system is part of a much larger organisational information system.

**Information System Development**

The information system development follows the complete Software Development Life Cycle (SDLC). The information System Life Cycle is quite fuzzy for information systems where databases are a major integral component. What does it typically include? Well typically it includes all the basic phases involved in the waterfall model of SDLC.

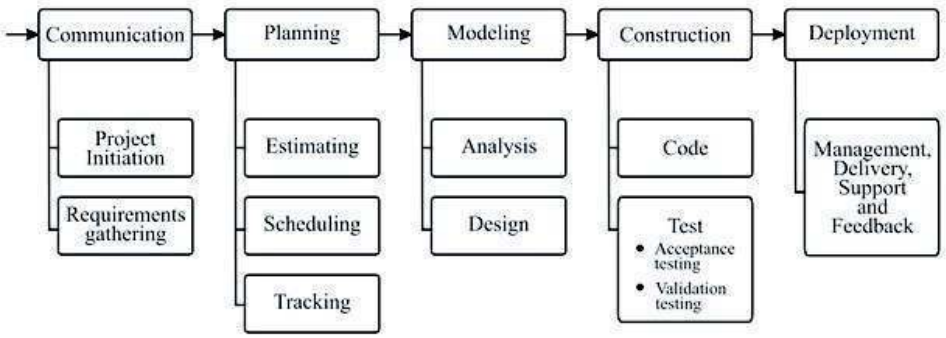



**Figure 1: The information system life cycle**

Let us define these phases briefly.

**(1) Communication:** This basically includes:

- Analysing potential application areas.
- Identifying the economics of information.
- Performing preliminary cost-benefit studies.
- Determining complexity of data and processes.
- Setting up priorities among applications.
- Gathering detailed requirements by interacting with users and user groups.-
- Identification of inter-application dependencies, communication and reporting procedures.

**(2) Planning:** This basically includes:

- Estimating the cost and time.
- Estimating the resource requirements.
- Identification of step-by-step (micro) procedure for each phase of SDLC.
- Scheduling of resources as per the needs.
- Tracking the resources as the project progresses.

**(3) Modeling:** Now let us see what we do in modeling:

- We create and refine analysis and design models.
- The information system is designed, it includes:
  - Design of the database system and the design of Application systems.
  - Design of information needs at different levels, system architecture design.
  - Forms and report design.
  - Design of Database Transactions and processes in detail.

**(4) Construction:** Construction mainly includes:

- Coding and implementing the system as per the detailed design.
- Creation of database and loading some sample data.
- Implementation of database transactions.
- Testing of all the above.
- Checking acceptability of the system in meeting user's requirements and performance criteria.
- The system is tested against performance criteria and behaviour specifications.

**(5) Deployment:** This is the last stage but necessarily an important one:

- Operational phase checks whether all system functions are operational and have been validated.
- New requirements and applications are validated and tested.
- Maintenance and monitoring of system performance is an important activity done here.

Thus, information system development is a major system development life cycle. However, in this unit we will focus our attention mainly on database application system development.

### ☞ Check Your Progress 1

1) Why is the DBMS most suitable for Information System implementation?

……

……

……

2) Which of the information System Life Cycle stages are important for database design?

……

……

……

## 2.3 DATABASE DESIGN AND IMPLEMENTATION

The database application design activity in an organisation is very important. This activity determines the overall quality of the information system.

The database application design involves database design and application system design. The *Figure 2* shows the steps of these processes and the linkages for database application design and implementation.

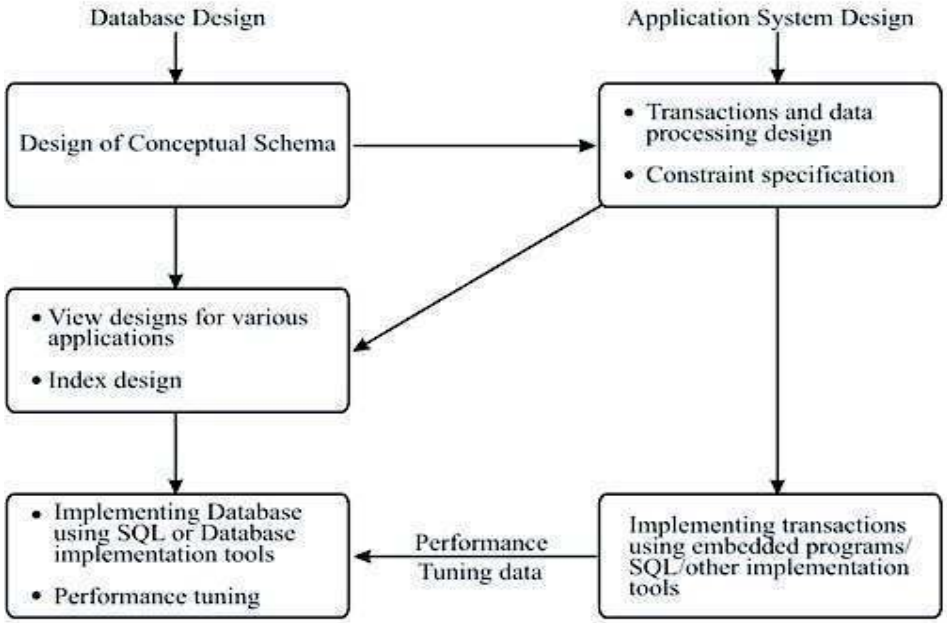



**Figure 2: Database application system design and implementation**

### 2.3.1 The Database Application Life Cycle

The database application life cycle defines the phases of development of a database application. Let us list the activities that help us to create a database:

**(1)** **System definition:** In this activity, the scope of the database system, its users and its applications are defined with the help of users information. Other activities of this phase are:

- to find the interfaces for various categories of users.
- to find the response time constraints.
- to identify the storage and processing needs.

Before we discuss the further activities, it is necessary to know that this part of the database application design and implementation is part of the information system life cycle.

**(2)** **Design of Conceptual Database Application:** It involves design of the complete conceptual schema of the database and the transaction and Data processing. We will discuss it in more detail in the next sub-section.

**(3)** **Logical and Physical Database Design:** This involves creation of views, creation of storage structure, indexes etc. This is also discussed in more detail later.

**(4)** **Database Implementation:** This is the process by which we implement the following:

- Conceptual Database
- External Database
- Internal Database
- The mappings among these designs
- Creating empty database files
- Implementing the software applications.

**(5)** **Application Conversion:** Any software application from a previous system is converted to the new system. The data is populated either by loading the data directly or by converting existing files into the database system format.

**(6)** **Testing and Validations:** The new system, which is made needs to be tested and validated for good results. Testing and validation is done for this particular purpose.

**(7)** **Operations:** During this phase, the old as well as the new systems are operated in parallel.

**(8)** **Monitoring and Maintenance:** Change is an unchanging phenomenon. Similarly, modifications and reorganisation are unavoidable and may be needed from time to time. During this phase, the system is constantly monitored and maintained.

Let us now discuss the two major stages as above – viz., conceptual Database Application Design and Logical and Physical Design in more detail. Performance tuning is a specialised area and DBMS specific so this topic has not been discussed in this unit.

### 2.3.2 Conceptual Database Application Design

The process of conceptual database design is given in *Figure 3*.

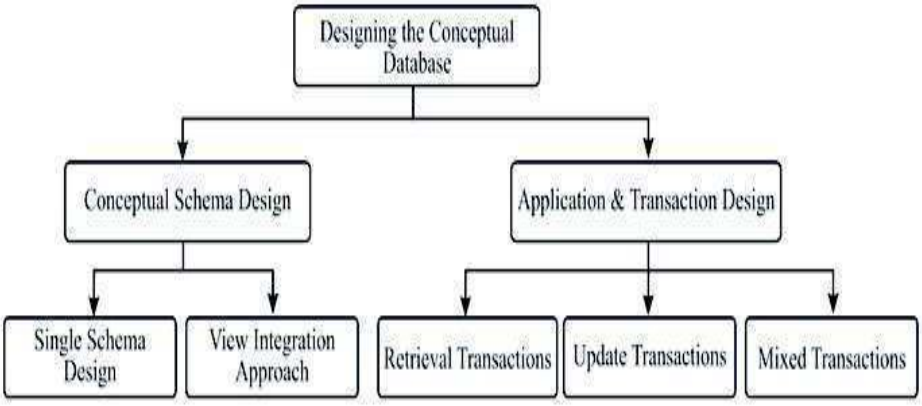



**Figure 3: The conceptual database design**

**Conceptual Schema Design**

The following details are produced during the conceptual database design:

- Database structures and data types associated with each field.
- Keys and constraints.
- Data interrelationships such as referential constraints
- Data dictionary.

But how do we do the conceptual schema design?

The process of conceptual schema design requires primarily modification and conversion of E-R diagram into tables keeping the following points in consideration:

- Proper identification of entities, relationship, and attributes
- Identification of key attributes
- Identification of cardinality of relationship
- Identifying weak entities, specialisation/generalisation, etc.

We have already discussed how an ER diagram is converted into tables in MCS-023 Block-1, Unit-2. Let us discuss the two common approaches for conceptual schema design:

(a) Single Schema Design

(b) View Integration

**(i) Single Schema Design**: This process results in the development of single schema. This process is explained with the help of an example:

Consider the following EER diagram of a University:

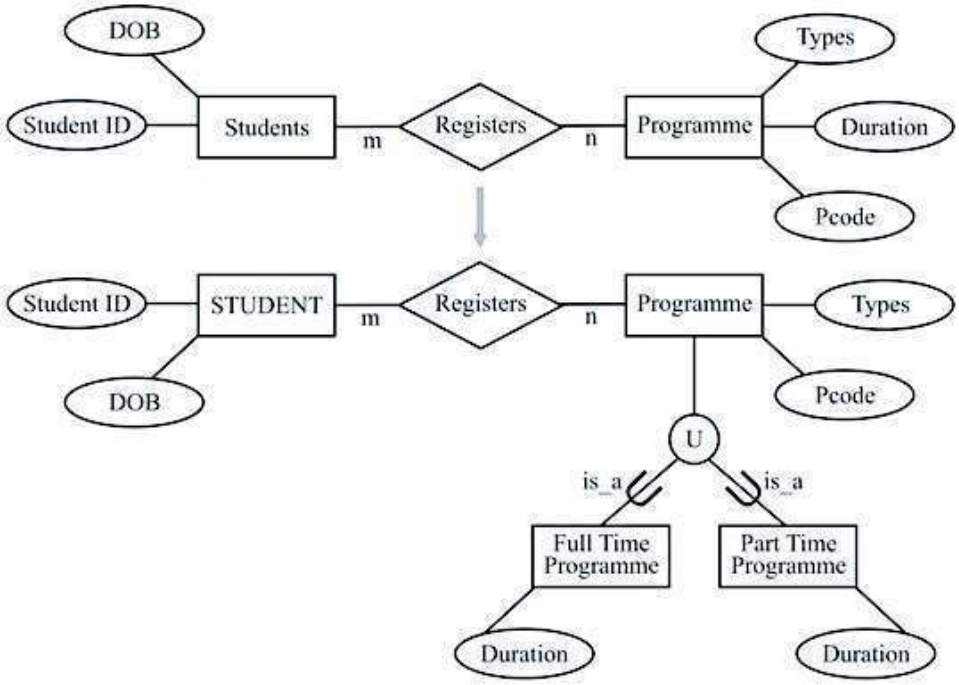



**Figure 4: Design of database**

Programme Types are: Full time or part time.

Please note the ER diagram is further refined to have an 'is-a' relationship. The simple conceptual tables can now be designed in such case as:

STUDENT (st-id, dob)
FULL-TIMEPROG (p-code, duration)
PART-TIMEPROG (p-code, duration)
REGISTERS (p-code, st-id)

Please note that the design above is not complete; a complete design would require all types defined and constraints identified. All such information is made available by the detailed analysis document.

**(ii) View Integration:** A commercial database may be very complex. It may be a good idea to model this database as per the viewpoints of various stakeholders, thus creating many views. A conceptual schema may be an integration of all such views. Such integration may have the following steps:

- Find out the common entities and relationships of interest.
- Find out if any conflicting requirements exist, such as name, type, constraints etc.
- Modify and manage the views.
- Refine the final model, if necessary.

We do not attempt a detailed discussion of this topic here, since it is beyond the scope of this unit.

**Transaction Design**

Transaction design specifies the functional requirements of a database system. The three types of transactions that may need to be designed are:



**(i) Retrieval Transaction:** These transactions will not update data in a database but help to access some information, for example, finding the status of a booking in a Railway reservation database. Such transactions can be executed concurrently without any problems.

**(ii) Updating Transactions:** Such transaction modifies some data value and requires careful concurrency control. For example, booking a ticket in a railway reservation system would update a lot of data items.

**(iii) Mixed Transactions:** A third type of transaction may have both these characteristics, that is, it reads some data and also updates some data. These are called **mixed transactions.**

The dataflow models created during the modeling phase help in identifying the transactions / procedure etc. on a database.

**Selecting the desirable DBMS**

Once we are through with conceptual design, we need to make a choice of the DBMS. Some of the factors on which the selection depends are:

**(1) Technical:** The technical issues we need to consider:

- Type of DBMS (Relational, object-relational etc.)
- Storage Structures and Access paths supported by DBMS.
- The simplicity of user/ programmer interface.
- Support for SQL.
- Availability of development Tools.
- Ability to interface with other DBMS like conformance to ODBC.
- Support for client-server architecture.

**(2) Non-Technical:** These include factors like cost and status of the organisation, which is supplying the product. The following cost factors may be considered while choosing DBMS:

- **Software Acquisition Cost:** It not only includes the cost of software, but also any utility cost, development tool cost or any other cost that the selling organisation may charge.
- **Additional Hardware Acquisition Cost:** If the DBMS requires additional memory, disk controllers, etc. then it must be included in the overall cost calculation.
- **Maintenance Cost:** The cost for maintaining and keeping the database perfect.
- **Operating Cost:** The cost related to continuous operation of the DBMS. It includes hiring cost of personnel employed only for DBMS such as DBA. It may also include the training cost.

### 2.3.3 Logical and Physical Database Design

The next phase aims to create an external schema and physical schema, including the mapping. Some of the important considerations at this stage of database design are:

- The database views should be identified.
- The security considerations and constraints on the view should be identified and duly specified.
- The performance criteria for views may be identified.
- The physical database design which includes specific storage structure (e.g., clustered file organisation) and access paths (indexes etc.) may be identified.

But why is physical database design required?

A high-performance *transactions processing systems* requires continuous operation of the system. Transactions performance, in terms of the average number of transactions per minute and maximum transaction response time, is critical. Hence, a careful physical database design that meets the organisation's transaction processing needs is a must.

Why do we choose a specific physical structure?

We do it so that the database applications can achieve good performance. The physical database design process is restricted to choosing the most appropriate structure for the database files from among the options offered by that DBMS. Let us discuss the criteria on which the choice of physical database design lie:

**(1) Response Time:** The elapsed time between the start of a database transaction to receive a response on it.

Please note that response time is also a factor of non-DBMS controlled issues like:

- Access Time of the database
- Present user Load
- Scheduling operation of operating system
- Communication Delays on a client-server system etc.

**(2) Space Utilisation:** This is the amount of storage space used by the database files including index. It also includes the storage of the physical database on the disk.

**(3) Transaction Throughput:** The average number of transactions that are processed per minute. It is measured under peak conditions on the system. This is very important in applications like Railway reservations or banking.

The outcome of the physical database design phase is an *initial* determination of storage structures and access paths for the database files. These structures do change in the lifetime of a database.

## ☞ Check Your Progress 2

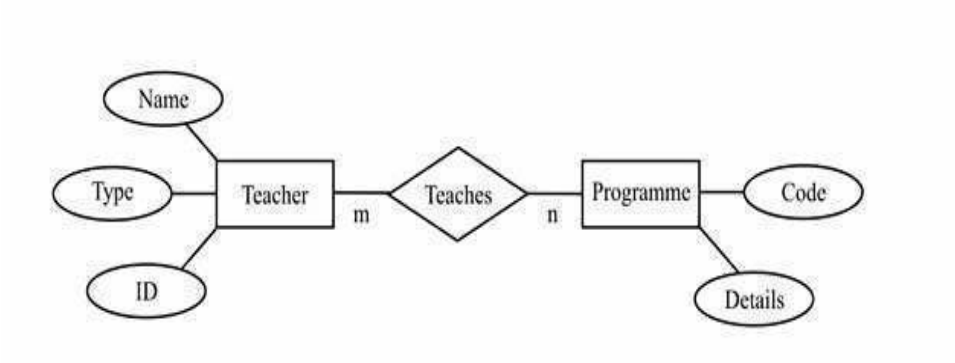
1) What are the steps for design of database system?

……

……

……

2) Consider the following EER diagram.





'Type' can be regular or visiting faculty. A visiting faculty members can teach only one programme. Make a suitable database application design for this.

……

……

……

3) What are the criteria for physical design?

……

……

……

## 2.4 UML DIAGRAMS: AN AID TO DATABASE DESIGN SPECIFICATIONS

First of all, before defining UML, we should first ask the question why UML? When there are so many database implementation tools/ languages available why has UML become so popular?

Let us consider this issue in detail.

Databases are an integral part of any information system and there exists many database implementation tools. Thus, the industry is in need of a standard approach that spawns all the phases of SDLC viz., requirement analysis, modeling, design, implementation and deployment. Also, the industry is looking for techniques to automate the production of software having an improved quality and reduced cost and time.

Since UML is accepted by the Object Management Group (OMG) as the standard for modeling object-oriented programs it becomes one of the automatic choices. Now the next question is why only UML? UML has quickly become one of the popularly used tools for modeling business and software application needs. The UML became popular due to the following reasons:

1) It is very flexible. It allows for many different types of modeling. Some of them include:

   - Business Process modeling event workflow,
   - Sequencing of events,
   - Defining database architecture etc.

2) It is language and platform - independent. It allows software architects to model any application, on any operating system in any programme language or network.

3) UML supports object-oriented methodologies. Please note however, that even if the concepts are based on object-oriented methods, the resulting structure and behaviour of UML can be used to design both relational and object-oriented models.

4) The integration of various SDLC stages through UML tools has brought the analysts, modeler, designers, and the software application developers closer to each other.

### Types of UML Diagrams

You have studied UML diagrams in course MCS-032. Let us briefly recapitulate those concepts again. *Figure 5* defines the various types of UML diagrams.

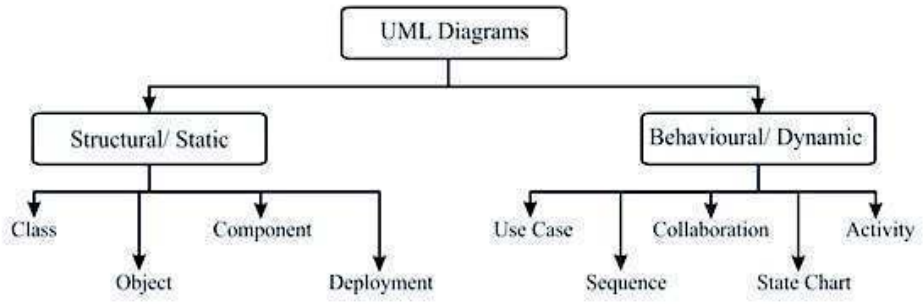



**Figure 5: The UML diagrams**

**Structural/Static Diagrams:** These diagrams are used to model the structural or static relationship among various objects or their components.

| Class | Object | Component | Deployment |
|---|---|---|---|
| * backbone of almost every object-oriented method, including UML.<br>* describes the static class structure of a system viz., classes, interfaces, dependencies etc. | * subset of class diagrams, sometimes treated as separate techniques.<br>* organises elements of a system into related groups to minimise dependencies. | * describes the organisation of physical software components including source code, run time code and executables. | * depicts the physical resources in a system<br>* includes nodes, components and connection. |

**Figure 6: The structural/static diagrams**

**Behavioral/dynamic Diagrams:** These diagrams are used to model the behavioural or dynamic relationship among various objects or their components.

| Use Case | Sequence | Collaboration | State Chart | Activity |
|---|---|---|---|---|
| * models the functionality of system using Actors and use cases. | * describes interaction between classes in terms of an exchange of messages over time. | * represents interactions between objects as a sequence of messages.<br>* describes both static structure and dynamic behaviour | * describes the dynamic behaviour of a system in response to external stimuli.<br>* useful in modeling objects where states are triggered by specific events. | * defines an activity representation operation on some class in the system that results in a change in the state of the system. |

**Figure 7: The behavioural/dynamic diagrams**

**Use of UML Diagrams:** During the construction of an application system design we use all the major UML diagrams. But how are these diagrams useful for the database design.

Let us first discuss the behavioural/dynamic diagrams of UML. The use Case Diagrams are used to gather information during the requirement analysis phase. Likewise the sequence diagram is used to visualise the execution of the use cases, while the collaboration diagram defines the sequence of messages. The state Chart Diagram and activity diagram are used to graphically represent some of the states and activities of a system.

Thus, these diagrams are very useful for providing the behavioural or functional information about databases. Thus, they may be useful for transaction and processing design. For example, a use case diagram of a student of an on-line University may be:

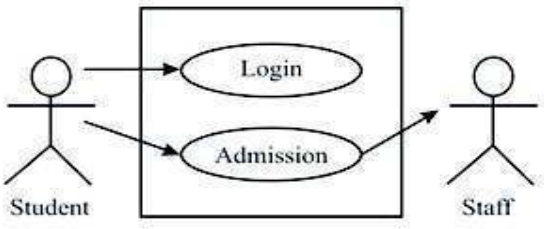



**Figure 8: A simple use case diagram for a university**

This use case diagram is indicating that the student and staff may have to login first, thus, the implementation of such a system would require a table for user-name and password and for taking admission a student may need to have a valid user name and password. It also indicates that the process of admission will be certified by a staff member, who will allow the student to take admission after due checking. Thus, UML behavioral diagrams may be useful to define processing/transactions. They also provide some information about entities and attributes. However, the UML diagram is closely associated with database structure design in the class diagram. Let us discuss these diagrams in more detail in respect of database design and implementation.

### 2.4.1 UML Class Diagrams and Database Design

To begin with, let us first recollect what we have learnt about class diagrams. The class diagrams sometimes may be considered as an alternative notation to E-R diagram. In UML class diagram contains:

- 'Class' – is displayed as a box. It has three sections:

  "Top Section" – Class Name
  "Middle Section" – Attributes
  "Last Section" – Operations
- Associations: Relationship Types are called Associations.
- Links: Relationship instances are called Links.
- Binary Association: Represented as a line connecting the participating classes.
- Linked Attribute: Connected to the Association's line by dashed line.
- Multiplicities: The (Minimum and Maximum) constraints.
- Reflexive Association: A Recursive Relationship.
- Aggregation: To represent a relationship between a whole object and its component parts.
- Different unidirectional and bi-directional Associations.

The specialisation/generalisation can be represented in UML class diagrams as:

- Basic notation is to connect the subclass by vertical lines to horizontal lines.
- A blank triangle indicates a specialisation /generalisation with the **disjoint constraint** and a filled triangle indicates an overlapping constraint.
- The root super class is known as the base class and the leaf nodes are called **terminal nodes**.

Let us explain the role of class diagrams with generalisation/specialisation hierarchy in a database structure design with the help of an example. Consider the following UML class diagram for a University.

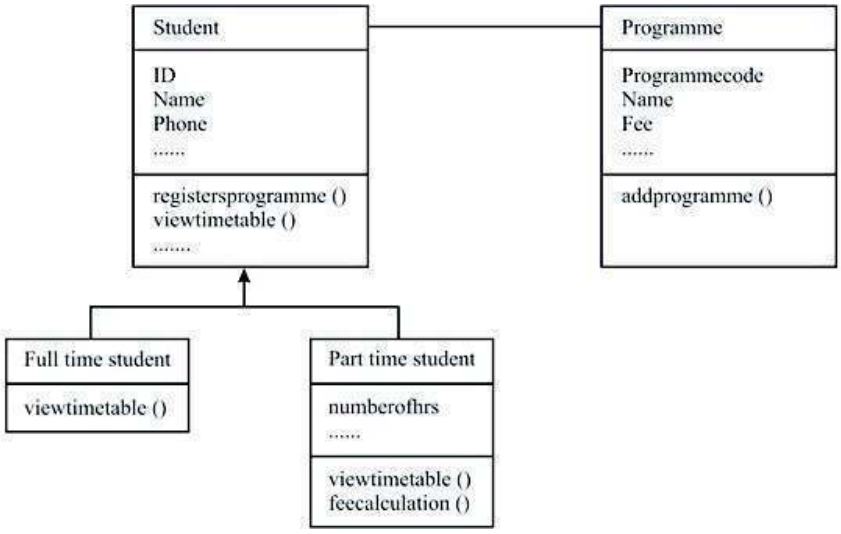



**Figure 9: A UML class diagram**

Please note that in the diagram above, you can make clear-cut table design. One such possible design may be:

STUDENT (ID, Name, Phone, type (PT, FT))
PT (ID, numberofhrs)
PROGRAMME (Programmecode, Name, Fee)
Stuprog (ID, Programmecode)

You can also identify several functions/data processing functions/ triggers relating to functions of classes. For example, feecalculation( ) may need to be implemented as a stored procedure while calculating the fee of part-time students, whereas addprogramme( ) is a simple data entry function. Thus, UML Class diagram with generalisation/specialisation is a very useful database design tool.

### 2.4.2 UML Based Design Tools

There are many tools used in the industry to develop information system. These tools provide the initial specification in UML that eventually leads to the database development. These tools provide support for conceptual, logical and physical database modeling design. Some of the features of these tools are:

- Support for Reverse Engineering: These tools allow the user to create a data model based on the already existing database structure.

- Forward Engineering and creation of database definitions: The tools can read the schema of the data models available in UML diagram – class diagram and create the database and the data storage model and generate appropriate DDL code for the same.

- Conceptual Design in UML notation: It allows modeling of databases using UML notations.

You must search for different tools and the facilities provided by them on the World Wide Web.

## 2.5 AUTOMATED DATABASE DESIGN AND IMPLEMENTATION TOOLS

Database design was initially handled manually. But as the complexity and volume of data increased, automation of the same became essential. So, why was Database design automated? The reasons are:

- As the complexity of a system increases, the number of options or in other words different design models that can represent the information model also increase rapidly. Hence it becomes difficult to deal with the complexity and the corresponding design alternatives manually.

- The size and volume of some databases may contain hundreds of entity types and relationship types. Manually managing such a design becomes extremely difficult.

Thus, because of the above two reasons, the Database Design Process had to be automated.

However, database design is not a process in isolation and as discussed earlier, database application design is part of SDLC. Thus, database design tools are part of tools or commercial database management system development environment.

Some of the essential facilities that need to be provided by the database design tools are:

**(1) Facility for drawing a conceptual schema Design:** It basically allows the database designers to represent the database diagrammatically. Thus, a tool must support the representation of the following:

- Entity Types
- Relationship Types
- Cardinality Constraints
- Attributes, Keys, foreign keys etc.
- Showing inheritance hierarchies or generalisation/specialisation hierarchy.

The diagrams should be stored and should allow modifications whenever necessary.

**(2) Allows Model Mapping:** Mapping Algorithms for external schemas, security etc. should be creatable and implementable. Such mapping is mostly system-specific.

**(3) Support of Normalisation**: Normalisation process uses a set of functional dependencies. Such FDs should be represented with the help of this tool at the time of logical design.

What should be the characteristics of good design tools?

A good design tool must have the following features:

**(1) An easy Interface:** Graphical user interfaces are commonly used by such tools so that the designer can concentrate more on complex high level abstraction and concepts



rather than lower level implementations. Different interfaces may be tailored to suit beginners and expert designers.

**(2) Analytical Capabilities:** Such tools need to provide analytical capabilities for tasks that are difficult to perform manually. One such capability may be evaluating design alternatives. Such capabilities may also detect conflicting constraints among views. The designer should also be able to compare multiple alternatives. This helps in the process of physical design in a given system.

**(3) Display of Design:** Some of the features expected here are:

- diagrammatic display results such as schemas representation.
- well laid out diagrams (easily creatable automatically).
- easy to read design.
- the design should be automatically implemented as tables, lists, or reports.

**(4) Support for Design Verification:** The tool should enable verification of the resulting design. It should allow checking whether the design satisfies the initial requirements. We have avoided detailing any proprietary tool here, but you can refer to websites for many such tools.

The CASE tools address both schema design and application design concurrently.

### ☞ Check Your Progress 3

1) Which of the UML diagrams help in database schema design the most and why?

……

……

……

2) What are the various UML based design tools?

……

……

……

3) What are the features of automated database design and implementation tools?

……

……

……

## 2.6 SUMMARY

In this unit, we have discussed the database implementation and tools, Information system and organisation, role of UML Diagrams in database design and Automated Database design tools.

The database management systems are used to implement the information system in an organisation. The Information systems are developed by following the complete SDLC phases. The database application design and implementation involves the conceptual database design, physical and logical design, procedural design and implementation. We have briefly reviewed the use of UML diagrams in different categories such as structural/static and behavioural/dynamic. We have also explained how the class diagrams with specialisation/generalisation hierarchies can be used to

design database structure. We also gave an idea to the learner with different types of database design and implementation tools.

## 2.7 SOLUTIONS/ANSWERS

### Check Your Progress 1

1) The following characteristics make DBMS a very valuable tool for information systems:

- Data independence
- Data sharing under secure environment
- Simple query mechanism
- Support for transaction and recovery
- Availability of tools for knowledge extraction.

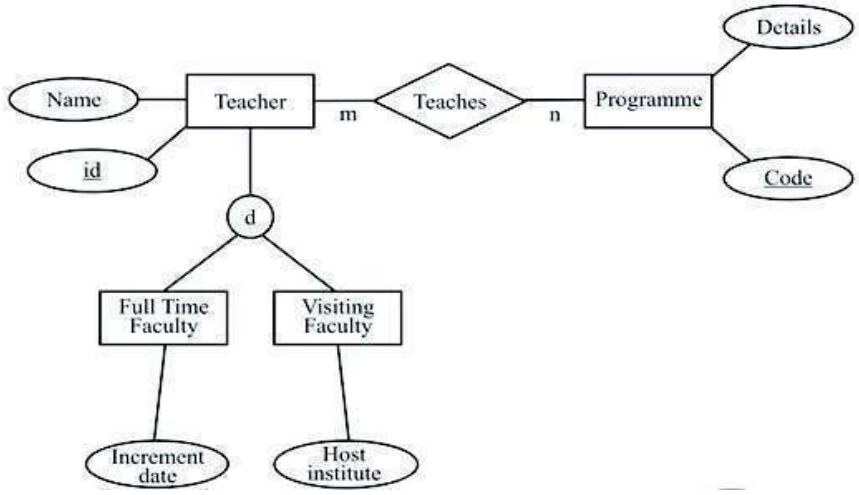
2) The stages where a lot of useful input is generated for database design are communication and modeling.

### Check Your Progress 2

1) Design steps:

- Conceptual Schema Design
- Logical and physical schema design
- Data processing and transaction design.

2) The modified EER diagram is:




TEACHER (id, Name, Increment-date)
VISITING (id, Name, Host-institute)
TEACHES (id, code)
PROGRAMME (code, details)

Transaction/constraints: The **id** in TEACHES, if exists in id in visiting then COUNT on 'id' in teaches should not exceed 1.

3) The criteria are:

- Response time
- Space utilisation
- Transaction throughput.



## Check Your Progress 3

1) The class diagram – it resembles ER diagram and also shows the possible transactions. Thus, it is useful for schema design as well as application design.

2) There are many open sources and proprietary UML based design tools available. You need to find them from a website.

3) They should have facilities for:

- Drawing conceptual schema
- Mapping implementation
- Normalisation
- Simple User interaction and displays
- Task analysis and
- Design verification.

# UNIT 3 ADVANCED SQL

**Structure** **Page Nos.**


3.0 Introduction 47
3.1 Objectives 47
3.2 Assertions and Views 47
3.2.1 Assertions
3.2.2 Views
3.3 Embedded SQL and Dynamic SQL 51
3.3.1 Embedded SQL
3.3.2 Cursors and Embedded SQL
3.3.3 Dynamic SQL
3.3.4 SQLJ
3.4 Stored Procedures and Triggers 58
3.4.1 Stored Procedures
3.4.2 Triggers
3.5 Advanced Features of SQL 61
3.6 Summary 62
3.7 Solutions/Answers 62


## 3.0 INTRODUCTION

The Structured Query Language (SQL) is a standard query language for database systems. It is considered as one of the factors contributing to the success of commercial database management systems, primarily, because of the availability of this standard language on most commercial database systems. We have described SQL in details in the course MCS-023 Block-2, Unit-1 where we discussed data definition, data manipulation, data control, queries, joins, group commands, sub-queries, etc.

In this unit, we provide details of some of the advanced features of Structured Query Language. We will discuss Assertions and Views, Triggers, Standard Procedure and Cursors. The concepts of embedded and dynamic SQL and SQLJ, which is used along with JAVA, are also been introduced. Some of the advanced features of SQL have been covered. We will provide examples in various sections rather than including a separate section of examples. The examples given here are in a SQL3 standard and will be applicable for any commercial database management system that supports SQL3 standards.

## 3.1 OBJECTIVES

After going through this unit, you should be able to:

- define Assertions and explain how they can be used in SQL;
- explain the concept of views, SQL commands on views and updates on views;
- define and use Cursors;
- discuss Triggers and write stored procedures, and
- explain Dynamic SQL and SQLJ.

## 3.2 ASSERTIONS AND VIEWS

One of the major requirements in a database system is to define constraints on various tables. Some of these simple constraints can be specified as primary key, NOT NULL, check value and range constraints. Such constraints can be specified within



the data or table creation statements with the help of statements like NOT NULL, PRIMARY KEY, UNIQUE, CHECK etc. Referential constraints can be specified with the help of foreign key constraints. However, there are some constraints, which may relate to more than one field or table. These are called assertions.

In this section we will discuss about two important concepts that we use in database systems viz., views and assertions. Assertions are general constraints while views are virtual tables. Let us discuss about them in more detail.

### 3.2.1 Assertions

Assertions are constraints that are normally of general nature. For example, the age of the student in a hypothetical University should not be more than 25 years or the minimum age of the teacher of that University should be 30 years. Such general constraints can be implemented with the help of an assertion statement. The syntax for creating assertion is:

Syntax:

CREATE ASSERTION <Name>

CHECK (<Condition>);

Thus, the assertion on age for the University as above can be implemented as:

```
CREATE ASSERTION age-constraint
CHECK (NOT EXISTS (
                SELECT *
                FROM STUDENT s
                WHERE s.age > 25
                OR s.age > (
                        SELECT MIN (f.age)
                        FROM FACULTY f
                ));
```

The assertion name helps in identifying the constraints specified by the assertion. These names can be used to modify or delete an assertion later. But how are these assertions enforced? The database management system is responsible for enforcing the assertion on to the database such that the constraints stated in the assertion are not violated. Assertions are checked whenever a related relation changes.

Now try writing an assertion for a university system that stores a database of faculty as:

FACULTY (code, name, age, basic salary, medical-allow, other benefits)

MEDICAL-CLAIM (code, date, amount, comment)

Assertion: The total medical claims made by a faculty member in the current financial year should not exceed his/her medial allowance.

```
CREATE ASSERTION med-claim
CHECK (NOT EXISTS (
        SELECT code, SUM (amount), MIN(medical-allow)
        FROM (FACULTY NATRUAL JOIN MEDICAL-CLAIM)
        WHERE date>"31-03-2006"
        GROUP BY code
        HAVING MIN(medical-allow) < SUM(amount)
));
```

**OR**

```
CREATE ASSERTION med-claim
CHECK (NOT EXISTS (
        SELECT  *
        FROM FACULT f
        WHERE (f.code IN
        (SELECT code, SUM(amount)
        FROM MEDICAL-CLAIM m
WHERE date>"31-03-2006" AND f.code=m.code AND
f.medical-allow<SUM(amount)
```

Please analyse both the queries above and find the errors if any.

So, now you can create an assertion. But how can these assertions be used in database systems? The general constraints may be designed as assertions, which can be put into the stored procedures. Thus, any violation of an assertion may be detected.

### 3.2.2 Views

A view is a virtual table, which does not actually store data. But if it does not store any data, then what does it contain?

A view actually is a query and thus has a SELECT FROM WHERE ….. clause which works on physical table which stores the data. Thus, the view is a collection of relevant information for a specific entity. The 'view' has been introduced as a topic in MCS-023, Block 2, Unit-1. Let us recapitulate the SQL commands for creating views, with the help of an example.

Example: A student's database may have the following tables:

STUDENT (name, enrolment-no, dateofbirth)
MARKS (enrolment-no, subjectcode, smarks)

For the database above a view can be created for a Teacher who is allowed to view only the performance of the student in his/her subject, let us say MCS-043.

```
CREATE VIEW SUBJECT-PERFORMANCE AS

        (SELECT  s.enrolment-no, name, subjectcode, smarks
        FROM STUDENT s, MARKS m
        WHERE s.enrolment-no = m.enrolment-no AND
        subjectcode 'MCS-043' ORDER BY s.enrolment-no;
```

A view can be dropped using a DROP statement as:

```
DROP VIEW SUBJECT-PERFORMANCE;
```

The table, which stores the data on which the statement of the view is written, is sometimes referred to as the base table. You can create views on two or more base tables by combining the data using joins. Thus, a view hides the logic of joining the tables from a user. You can also index the views too. This may speed up the performance. Indexed views may be beneficial for very large tables. Once a view has been created, it can be queried exactly like a base table. For example:

```
SELECT   *
FROM  STUDENT-PERFORMANCE
WHERE   smarks >50
```

How the views are implemented?



There are two strategies for implementing the views. These are:

- Query modification
- View materialisation.

In the query modification strategy, any query that is made on the view is modified to include the view defining expression. For example, consider the view STUDENT-PERFORMANCE. A query on this view may be: The teacher of the course MCS-043 wants to find the maximum and average marks in the course. The query for this in SQL will be:

SELECT MAX(smarks), AVG(smarks)
FROM SUBJECT-PERFORMANCE

Since SUBJECT-PERFORMANCE is itself a view the query will be modified automatically as:

SELECT MAX (smarks), AVG (smarks)
FROM STUDENT s, MARKS m
WHERE s.enrolment-no=m.enrolment-no AND subjectcode= “MCS-043”;

However, this approach has a major disadvantage. For a large database system, if complex queries have to be repeatedly executed on a view, the query modification will have to be done each time, leading to inefficient utilisation of resources such as time and space.

The view materialisation strategy solves this problem by creating a temporary physical table for a view, thus, materialising it. However, this strategy is not useful in situations where many database updates are made on the tables, which are used for view creation, as it will require suitable updating of a temporary table each time the base table is updated.

Can views be used for Data Manipulations?

Views can be used during DML operations like INSERT, DELETE and UPDATE. When you perform DML operations, such modifications need to be passed to the underlying base table. However, this is not allowed on all the views. Conditions for the view that may allow Data Manipulation are:

A view allows data updating, if it follows the following conditions:

1) If the view is created from a single table, then:
   - For INSERT operation, the PRIMARY KEY column(s) and all the NOT NULL columns must be included in the view.
   - View should not be defined using any aggregate function or GROUP BY or HAVING or DISTINCT clauses. This is due to the fact that any update in such aggregated attributes or groups cannot be traced back to a single tuple of the base table. For example, consider a view avgmarks (coursecode, avgmark) created on a base table student(st_id, coursecode, marks). In the avgmarks table changing the class average marks for coursecode “MCS 043” to 50 from a calculated value of 40, cannot be accounted for a single tuple in the Student base table, as the average marks are computed from the marks of all the Student tuples for that coursecode. Thus, this update will be rejected.
2) The views in SQL that are defined using joins are normally NOT updatable in general.
3) WITH CHECK OPTION clause of SQL checks the updatability of data from views, therefore, must be used with views through which you want to update.

**Views and Security**

Views are useful for security of data. A view allows a user to use the data that is available through the view; thus, the hidden data is not made accessible. Access privileges can be given on views. Let us explain this with the help of an example. Consider the view that we have created for teacher-STUDENT-PERFORMANCE. We can grant privileges to the teacher whose name is 'ABC' as:

GRANT SELECT, INSERT, DELETE ON STUDENT-PERFORMANCE TO ABC WITH GRANT OPTION;

Please note that the teacher ABC has been given the rights to query, insert and delete the records on the given view. Please also note s/he is authorised to grant these access rights (WITH GRANT OPTION) to any data entry user so that s/he may enter data on his/her behalf. The access rights can be revoked using the REVOKE statement as:

REVOKE ALL ON STUDENT-PERFORMANCE FROM ABC;

### ☞ Check Your Progress 1

1) Consider a constraint – the value of the age field of the student of a formal University should be between 17 years and 50 years. Would you like to write an assertion for this statement?

……

……

……

2) Create a view for finding the average marks of the students in various subjects, for the tables given in section 3.2.2.

……

……

……

3) Can the view created in problem 2 be used to update subjectcode?

……

……

……

## 3.3 EMBEDDED SQL AND DYNAMIC SQL

SQL commands can be entered through a standard SQL command level user interface. Such interfaces are interactive in nature and the result of a command is shown immediately. Such interfaces are very useful for those who have some knowledge of SQL and want to create a new type of query. However, in a database application where a naïve user wants to make standard queries, that too using GUI type interfaces, probably an application program needs to be developed. Such interfaces sometimes require the support of a programming language environment.

Please note that SQL normally does not support a full programming paradigm (although the latest SQL has full API support), which allows it a full programming interface. In fact, most of the application programs are seen through a programming interface, where SQL commands are put wherever database interactions are needed. Thus, SQL is embedded into programming languages like C, C++, JAVA, etc.

Let us discuss the different forms of embedded SQL in more detail.



### 3.3.1 Embedded SQL

The embedded SQL statements can be put in the application program written in C, Java or any other host language. These statements sometime may be called static. Why are they called static? The term 'static' is used to indicate that the embedded SQL commands, which are written in the host program, do not change automatically during the lifetime of the program. Thus, such queries are determined at the time of database application design. For example, a *query statement* embedded in C to determine the status of train booking for a train will not change. However, this query may be executed for many different trains. Please note that it will only change the input parameter to the query that is train-number, date of boarding, etc., and not the query itself.

But how is such embedding done? Let us explain this with the help of an example.

Example: Write a C program segment that prints the details of a student whose enrolment number is input.

Let us assume the relation

STUDENT (enrolno:char(9), name:Char(25), phone:integer(12), prog-code:char(3))

```
/* add proper include statements*/
/*declaration in C program */
EXEC SQL BEGIN DECLARE SECTION;
        Char enrolno[10], name[26], p-code[4];
        int phone;
        int SQLCODE;
        char SQLSTATE[6]
EXEC SQL END DECLARE SECTION;

/* The connection needs to be established with SQL*/
/* program segment for the required function */
printf ("enter the enrolment number of the student");
scanf ("% s", &enrolno);
EXEC SQL
        SELECT  name,  phone,  prog-code INTO
                :name, :phone, :p-code
        FROM STUDENT
        WHERE  enrolno = :enrolno;
        If (SQLCODE ==0)
            printf  ("%d,  %s, %s, %s", enrolno, name, phone, p-code)
            else
            printf ("Wrong Enrolment Number");
```

Please note the following points in the program above:

- The program is written in the host language 'C' and contains embedded SQL statements.
- Although in the program an SQL query (SELECT) has been added. You can embed any DML, DDL or views statements.
- The distinction between an SQL statement and host language statement is made by using the key word EXEC SQL; thus, this key word helps in identifying the Embedded SQL statements by the pre-compiler.
- Please note that the statements including (EXEC SQL) are terminated by a semi-colon (;),
- As the data is to be exchanged between a host language and a database, there is a need of shared variables that are shared between the environments. Please note that enrolno[10], name[20], p-code[4]; etc. are shared variables, colon (:) declared in 'C'.



- Please note that the shared host variables enrolno is declared to have char[10] whereas, an SQL attribute enrolno has only char[9]. Why? Because in 'C' conversion to a string includes a '\ 0' as the end of the string.
- The type mapping between 'C' and SQL types is defined in the following table:

| 'C' TYPE | SQL TYPE |
|---|---|
| long | INTEGER |
| short | SMALLINT |
| float | REAL |
| double | DOUBLE |
| char [ i+1] | CHAR (i) |

- Please also note that these shared variables are used in SQL statements of the program. They are prefixed with the colon (:) to distinguish them from database attribute and relation names. However, they are used without this prefix in any C language statement.
- Please also note that these shared variables have almost the same name (except p-code) as that of the attribute name of the database. The prefix colon (:) this distinguishes whether we are referring to the shared host variable or an SQL attribute. Such similar names is a good programming convention as it helps in identifying the related attribute easily.
- Please note that the shared variables are declared between BEGIN DECLARE SECTION and END DECLARE SECTION and there typed is defined in 'C' language.

Two more shared variables have been declared in 'C'. These are:

- SQLCODE as int
- SQLSTATE as char of size 6
- These variables are used to communicate errors and exception conditions between the database and the host language program. The value 0 in SQLCODE means successful execution of SQL command. A value of the SQLCODE =100 means 'no more data'. The value of SQLCODE if less than 0 indicates an error. Similarly, SQLSTATE is a 5 char code the 6$^{th}$ char is for '\0' in the host language 'C'. Value "00000" in an SQLSTATE indicate no error. You can refer to SQL standard in more detail for more information.
- In order to execute the required SQL command, connection with the database server need to be established by the program. For this, the following SQL statement is used:

  CONNECT <name of the server> AS <name of the connection>
  AUTHORISATION <username, password>,

  TO DISCONNECT we can simply say

  DISCONNECT <name of the connection>;

However, these statements need to be checked in the commercial database management system, which you are using.

*Execution of SQL query in the given program*: To create the SQL query, first, the given value of enrolment number is transferred to SQL attribute value, the query then is executed and the result, which is a single tuple in this case, is transferred to shared host variables as indicated by the key word INTO after the SELECT statement.

The SQL query runs as a standard SQL query except the use of shared host variables. Rest of the C program has very simple logic and will print the data of the students whose enrolment number has been entered.



Please note that in this query, as the enrolment number is a key to the relation, only one tuple will be transferred to the shared host variables. But what will happen if more than one tuple results on the execution of embedded SQL query. Such situations are handled with the help of a cursor. Let us discuss such queries in more detail.

### 3.3.2 Cursors and Embedded SQL

Let us first define the terminate 'cursor'. The Cursor can be defined as a pointer to the current tuple. It can also be defined as a portion of RAM allocated for the internal processing that has been set aside by the database server for database interactions. This portion may be used for query processing using SQL. But, what size of memory is to be allotted for the cursor? Ideally, the size allotted for the cursor should be equal to the memory required to hold the tuples that result from a query. However, the available memory puts a constraint on this size. Whenever a query results in a number of tuples, we can use a cursor to process the currently available tuples one by one. How? Let us explain the use of the cursor with the help of an example:

Since most of the commercial RDBMS architectures are client-server architectures, on execution of an embedded SQL query, the resulting tuples are cached in the cursor. This operation is performed on the server. Sometimes the cursor is opened by RDBMS itself – these are called **implicit** cursors. However, in embedded SQL you need to declare these cursors explicitly – these are called **explicit cursors.** Any cursor needs to have the following operations defined on them:

DECLARE – to declare the cursor
OPEN AND CLOSE - to open and close the cursor
FETCH – get the current records one by one till end of tuples.

In addition, cursors have some attributes through which we can determine the state of the cursor. These may be:

ISOPEN – It is true if cursor is OPEN, otherwise false.
FOUND/NOT FOUND – It is true if a row is fetched successfully/not successfully.
ROWCOUNT – It determines the number of tuples in the cursor.

Let us explain the use of the cursor with the help of an example:

Example: Write a C program segment that inputs the final grade of the students of MCA programme.

Let us assume the relation:

STUDENT (enrolno:char(9), name:Char(25), phone:integer(12),
prog-code:char(3)); grade: char(1));

The program segment is:

```
/* add proper include statements*/
/*declaration in C program */

EXEC SQL BEGIN DECLARE SECTION;
        Char enrolno[10], name[26], p-code[4], grade /* grade is just one character*/
        int phone;
        int SQLCODE;
        char SQLSTATE[6]
EXEC SQL END DECLARE SECTION;

/* The connection needs to be established with SQL*/
/* program segment for the required function */
```

```
printf ("enter the programme code);
scanf ("%s, &p-code);
EXEC SQL DECLARE CURSOR GUPDATE
        SELECT enrolno, name, phone, grade
        FROM STUDENT
        WHERE progcode =: p-code
        FOR UPDATE OF grade;
EXEC SQL OPEN GUPDATE;
EXEC SQL FETCH FROM GUPDATE
        INTO :enrolno, :name, :phone, :grade;
WHILE (SQLCODE==0) {
        printf ("enter grade for enrolment number, "%s", enrolno);
        scanf ("%c", grade);
        EXEC SQL
                UPDATE STUDENT
                SET grade=:grade
                WHERE CURRENT OF GUPDATE
        EXEC SQL FETCH FROM GUPDATE;
}
EXEC SQL CLOSE GUPDATE;
```

- Please note that the declared section remains almost the same. The cursor is declared to contain the output of the SQL statement. Please notice that in this case, there will be many tuples of students database, which belong to a particular programme.
- The purpose of the cursor is also indicated during the declaration of the cursor.
- The cursor is then opened and the first tuple is fetch into shared host variable followed by SQL query to update the required record. Please note the use of CURRENT OF which states that these updates are for the current tuple referred to by the cursor.
- WHILE Loop is checking the SQLCODE to ascertain whether more tuples are pending in the cursor.
- Please note the SQLCODE will be set by the last fetch statement executed just prior to while condition check.

How are these SQL statements compiled and error checked during embedded SQL?

- The SQL pre-compiler performs the type of checking of the various shared host variables to find any mismatches or errors on each of the SQL statements. It then stores the results into the SQLCODE or SQLSTATE variables.

Is there any limitation on these statically embedded SQL statements?

They offer only limited functionality, as the query must be known at the time of application development so that they can be pre-compiled in advance. However, many queries are not known at the time of development of an application; thus we require dynamically embedded SQL also.

### 3.3.3 Dynamic SQL

Dynamic SQL, unlike embedded SQL statements, are built at the run time and placed in a string in a host variable. The created SQL statements are then sent to the DBMS for processing. Dynamic SQL is generally slower than statically embedded SQL as they require complete processing including access plan generation during the run time.

However, they are more powerful than *embedded SQL* as they allow run time application logic. The basic advantage of using dynamic embedded SQL is that we need not compile and test a new program for a new query.



Let us explain the use of dynamic SQL with the help of an example:

Example: Write a dynamic SQL interface that allows a student to get and modify permissible details about him/her. The student may ask for subset of information also. Assume that the student database has the following relations.

STUDENT (enrolno, name, dob)

RESULT (enrolno, coursecode, marks)

In the table above, a student has access rights for accessing information on his/her enrolment number, but s/he cannot update the data. Assume that user names are enrolment number.

Solution: A sample program segment may be (please note that the syntax may change for different commercial DBMS).

```
/* declarations in SQL */
EXEC SQL BEGIN DECLARE SECTION;
        char inputfields (50);
        char tablename(10)
        char sqlquery ystring(200)
EXEC SQL END DECLARE SECTION;
        printf  ("Enter the fields you want to see \n");
        scanf ("SELECT%s", inputfields);
        printf ("Enter the name of table STUDENT or RESULT");
        scanf ("FROM%s", tablename);
        sqlqueryystring = "SELECT" +inputfields +" "+
                "FROM" + tablename
                + "WHERE enrolno + :USER"
/*Plus is used as a symbol for concatenation operator; in some DBMS it may be ||*/
/* Assumption: the user name is available in the host language variable USER*/

EXEC SQL PREPARE sqlcommand FROM :sqlqueryystring;
EXEC SQL EXECUTE sqlcommand;
```

Please note the following points in the example above.

- The query can be entered completely as a string by the user or s/he can be suitably prompted.
- The query can be fabricated using a concatenation of strings. This is language dependent in the example and is not a portable feature in the present query.
- The query modification of the query is being done keeping security in mind.
- The query is prepared and executed using a suitable SQL EXEC commands.

### 3.3.4 SQLJ

Till now we have talked about embedding SQL in C, but how can we embed SQL statements into JAVA Program? For this purpose we use SQLJ. In SQLJ, a preprocessor called SQLJ translator translates SQLJ source file to JAVA source file. The JAVA file compiled and run on the database. Use of SQLJ improves the productivity and manageability of JAVA Code as:

- The code becomes somewhat compact.
- No run-time SQL syntax errors as SQL statements are checked at compile time.
- It allows sharing of JAVA variables with SQL statements. Such sharing is not possible otherwise.

Please note that SQLJ cannot use dynamic SQL. It can only use simple embedded SQL. SQLJ provides a standard form in which SQL statements can be embedded in

JAVA program. SQLJ statements always begin with a #sql keyword. These embedded SQL statements are of two categories – Declarations and Executable Statements.

Declarations have the following syntax:

```
#sql <modifier> context context_classname;
```

The executable statements have the following syntax:

```
#sql {SQL operation returning no output};

        OR

#sql result = {SQL operation returning  output};
```

**Example:**

Let us write a JAVA function to print the student details of student table, for the student who have taken admission in 2005 and name are like 'Shyam'. Assuming that the first two digits of the 9-digit enrolment number represents a year, the required input conditions may be:

- The enrolno should be more than "05000000" and
- The name contains the sub string "Shyam".

Please note that these input conditions will not be part of the Student Display function, rather will be used in the main ( ) function that may be created separately by you. The following display function will accept the values as the parameters supplied by the main ( ).

```
Public void  DISPSTUDENT (String enrolno, String name, int phone)
{
    try {
                if ( name equals ( " " ) )
                    name =  "%";
                if (enrolno  equals (" ") )
                    enrolno  = "%";
                 SelRowIter       srows = null;
                # sql srows = { SELECT Enrolno, name, phone
                        FROM STUDENT
                 WHERE   enrolno > :enrolno AND name like :name
                  };
                 while ( srows.next ( ) ) {
                        int enrolno          =          srows. enrolno ( );
                        String name          =          srows.name ( );

                    System.out.println ( "Enrollment_No  =  " + enrolno);
                    System.out.println ("Name =" +name);
                    System.out.println ("phone =" +phone);
            }
    } Catch (Exception e) {
            System.out.println ( " error accessing database" + e.to_string);
    }
}
```

### ☞ Check Your Progress 2

1) A University decided to enhance the marks for the students by 2 in the subject MCS-043 in the table: RESULT (enrolno, coursecode, marks). Write a segment of embedded SQL program to do this processing.

……

……

……

2) What is dynamic SQL?

……

……

……

3) Can you embed dynamic SQL in JAVA?

……

……

……

## 3.4 STORED PROCEDURES AND TRIGGERS

In this section, we will discuss some standard features that make commercial databases a strong implementation tool for information systems. These features are triggers and stored procedures. Let us discuss about them in more detail in this section.

### 3.4 .1 Stored Procedures

Stored procedures are collections of small programs that are stored in compiled form and have a specific purpose. For example, a company may have rules such as:

- A code (like enrolment number) with one of the digits as the check digit, which checks the validity of the code.
- Any date of change of value is to be recorded.

These rules are standard and need to be made applicable to all the applications that may be written. Thus, instead of inserting them in the code of each application they may be put in a stored procedure and reused.

The use of procedure has the following advantages from the viewpoint of database application development.

- They help in removing SQL statements from the application program thus making it more readable and maintainable.
- They run faster than SQL statements since they are already compiled in the database.

Stored procedures can be created using CREATE PROCEDURE in some commercial DBMS.

**Syntax:**

CREATE [or replace] PROCEDURE [user]PROCEDURE_NAME

[(argument datatype

[, argument datatype]….)]



BEGIN

Host Language statements;

END;

For example, consider a data entry screen that allows entry of enrolment number, first name and the last name of a student combines the first and last name and enters the complete name in upper case in the table STUDENT. The student table has the following structure:

STUDENT (enrolno:char(9), name:char(40));

The stored procedure for this may be written as:

```
CREATE PROCEDURE studententry (
        enrolment   IN  char (9);
        f-name      INOUT  char (20);
        l-name      INOUT   char (20)
BEGIN
 /* change all the characters to uppercase and trim the length */
        f-name  TRIM =  UPPER (f-name);
        l-name   TRIM =  UPPER (l-name);
        name    TRIM  =  f-name || . . || l-name;
        INSERT INTO CUSTOMER
        VALUES (enrolment, name);
END;
```

INOUT used in the host language indicates that this parameter may be used both for input and output of values in the database.

While creating a procedure, if you encounter errors, then you can use the ***show errors*** command. It shows all the error encountered by the most recently created procedure object.

You can also write an SQL command to display errors. The syntax of finding an error in a commercial database is:

```
SELECT  *
FROM USER_ERRORS
WHERE Name='procedure name' and type='PROCEDURE';
```

Procedures are compiled by the DBMS. However, if there is a change in the tables, etc. referred to by the procedure, then the procedure needs to be recompiled. You can recompile the procedure explicitly using the following command:

```
ALTER PROCEDURE procedure_name COMPILE;
```

You can drop a procedure by using DROP PROCEDURE command.

### 3.4.2 Triggers

Triggers are somewhat similar to stored procedures except that they are activated automatically. When a trigger is activated? A trigger is activated on the occurrence of a particular event. What are these events that can cause the activation of triggers?



These events may be database update operations like INSERT, UPDATE, DELETE etc. A trigger consists of these essential components:

- An event that causes its automatic activation.
- The condition that determines whether the event has called an exception such that the desired action is executed.
- The action that is to be performed.

Triggers do not take parameters and are activated automatically, thus, are different to stored procedures on these accounts. Triggers are used to implement constraints among more than one table. Specifically, the triggers should be used to implement the constraints that are not implementable using referential integrity/constraints. An instance, of such a situation may be when an update in one relation affects only few tuples in another relation. However, please note that you should not be over enthusiastic for writing triggers – if any constraint is implementable using declarative constraints such as PRIMARY KEY, UNIQUE, NOT NULL, CHECK, FOREIGN KEY, etc. then it should be implemented using those declarative constraints rather than triggers, primarily due to performance reasons.

You may write triggers that may execute once for each row in a transaction – called Row Level Triggers or once for the entire transaction Statement Level Triggers. Remember, that you must have proper access rights on the table on which you are creating the trigger. For example, you may have all the rights on a table or at least have UPDATE access rights on the tables, which are to be used in trigger. The following is the syntax of triggers in one of the commercial DBMS:

CREATE TRIGGER <trigger_name>
[BEFORE | AFTER|
<Event>
ON <tablename>
[WHEN <condition> | FOR EACH ROW]

<Declarations of variables if needed is – may be used when creating trigger using host language>

BEGIN
    <SQL statements OR host language SQL statements>

[EXCEPTION]

    <Exceptions if any>
END:

Let us explain the use of triggers with the help of an example:

**Example:**

Consider the following relation of a students database

STUDENT(<u>enrolno,</u> name, phone)
RESULT (<u>enrolno, coursecode</u>, marks)
COURSE (<u>course-code</u>, c-name, details)

Assume that the marks are out of 100 in each course. The passing marks in a subject are 50. The University has a provision for 2% grace marks for the students who are failing marginally – that is if a student has 48 marks, s/he is given 2 marks grace and if a student has 49 marks then s/he is given 1 grace mark. Write the suitable trigger for this situation.

Please note the requirements of the trigger:

Event: UPDATE of marks



OR
INSERT of marks
Condition: When student has 48 OR 49 marks

Action: Give 2 grace marks to the student having 48 marks and 1 grace mark to the student having 49 marks.

The trigger for this thus can be written as:

```
CREATE TRIGGER grace
AFTER INSERT OR UPDATE OF marks ON RESULT
WHEN (marks = 48 OR marks =49)
UPDATE RESULT
SET marks =50;
```

We can drop a trigger using a DROP TRIGGER statement

```
DROP TRIGGER trigger_name;
```

The triggers are implemented in many commercial DBMS. Please refer to them in the respective DBMS for more details.

## 3.5 ADVANCED FEATURES OF SQL

The latest SQL standards have enhanced SQL tremendously. Let us touch upon some of these enhanced features. More details on these would be available in the sources mentioned in 'further readings'.

**SQL Interfaces:** SQL also has a good programming level interfaces. The SQL supports a library of functions for accessing a database. These functions are also called the Application Programming Interface (API) of SQL. The advantage of using an API is that it provides flexibility in accessing multiple databases in the same program irrespective of DBMS, while the disadvantage is that it requires more complex programming. The following are two common functions called interfaces:

SQL/CLI (SQL – call level interface) is an advanced form of Open Database Connectivity (ODBC).

Java Database Connectivity (JDBC) – allows object-oriented JAVA to connect to the multiple databases.

**SQL support for object-orientation:** The latest SQL standard also supports the object-oriented features. It allows creation of abstract data types, nested relations, object identifies etc.

Interaction with Newer Technologies: SQL provides support for XML (eXtended Markup Language) and Online Analytical Processing (OLAP) for data warehousing technologies.

### ☞ Check Your Progress 3

1) What is stored procedure?

……

……

……

2) Write a trigger that restricts updating of STUDENT table outside the normal working hours/holiday.

……

……

……

## 3.6 SUMMARY

This unit has introduced some important advanced features of SQL. The unit has also provided information on how to use these features.

An assertion can be defined as the general constraint on the states of the database. These constraints are expected to be satisfied by the database at all times. The assertions can be stored as stored procedures.

Views are the external schema windows of the data from a database. Views can be defined on a single table or multiple tables and help in automatic security of hidden data. All the views cannot be used for updating data in the tables. You can query a view.

The embedded SQL helps in providing a complete host language support to the functionality of SQL, thus making application programming somewhat easier. An embedded query can result in a single tuple, however, if it results in multiple tuple then we need to use cursors to perform the desired operation. Cursor is a sort of pointer to the area in the memory that contains the result of a query. The cursor allows sequential processing of these result tuples. The SQLJ is the embedded SQL for JAVA. The dynamic SQL is a way of creating queries through an application and compiling and executing them at the run time. Thus, it provides dynamic interface and hence the name.

Stored procedures are the procedures that are created for some application purpose. These procedures are precompiled procedures and can be invoked from application programs. Arguments can be passed to a stored procedure and it can return values. A trigger is also like a stored procedure except that it is invoked automatically on the occurrence of a specified event.

You should refer to the books further readings list for more details relating to this unit.

## 3.7 SOLUTIONS/ANSWERS

### Check Your Progress 1

1) The constraint is a simple Range constraint and can easily be implemented with the help of declarative constraint statement. (You need to use CHECK CONSTRAINT statement). Therefore, there is no need to write an assertion for this constraint.

2)
```
CREATE VIEW avgmarks AS (
  SELECT subjectcode, AVG(smarks)
  FROM MARKS
  GROUP BY subjectcode);
```

3) No, the view is using a GROUP BY clause. Thus, if we try to update the subjectcode. We cannot trace back a single tuple where such a change needs to take place. Please go through the rules for data updating though views.

**Check Your Progress 2**

1)

```
/*The table is RESULT (enrolno.coursecode, marks). */
EXEC SQL BEGIN DECLARE SECTION;
        char enrolno [10], coursecode[7]; /* grade is just one character*/
        int marks;
        char SQLSTATE[6]
EXEC SQL END DECLARE SECTION;
/*The connection needs to be established with sQL*/
/* program segment for the required function*/
        printf ("enter the course code for which 2 grace marks are to be added")
        scanf ("%s", &coursecode);
EXEC SQL DECLARE CURSOR GRACE
            SELECT enrolno, coursecode, marks
            FROM RESULT
            WHERE coursecode=:coursecode
            FOR UPDATE OF marks;
EXEC SQL OPEN GRACE;
EXEC SQL FETCH FROM GRACE
            INTO :enrolno, :coursecode, :marks;
WHILE (SQL CODE==0) {
            EXEC SQL
                    UPDATE RESULT
                    SET marks = marks+2
                    WHERE CURRENT OF GRACE;
            EXEC SQL FETCH FROM GRACE;
}
EXEC SQL CLOSE GRACE;
```

An alternative implementation in a commercial database management system may be:

```
DECLARE CURSOR grace IS
            SELECT enrolno, coursecode, marks
            FROM RESULT
            WHERE coursecode ='MCS043';
str_enrolno RESULT.enrolno%type;
str_coursecode RESULT.coursecode%type;
str_marks RESULT.marks%type;
BEGIN
            OPEN   grace;
IF GRACE %OPEN THEN
            LOOP
                FETCH grace INTO str_enrolno, str_coursecode, str_marks;
                Exit when grace%NOTFOUND;
                UPDATE student SET marks=str_marks +2;
                INSERT INTO resultmcs-43 VALUES (str_enrolno,
                str_coursecode, str_marks);
            END LOOP;
            COMMIT;
            CLOSE grace;
            ELSE
                Dbms_output.put_line ('Unable to open cursor');
            END IF;
END;
```



2) Dynamic SQL allows run time query making through embedded languages. The basic step here would be first to create a valid query string and then to execute that query string. Since the queries are compiled and executed at the run time thus, it is slower than simple embedded SQL.

3) No, at present JAVA cannot use dynamic SQL.

## Check Your Progress 3

1) Stored procedure is a compiled procedure in a host language that has been written for some purpose.

2) The trigger is some pseudo DBMS may be written as:

```
CREATE  TRIGGER resupdstrudent
        BEFORE INSERT OR UPDATE OR DELETE ON STUDENT
BEGIN
    IF (DAY ( SYSDATE) IN ('SAT', 'SUN')) OR
        (HOURS (SYSDATE) NOT BETWEEN  '09:00' AND 18:30')
    THEN
        RAISE_EXCEPTION AND OUTPUT ('OPERATION NOT
                                    ALLOWED AT THIS TIME/DAY');
    END IF;
END
```

Please note that we have used some hypothetical functions, syntax of which will be different in different RDBMS.

# UNIT 4 DATABASE SYSTEM CATALOGUE

**Structure** **Page Nos.**


4.0 Introduction 65
4.1 Objectives 66
4.2 Catalogue for Relational Database Management System 66
4.3 Data Dictionary and Data Repository System 68
  4.3.1 Data Dictionary Features
  4.3.2 Data Dictionary Benefits
  4.3.3 Disadvantages of Data Dictionary
4.4 System Catalogue in a Commercial Database Management System 70
  4.4.1 Views with the Prefix USER
  4.4.2 Views with the Prefix ALL
  4.4.3 Views with the Prefix DBA
  4.4.4 SYS, Owner of the Data Dictionary
4.5 Catalogue in Distributed Database and Object Oriented Database Systems 75
  4.5.1 Catalogue in Distributed Database Systems
  4.5.2 Catalogue in Object Oriented Database Systems
4.6 Role of System Catalogue in Database Administration 77
4.7 Summary 78
4.8 Solutions/Answers 78


## 4.0 INTRODUCTION

The *system catalogue* is a collection of tables and views that contain important information about a database. It is the place where a relational database management system stores schema metadata, such as information about tables and columns, and internal bookkeeping information. A system catalogue is available for each database. Information in the system catalogue defines the structure of the database. For example, the *DDL* (data dictionary language) for all tables in the database is stored in the system catalogue. Most system catalogues are copied from the template database during database creation, and are thereafter database-specific. A few catalogues are physically shared across all databases in an installation; these are marked in the descriptions of the individual catalogues.

The system catalogue for a database is actually part of the database. Within the database are objects, such as tables, indexes, and views. The system catalogue is basically a group of objects that contain information that defines other objects in the database, the structure of the database itself, and various other significant information.

The system catalogue may be divided into logical groups of objects to provide tables that are accessible by not only the database administrator, but by any other database user as well. A user typically queries the system catalogue to acquire information on the user's own objects and privileges, whereas the *DBA* needs to be able to inquire about any structure or event within the database. In some implementations, there are system catalogue objects that are accessible only to the database administrator.

The terms system catalogue and data dictionary have been used interchangeably in most situations. We will make a distinction in the two terms in the context of data dictionary system, which is an implementation of the data dictionary or system catalogue. This unit provides more information on the system catalogue or data dictionary.

## 4.1 OBJECTIVES

After going through this unit, you should be able to:

- define the system catalogue and its content;
- describe the use of catalogue in a commercial DBMS;
- define the data dictionary system and its advantages and disadvantages, and
- define the role of catalogue in system administration.

## 4.2 CATALOGUE FOR RELATIONAL DATABASE MANAGEMENT SYSTEM

In database management systems, a file defines the basic organisation of a database. A data dictionary contains a list of all the files in the database, the number of records in each file, and the names and types of each field. Most database management systems keep the data dictionary hidden from users to prevent them from accidentally destroying its contents.

The information stored in a catalogue of an RDBMS includes:

- the relation names,
- attribute names,
- attribute domains (data types),
- descriptions of constraints (primary keys, secondary keys, foreign keys, NULL/ NOT NULL, and other types of constraints),
- views, and
- storage structures and indexes (index name, attributes on which index is defined, type of index etc).

Security and authorisation information is also kept in the catalogue, which describes:

- authorised user names and passwords,
- each user's privilege to access specific database relations and views,
- the creator and owner of each relation. The privileges are granted using GRANT command. A listing of such commands is given in *Figure 1*.

The system catalogue can also be used to store some statistical and descriptive information about relations. Some such information can be:

- number of tuples in each relation,
- the different attribute values,
- storage and access methods used in relation.

All such information finds its use in query processing.

In relational DBMSs the catalogue is stored as relations. The DBMS software is used for querying, updating, and maintaining the catalogue. This allows DBMS routines (as well as users) to access. The information stored in the catalogue can be accessed by the DBMS routines as well as users upon authorisation with the help of the query language such as SQL.

Let us show a catalogue structure for base relation with the help of an example. *Figure 1* shows a relational schema, and *Figure 2* shows its catalogue. The catalogue here has stored - relation names, attribute names, attribute type and primary key as well as foreign key information.

**Student**
(RollNo, Name, Address, PhoneNo, DOB, email, CourseNo, DNo)

**Course**
(CourseNo, CourseTitle, Professor)

**Dept**
(DeptNo, DeptName, Location)

**Grade**
(RollNo, CourseNo, Grade)

**Figure 1: A sample relation**

The description of the relational database schema in the *Figure 1* is shown as the tuples (contents) of the catalogue relation in *Figure 2.* This entry is called as CAT_ENTRY. All relation names should be unique and all attribute names *within a particular relation* should also unique. Another catalogue relation can store information such as tuple size, current number of tuples, number of indexes, and creator name for each relation.

| Relation_Name | Attribute | Attr_Type | PK | FK | FK_REL |
|---|---|---|---|---|---|
| Student | RollNo | INTEGER(10) | Yes | No | |
| Student | Name | VARCHAR2(30) | No | No | |
| Student | Address | VARCHAR2(50) | No | No | |
| Student | PhoneNo | VARCHAR2(10) | No | No | |
| Student | DOB | Date/Time | No | No | |
| Student | email | VARCHAR2(30) | No | No | |
| Student | CourseNo | INTEGER(10) | No | Yes | Course |
| Student | DNo | INTEGER(10) | No | Yes | Dept |
| Course | CourseNo | INTEGER(10) | Yes | No | |
| Course | CourseTitle | VARCHAR2(10) | No | No | |
| Course | Professor | VARCHAR2(30) | No | No | |
| Dept | DeptNo | INTEGER(10) | Yes | No | |
| Dept | DeptName | VARCHAR2(20) | No | No | |
| Dept | Location | VARCHAR2(30) | No | No | |
| Grade | RollNo | INTEGER(10) | Yes | Yes | |
| Grade | CourseNo | INTEGER(10) | Yes | Yes | |
| Grade | Grade | VARCHAR(2) | No | No | |

**Figure 2: One sample catalogue table**

Data dictionaries also include data on the secondary keys, indexes and views. The above could also be extended to the secondary key, index as well as view information by defining the secondary key, indexes and views. Data dictionaries do not contain any actual data from the database, it contains only book-keeping information for managing it. **Without a data dictionary, however, a database management system cannot access data from the database.**

The Database Library is built on a *Data Dictionary*, which provides a complete description of record layouts and indexes of the database, for validation and efficient data access. The data dictionary can be used for automated database creation, including building tables, indexes, and referential constraints, and granting access rights to individual users and groups. The database dictionary supports the concept of Attached Objects, which allow database records to include compressed BLOBs (Binary Large Objects) containing images, texts, sounds, video, documents, spreadsheets, or programmer-defined data types.

The data dictionary stores useful metadata, such as field descriptions, in a format that is independent of the underlying database system. Some of the functions served by the Data Dictionary include:

- ensuring efficient data access, especially with regard to the utilisation of indexes,
- partitioning the database into both logical and physical regions,
- specifying validation criteria and referential constraints to be automatically enforced,
- supplying pre-defined record types for Rich Client features, such as security and administration facilities, attached objects, and distributed processing (i.e., grid and cluster supercomputing).

### ☞ Check Your Progress 1

**State whether the following are True or False:**

1) A System Catalogue does not store the information on the relationships between two entities. ☐

2) Integrity constraints are applied to relations only and are not stored separately in the Data Dictionary. ☐

3) The system catalogue is a collection of tables and views that contain important information about a database. ☐

4) The information stored in the catalogue cannot be accessed by the users. ☐

5) Structured Query Language can be used to access information from the system catalogues. ☐

## 4.3 DATA DICTIONARY AND DATA REPOSITORY SYSTEM

The terms data dictionary and data repository are used to indicate a more general software utility than a catalogue. *A* **catalogue** is closely coupled with the DBMS software; it provides the information stored in it to users and the DBA, but it is *mainly* accessed by the various software modules of the DBMS itself, such as DDL and DML compilers, the query optimiser, the transaction processor, report generators, and the constraint enforcer. On the other hand, a Data Dictionary is a data structure that stores meta-data, i.e., data about data. The software package for a *stand-alone* **data dictionary** or **data repository** may interact with the software modules of the DBMS, but it is *mainly* used by the designers, users, and administrators of a computer system for information resource management. These systems are used to maintain information on system hardware and software configurations, documentation, applications, and users, as well as other information relevant to system administration.

If a data dictionary system is used *only* by designers, users, and administrators, and not by the DBMS software, it is called a **passive data dictionary**; otherwise, it is called an **active data dictionary** or **data directory**. An active data dictionary is automatically updated as changes occur in the database. A passive data dictionary must be manually updated.

The data dictionary consists of record types (tables) created in the database by system-generated command files, tailored for each supported back-end DBMS. Command files contain SQL statements for CREATE TABLE, CREATE UNIQUE

INDEX, ALTER TABLE (for referential integrity), etc., using the specific SQL statement required by that type of database.

### 4.3.1 Data Dictionary Features

A comprehensive data dictionary product will include:

- support for standard entity types (elements, records, files, reports, programs, systems, screens, users, terminals, etc.), and their various characteristics (e.g., for elements, the dictionary might maintain Business name, Business definition, name, Data type, Size, Format, Range(s), Validation criteria, etc.)

- support for user-designed entity types (this is often called the "extensibility" feature); this facility is often exploited in support of data modelling, to record and cross-reference entities, relationships, data flows, data stores, processes, etc.

- the ability to distinguish between versions of entities (e.g., test and production)
- enforcement of in-house standards and conventions.

- comprehensive reporting facilities, including both "canned" reports and a reporting language for user-designed reports; typical reports include:

    - detail reports of entities
    - summary reports of entities
    - component reports (e.g., record-element structures)
    - cross-reference reports (e.g., element keyword indexes)
    - where-used reports (e.g., element-record-program cross-references).

- a query facility, both for administrators and casual users, which includes the ability to perform generic searches on business definitions, user descriptions, synonyms, etc.

- language interfaces, to allow, for example, standard record layouts to be automatically incorporated into programs during the compile process.

- automated input facilities (e.g., to load record descriptions from a copy library).

- security features

- adequate performance tuning abilities

- support for DBMS administration, such as automatic generation of DDL (Data Definition Language).

### 4.3.2 Data Dictionary Benefits

The benefits of a fully utilised data dictionary are substantial. A data dictionary has the potential to:

- facilitate data sharing by

    - enabling database classes to automatically handle multi-user coordination, buffer layouts, data validation, and performance optimisations,

    - improving the ease of understanding of data definitions,

    - ensuring that there is a single authoritative source of reference for all users
- facilitate application integration by identifying data redundancies,
- reduce development lead times by

  - simplifying documentation
  - automating programming activities.
- reduce maintenance effort by identifying the impact of change as it affects:
  - users,
  - data base administrators,
  - programmers.
- improve the quality of application software by enforcing standards in the development process
- ensure application system longevity by maintaining documentation beyond project completions
- data dictionary information created under one database system can easily be used to generate the same database layout on any of the other database systems BFC supports (Oracle, MS SQL Server, Access, DB2, Sybase, SQL Anywhere, etc.)

These benefits are maximised by a *fully utilised* data dictionary. As the next section will show, our environment is such that not all of these benefits are immediately available to us.

### 4.3.3 Disadvantages of Data Dictionary

A DDS is a useful management tool, but it also has several disadvantages.
It needs careful planning. We would need to define the exact requirements designing its contents, testing, implementation and evaluation. The cost of a DDS includes not only the initial price of its installation and any hardware requirements, but also the cost of collecting the information entering it into the DDS, keeping it up-to-date and enforcing standards. The use of a DDS requires management commitment, which is not easy to achieve, particularly where the benefits are intangible and long term.

## 4.4 SYSTEM CATALOGUE IN A COMMERCIAL DATABASE MANAGEMENT SYSTEM

In this section we will discuss the system catalogue in the context of one of the commercial DBMS – Oracle. Although this section discusses the features of this commercial DBMS, yet we have tried to keep the details as general as possible. The collection of metadata is called the **data dictionary**. The metadata is information about schema objects, such as **Tables,** indexes, views, triggers and more. A data dictionary contains:

- the definitions of all schema objects in the database (tables, views, indexes, clusters, synonyms, sequences, procedures, functions, packages, triggers, and so on),
- amount of space has been allocated for, and is currently used by, the schema objects,
- default values for columns,
- integrity constraint information,
- the names of database users,
- privileges and roles each user has been granted,
- auditing information, such as who has accessed or updated various schema objects,
- other general database information.

Not only is the data dictionary central to every database, it is an important tool for all users, from end users to application designers and database administrators. SQL statements are used to access the data dictionary. Because the data dictionary is read-only, only SELECT statements can be used against its tables and views.

The data dictionary is structured in tables and views, just like other database data. All the data dictionary tables and views for a given database are stored in that database's SYSTEM table space. Access to the data dictionary is allowed through numerous views, which may be divided into three categories: USER, ALL, and DBA.

### 4.4.1 Views with the Prefix USER

The views most likely to be of interest to typical database users are those with the prefix USER. These views are as follows:

- refer to the user's own private environment in the database, including information about schema objects created by the user, grants made by the user, and so on,
- display only rows pertinent to the user,
- have columns identical to the other views, except that the column OWNER is implied,
- return a subset of the information in the ALL views,
- can have abbreviated PUBLIC synonyms for the sake of convenience.

For example, the following query returns all the objects contained in your schema:

**SELECT object_name, object_type FROM USER_OBJECTS;**

### 4.4.2 Views with the Prefix ALL

Views with the prefix ALL refer to the user's overall perspective of the database. These views return information about schema objects to which the user has access through public or explicit grants of privileges and roles, in addition to schema objects that the user owns. For example, the following query returns information on all the objects to which the user has access:

**SELECT owner, object_name, object_type FROM ALL_OBJECTS;**

### 4.4.3 Views with the Prefix DBA

Views with the prefix DBA show a global view of the entire database. Synonyms are not created for these views, because DBA views should be queried only by administrators. Therefore, to query DBA views, administrators must prefix the view name with its owner, SYS, as in the following:

**SELECT owner, object_name, object_type FROM SYS.DBA_OBJECTS;**

### 4.4.4 SYS, Owner of the Data Dictionary

The user named SYS owns all base tables and user-accessible views of the data dictionary. No general user should *ever* alter (UPDATE, DELETE, or INSERT) any rows or schema objects contained in the SYS schema, because such activity can compromise data integrity. The security administrator must keep a strict watch of this central account. Altering or manipulating the data in the data dictionary tables can permanently and detrimentally affect the operation of a database.

**Use of Data Dictionary**

The data dictionary in Oracle has three primary uses:

- Oracle accesses the data dictionary to find information about users, schema objects, and storage structures.

- Oracle modifies the data dictionary every time that a data definition language (DDL) statement is issued.
- Any Oracle user can use the data dictionary as a read-only reference for information about the database.

The system catalogue in Oracle contains information on all three levels of database schemas: external which has view definitions, conceptual - which defines base tables, and internal schema - giving the storage and index descriptions. Some of the catalogue views relating to each of the three schema levels are given below. The catalogue (metadata) data can be retrieved through SQL statements just as the user's actual data.

The first query is the example of the conceptual schema information where an object owned by a particular user, 'KARAN' can be obtained:

SELECT * FROM ALL-CATALOGUE WHERE OWNER = 'KARAN';

The result of this query could be as shown in *Figure 3*, which indicates that three base tables are owned by KARAN: STUDENT, COURSE, and DEPT, plus a view STUCRS. The columns are retrieved as shown.

| OWNER | TABLENAME | TABLE_TYPE |
|---|---|---|
| KARAN | STUDENT | TABLE |
| KARAN | COURSE | TABLE |
| KARAN | DEPT | TABLE |
| KARAN | GRADE | TABLE |
| KARAN | STUCRS | VIEW |

**Figure 3: Result of query about particular user**

To find some of the information describing the columns of the GRADE table *for* 'KARAN', the following query could be submitted:

```
SELECT COLUMN-NAME, DATA-TYPE, DATA-LENGTH, NUM-DISTINCT,
        LOW-VALUE, HIGH-VALUE

FROM USER-TAB-COLUMNS
WHERE TABLE-NAME = 'GRADE';
```

The result of this query could be shown as in *Figure 4*. Because the USER- TAB-COLUMNS table of the catalogue has the prefix USER, this query must be submitted by the owner of the DEPT table. The NUM-DISTINCT column specifies the number of distinct values for a given column and LOW-VALUE and HIGH-VALUE show the lowest and highest values for the given column.

| COLUMN-NAME | DATA-TYPE | DATA-LENGTH | NUM-DISTINCT | LOW-VALUE | HIGH-VALUE |
|---|---|---|---|---|---|
| ROLL NO. | NUMBER | 10 | 50 | 1234A | 9999A |
| COURSE NO. | NUMBER | 10 | 50 | AB012 | MG999 |
| GRADE | VARCHAR | 2 | 8 | F | A |

**Figure 4: Result of query on column details on GRADE table**

A general listing of the SELECT clause used in Data Dictionary to retrieve the information is given in *Figure 5*. It includes the various entity, data type, primary key (pk), foreign key (fk), alternate key (ak), and attribute definition for the retrieval.

**Listing: The Dictionary SELECTs**

```
select table_name as table_name,
   as column_name,
   entity_definition as entity_definition,
   entity_note as entity_note,
   as column_datatype,
   as default_value,
   as pk, as fk, as ak,
   as attribute_definition,
   as sample_instance_data,
        query as query
   from entity
   union
   select table_usage,
        column_name,
        column_datatype,
        null_option,
        default_value,
        pk, fk, ak,
        attribute_definition,
   from attribute
   order by 1, 2
```

**Figure 5: A general listing for SELECT clauses on data dictionary**

At runtime each detail row displays either table data, or column data, but not both. A listing of GRANT commands is given in *Figure 6*. This gives the privileges to the users according to the GRANT commands.

## Listing: Generating Grant Commands

```
select string ( 'grant all on ',
     table_name, ' to administration;' )
     as "GRANT COMMAND" from entity
union select string ( 'grant delete on ',
     table_name, ' to endusers;' )
from entity where query like '% delete%'
union select string ( 'grant insert on ',
     table_name, ' to endusers;' )
from entity where query like '% insert%'
union select string ( 'grant select on ',
     table_name, ' to endusers;' )
from entity where query like '% select%'
union select string ( 'grant update on ',
     table_name, ' to endusers;' )
from entity where query like '% update%'
order by 1;
```

**Figure 6: A generalised listing of GRANT command**

In a simple data dictionary each table definition is immediately followed by an alphabetic list of its columns, together with their data types, null versus not null settings, default values and an indication of whether it is part of a primary, foreign and/or alternate key (PK, FK, AK). As with the tables, a concise free-form description appears directly beneath each column name.

Some of these details are specific to the needs of one particular data base application. Their primary importance is to the human reader, not computer programme. Many of these details are available elsewhere; for example, the data types are visible in the database diagram shown in *Figure 7*, and the default values and access privileges may be discovered by querying the system catalogue *(Figure 8)*. The dictionary

serves to gather the important information together in one place and, of course, to provide the table and column descriptions that make a dictionary what it is.

| **Addr_Info Table**<br>Business or any other address Information<br>End Users have SELECT, INSERT and UPDATE rights |
|---|
| **Contact_Addr_id** INTEGER NOT NULL (PK) (FK) Auto number<br>Automatically generated Number<br>Unique on the defined database |
| **Date/Time Updated** TIMESTAMP NOT NULL DEFAULT<br>Date/Time this contact was added or updated |
| **Fax** CHAR (12) NOT NULL<br>Fax number of the contact |
| **First_name** CHAR(20) Not NULL<br>First Name of the contact |
| **Website** CHAR(100) NOT NULL<br>WWW URL of the contact |

**Figure 7: A sample database diagram with description**

The detailed Data dictionary is stored as given in *Figure 8*

**Addr_Info**
Addr_id:CHAR(6) NOT NULL(FK)
Contact_Addr_id INTEGER NOT NULL(PK)
First_name CHAR (20) NOT NULL
Last_name CHAR (20) NOT NULL
Title CHAR (10) NOT NULL
Addr1 CHAR (50) NOT NULL
Addr2 CHAR (20) NOT NULL
City CHAR (20) NOT NULL
State CHAR (20) NOT NULL (FK)
Pin_Code CHAR (6) NOT NULL
PhoneNo CHAR (12) NOT NULL
Fax CHAR (12) NOT NULL
Email CHAR (50) NOT NULL
Website CHAR (100)
Date/Time updated TIMESTAMP NOT NULL

**Figure 8: A detailed data dictionary**

The main reason to store dictionary information in the database diagram itself is the ease of data entry. It is quite hard to ensure correctness and completeness if the dictionary data is stored separately from the diagram.

The text stored in the query column has been written to be both readable by human beings and usable by the SQL command in *Figure 7.* Precise positive statements are used, such as “End users have select, insert and update rights” rather than vague or negative sentences like “End users can’t remove any rows from this table”.

A data dictionary is a valuable tool if it contains practical and useful information, and if it is kept up to date and accurate. That implies it must reflect the physical reality of the database rather than simple logical view, and that it must be easy to update and generate.

## ☞ Check Your Progress 2

1) List the various types of views through which access to the data dictionary is allowed.

……

……

……

……

2) What are the uses of Data Dictionary in Oracle?

……

……

……

……

3) What is the difference between active and passive Data Dictionary?

……

……

……

……

## 4.5 CATALOGUE IN DISTRIBUTED DATABASE AND OBJECT ORIENTED DATABASE SYSTEMS

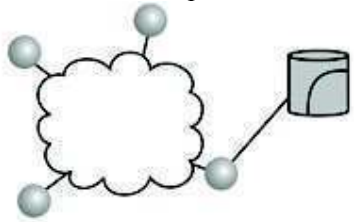
The applications of the database systems over the time were required to support distributed nature of organisation and object oriented system design. Let us discuss the issues of data dictionary in the context of these advanced database systems.

### 4.5.1 Catalogue in Distributed Database Systems

The data dictionary stores useful metadata on database relations. In a Distributed database systems information on locations, fragmentations and replications is also added to the catalogues.

The distributed database catalogue entries must specify site(s) at which data is being stored in addition to data in a system catalogue in a centralised DBMS. Because of data partitioning and replication, this extra information is needed. There are a number of approaches to implementing a distributed database catalogue.

- Centralised: Keep one master copy of the catalogue,
- Fully replicated: Keep one copy of the catalogue at each site,
- Partitioned: Partition and replicate the catalogue as usage patterns demand,
- Centralised/partitioned: Combination of the above.



(a) Centralised

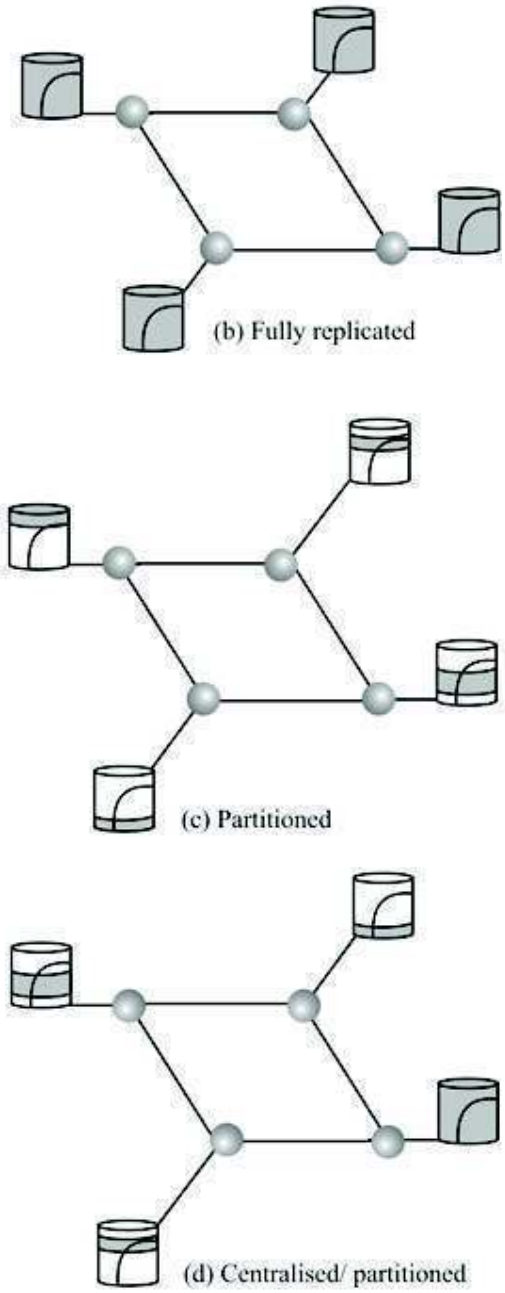



**Figure 9: A distributed system catalogue**

Catalogues are generally updated during creation and/or deletion of relations and modification of attributes. Certain characteristics of the data affect the access and query evaluation. For example, in some distributed catalogues such as R* information of locally stored objects including the fragments and replicates is stored on local catalogues. The object storage sites of any object are maintained at the place where the object was first created and the new sites where the object is moved is updated. This is a partitioned scheme where the catalogue is partitioned and replicated as per the demands of usage patterns.

In some distributed systems local and global relations are different. Global relations can be accessed from all sites. A catalogue containing information of a global nature is maintained at all sites. This is a fully replicated or redundant scheme where - one copy of the catalogue is kept at each site.

Distributed catalogue also store the information on local-to-global name mappings, file organisation and access methods, general as well as integrity constraints details and database statistics.

Remote catalogue information can be requested by any site and stored for later use. However, this stored catalogue may become inconsistent over time as the remote catalogue gets updated. Any query accessing the stored catalogue yields out-of-date information. Hence, the stored remote catalogue has to be updated from time to time.

### 4.5.2 Catalogue in Object Oriented Database Systems

The concept of object-oriented databases will be introduced in Unit 1 of Block 3 of this course. An object oriented database systems brings together the features of object-oriented programming and the advantages of database systems under one persistent DBMS interface. Thus, they are very useful in applications with complex interrelationships, complex classes, inheritance etc. However, as far as data dictionary is concerned, it now additionally needs to describe few more classes, objects and their inter-relationships. Thus, a data dictionary for such a system may be more complex from the viewpoint of implementation; however, from the users point of view it has almost similar concepts. This data dictionary should now also store class definitions including the member variables, member functions, inheritances and relationships of various class objects.

## 4.6 ROLE OF SYSTEM CATALOGUE IN DATABASE ADMINISTRATION

The database administration is a specialised database activity that is performed by a database administrator. The system catalogue has an important role to play in the database administration. Some of the key areas where the system catalogue helps the database administrator are defined below:

**Enforcement of Database Integrity:** System catalogue is used to store information on keys, constraints, referential integrity, business rules, triggering events etc. on various tables and related objects. Thus, integrity enforcement would necessarily require the use of a data dictionary.

**Enforcement of Security:** The data dictionary also stores information on various users of the database systems and their access rights. Thus, enforcement of any security policy has to be processed through the data dictionary.

**Support for Database System Performance:** The data dictionary contains information on the indexes, statistics etc. Such information is very useful for query optimisation. Also such information can be used by the database administrator to suggest changes in the internal schema.

Data dictionary can also support the process of database application development and testing (although this is not a direct relationship to database administration) as they contain the basic documentation while the systems are in the process of being developed.

### ☞ Check Your Progress 3

1) List the disadvantages of a data dictionary.

……

……

……

……

2) What are the approaches to implementing a distributed database catalogue?

……

……

……

……

3) How are Object Oriented catalogues implemented?

……

……

……

……

## 4.7 SUMMARY

This unit provides an detailed view of a data dictionary in a DBMS. The data dictionary is one of the most important implementation tools in a database system. The system catalogue provides all the information on various entities, attributes, database statistics etc. It is a very useful tool if implemented actively in a database system. However, active implementation of a data dictionary is costly.

In this unit we have discussed concepts related to data dictionary and its use in oracle by the different types of users. We have also presented information on the data dictionary system and its advantages and disadvantages. We have provided a brief introduction to system catalogue in distributed systems and how data dictionary is useful in system administration.

## 4.8 SOLUTIONS/ANSWERS

### Check Your Progress 1

1) False 2) False 3) True 4) False 5) True

### Check Your Progress 2

1) a) Views with the Prefix USER b) Views with the Prefix ALL c) Views with the Prefix DBA

2) The data dictionary in Oracle has three primary uses:

- Oracle accesses the data dictionary to find information about users, schema objects, and storage structures.
- Oracle modifies the data dictionary every time that a data definition language (DDL) statement is issued.

3) If a data dictionary system is used *only* by designers, users, and administrators, not by the DBMS software, it is called a **passive data dictionary**; otherwise, it is called an **active data dictionary** or **data directory**. An active data dictionary is automatically updated as changes occur in the database. A passive data dictionary must be manually updated.

### Check Your Progress 3

1) (a) careful planning, (b) defining the exact requirements designing its contents, (c) testing, (d) implementation, (e) evaluation, (f) cost of a DDS includes not only the initial price of its installation and any hardware requirements, but also the cost of collecting the information entering into the DDS, (g) keeping it up-to-date and enforcing standards.

2) a) Centralised: Keeps one master copy of the catalogue
b) Fully replicated: Keeps one copy of the catalogue at each site

c) Partitioned: Partitions and replicates the catalogue as per the demands of the usage pattern
d) Centralised/partitioned: Combination of the above

3) In addition to system catalogues that may be used in simple relational database system, an object oriented system catalogue needs to define complex user defined classes, inheritance hierarchy and complex inter-relationships.

# UNIT 1 QUERY PROCESSING AND EVALUATION

**Structure** **Page Nos.**


1.0 Introduction 5
1.1 Objectives 5
1.2 Query Processing : An Introduction 6
1.2.1 Optimisation
1.2.2 Measure of Query Cost
1.3 Select Operation 7
1.4 Sorting 9
1.5 Join Operation 12
1.5.1 Nested-Loop Join
1.5.2 Block Nested-Loop Join
1.5.3 Indexed Nested-Loop Join
1.5.4 Merge-Join
1.5.5 Hash-Join
1.5.6 Complex Join
1.6 Other Operations 18
1.7 Representation and Evaluation of Query Expressions 19
1.8 Creation of Query Evaluation Plans 20
1.8.1 Transformation of Relational Expressions
1.8.2 Query Evaluation Plans
1.8.3 Choice of Evaluation Plan
1.8.4 Cost Based Optimisation
1.8.5 Storage and Query Optimisation
1.9 View And Query Processing 24
1.9.1 Materialised View
1.9.2 Materialised Views and Query Optimisation
1.10 Summary 26
1.11 Solutions/Answers 26


## 1.0 INTRODUCTION

The Query Language – SQL is one of the main reasons of success of RDBMS. A user just needs to specify the query in SQL that is close to the English language and does not need to say how such query is to be evaluated. However, a query needs to be evaluated efficiently by the DBMS. But how is a query-evaluated efficiently? This unit attempts to answer this question. The unit covers the basic principles of query evaluation, the cost of query evaluation, the evaluation of join queries, etc. in detail. It also provides information about query evaluation plans and the role of storage in query evaluation and optimisation. This unit thus, introduces you to the complexity of query evaluation in DBMS.

## 1.1 OBJECTIVES

After going through this unit, you should be able to:

- measure query cost;
- define algorithms for individual relational algebra operations;
- create and modify query expression;
- define evaluation plan choices, and
- define query processing using views.

## 1.2 QUERY PROCESSING: AN INTRODUCTION

Before defining the measures of query cost, let us begin by defining query processing. Let us take a look at the *Figure 1*.

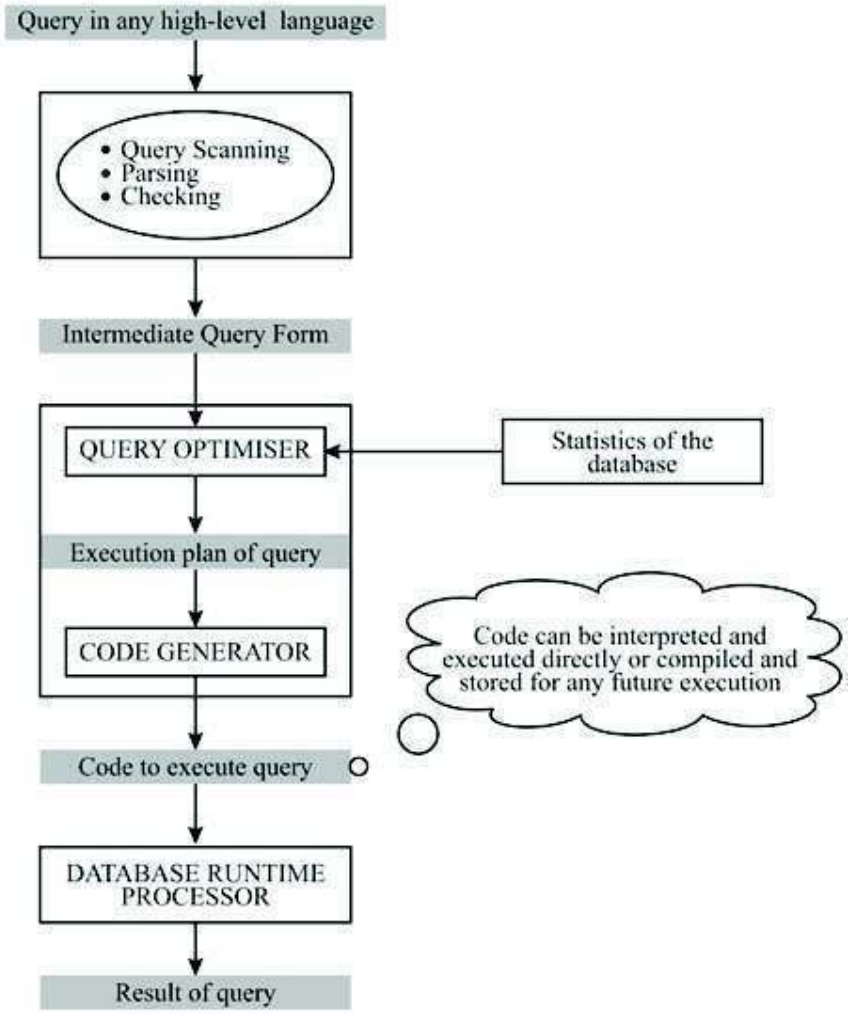



**Figure 1: Query processing**

In the first step Scanning, Parsing, and Validating is done to translate the query into its internal form. This is then further translated into relational algebra (an intermediate query form). Parser checks syntax and verifies relations. The query then is optimised with a query plan, which then is compiled into a code that can be executed by the database runtime processor.

We can define query evaluation as the query-execution engine taking a query-evaluation plan, executing that plan, and returning the answers to the query. The query processing involves the study of the following concepts:

- how to measure query costs?
- algorithms for evaluating relational algebraic operations.
- how to evaluate a complete expression using algorithms on individual operations?

### 1.2.1 Optimisation

A relational algebra expression may have many equivalent expressions. For example, $\sigma_{(salary < 5000)}(\pi_{salary}(EMP))$ is equivalent to $\pi_{salary}(\sigma_{salary <5000}(EMP))$.

Each relational algebraic operation can be evaluated using one of the several different algorithms. Correspondingly, a relational-algebraic expression can be evaluated in many ways.

An expression that specifies detailed evaluation strategy is known as evaluation-plan, for example, you can use an index on *salary* to find employees with *salary <5000* or we can perform complete relation scan and discard employees with salary ≥ 5000. The basis of selection of any of the scheme will be the cost.

**Query Optimisation:** Amongst all equivalent plans choose the one with the lowest cost. Cost is estimated using statistical information from the database catalogue, for example, number of tuples in each relation, size of tuples, etc.

Thus, in query optimisation we find an evaluation plan with the lowest cost. The cost estimation is made on the basis of heuristic rules.

### 1.2.2 Measure of Query Cost

Cost is generally measured as total elapsed time for answering the query. There are many factors that contribute to time cost. These are *disk accesses*, CPU time, or even network *communication*.

Typically disk access is the predominant cost as disk transfer is a very slow event and is also relatively easy to estimate. It is measured by taking into account the following activities:

| | | |
|---|---|---|
| Number of seeks | × | average-seek-cost |
| Number of blocks read | × | average-block-read-cost |
| Number of blocks written | × | average-block-written-cost. |

Please note that the cost for writing a block is higher than the cost for reading a block. This is due to the fact that the data is read back after being written to ensure that the write was successful. However, for the sake of simplicity we will just use *number of block transfers from disk as the cost measure.* We will also ignore the difference in cost between sequential and random I/O, CPU and communication costs. The I/O cost **depends** on the search criteria i.e., point/range query on an ordering/other fields and the file structures: heap, sorted, hashed. It is also dependent on the use of indices such as primary, clustering, secondary, B+ tree, multilevel, etc. There are other cost factors also, these may include buffering, disk placement, materialisation, overflow / free space management etc.

In the subsequent section, let us try to find the cost of various operations.

## 1.3 SELECT OPERATION

The selection operation can be performed in a number of ways. Let us discuss the algorithms and the related cost of performing selection operations.

**File scan:** These are the search algorithms that locate and retrieve records that fulfil a selection condition in a file. The following are the two basic files scan algorithms for selection operation:



1) *Linear search:* This algorithm scans each file block and tests all records to see whether they satisfy the selection condition.

The cost of this algorithm (in terms of block transfer) is defined as:

Cost of searching records satisfying a condition = Number of blocks in a database = $N_b$.

Cost for searching a key attribute value = Average number of block transfer for locating the value (on an average, half of the file needs to be traversed) so it is = $N_b/2$.

Linear search can be applied regardless of selection condition or ordering of records in the file, or availability of indices.

2) *Binary search:* It is applicable when the selection is an equality comparison on the attribute on which file is ordered. Assume that the blocks of a relation are stored continuously then, cost can be estimated as:

Cost = Cost of locating the first tuple by a binary search on the blocks + sequence of other blocks that continue to satisfy the condition.

$$= \lceil \log_2 (N_b) \rceil + \frac{\text{average number of tuples with the same value}}{\text{Blocking factor (Number of tuples in a block) of the relation}}$$

These two values may be calculated from the statistics of the database.

**Index scan:** Search algorithms that use an index are restricted because the selection condition must be on the search-key of the index.

3) (a) *Primary index-scan for equality:* This search retrieves a single record that satisfies the corresponding equality condition. The cost here can be calculated as:

Cost = Height traversed in index to locate the block pointer +1(block of the primary key is transferred for access).

(b) *Hash key:* It retrieves a single record in a direct way thus, cost in hash key may also be considered as Block transfer needed for finding hash target +1

4) *Primary index-scan for comparison:* Assuming that the relation is sorted on the attribute(s) that are being compared, (< , > etc.), then we need to locate the first record satisfying the condition after which the records are scanned forward or backward as the condition may be, displaying all the records. Thus cost in this case would be:

Cost = Number of block transfer to locate the value in index + Transferring all the blocks of data satisfying that condition.

Please note we can calculate roughly (from the cost point of view) the number of blocks satisfying the condition as:

Number of values that satisfy the condition × average number of tuples per attribute value/blocking factor of the relation.

5) *Equality on clustering index* to retrieve multiple records: The cost calculations in this case are somewhat similar to that of algorithm (4).



6) (a) *Equality on search-key of secondary index:* Retrieves a single record if the search-key is a candidate key.

*Cost = cost of accessing index* + 1.

It retrieves multiple records if search-key is not a candidate key.

Cost = *cost of accessing index + number of records retrieved* (It can be very expensive).

Each record may be on a different block, thus, requiring one block access for each retrieved record (this is the worst case cost).

(b) *Secondary index, comparison:* For the queries of the type that use comparison on secondary index value ≥ a value, then the index can be used to find first index entry which is greater than that value, scan index sequentially from there till the end and also keep finding the pointers to records.

For the ≤ *type query* just scan leaf pages of index, also keep finding pointers to records, till first entry is found satisfying the condition.

In either case, retrieving records that are pointed to, may require an I/O for each record. Please note linear file scans may be cheaper if many records are to be fetched.

**Implementation of Complex Selections**

**Conjunction:** Conjunction is basically a set of AND conditions.

7) *Conjunctive selection using one index:* In such a case, select any algorithm given earlier on one or more conditions. If possible, test other conditions on these tuples after fetching them into memory buffer.

8) *Conjunctive selection using multiple-key index:* Use appropriate composite (multiple-key) index if they are available.

9) *Conjunctive selection by intersection of identifiers* requires indices with record pointers. Use corresponding index for each condition, and take the intersection of all the obtained sets of record pointers. Then fetch records from file if, some conditions do not have appropriate indices, test them after fetching the tuple from the memory buffer.

**Disjunction:** Specifies a set of OR conditions.

10) *Disjunctive selection by union of identifiers* is applicable if *all* conditions have available indices, otherwise use linear scan. Use corresponding index for each condition, take the union of all the obtained sets of record pointers, and eliminate duplicates, then fetch data from the file.

**Negation:** Use linear scan on file. However, if very few records are available in the result and an index is applicable on attribute, which is being negated, then find the satisfying records using index and fetch them from the file.

## 1.4 SORTING

Now we need to take a look at sorting techniques that can be used for calculating costing. There are various methods that can be used in the following ways:

1) Use an existing applicable ordered index (e.g., B+ tree) to read the relation in sorted order.



2) Build an index on the relation, and then use the index to read the relation in sorted order. (Options 1&2 may lead to one block access per tuple).

3) For relations that fit in the memory, techniques like quicksort can be used.

4) For relations that do not fit in the memory, *external sort-merge* is a good choice.

Let us go through the algorithm for External Sort-Merge.

i) **Create Sorted Partitions:**

Let *i* be 0 initially.

**Repeat steps (a) to (d) until the end of the relation:**

(a) Read *M* blocks of relation into the memory. (Assumption M is the number of available buffers for the algorithm).

(b) Sort these buffered blocks using internal sorting.

(c) Write sorted data to a temporary file – temp (i)

(d) *i = i + 1;*

Let the final value of *i* be denoted by *N;*

Please note that each temporary file is a sorted partition.

ii) **Merging Partitions (N-way merge):**

// We assume (for now) that $N < M$.
// Use *N* blocks of memory to buffer temporary files and 1 block to buffer output.

Read the first block of each temporary file (partition) into its input buffer block;

**Repeat steps (a) to (e) until all input buffer blocks are empty;**

(a) Select the first record (in sort order) among all input buffer blocks;

(b) Write the record to the output buffer block;

(c) **If** the output buffer block is full then write it to disk and empty it for the next set of data. This step may be performed automatically by the Operating System;

(d) Delete the record from its input buffer block;

(e) **If** the buffer block becomes empty **then** read the next block (if any) of the temporary file into the buffer.

If $N \geq M$, several merge *passes* are required, in each pass, contiguous groups of $M - 1$ partitions are merged and a pass reduces the number of temporary files temp (i) by a factor of $M - 1$. For example, if M=11 and there are 90 temporary files, one pass reduces the number of temporary files to 9, each temporary file begin 10 times the size of the earlier partitions.

Repeated passes are performed till all partitions have been merged into one.

*Figure 2* shows an example for external sorting using sort-merge.

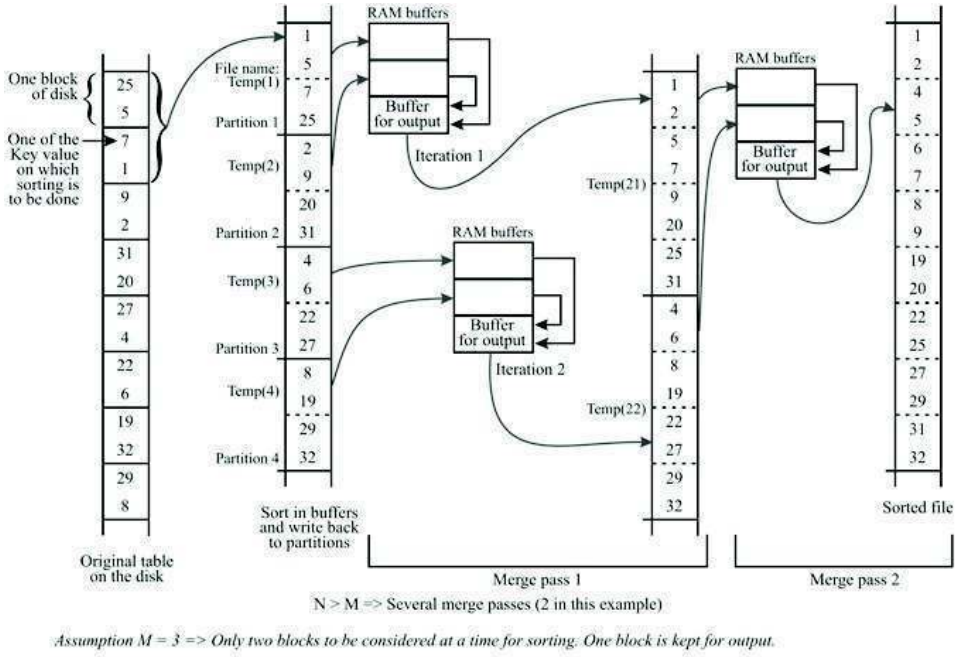



**Figure 2: External merge-sort example**

## Cost Analysis:

Cost Analysis is may be performed, according to the following activities:

- Assume the file has a total of Z blocks.
- Z block input to buffers + Z block output – for temporary file creation.
- Assuming that N ≥ M, then a number of merge passes are required
- Number of merge passes = $\lceil \log_{M-1} (Z/M) \rceil$. Please note that of M buffers 1 is used for output.
- So number of block transfers needed for merge passes $=2\times Z (\lceil \log_{M-1} (Z/M) \rceil)$ as all the blocks will be read and written back of the buffer for each merge pass.
- Thus, the total number of passes for sort-merge algorithm $= 2Z + 2Z (\lceil \log_{M-1} (Z/M) \rceil) = 2Z \times (\lceil \log_{M-1} (Z/M) \rceil + 1)$.

## ☞ Check Your Progress 1

1) What are the basic steps in query processing?

……

……

……

2) How can the cost of query be measured?

……

……

……

3) What are the various methods adopted in select operation?

……

……

……



4) Define External-Sort-Merge algorithm.

……

……

……

# 1.5 JOIN OPERATION

There are number of algorithms that can be used to implement joins:

- Nested-loop join
- Block nested-loop join
- Indexed nested-loop join
- Merge-join
- Hash-join
- Complex join.

The choice of join algorithm is based on the cost estimate. Let us use the following information to elaborate the same:

MARKS (enroll no, subject code, marks):10000 rows, 500 blocks
STUDENT (enroll no, name, dob): 2000 rows, 100 blocks

## 1.5.1 Nested-Loop Join

Let us show the algorithm for the given relations.

To compute the theta or equi-join

```
for each tuple s in STUDENT
{
        for each tuple m in MARKS
        {
                test s.enrolno = m.enrolno to see if they satisfy the join
                condition if they do, output joined tuple to the result.
        };
};
```

- In the nested loop join there is one *outer* relation and one *inner* relation.
- It does not use or require indices. It can be used with any kind of join condition. However, it is expensive as it examines every pair of tuples in the two relations.
- If there is only enough memory to hold one block of each relation, the number of disk accesses can be calculated as:

  For each tuple of STUDENT, all the MARKS tuples (blocks) that need to be accessed.

However, if both or one of the relations fit entirely in the memory, block transfer will be needed only once, so the total number of transfers in such a case, may be calculated as:

= Number of blocks of STUDENT + Number of blocks of MARKS
= 100 + 500 = 600.

If only the smaller of the two relations fits entirely in the memory then use that as the inner relation and the bound still holds.



Cost for the worst case:

Number of tuples of outer relation × Number of blocks of inner relation + Number of blocks of outer relation.

2000 ∗ 500 + 100 = 1,000,100 with STUDENT as outer relation.

There is one more possible bad case when MARKS is on outer loop and STUDENT in the inner loop. In this case, the number of Block transfer will be:

10000 ∗ 100+500 = 1,000,500 with MARKS as the outer relation.

### 1.5.2 Block Nested-Loop Join

This is a variant of nested-loop join in which a complete block of outer loop is joined with the block of inner loop.

The algorithm for this may be written has:

```
for each block s of  STUDENT
{
    for each block m of MARKS
    {
        for each tuple si in s
        {
            for each tuple mi in m
            {
                Check if (si and mi) satisfy the join condition
                        if they do output joined tuple to the result
            };
        };
    };
};
```

Worst case of block accesses in this case = Number of Blocks of outer relation (STUDENT) ×Number of blocks of inner relation (MARKS) + Number of blocks of outer relation (STUDENT).

Best case: Blocks of STUDENT + Blocks of MARKS

Number of block transfers assuming worst case:

100*500 + 100 = 50,100 (much less than nested-loop join)

Number of block transfers assuming best case:

400 + 100 = 500 (same as with nested-loop join)

**Improvements to Block Nested-Loop Algorithm**

The following modifications improve the block Nested method:

Use $M - 2$ disk blocks as the blocking unit for the outer relation, where $M$ = memory size in blocks.

Use one buffer block to buffer the inner relation.

Use one buffer block to buffer the output.

This method minimizes the number of iterations.

### 1.5.3 Indexed Nested-Loop Join

Index scans can replace file scans if the join is an equi-join or natural join, and an index is available on the inner relation's join attribute.

For each tuple *si* in the outer relation STUDENT, use the index to look up tuples in MARKS that satisfy the join condition with tuple *si.*

In a worst case scenarios, the buffer has space for only one page of STUDENT, and, for each tuple in MARKS, then we should perform an index lookup on *MARKS index.*

Worst case: Block transfer of STUDENT+ number of records in STUDENT * cost of searching through index and retrieving all matching tuples for each tuple of STUDENT.

If a supporting index does not exist than it can be constructed as and when needed.

If indices are available on the join attributes of both STUDENT and MARKS, then use the relation with fewer tuples as the outer relation.

**Example of Index Nested-Loop Join Costs**

Compute the cost for STUDENT and MARKS join, with STUDENT as the outer relation. Suppose MARKS has a primary B+-tree index on enroll no, which contains 10 entries in each index node. Since MARKS has 10,000 tuples, the height of the tree is 4, and one more access is needed to the actual data. The STUDENT has 2000 tuples. Thus, the cost of indexed nested loops join as:

100 + 2000 * 5 = 10,100 disk accesses

### 1.5.4 Merge-Join

The merge-join is applicable to equi-joins and natural joins only. It has the following process:

1) Sort both relations on their join attribute (if not already sorted).
2) Merge the sorted relations to join them. The join step is similar to the merge stage of the sort-merge algorithm, the only difference lies in the manner in which duplicate values in join attribute are treated, i.e., every pair with same value on join attribute must be matched.

STUDENT

| Enrol no | Name | ----- |
|---|---|---|
| 1001 | Ajay | …. |
| 1002 | Aman | ….. |
| 1005 | Rakesh | ….. |
| 1100 | Raman | ….. |
| …… | …….. | …… |

MARKS

| Enrol no | subject code | Marks |
|---|---|---|
| 1001 | MCS-011 | 55 |
| 1001 | MCS-012 | 75 |
| 1002 | MCS-013 | 90 |
| 1005 | MCS-015 | 75 |
| ……. | …….. | …… |

**Figure 3: Sample relations for computing join**

The number of block accesses:

Each block needs to be read only once (assuming all tuples for any given value of the join attributes fit in memory). Thus number of block accesses for merge-join is:

Blocks of STUDENT + Blocks of MARKS + the cost of sorting if relations are unsorted.

### Hybrid Merge-Join

This is applicable only when the join is an equi-join or a natural join and one relation is sorted and the other has a secondary B+-tree index on the join attribute.

The algorithm is as follows:

> Merge the sorted relation with the leaf entries of the B+-tree. Sort the result on the addresses of the unsorted relation's tuples. Scan the unsorted relation in physical address order and merge with the previous results, to replace addresses by the actual tuples. Sequential scan in such cases is more efficient than the random lookup method.

### 1.5.5 Hash-Join

This is applicable to both the equi-joins and natural joins. A hash function *h* is used to partition tuples of both relations, where *h* maps joining attribute (enroll no in our example) values to {0, 1, ..., *n-1*}.

The join attribute is hashed to the join-hash partitions. In the example of *Figure 4* we have used mod 100 function to hashing, and n = 100.

**Error!**

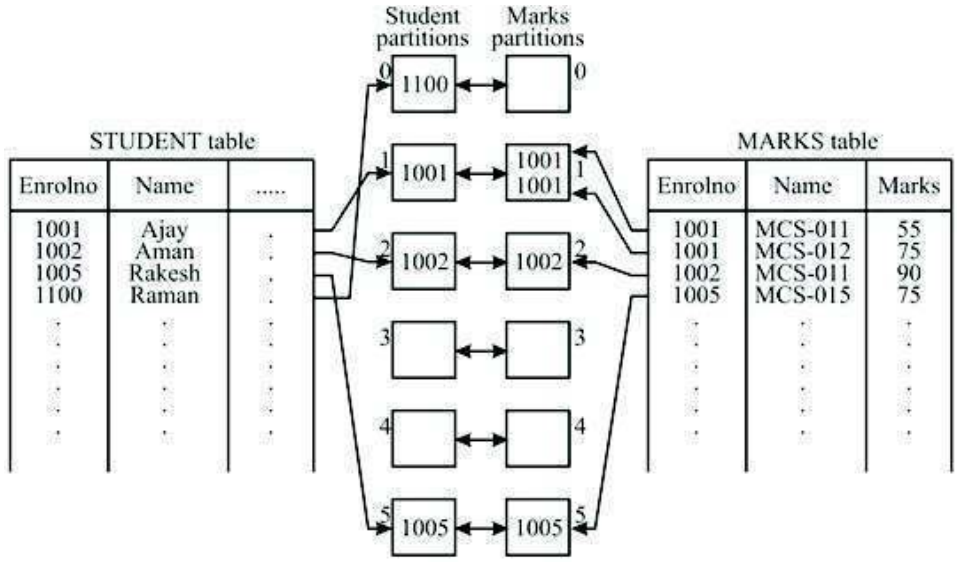



**Figure 4: A hash-join example**

Once the partition tables of STUDENT and MARKS are made on the enrolment number, then only the corresponding partitions will participate in the join as:

A STUDENT tuple and a MARKS tuple that satisfy the join condition will have the same value for the join attributes. Therefore, they will be hashed to equivalent partition and thus can be joined easily.

#### Algorithm for Hash-Join

The hash-join of two relations *r* and *s* is computed as follows:

- Partition the relation *r* and *s* using hashing function *h*. (When partitioning a relation, one block of memory is reserved as the output buffer for each partition).

- For each partition *si* of *s,* load the partition into memory and build an in-memory hash index on the join attribute.



- Read the tuples in *ri* from the disk, one block at a time. For each tuple in *ri* locate each matching tuple in *si* using the in-memory hash index and output the concatenation of their attributes.

In this method, the relation *s* is called the *build* relation and *r* is called the *probe* relation. The value *n* (the number of partitions) and the hash function *h* is chosen in such a manner that each *si* should fit in to the memory. Typically *n* is chosen as Number of blocks of s/Number of memory buffers] *f (M) where f is a "fudge factor", typically around 1.2. The probe relation partitions *ri* need not fit in memory.

*Average* size of a partition *si* will be less than *M* blocks using the formula for *n* as above thereby allowing room for the index. If the build relation *s* is very huge, then the value of *n* as given by the above formula may be greater than *M– 1* i.e., the number of buckets is > the number of buffer pages. In such a case, the relation *s* can be recursively partitioned, instead of partitioning *n* ways, use *M* – 1 partitions for *s and* further partition the *M* – 1 partitions using a different hash function. You should use the same partitioning method on *r.* This method is rarely required, as recursive partitioning is not needed for relations of 1GB or less for a memory size of 2MB, with block size of 4KB.

### Cost calculation for Simple Hash-Join

(i) Cost of partitioning *r* and *s*: all the blocks of *r* and *s* are read once and after partitioning written back, so cost 1 = 2 (blocks of *r* + blocks of *s*).

(ii) Cost of performing the hash-join using build and probe will require at least one block transfer for reading the partitions
Cost 2 = (blocks of *r* + blocks of *s*)

(iii) There are a few more blocks in the main memory that may be used for evaluation, they may be read or written back. We ignore this cost as it will be too less in comparison to cost 1 and cost 2.
Thus, the total cost = cost 1 + cost 2
= 3 (blocks of *r* + blocks of *s*)

Cost of Hash-Join requiring recursive partitioning:

(i) The cost of partitioning in this case will increase to number of recursion required, it may be calculated as:

Number of iterations required = ([$\log_{M-1}$ (blocks of s] –1)

Thus, cost 1 will be modified as:

= 2 (blocks of *r* + blocks of *s*) × ([$\log_{M-1}$ (blocks of s)] –1)

The cost for step (ii) and (iii) here will be the same as that given in steps (ii) and (iii) above.

Thus, total cost = 2*(*blocks of *r* + blocks of *s)* ( [$log_{M-1}$(blocks of *s*) – 1] ) + (blocks of *r* + blocks of *s)*.

Because *s* is in the inner term in this expression, it is advisable to choose the smaller relation as the build relation. If the entire build input can be kept in the main memory, *n* can be set to 1 and the algorithm need not partition the relations but may still build an in-memory index, in such cases the cost estimate goes down to (Number of blocks *r* + Number of blocks of *s*).

### Handling of Overflows

Even if *s* is recursively partitioned *hash-table overflow* can occur, i.e., some partition *si* may not fit in the memory. This may happen if there are many tuples in *s* with the same value for join attributes or a bad hash function.

Partitioning is said to be *skewed* if some partitions have significantly more tuples than the others. This is the overflow condition. The overflow can be handled in a variety of ways:

> *Resolution* (during the build phase): The overflow partition *s* is further partitioned using different hash function. The equivalent partition of *r* must be further partitioned similarily.
>
> *Avoidance* (during build phase): Partition build relations into many partitions, then combines them.

However, such approaches for handling overflows fail with large numbers of duplicates. One option of avoiding such problems is to use block nested-loop join on the overflowed partitions.

Let us explain the hash join and its cost for the natural join STUDENT ⋈ MARKS
Assume a memory size of 25 blocks ⇒ M=25;

SELECT build *s* as STUDENT as it has less number of blocks (100 blocks) and *r* probe as MARKS (500 blocks).

Number of partitions to be created for STUDENT = (block of STUDENT/M)* fudge factor (1.2) = (100/25) × 1.2 = 4.8

Thus, STUDENT relation will be partitioned into 5 partitions of 20 blocks each. MARKS will also have 5 partitions of 100 blocks each. The 25 buffers will be used as –20 blocks for one complete partition of STUDENT plus 4 more blocks for one block of each of the other 4 partitions. One block will be used for input of MARKS partitions.

The total cost = 3(100+500) = 1800 as no recursive partitioning is needed.

#### Hybrid Hash-Join

This is useful when the size of the memory is relatively large, and the build input is larger than the memory. Hybrid hash join keeps the first partition of the build relation in the memory. The first partition of STUDENT is maintained in the first 20 blocks of the buffer, *and not written to the disk*. The first block of MARKS is used right away for probing, instead of being written and read back. Thus, it has a cost of 3(80 + 400) + 20 +100 = 1560 block transfers for hybrid hash-join, instead of 1800 with plain hash-join.

Hybrid hash-join is most useful if *M* is large, such that we can have bigger partitions.

### 1.5.6 Complex Joins

A join with a conjunctive condition can be handled, either by using the nested loop or block nested loop join, or alternatively, the result of one of the simpler joins (on a few conditions) can be computed and the final result may be evaluated by finding the tuples that satisfy the remaining conditions in the result.

A join with a disjunctive condition can be calculated either by using the nested loop or block nested loop join, or it may be computed as the union of the records in individual joins.



## 1.6 OTHER OPERATIONS

There are many other operations that are performed in database systems. Let us introduce these processes in this section.

**Duplicate Elimination:** Duplicate elimination may be implemented by using hashing or sorting. On sorting, duplicates will be adjacent to each other thus, may be identified and deleted. An optimised method for duplicate elimination can be deletion of duplicates during generation as well as at intermediate merge steps in external sort-merge. Hashing is similar – duplicates will be clubbed together in the same bucket and therefore may be eliminated easily.

**Projection:** It may be implemented by performing the projection process on each tuple, followed by duplicate elimination.

**Aggregate Function Execution:** Aggregate functions can be implemented in a manner similar to duplicate elimination. Sorting or hashing can be used to bring tuples in the same group together, and then aggregate functions can be applied to each group. An optimised solution could be to combine tuples in the same group during part time generation and intermediate merges, by computing partial aggregate values. For count, min, max, sum, you may club aggregate values on tuples found so far in the group. When combining partial aggregates for counting, you would need to add up the aggregates. For calculating the average, take the sum of the aggregates and the count/number of aggregates, and then divide the sum with the count at the end.

**Set operations** ($\cup$, $\cap$ and —) can either use a variant of merge-join after sorting, or a variant of hash-join.

Hashing:

1) Partition both relations using the same hash function, thereby creating *r0, .., rn – 1* and *s0,.., sn – 1*

2) Process each partition *i* as follows: Using a different hashing function, build an in-memory hash index on *ri* after it is brought into the memory.

   $r \cup s$: Add tuples in *si* to the hash index if they are not already in it. At the end of *si* add the tuples in the hash index to the result.

   $r \cap s$: Output tuples in *si* to the result if they are already there in the hash index.
   *r – s:* For each tuple in *si,* if it is there in the hash index, delete it from the index.

   At end of *si* add remaining tuples in the hash index to the result.

There are many other operations as well. You may wish to refer to them in further readings.

### ☞ Check Your Progress 2

1) Define the algorithm for Block Nested-Loop Join for the worst-case scenario.

   ……

   ……

   ……

2) Define Hybrid Merge-Join.

   ……

   ……

   ……

3) Give the method for calculating the cost of Hash-Join?

……

……

……

4) Define other operations?

……

……

……

## 1.7 REPRESENTATION AND EVALUATION OF QUERY EXPRESSIONS

Before we discuss the evaluation of a query expression, let us briefly explain how a SQL query may be represented. Consider the following student and marks relations:

STUDENT (enrolno, name, phone)
MARKS (enrolno, subjectcode, grade)

To find the results of the student(s) whose phone number is '1129250025', the following query may be given.

```
SELECT enrolno, name, subjectcode, grade
FROM STUDENT s, MARKS m
WEHRE s.enrolno=m.enrolno AND phone= '1129250025'
```

The equivalent relational algebraic query for this will be:

$\pi_{\text{enrolno, name, subjectcode}} ( \sigma_{\text{phone='1129250025'}} (\text{STUDENT}) \bowtie \text{MARKS})$

This is a very good internal representation however, it may be a good idea to represent the relational algebraic expression to a query tree on which algorithms for query optimisation can be designed easily. In a query tree, nodes are the operators and relations represent the leaf. The query tree for the relational expression above would be:

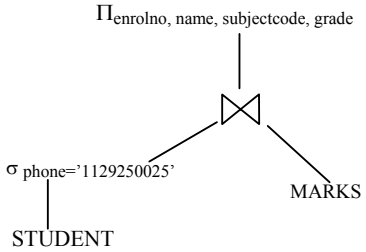



**Figure 5: A Sample query tree**

In the previous section we have seen the algorithms for individual operations. Now let us look at the methods for evaluating an entire expression. In general we use two methods:

- Materialisation
- Pipelining.

**Materialisation**: Evaluate a relational algebraic expression from the bottom-up, by explicitly generating and storing the results of each operation in the expression. For example, in *Figure 5* compute and store the result of the selection operation on STUDENT relation, then take the join of this result with MARKS relation and then finally compile the projection operation.



Materialised evaluation is always possible though; the cost of writing/reading results to/from disk can be quite high.

**Pipelining**

Evaluate operations in a multi-threaded manner, (i.e., passes tuples output from one operation to the next parent operation as input) even as the first operation is being executed. In the previous expression tree, it does not store (materialise) results instead, it passes tuples directly to the join. Similarly, does not store results of join, and passes tuples directly to the projection. Thus, there is no need to store a temporary relation on a disk for each operation. Pipelining may not always be possible or easy for sort, hash-join.

One of the pipelining execution methods may involve a buffer filled by the result tuples of lower level operation while, records may be picked up from the buffer by the higher level operation.

**Complex Joins**

When an expression involves three relations then we have more than one strategy for the evaluation of the expression. For example, join of relations such as:

STUDENT $\bowtie$ MARKS $\bowtie$ SUBJECTS
may involve the following three strategies:

**Strategy 1:** Compute STUDENT $\bowtie$ MARKS*; use* result to compute result $\bowtie$ SUBJECTS

**Strategy 2:** Compute MARKS $\bowtie$ SUBJECTS first, and then join the result with STUDENT

**Strategy 3:** Perform the pair of joins at the same time. This can be done by building an index of enroll no in STUDENT and on subject code in SUBJECTS.

For each tuple *m* in MARKS, look up the corresponding tuples in STUDENT and the corresponding tuples in SUBJECTS. Each tuple of MARKS will be examined only once.

Strategy 3 combines two operations into one special-purpose operation that may be more efficient than implementing the joins of two relations.

## 1.8 CREATION OF QUERY EVALUATION PLANS

We have already discussed query representation and its final evaluation in earlier section of this unit, but can something be done during these two stages that optimises the query evaluation? This section deals with this process in detail.

Generation of query-evaluation plans for an expression involves several steps:

1) Generating logically equivalent expressions using **equivalence rules**
2) Annotating resultant expressions to get alternative query plans
3) Choosing the cheapest plan based on **estimated cost.**

The overall process is called **cost based optimisation.**

The cost difference between a good and a bad method of evaluating a query would be enormous. We would therefore, need to estimate the cost of operations and statistical information about relations. For example a number of tuples, a number of distinct values for an attribute etc. Statistics helps in estimating intermediate results to compute the cost of complex expressions.



Let us discuss all the steps in query-evaluation plan development in more details next.

### 1.8.1 Transformation of Relational Expressions

Two relational algebraic expressions are said to be **equivalent** if on every legal database instance the two expressions generate the same set of tuples (order of tuples is irrelevant).

Let us define certain equivalence rules that may be used to generate equivalent relational expressions.

**Equivalence Rules**

1) The conjunctive selection operations can be equated to a sequence of individual selections. It can be represented as:

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

2) The selection operations are commutative, that is,

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

3) Only the last of the sequence of projection operations is needed, the others can be omitted.

$$\pi_{attriblist1}(\pi_{attriblist2}(\pi_{attriblist3} \ldots..(E) ..) = \pi_{attriblist1}(E)$$

4) The selection operations can be combined with Cartesian products and theta join operations.

$$\sigma_{\theta 1}(E_1 \times E_2) = E_1 \bowtie_{\theta 1} E_2$$

and

$$\sigma_{\theta 2}(E_1 \bowtie_{\theta 1} E_2) = E_1 \bowtie_{\theta 2 \wedge \theta 1} E_2$$

5) The theta-join operations and natural joins are commutative.

$$E1 \bowtie_{\theta} E2 = E2 \bowtie_{\theta} E1$$

6) The Natural join operations are associative. Theta joins are also associative but with the proper distribution of joining condition:

$$(E1 \bowtie E2) \bowtie E3 = E1 \bowtie (E2 \bowtie E3)$$

7) The selection operation distributes over the theta join operation, under conditions when all the attributes in selection predicate involve only the attributes of one of the expressions ($E1$) being joined.

$$\sigma_{\theta 1}(E1 \bowtie_{\theta} E2) = (\sigma_{\theta 1}(E1)) \bowtie_{\theta} E2)$$

8) The projections operation distributes over the theta join operation with only those attributes, which are present in that relation.

$$\pi_{attrlist1 \cup attrlist2}(E1 \bowtie_{\theta} E2) = (\pi_{attrlist1}(E1) \bowtie_{\theta} \pi_{attrlist2}(E2))$$

9) The set operations of union and intersection are commutative. But set difference is not commutative.

$E1 \cup E2 = E2 \cup E$ and similarly for the intersection.

10) Set union and intersection operations are also associative.

$(E1 \cup E2) \cup E3 = E1 \cup (E2 \cup E3)$ and similarly for intersection.

11) The selection operation can be distributed over the union, intersection, and set-differences operations.

$$\sigma_{\theta 1}(E1 - E2) = ((\sigma_{\theta 1}(E1) - (\sigma_{\theta 1}(E2))$$

12) The projection operation can be distributed over the union.

$$\pi_{attriblist1}(E1 \cup E2) = \pi_{attriblist1}(E1) \cup \pi_{attriblist1}(E2)$$

The rules as above are too general and a few heuristics rules may be generated from these rules, which help in modifying the relational expression in a better way. These rules are:

(1) Combining a cascade of selections into a conjunction and testing all the predicates on the tuples at the same time:

$\sigma_{\theta 2}(\sigma_{\theta 1}(E))$ convert to $\sigma_{\theta 2 \wedge \theta 1}(E)$

(2) Combining a cascade of projections into single outer projection:

$\pi_4(\pi_3(\ldots\ldots(E))) = \pi_4(E)$

(3) Commutating the selection and projection or vice-versa sometimes reduces cost

(4) Using associative or commutative rules for Cartesian product or joining to find various alternatives.

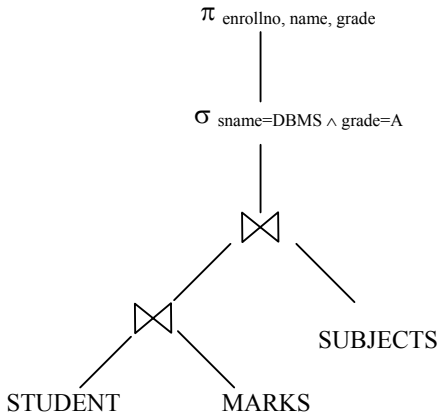
(5) Moving the selection and projection (it may have to be modified) before joins. The selection and projection results in the reduction of the number of tuples and therefore may reduce cost of joining.

(6) Commuting the projection and selection with Cartesian product or union.

Let us explain use of some of these rules with the help of an example. Consider the query for the relations:

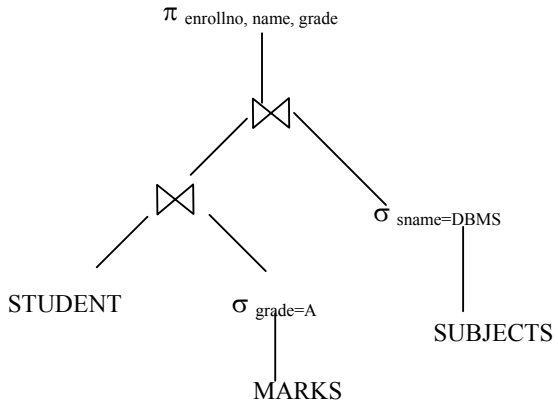
STUDENT (enrollno, name, phone)
MARKS (enrollno, subjectcode, grade)
SUBJECTS (subjectcode, sname)

Consider the query: Find the enrolment number, name, and grade of those students who have secured an A grade in the subject DBMS. One of the possible solutions to this query may be:

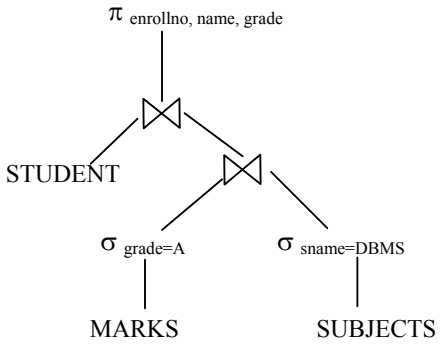



The selection condition may be moved to the join operation. The selection condition given in the *Figure* above is: *sname = DBMS and grade = A*. Both of these

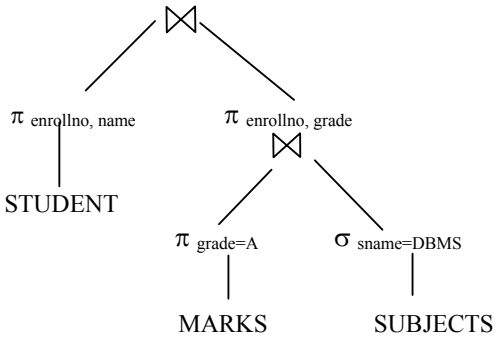
conditions belong to different tables, as *s name* is available only in the SUBJECTS table and *grade* in the MARKS table. Thus, the conditions of selection will be mapped accordingly as shown in the *Figure* below. Thus, the equivalent expression will be:




The expected size of SUBJECTS and MARKS after selection will be small so it may be a good idea to first join MARKS with SUBJECTS. Hence, the associative law of JOIN may be applied.




Finally projection may be moved inside. Thus the resulting query tree may be:




Please note the movement of the projections.

## Obtaining Alternative Query Expressions

Query optimisers use equivalence rules to systematically generate expressions equivalent to the given expression. Conceptually, they generate all equivalent expressions by repeatedly executing the equivalence rules until no more expressions are to be found. For each expression found, use all applicable equivalence rules and add newly generated expressions to the set of expressions already found. However, the approach above is very expensive both in time space requirements. The heuristics rules given above may be used to reduce cost and to create a few possible but good equivalent query expression.

### 1.8.2 Query Evaluation Plans

Let us first define the term Evaluation Plan.

An evaluation plan defines exactly which algorithm is to be used for each operation, and how the execution of the operation is coordinated. For example, *Figure 6* shows the query tree with evaluation plan.

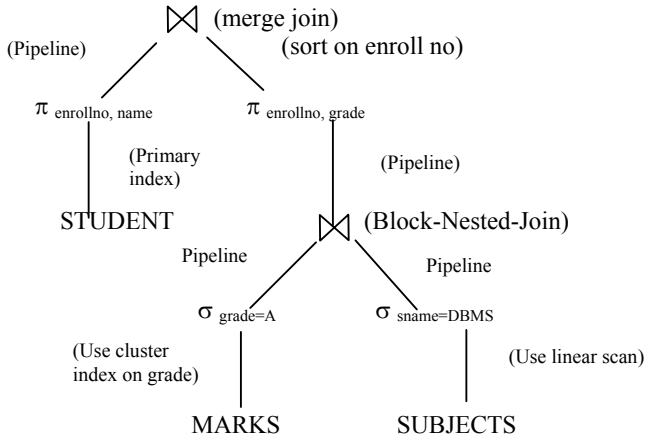



**Figure 6: Query evaluation plan**

### 1.8.3 Choice of Evaluation Plans

For choosing an evaluation technique, we must consider the interaction of evaluation techniques. Please note that choosing the cheapest algorithm for each operation independently may not yield best overall algorithm. For example, merge-join may be costlier than hash-join, but may provide a sorted output which reduces the cost for a higher level aggregation. A nested-loop join may provide opportunity for pipelining. Practical query optimisers incorporate elements of the following two broad approaches:

1) Searches all the plans and chooses the best plan in a cost-based fashion.
2) Uses heuristic rules to choose a plan.

### 1.8.4 Cost Based Optimisation

Cost based optimisation is performed on the basis of the cost of various individual operations that are to be performed as per the query evaluation plan. The cost is calculated as we have explained in the earlier section with respect to the method and operation (JOIN, SELECT, etc.).

### 1.8.5 Storage and Query Optimisation

Cost calculations are primarily based on disk access, thus, storage has an important role to play in cost. In addition, some of the operations also require intermediate storage thus; cost is further enhanced in such cases. The cost of finding an optimal query plan is usually more than offset by savings at query-execution time, particularly by reducing the number of slow disk accesses.

## 1.9 VIEWS AND QUERY PROCESSING

A view must be prepared, passing its parameters, which describe the query and the manner in which tuples should be evaluated. This takes the form of a pre-evaluation window, which gives the user of the program the ability to trade off memory usage for faster navigation, or an attempt to balance both of these resources.

The view may maintain the complete set of tuples following evaluation. This requires a lot of memory space, therefore, it may be a good idea to partially pre-evaluate it. This is done by hinting at how the number of that should be present in the evaluated view tuples before and after the current tuple. However, as the current tuple changes, further evaluation changes, thus, such scheme is harder to plan.

### 1.9.1 Materialised View

A materialised view is a view whose contents are computed and stored.

Materialising the view would be very useful if the result of a view is required frequently as it saves the effort of finding multiple tuples and totalling them up.

Further the task of keeping a materialised view up-to-date with the underlying data is known as materialised view maintenance. Materialised views can be maintained by recomputation on every update. A better option is to use incremental view maintenance, that is, where only the affected part of the view is modified. View maintenance can be done manually by defining triggers on insert, delete, and update of each relation in the view definition. It can also be written manually to update the view whenever database relations are updated or supported directly by the database.

### 1.9.2 Materialised Views and Query Optimisation

We can perform query optimisation by rewriting queries to use materialised views. For example, assume that a materialised view of the join of two tables *b* and *c* is available as:

$$a = b \ \ NATURAL\ JOIN \ \ c$$

Any query that uses natural join on b and c can use this materialised view '*a*' as:

Consider you are evaluating a query:

$$z = r\ NATURAL\ JOIN \ \ b \ \ NATURAL\ JOIN \ \ c$$

Then this query would be rewritten using the materialised view 'a' as:

$$z = r \ \ NATURAL\ JOIN \ \ a$$

Do we need to perform materialisation? It depends on cost estimates for the two alternatives viz., use of a materialised view by view definition, or simple evaluation.

Query optimiser should be extended to consider all the alternatives of view evaluation and choose the best overall plan. This decision must be made on the basis of the system workload. Indices in such decision-making may be considered as specialised views. Some database systems provide tools to help the database administrator with index and materialised view selection.

### ☞ Check Your Progress 3

1) Define methods used for evaluation of expressions?

……

……

……

2) How you define cost based optimisation?

……

……

……



3) How you define evaluation plan?

……

……

……

## 1.10 SUMMARY

In this unit we have discussed query processing and evaluation. A query in a DBMS is a very important operation, as it needs to be efficient. Query processing involves query parsing, representing query in alternative forms, finding the best plan of evaluation of a query and then actually evaluating it. The major query evaluation cost is the disk access time. In this unit, we have discussed the cost of different operations in details. However, an overall query cost will not be a simple addition of all such costs.

## 1.11 SOLUTIONS/ANSWERS

### Check Your Progress 1

1) (a) In the first step scanning, parsing & validating is done
   (b) Translation into relational algebra
   (c) Parser checks syntax, verifies relations
   (d) Query execution engine take a query evaluation plan, executes it and returns answer to the query.

2) Query cost is measured by taking into account following activities:

   * Number of seeks * Number of blocks * Number of block written

   For simplicity we just use number of block transfers from disk as the cost measure. During this activity we generally ignore the difference in cost between sequential and random I/O and CPU/communication cost.

3) Selection operation can be performed in a number of ways such as:

| **File Scan** | **Index Scan** |
|---|---|
| 1. Linear Search | 1. (a) Primary index, equality<br>(b) Hash Key |
| 2. Binary Search | 2. Primary index, Comparison<br>3. Equality on clustering index<br>4. (a) Equality on search key of secondary index<br>(b) Secondary index comparison |

4) Algorithm for External Sort-Merge

   1. Create sorted partitions

      (a) Read M blocks of relation into memory
      (b) Write sorted data to partition $R_i$

   2. Merge the partitions (N-way Merge) until all input buffer blocks are empty.

### Check Your Progress 2

1) For each block $B_r$ of r {
   For each block $B_s$ of s {
   For each tuple $t_i$ in Br {
   For each tuple $t_i$ in Bs {
   Test pair($t_i$,$s_i$) to see if they satisfy the join condition



```
                    If they do, add the joined tuple to result
                        };
                    };
                };
            };
```

2) Hybrid Merge-Join is applicable when the join is equi-join or natural join and one relation is sorted.

   Merge the sorted relation with leaf of B+tree.

   (i) Sort the result on address of sorted relations tuples.
   (ii) Scan unsorted relation and merge with previous result.

3) Cost of Hash-join.
   (i) If recursive partitioning not required
   3(Blocks of r +blocks of s)
   (ii) If recursive partitioning required then
   2(blocks of r + blocks of s $(\lceil \log_{m-1}(\text{blocks of s}) \rceil - 1)$ + blocks of r+blocks of s

4) Other operations

   (a) Duplicate elimination
   (b) Projection
   (c) Aggregate functions.

## Check Your Progress 3

1) Methods used for evaluation of expressions:

   (a) Materialisation
   (b) Pipelining

2) Cost based optimisation

   (a) Generalising logically equivalent expressions using equivalence rules
   (b) Annotating resultant expressions to get alternative query plans.
   (c) Choosing the cheapest plan based on estimated cost.

3) Evaluation plan defines exactly what algorithms are to be used for each operation and the manner in which the operations are coordinated.

# UNIT 2 TRANSACTION MANAGEMENT AND RECOVERY

**Structure** **Page Nos.**


2.0 Introduction 28
2.1 Objectives 28
2.2 Transaction Processing 29
2.3 Enhanced Lock Based Protocols and Timestamp Based Protocols 31
2.3.1 Locking Protocols
2.3.2 Timestamp-Based Protocols
2.3.3 Multiple Granularity
2.4 Multi-Version Schemes 41
2.4.1 Multi-Version Timestamp Ordering
2.4.2 Multi-Version Two-Phase Locking
2.5 Deadlock Handling 42
2.5.1 Deadlock Prevention
2.5.2 Deadlock Detection
2.5.3 Deadlock Recovery
2.6 Weak Levels of Consistency 44
2.7 Concurrency in Index Structures 44
2.8 Failure Classification 45
2.9 Recovery Algorithms 46
2.9.1 Log-Based Recovery
2.9.2 Shadow Paging
2.9.3 Recovery with Concurrent Transactions
2.10 Buffer Management 52
2.10.1 Log Record Buffering
2.10.2 Database Buffering
2.11 Advanced Recovery Techniques 53
2.12 Remote Backup Systems 55
2.13 E-Transactions 56
2.14 Summary 56
2.15 Solutions/Answers 57


## 2.0 INTRODUCTION

You have already been introduced to some of the features of transactions and recovery in MCS-023, Block-2. Transactions are one of the most important components of a database management system application. A real challenge for a DBMS is not losing its integrity even when multiple transactions are going on. This unit provides a description on how concurrent transactions are handled in a DBMS. In addition, this unit also covers aspects related to recovery schemes when a number of concurrent transactions are going on. We will also explore the advanced features of DBMS in respect to Transaction Management and the recovery, transaction processing and the different methods of transaction processing. We have tried to highlight the problems that arise during concurrent transactions and the manner in which such problems may be avoided. We have explained the recovery method to ensure the consistency and ACID properties of transactions.

## 2.1 OBJECTIVES

After going through this unit, you should be able to:

- define transaction processing;
- describe the various protocols for concurrent transactions;
- explain the process of deadlock handling in DBMS;

- describe various recovery schemes when there is a failure, and
- define the buffer management system of the DBMS.

Formatted: Centered

and Recovery



## 2.2 TRANSACTION PROCESSING

Transactions processing involves a type of computer process in which the computer responds immediately to the user's request. Each request is considered as a *transaction.* For example, each operation performed by you through the ATM at the bank is transaction.

A database system needs to have a transaction system, which ensures consistent and error free transaction execution. Transaction Systems handle errors in a safe and consistent manner, but there are certain errors that cannot be avoided (e.g., Network errors or Database deadlocks) so a method that can handle these errors must exist. It is moreover, not possible to simply abort a current process. The updates of partial transactions may be reflected in the database leaving it in an inconsistent state. This could render the entire system unusable.

That is why transaction was introduced. A Transaction runs like a sub-program that modifies the database, leading from one consistent state to another. A Transaction must be atomic (i.e., either all modifications are done, or none of them are done). Transaction Systems are designed to guarantee that Transactions are atomic.

A transaction may be defined as a collection of operations on the database that performs a single logical function in a database application or it should be an inseparable list of database operations or Logical Unit of Work (LUW). It should not violate any database integrity constraints.

When a transaction processing system creates a transaction, it will ensure that the transaction has certain characteristics. These characteristics are known as the ACID properties.

ACID is an acronym for ATOMICITY, CONSISTENCY, ISOLATION and DURABILITY.

**ACID Properties**

**ATOMICITY:** The atomicity property identifies the transaction as atomic. An atomic transaction is either fully completed, or is not begun at all.

**CONSISTENCY:** A transaction enforces consistency in the system state, by ensuring that at the end of the execution of a transaction, the system is in a valid state. If the transaction is completed successfully, then all changes to the system will have been made properly, and the system will be in a valid state. If any error occurs in a transaction, then any changes already made will be automatically rolled back.

**ISOLATION:** When a transaction runs in isolation, it appears to be the only action that the system is carrying out at a time. If there are two transactions that are performing the same function and are running at the same time, transaction isolation will ensure that each transaction thinks that it has exclusive use of the system.

**DURABILITY:** A transaction is durable in that once it has been successfully completed, all of the changes it has made to the system are permanent. There are safeguards that will prevent the loss of information, even in the case of system failure.

Now, let us discuss some of the traditional and advanced transaction processing methods.

## Traditional Transaction Processing Methods

- **Batch Processing** is a method of computerised processing in which business transactions are accumulated over a period of time and prepared for processing as a single unit.

- **On-line Transaction Processing (OLTP)** is a method of computerised processing in which each transaction is processed immediately and the affected records are updated.

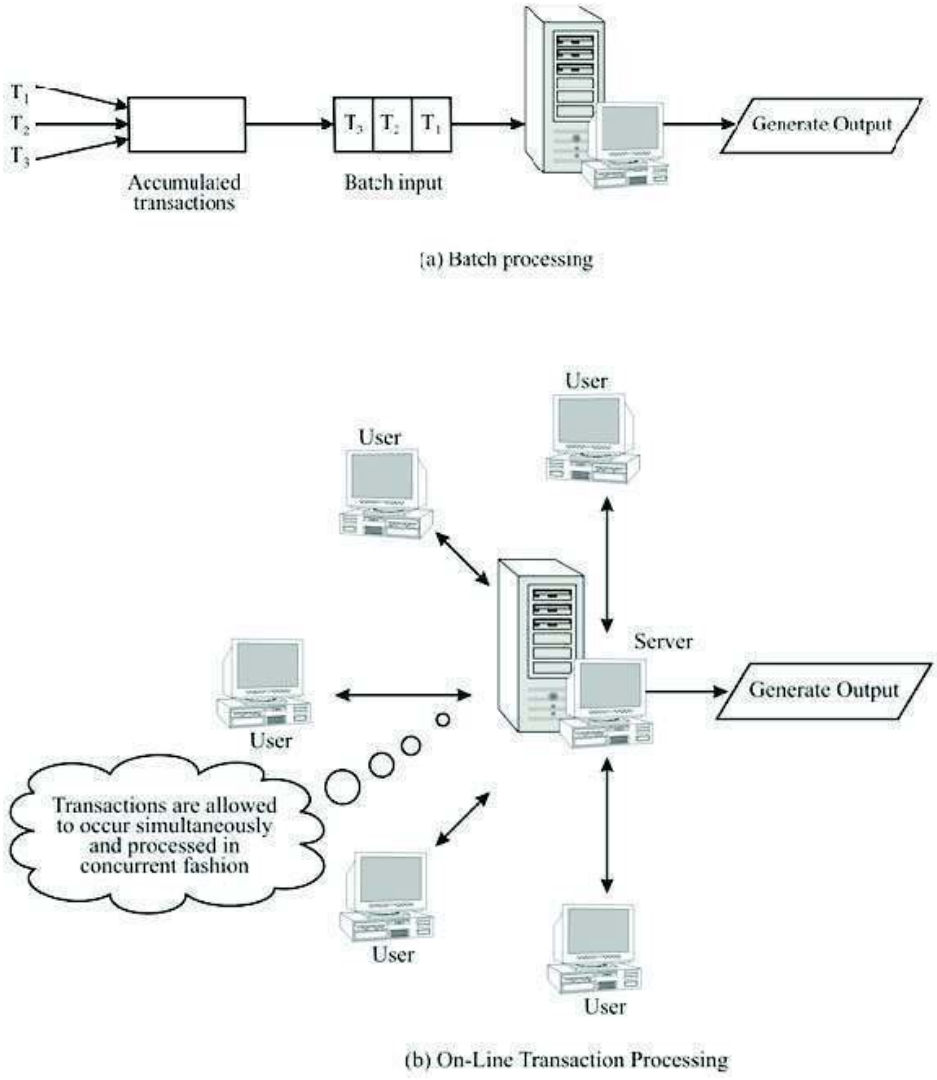



**Figure 1: Batch vs. online transaction**

The following are the Transaction Processing Activities:

- **Data collection:** Capturing data necessary for the transaction.
- **Data editing:** Checking validity and completeness.
  For example, a database has a value of 400 hours/week instead of 40 hours/week.
- **Data correction:** Correcting the incorrect data.
- **Data manipulation:** Calculating, summarising.
- **Data storage:** Updating the database as desired by the transaction.
- **Document production and reports:** Creating end results (paychecks).

**Advanced Transaction Processing**

In contrast to the traditional transaction system there are more flexible and user oriented processing methods such as:

- Remote Backup Systems
- Transaction-Processing Monitors
- High-Performance Transaction Systems
- Real-Time Transaction Systems
- Weak Levels of Consistency (Section 2.6)
- Transactional Workflows.

**Formatted:** Centered

and Recovery

**Comment:** Please include linking para. Please note that a student has to read and understand this unit. Thus, our language should be as if we are teaching in the class. You can also ask few questions and provide information to that in text questions. Broadly the language should be conversational. Please look in the complete unit from that viewpoint

**Remote Backup Systems**

Remote backup systems provide high availability by allowing transaction processing to continue even if the primary site is destroyed.

Backup site must be capable of detecting the failure, when the primary site has failed. To distinguish primary site failure from link failure it maintains several communication links between the primary and the remote backup sites.

Prior to the transfer control back to old primary when it recovers, the old primary must receive redo logs from the old backup and apply all updates locally.

**Transaction Processing Monitors**

TP monitors were initially developed as multithreaded servers to support large numbers of terminals from a single process. They provide infrastructure for building and administering complex transaction processing systems with a large number of clients and multiple servers.

A transaction-processing monitor has components for Input queue authorisation, output queue, network, lock manager, recovery manager, log manager, application servers, database manager and resource managers.

**High-Performance Transaction Systems**

High-performance hardware and parallelism helps improve the rate of transaction processing. They involve the following features for concurrency control that tries to ensure correctness without serialisability. They use database consistency constraints to split the database into sub databases on which concurrency can be managed separately.

**Real-Time Transaction Systems**

In systems with real-time constraints, correctness of execution involves both database consistency and the satisfaction of deadlines. Real time systems are classified as:

**Hard:** The task has zero value if it is completed after the deadline.
**Soft :** The task has diminishing value if it is completed after the deadline.

**Transactional Workflows** is an activity which involves the coordinated execution of multiple task performed different processing entities e.g., bank loan processing, purchase order processing.

## 2.3 ENHANCED LOCK BASED PROTOCOLS AND TIMESTAMP BASED PROTOCOLS

A lock based mechanism is used to maintain consistency when more than one transaction processes is executed. This mechanism controls concurrent access to data items. As per this mechanism the data items can be locked in two modes:

1) **Exclusive (X) mode:** Data item can be both read as well as written. X-lock is requested using **lock-X** instruction.

2) **Shared (S) mode:** Data item can only be read. S-lock is requested using **lock-S** instruction.

Lock requests are made to the concurrency-control manager. The transaction can proceed only after request is granted. The Lock-compatibility matrix shows when lock request will be granted (True).

| | **Unlocked** | **S** | **X** |
|---|---|---|---|
| **S** | True | True | True |
| **X** | True | False | False |

The above matrix shows that if an item is locked in shared mode then only a shared lock request can be permitted. In general, a transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions. Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive lock on the item, then no other transaction may hold any lock on the item. If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted. The following is an example of a transaction that requests and is granted/permitted locks:

$T_2$: **lock-S***(X)*;
**read** *(X)*;
**unlock***(X)*;
**lock-S***(Y)*;
**read** *(Y)*;
**unlock***(Y)*;
**display***(X+Y)*

Locking as above is not sufficient to guarantee serialisability — if *X* and *Y* are updated in-between the read of *X* and *Y* by some other transaction, the displayed sum would be wrong. A locking **protocol** is a set of rules to be followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.

### Pitfalls of Lock-Based Protocols

Consider the partial schedule:

| $T_A$ | $T_B$ |
|---|---|
| lock-x(A) | |
| read(X) | |
| X=X-1000 | |
| write (X) | lock-S (Y) |
| | read (Y) |
| | lock-S (X) |
| lock-X(Y) | |

Neither $T_A$ nor $T_B$ can make progress — executing **lock-S***(X)* causes $T_B$ to wait for $T_A$ to release its lock on *X*, while executing **lock-X***(Y)* causes $T_A$ to wait for $T_B$ to release its lock on *Y*. Such a situation is called a **deadlock**. To handle a deadlock one of $T_A$ or $T_B$ must be rolled back and its lock released. The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil. **Starvation** may also occur, if the concurrency control manager is badly designed. For example:

- A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item.
- The same transaction is repeatedly rolled back due to deadlocks.

These concurrency control manager needs to be designed to prevent starvation.

### 2.3.1 Locking Protocols





Formatted: Centered

They are primarily two types of looking protocols:

- Two phase locking
- Tree locking protocols.

Let us discuss them in more detail.

**The Two-Phase Locking Protocols**

This is a protocol, which ensures conflict-serialisable schedules read before write of transaction kindly (these transactions read a value before writing it). It consists of two phases:

Phase 1: Growing Phase

- Transaction may obtain locks
- Transaction may not release locks.

Phase 2: Shrinking Phase

- Transaction may release locks
- Transaction may not obtain locks.

The protocol assures serialisability. It can be proved that the transactions can be serialised in the order of their **lock points** (i.e., the point where a transaction acquires its final lock). Two-phase locking *does not* ensure freedom from deadlocks. Cascading roll-back is possible under two-phase locking as uncommitted values can be seen. To avoid this, follow a modified protocol called **strict two-phase locking**. Here a transaction must hold all its exclusive locks till it commits/aborts. **Rigorous two-phase locking** is even stricter, here *all* locks are held till commit/abort. In this protocol transactions can be serialised in the order in which they commit.

There can be serialisable schedules that cannot be obtained if two-phase locking is used. However, in the absence of extra information (e.g., ordering of access to data), two-phase locking is needed for serialisability in the following sense:

> *Given a transaction $T_i$ that does not follow two-phase locking, we may find another transaction $T_j$ that uses two-phase locking, and a schedule for $T_i$ and $T_j$ that is not conflict serialisable.*

**Lock Conversions**

Sometimes shared lock may need to be upgraded to an exclusive lock. The two-phase locking with lock conversions can be defined as:

First Phase:

- can acquire a **lock-S** on item
- can acquire a **lock-X** on item
- can convert a **lock-S** to a **lock-X** (**upgrade**)

Second Phase:

- can release a **lock-S**
- can release a **lock-X**
- can convert a **lock-X** to a **lock-S** (**downgrade**)

This protocol assures serialisability. But still relies on the programmer to insert the various locking instructions.

**Automatic Acquisition of Locks**

A transaction $T_i$ issues the standard read/write instruction, without explicit locking calls.

The operation **read**($D$) is processed as:

```
if Ti has a lock on D
  then
       read(D)
  else
       begin
         if necessary wait until no other
            transaction has a lock-X on D
         grant Ti a  lock-S on D;
         read(D)
       end
```

**write***(D)* is processed as:

```
if Ti has a  lock-X on D
  then
    write(D)
  else
   begin
     if necessary wait until no other transaction has any lock on D,
     if Ti has a lock-S on D
        then
          upgrade lock on D  to lock-X
        else
          grant Ti a lock-X on D
        write(D)
   end;
```

All locks are released after commit or abort.

**Implementation of Locking**

A **Lock manager** can be implemented as a separate process to which transactions send lock and unlock requests. The lock manager replies to a lock request by sending a lock grant messages (or a message asking the transaction to roll back, in case of a deadlock)

The requesting transaction waits until its request is answered. The lock manager maintains a data structure called a **lock table** to record granted locks and pending requests. The lock table is usually implemented as an in-memory hash table indexed on the name of the data item being locked.

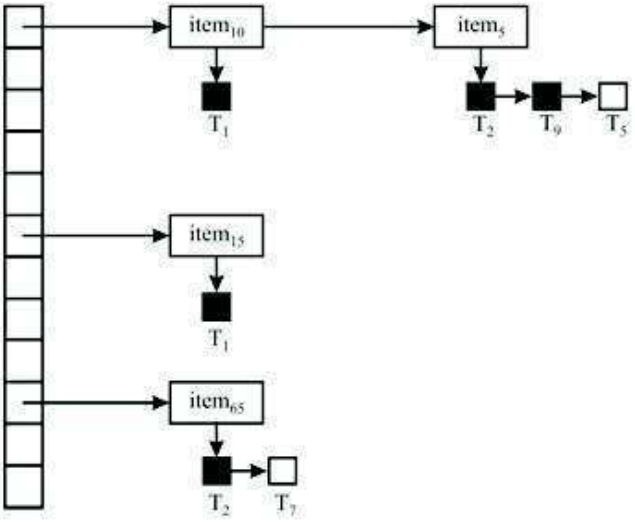



**Figure 2: Lock table**



... [4]

### Lock Table

Black rectangles in the table indicate granted locks, white ones indicate waiting requests. A lock table also records the type of lock granted or requested. A new request is added to the end of the queue of requests for the data item, and granted if it is compatible with all earlier locks. Unlock requests result in the request being deleted, and later requests are checked to see if they can now be granted. If transaction aborts, all waiting or granted requests of the transaction are deleted. The lock manager may then keep a list of locks held by each transaction, to implement this efficiently.

### Graph-Based Protocols

Graph-based protocols are an alternative to two-phase locking. They impose a partial ordering ($\rightarrow$) on the set **D** = $\{d_1, d_2, ..., d_h\}$ of all data items.

- If $d_i \rightarrow d_j$ then any transaction accessing both $d_i$ and $d_j$ must access $d_i$ before accessing $d_j$ this implies that the set **D** may now be viewed as a directed acyclic graph, called a *database graph*.

The *tree-protocol* is a simple kind of graph protocol.

### Tree Protocol

In a tree based protocol only exclusive locks are permitted. The first lock by $T_i$ may be on any data item. Subsequently, a data $Q$ can be locked by $T_i$ only if the parent of $Q$ is currently locked by $T_i$. Data items may be unlocked at any time. *Figure 3* shows a tree based protocol for the following lock (exclusive) – unlock requests from various transactions.

| Transaction T1 | Transaction T2 | Transaction T3 |
|---|---|---|
| Lock A | Lock B | Lock A |
| Lock B | Lock D | Lock C |
| Lock E | Lock G | Lock F |
| Unlock A | Unlock B | Unlock A |
| Unlock B | Unlock D | Unlock C |
| Unlock E | Unlock G | Unlock F |

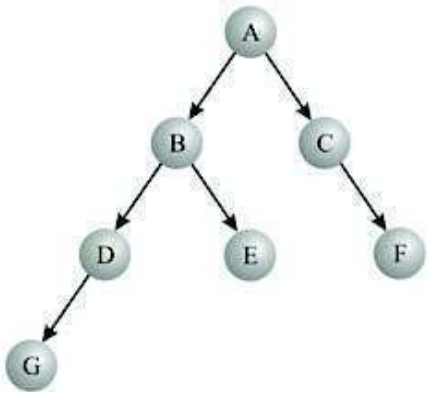



**Figure 3: Tree based protocol**

The tree protocol ensures conflict serialisability as well as freedom from deadlock for these transactions. Please note transaction T3 will get A only after it is set free by transaction T1. Similarly, T2 will get B when it is set free by T1. Unlocking may occur earlier in the tree-locking protocol than in the two-phase locking protocol. Thus Tree protocol is a protocol:

- with a shorter waiting time and increased concurrency,
- that is deadlock-free and does not require rollbacks,
- where aborting a transaction can still lead to cascading rollbacks.



However, in the tree-locking protocol, a transaction may have to lock data items that it does not access. Thus, it has:

- increased locking overhead, and additional waiting time.
- potential decrease in concurrency.

Schedules that are not possible in two-phase locking are possible under tree based protocol and vice versa.

**Insert and Delete Operations**

If two-phase locking is used:

- A **delete** operation may be performed only if the transaction deleting the tuple has an exclusive lock on the tuple to be deleted.
- A transaction that inserts a new tuple into the database is given an X-mode lock on the tuple.

Insertions and deletions can lead to the **phantom phenomenon**.

- A transaction that scans a relation (e.g., find all accounts in Bombay) and a transaction that inserts a tuple in the relation (e.g., insert a new account at Bombay) may be in conflict with each other, despite the fact that the two transactions are not accessing any tuple in common.
- If only tuple locks are used, non-serialisable schedules may be the result: the scan than transaction may not see the new account, yet may be serialised before the insert transaction.

The transaction scanning the relation reads the information that indicates what tuples the relation contains, while a transaction inserting a tuple updates the same information.

- The information should be locked.

One solution to such a problem may be:

- Associating a data item with the relation helps represent the information about the tuples the relation contains.
- Transactions scanning the relation acquires a shared lock in the data item.
- Transactions inserting or deleting a tuple acquires an exclusive lock on the data item. (Note: locks on the data item do not conflict with locks on individual tuples).

The above mentioned protocol provides very low concurrency for insertions/deletions. Index locking protocols provide higher concurrency while preventing the phantom phenomenon, (by requiring locks on certain index buckets).

**Index Locking Protocol:** Every relation must have at least one index. Access to a relation must be made only through one of the indices on the relation. A transaction $T_i$ that performs a lookup must lock all the index buckets that it accesses, in S-mode. A transaction $T_i$ may not insert a tuple $t_i$ into a relation $r$ without updating all indices to $r$. $T_i$ must perform a lookup on every index to find all index buckets that could have possibly contained a pointer to tuple $t_i$, had it existed already, and obtain locks in X-mode on all these index buckets. $T_i$ must also obtain locks in X-mode on all index buckets that it modifies. The rules of the two-phase locking protocol must be observed for index locking protocols to be effective.

### 2.3.2 Timestamp-Based Protocols

Each transaction is issued a timestamp when it enters the system. If an old transaction $T_i$ has time-stamp TS($T_i$), a new transaction $T_j$ is assigned time-stamp TS($T_j$) such that TS($T_i$) <TS($T_j$). This protocol manages concurrent execution in such a manner that the time-stamps determine the serialisability order. In order to assure such behaviour, the protocol needs to maintain for each data $Q$ two timestamp values:

- **W-timestamp**($Q$) is the largest time-stamp of any transaction that executed **write**($Q$) successfully.
- **R-timestamp**($Q$) is the largest time-stamp of any transaction that executed **read**($Q$) successfully.

The timestamp ordering protocol executes any conflicting **read** and **write** operations in timestamp order. Suppose a transaction $T_i$ issues a **read**($Q$).

1) If TS($T_i$) <**W**-timestamp($Q$), $\Rightarrow$ $T_i$ needs to read a value of $Q$ that was already overwritten. Action: reject **read** operation and rolled back $T_i$.
2) If TS($T_i$)≥ **W**-timestamp($Q$), $\Rightarrow$ It is OK. Action: Execute **read** operation and set R-timestamp($Q$) to the maximum of R-timestamp($Q$) and TS($T_i$).

Suppose that transaction $T_i$ issues **write** ($Q$):

1) If TS($T_i$) < R-timestamp($Q$), $\Rightarrow$ the value of $Q$ that $T_i$ is writing was used previously, this value should have never been produced. Action: Reject the **write** operation. Roll back $T_i$
2) If TS($T_i$) < W-timestamp($Q$), $\Rightarrow$ $T_i$ is trying to write an obsolete value of $Q$. Action: Reject **write** operation, and roll back $T_i$.
3) Otherwise, execute **write** operation, and set W-timestamp($Q$) to TS($T_i$).

**Correctness of Timestamp-Ordering Protocol**

The timestamp-ordering protocol guarantees serialisability if all the arcs (please refer to MCS-023, Block-2 for rules on drawing precedence graph arcs) in the precedence graph of the form as shown in *Figure 4* and there is no cycle.

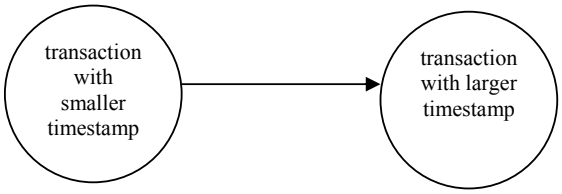


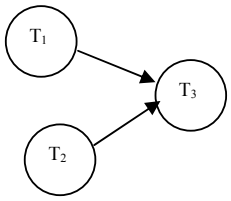

**Figure 4: Arcs of a precedence graph in timestamp ordering protocol**

Timestamp protocol ensures freedom from a deadlock as no transaction can wait. But the schedule may not be cascade-free, and may not even be recoverable.

**Example:** Use of the timestamp based Protocol is shown in the example below. Assume the following table showing a partial schedule for several data items for transactions with timestamps 1, 2, 3. Please note that all the transactions $T_1$ and $T_2$ allowed the read values of X and Y which are written by a transaction with a lower timestamp than that of these transactions. The transaction $T_3$ will also be allowed to write the values of X and Y as the read timestamp value of these variables is 2 and since it reads X and Y itself so the read timestamp becomes 3 for X and Y. Thus, no later transaction have the read values and hence $T_3$ will be allowed to write the values of X and Y.

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| | | read (X) |
| read (Y) | read (Y) | |
| | | read (Y) |
| | read (X) | |
| | abort | |
| read (X) | | |
| | | write (X) |
| | | write (Y) |

The precedence graph of these transactions would be (Please recollect the rules of drawing precedence graph).




Thus, the partial ordering of execution is $T_1$, $T_2$, $T_3$, or $T_2$, $T_1$, $T_3$.

**Recoverability and Cascade Freedom**

*Problem with timestamp-ordering protocol*:

- Suppose $T_i$ aborts, but $T_j$ has read a data item written by *Ti* then $T_j$ must also be aborted; if $T_j$ had committed earlier, the schedule is **not recoverable.**
- Also, any transaction that has read a data item written by $T_j$ must abort too.

Thus, there may occur a cascading rollback – that is, a chain of rollbacks.

*Solution:*

- A transaction is structured in such a manner that all the writes are performed at the end of its processing.
- All such writes of a transaction form an atomic action; no transaction may execute while a transaction is being written.
- A transaction that aborts or is rolled back is restarted with a new timestamp.

**Thomas' Write Rule**: Modified versions of the timestamp-ordering protocol in which obsolete write operations may be ignored under certain circumstances. When $T_i$ attempts to write data item $Q$, if TS ($T_i$) < W-timestamp($Q$), then $T_i$ is attempting to write an obsolete value of {$Q$}. Hence, rather than rolling back $T_i$ as the timestamp ordering protocol would have done, this {write} operation can be ignored. Otherwise this protocol is the same as the timestamp ordering protocol. Thomas' Write Rule allows greater potential concurrency.

**Validation-Based Protocol**

In certain cases when the write operations are minimum then a simple protocol may be used. In such a protocol an execution of transaction $T_i$ is done in three phases.

1) **Read and execution phase**: Transaction $T_i$ writes only to temporary local variables.

2) **Validation phase**: Transaction $T_i$ performs a "validation test" to determine whether local variables can be written without violating serialisability.

3) **Write phase**: If $T_i$ is validated, the updates are applied to the database otherwise $T_i$ is rolled back.

The three phases of concurrently executing transactions can be interleaved, but each transaction must go through the three phases in that order. This protocol is also known as **Optimistic Concurrency Control** since the transaction executes fully in the hope that all will go well during the validation.

Each transaction $T_i$ has 3 timestamps:

- **Start**($T_i$) : The time when $T_i$ starts the execution.
- **Validation**($T_i$): The time when $T_i$ started with the validation phase.
- **Finish**($T_i$) : The time when $T_i$ completed its write phase.

Serialisability order is determined by a timestamp given at the time of validation, to increase concurrency. Thus $TS(T_i)$ is given the value of **Validation**($T_i$).



... [6]

This protocol is useful and gives a greater degree of concurrency provided the probability of conflicts is low. That is because the serialisability order is not pre-decided and relatively less transactions will have to be rolled back.

**Validation Test for Transaction $T_j$**

If for all $T_i$ with TS ($T_i$) < TS ($T_j$) either one of the following condition holds:

- **finish**($T_i$) < **start**($T_j$)
- **start**($T_j$) < **finish**($T_i$) < **validation**($T_j$) **and** the set of data items written by $T_i$ does not intersect with the set of data items read by $T_j$, then validation succeeds and $T_j$ can be committed. Otherwise, validation fails and $T_j$ is aborted.

*Justification*: If the first condition is satisfied, and there is no execution overlapping, or if the second condition is satisfied and,

- the writes of $T_j$ do not affect reads of $T_i$ since they occur after $T_i$ has finished its reads.
- the writes of $T_i$ do not affect reads of $T_j$ since $T_j$ does not read any item written by $T_i$.

**Schedule Produced by Validation**

Example of schedule produced using validation:

| $T_A$ | $T_B$: Transfer of Rs.1000/- From account X to account Y |
|---|---|
| read (X) | read (X)<br>X=X−1000<br>read (Y)<br>Y=Y+1000 |
| | (validate)<br>write (X) |
| read (Y)<br>(validate)<br>The validation here may fail although no error has occurred. | write (Y) |

## 2.3.3 Multiple Granularity

In the concurrency control schemes, we used each individual data item as a unit on which synchronisation is performed. Can we allow data items to be of various sizes and define a hierarchy of data granularities, whereby the small granularities are nested within larger ones? It can be represented graphically as a tree (not tree-locking protocol). In such a situation, when a transaction locks a node in the tree explicitly, it implicitly locks all the node's descendents in the same mode. Granularity of locking (the levels in tree where locking is done) are:

- fine granularity (lower in the tree): It allows high concurrency but has high locking overhead,
- coarse granularity (higher in the tree): It has low locking overhead but also has low concurrency.

*Figure 5* shows an example of Granularity Hierarchy.

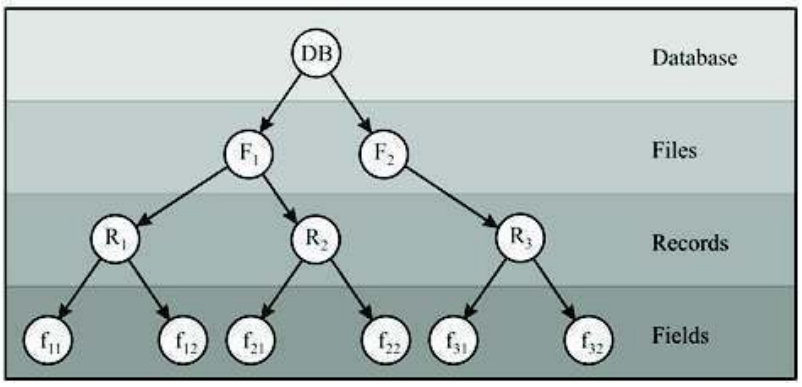



**Figure 5: Hierarchical locking**

The highest level in the example hierarchy is the entire database. The levels below are of file, record and field.

But how is locking done in such situations? Locking may be done by using intention mode locking.

**Intention Lock Modes**

In addition to S and X lock modes, there are three additional lock modes with multiple granularity:

- ***Intention-shared*** (IS): indicates explicit locking at a lower level of the tree but only with shared locks.
- ***Intention-exclusive*** (IX): indicates explicit locking at a lower level with exclusive or shared locks
- ***Shared and intention-exclusive*** (SIX): the sub tree rooted by that node is locked explicitly in shared mode and explicit locking is being done at a lower level with exclusive-mode locks.

Intention locks allow a higher-level node to be locked in share (S) or exclusive (X) mode without having to check all descendent nodes.

Thus, this locking scheme helps in providing more concurrency but lowers the lock overheads.

**Compatibility Matrix with Intention Lock Modes**: The following figure shows the compatible matrix that allows locking.

| | IS | IX | S | S IX | X |
|---|---|---|---|---|---|
| **IS** | ✓ | ✓ | ✓ | ✓ | × |
| **IX** | ✓ | ✓ | | × | × |
| **X** | ✓ | × | ✓ | × | × |
| **S IX** | ✓ | × | × | × | × |
| **X** | × | × | × | × | × |

## ☞ Check Your Progress 1

1) How do you define transaction processing methods? Also list the advanced transaction methods.

……

……

……

2) Why we use lock based protocol? What are the pitfalls of lock based protocol?

……

……

……

3) What is a timestamp? What is the need of time stamping protocol?

……

……

……

4) How are insert and delete operations performed in a tree protocol?

……

……

……

5) Define multiple granularity?

……

……

……

6) Test all the protocols discussed till now.

……

……

……

## 2.4 MULTI-VERSION SCHEMES

Multi-version schemes maintain old versions of data item to increase concurrency. These schemes are available for:

- Multi version Timestamp Ordering
- Multi-version Two-Phase Locking.

In multi-version schemes each successful **write** results in the creation of a new version of the data item written.

Timestamps can be used to label the versions. When a **read** (Q) operation is issued, you can select an appropriate version of Q based on the timestamp of the transaction, and return the value of the selected version. **Read** operation never has to wait as an appropriate version is returned immediately.

### 2.4.1 Multi-Version Timestamp Ordering

Each data item Q has a sequence of versions <$Q_1$, $Q_2$,...., $Q_m$>. Each version $Q_k$ contains three data fields:

**Content**−the value of version $Q_k$.
**W-timestamp** ($Q_k$) − timestamp of the transaction that created (wrote) the version $Q_k$
**R-timestamp** ($Q_k$) − largest timestamp of a transaction that successfully read version $Q_k$.

When a transaction $T_i$ creates a new version $Q_k$ of Q, $Q_k$'s W-timestamp and R-timestamp are initialised to TS($T_i$). R-timestamp of $Q_k$ is updated whenever a transaction $T_j$ reads $Q_k$, and TS($T_j$) > R-timestamp($Q_k$).

1) If transaction Ti issues a read(Q) , then the value returned is the content of version Qk.

2) If transaction Ti issues a write(Q), and if TS(Ti) < R-timestamp(Qk), then Transaction Ti is rolled back, otherwise, if TS(Ti)= W- timestamp(Qk), the contents of Qk are overwritten a new version of Q is created.

The following multi-version technique ensures serialisability.

Suppose that transaction $T_i$ issues a read (Q) or write(Q) operation. Let $Q_k$ denote the version of Q whose write timestamp is the largest write timestamp less than or equal to TS($T_i$).

1) If transaction $T_i$ issues a read (Q), then the value returned is the content of version $Q_k$.
2) If transaction $T_i$ issues a write (Q), and if TS($T_i$) < R-timestamp($Q_k$), then transaction $T_i$ is rolled back. Otherwise, if TS($T_i$) = W-timestamp($Q_k$), the contents of $Q_k$ are overwritten, then a new version of Q is created.

Read always succeeds. A write by $T_i$ is rejected if some other transaction $T_j$ (in the serialisation order defined by the timestamp values) should read $T_i$'s write, has already read a version created by a transaction older than $T_i$.

### 2.4.2 Multi-Version Two-Phase Locking

It differentiates between read-only transactions and update transactions. Update transactions acquire read and write locks, and hold all locks up to the end of the transaction. That is, update transactions follow rigorous two-phase locking.

1) Each successful **write** results in the creation of a new version of the data item written.
2) Each version of a data item has a single timestamp whose value is obtained from a counter-transaction that is incremented during commit processing.

Read-only transactions are assigned a timestamp. By reading the current value of transaction counter before they start execution; they follow the multi-version timestamp-ordering protocol for performing the read function.

When an update transaction wants to read a data item, it obtains a shared lock on it, and reads the latest version. When it wants to write an item, it obtains X lock on it; then creates a new version of the item and sets this version's timestamp to when update transaction $T_i$ completes and commit processing occurs:

1) $T_i$ sets timestamp on the versions it has created to transaction-counter + 1
2) $T_i$ increments transaction-counter by 1.

Read-only transactions that start after $T_i$ increments transaction-counter will see the values updated by $T_i$. Read-only transactions that start before $T_i$ increments the transaction counter, will see the value before the updates by $T_i$. Only serialisable schedules are produced using this method.

## 2.5 DEADLOCK HANDLING

Consider the following two transactions:

| | | | |
|---|---|---|---|
| $T_1$: | write (X) | $T_2$: | write(Y) |
| | write(Y) | | write(X) |

A schedule with deadlock is given as:

| T 1 | T 2 |
|---|---|
| **lock-X** on X | |
| write (X) | |
| | **lock-X** on Y |
| | write (X) |
| | wait for **lock-X** on X |
| wait for **lock-X** on Y | |

A system is deadlocked if a set of transactions in the set is waiting for another transaction in the set. Let us discuss certain mechanisms to deal with this deadlock.

### 2.5.1 Deadlock Prevention

*Deadlock prevention* protocols ensure that the system will never enter into a deadlock state. The basic prevention strategies are:

- The strategies require that each transaction lock all its data items before it begins execution (pre declaration).
- They impose partial ordering of all data items and require that a transaction can lock data items only in the order specified by the partial order (graph-based protocol).

Two common techniques for deadlock prevention are *wait-die* and *wound-wait.* In both the *wait-die* and *wound-wait* schemes, a rolled back transaction is restarted with its original timestamp. Older transactions thus have precedence over newer ones, and deadlock is hence avoided.

These schemes use transaction timestamps for the prevention of deadlock.

**Wait-die** scheme — non-preemptive

- Older transaction may wait for younger one to release data item. Younger transactions never wait for older ones; they are rolled back instead.
- A transaction may die several times before acquiring the needed data item

**Wound-wait** scheme — preemptive

- Older transaction wounds (forces rollback) of younger transaction instead of waiting for it. Younger transactions may wait for older ones.
- May be fewer rollbacks than wait-die scheme.

*Timeout-Based Schemes*:

The timeout-based schemes have the following characteristics:

1) A transaction waits for a lock only for a specified amount of time. After that, the wait times out and transaction are rolled back. Thus deadlocks are prevented.

2) Simple to implement; but starvation may occur. Also it is difficult to determine the good value of timeout interval.

### 2.5.2 Deadlock Detection

Deadlocks can be detected using a wait-for graph, which consists of a pair G = (V,E),

- V is a set of vertices (all the transactions in the system)
- E is a set of edges; each element is an ordered pair $T_i \rightarrow T_j$.

If $T_i \rightarrow T_j$ is in E, then there is a directed edge from $T_i$ to $T_j$, implying that $T_i$ is waiting for $T_j$ to release a data item. When $T_i$ requests a data item currently being held by $T_j$, then the edge $T_i \rightarrow T_j$ is inserted in the wait-for graph. This edge is removed only when $T_j$ is no longer holding a data item needed by $T_i$.

The system is in a state of deadlock if the wait-for graph has a cycle. You must make use of a deadlock-detection algorithm periodically to look for such cycles.

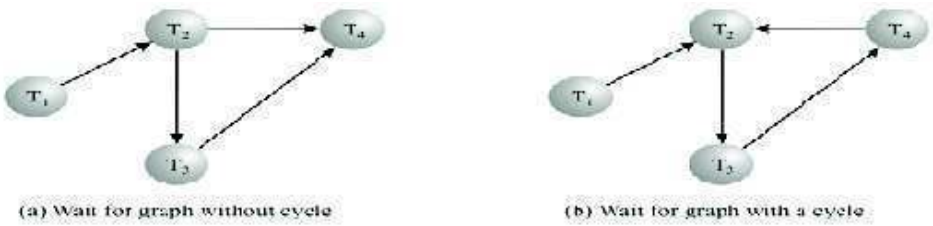



**Figure 6: Deadlock detection**

### 2.5.3 Deadlock Recovery

When a deadlock is detected, some transaction will have to be rolled back to break the deadlock. Selecting that transaction, as a victim will incur minimum cost. Rollback will determine the distance the transaction needs to be rolled back. Total rollback aborts the transaction and then restarts it. The more effective method of deadlock recovery is to rollback transaction only as far as necessary to break the deadlock. Starvation occurs if the same transaction is always selected as a victim. You can include the number of rollbacks in the cost factor calculation in order to avoid starvation.

## 2.6 WEAK LEVELS OF CONSISTENCY

The protocols such as strict two phase locking protocol restricts concurrency while transactions are being execution. Can we allow more concurrency by compromising on correctness/accurateness, which now needs to be ensured by database programmers rather than by the DBMS? We can operate on weak levels of consistency using the following mechanism:

**Degree-two consistency**

Degree-two consistency differs from two-phase locking in that S-locks may be released at any time, and locks may be acquired at any time, however,

- X-locks must be held till the transaction has ended
- Serialisability is not guaranteed. The programmer must ensure that no erroneous database state will occur.

One of the degree two-consistency level protocols is Cursor stability. It has the following rules:

- For reads, each tuple is locked, read, and lock is immediately released
- X-locks are held till end of transaction.

**Weak Levels of Consistency in SQL**

SQL allows non-serialisable executions.

- **Serialisable** is the default
- **Repeatable read** allows only committed records to be read, and repeating a read should return the same value (so read locks should be retained). (However, the phantom phenomenon need not be prevented) that is, T1 may see some records inserted by T2, but may not see others inserted by T2.
- **Read committed** same as degree two consistency, but most systems implement it as cursor-stability.
- **Read uncommitted** allows even uncommitted data to be read. This is the level, which has almost no restriction on concurrency but will result in all sorts of concurrency related problems.

## 2.7 CONCURRENCY IN INDEX STRUCTURES

Indices are unlike other database items in that their only job is to help in accessing data. Index-structures are typically accessed very often, much more than other database items. Treating index-structures like other database items leads to low concurrency. Two-phase locking on an index may result in transactions being executed practically one-at-a-time. It is acceptable to have nonserialisable concurrent access to an index as long as the accuracy of the index is maintained. In particular, the exact values read in an internal node of a $B^+$-tree are irrelevant so long as we land up in the correct leaf node. There are index concurrency protocols where locks on internal nodes are released early, and not in a two-phase fashion.

**Example of index concurrency protocol**: Use **crabbing** instead of two-phase locking on the nodes of the $B^+$-tree, as follows. During search/insertion/deletion:

- First lock the root node in shared mode.
- After locking all required children of a node in shared mode, release the lock on the node.
- During insertion/deletion, upgrade leaf node locks to exclusive mode.
- When splitting or coalescing requires changes to a parent, lock the parent in exclusive mode.

## 2.8 FAILURE CLASSIFICATION

A DBMS may encounter a failure. These failure may be of the following types:
**Transaction failure**. An ongoing transaction may fail due to:

- **Logical errors**: Transaction cannot be completed due to some internal error condition.
- **System errors**: The database system must terminate an active transaction due to an error condition (e.g., deadlock).

**System crash**: A power failure or other hardware or software failure causes the system to crash.

- **Fail-stop assumption**: Non-volatile storage contents are assumed to be uncorrupted by system crash.
- **Disk failure**: A head crash or similar disk failure destroys all or part of the disk storage capacity.
- **Destruction is assumed to be detectable**: Disk drives use checksums to detect failure.

All these failures result in the inconsistent state of a transaction. Thus, we need a recovery scheme in a database system, but before we discuss recovery. Let us briefly define the storage structure from a recovery point of view.

**Storage Structure**

There are various ways for storing information.

**Volatile storage**

- does not survive system crashes
- examples: main memory, cache memory

**Nonvolatile storage**

- survives system crashes
- examples: disk, tape, flash memory, non-volatile (battery backed up) RAM.

**Stable storage**

- a mythical form of storage that survives all failures
- approximated by maintaining multiple copies on distinct nonvolatile media.

**Stable-Storage Implementation**

A stable storage maintains multiple copies of each block on separate disks. Copies can be kept at remote sites to protect against disasters such as fire or flooding. Failure during data transfer can still result in inconsistent copies. A block transfer can result in:

- Successful completion
- Partial failure: destination block has incorrect information
- Total failure: destination block was never updated.

For protecting storage media from failure during data transfer we can execute output operation as follows (assuming two copies of each block):

1) Write the information on the first physical block.
2) When the first write successfully completes, write the same information on the second physical block.
3) The output is completed only after the second write is successfully completed.

Copies of a block may differ due to failure during output operation. To recover from this failure you need to first find the inconsistent blocks:

*One Expensive solution:* is comparing the two copies of every disk block.

*A Better solution may be to:*

- record in-progress disk writes on non-volatile storage (Non-volatile RAM or special area of disk),
- use this information during recovery to find blocks that may be inconsistent, and only compare copies of these, and
- used in hardware-RAID systems.

If either copy of an inconsistent block is detected with an error (bad checksum), overwrite it with the other copy. If both have no error, but are different, overwrite the second block with the first block.

### ☞ Check Your Progress 2

1) Define multi-version scheme?

……

……

2) Define dead locks? How can dead locks be avoided?

……

……

3) Define weak level of consistency in SQL?

……

……

4) How is stable storage implemented?

……

……

## 2.9 RECOVERY ALGORITHMS

Recovery algorithms are techniques to ensure database consistency and transaction atomicity and durability despite failures. Recovery algorithms have two parts:

1) Actions taken during normal transaction processing is to ensure that enough information exists to recover from failures,
2) Actions taken after a failure to recover the database contents to a state that ensures atomicity, consistency and durability.

While modifying the database, without ensuring that the transaction will commit, may leave the database in an inconsistent state. Let us consider an example transaction $T_1$ that transfers Rs.1000/- from account $X$ to account $Y$; goal is either to perform all database modifications made by $T_1$ or none at all. $T_1$ modifies x by subtracting Rs.1000/- and modifies Y by adding Rs.1000/-. A failure may occur after one of these modifications has been made, but before all of them are made. To ensure consistency despite failures, we have several recovery mechanisms.

Let us now explain two approaches: **log-based recovery** and **shadow paging**.

Formatted: Centered

and Recovery

### 2.9.1 Log-Based Recovery

A log is maintained on a stable storage media. The log is a sequence of **log records**, and maintains a record of update activities on the database. When transaction $T_i$ starts, it registers itself by writing a

$<T_i$ **start**>log record

*Before* $T_i$ executes **write**($X$), a log record $<T_i, X, V_1, V_2>$ is written, where $V_1$ is the value of $X$ before the write (undo value), and $V_2$ is the value to be written to $X$ *(redo value).*

- Log record notes that $T_i$ has performed a write on data item ***X***. ***X*** had value $V_1$ before the write, and will have value $V_2$ after the write.

- When $T_i$ finishes it last statement, the log record **<**$T_i$ **commit>** is written. We assume for now that log records are written directly to a stable storage media (that is, they are not buffered).

Two approaches for recovery using logs are:

- Deferred database modification.
- Immediate database modification.

#### Deferred Database Modification

The **deferred database modification** scheme records all the modifications to the log, but defers all the **write**s to after partial commit. Let us assume that transactions execute serially, to simplify the discussion.

A transaction starts by writing $<T_i$ ***start***> record to log. A **write**($X$) operation results in a log record $<T_i, X, V>$ being written, where $V$ is the new value for $X$. The write is not performed on $X$ at this time, but is deferred. When $T_i$ partially commits, $<T_i$ **commit**> is written to the log. Finally, the log records are read and used to actually execute the previously deferred writes. During recovery after a crash, a transaction needs to be redone if both $<T_i$ **start**> and$<T_i$ **commit**> are there in the log. Redoing a transaction $T_i$ (**redo**$T_i$) sets the value of all data items updated by the transaction to the new values. Crashes can occur while

- the transaction is executing the original updates, or
- while recovery action is being taken.

Example:

Transactions $T_1$ and $T_2$ ($T_1$ executes before $T_2$):

| $T_1$: **read** (X) | $T_2$ : **read** ($Z$) |
|---|---|
| $X = X - 1000$ | Z= $Z - 1000$ |
| **Write** (X) | **write** ($Z$) |
| **read** (Y) | |
| $Y = Y + 1000$ | |
| **write** (Y) | |

Formatted: French France

The following figure shows the log as it appears at three instances of time (Assuming that initial balance in X is 10,000/- Y is 8,000/- and Z has 20,000/-):

| | | |
|---|---|---|
| <$T_1$ start> | <$T_1$ start> | <$T_1$ start> |
| <$T_1$, X 9000> | <$T_1$, X 9000> | <$T_1$, X 9000> |
| <$T_1$, Y 9000> | <$T_1$, Y 9000> | <$T_1$, Y 9000> |
| | <$T_1$, Commit> | <$T_1$, Commit> |
| | <$T_2$ start> | <$T_2$ start> |
| | <$T_2$, Z 19000> | <$T_2$, Z 19000> |
| | | <$T_2$, Commit> |
| (a) | (b) | (c) |

Formatted: French France



If log on stable storage at the time of crash as per (a) (b) and (c) then in:

(a) No redo action needs to be performed.

(b) redo($T_1$) must be performed since <$T_1$ **commit**> is present

(c) redo($T_2$) must be performed followed by redo($T_2$) since

<$T_1$ **commit**> and <$T_2$ **commit**> are present.

Please note that you can repeat this sequence of redo operation as suggested in (c) any number of times, it will still bring the value of X, Y, Z to consistent redo values. This property of the redo operation is called **idempotent.**

**Immediate Database Modification**

The **immediate database modification** scheme allows database updates on the stored database even of an uncommitted transaction. These updates are made as the writes are issued (since undoing may be needed, update logs must have both the **old value** as well as the **new value**). Updated log records must be written *before* database item is written (assume that the log record is output directly to a stable storage and can be extended to postpone log record output, as long as prior to execution of an **output** (Y) operation for a data block Y all log records corresponding to items Y must be flushed to stable storage).

Output of updated blocks can take place at any time before or after transaction commit. Order in which blocks are output can be different from the order in which they are written.

Example:

| Log | Write operation | Output |
|---|---|---|
| <$T_1$ **start**> | | |
| <$T_1$, X, 10000, 9000> | | |
| $T_1$, Y, 8000, 9000 | | |
| | $X$ = 9000 | Output Block of X |
| | $Y$ = 9000 | Output Block of Y |
| <$T_1$ **commit**> | | |
| <$T_2$ **start**> | | |
| <$T_2$, Z, 20,000, 19,000> | | |
| | $Z$= 19000 | Output Block of Z |
| <$T_2$ **commit**> | | |

The recovery procedure in such has two operations instead of one:

- **undo**($T_i$) restores the value of all data items updated by $T_i$ to their old values, moving backwards from the last log record for $T_i$,
- **redo**($T_i$) sets the value of all data items updated by $T_i$ to the new values, moving forward from the first log record for $T_i$.

Both operations are **idempotent, *that is,*** even if the operation is executed multiple times the effect is the same as it is executed once. (This is necessary because operations may have to be re-executed during recovery).

When recovering after failure:

- Transaction $T_i$ needs to be undone if the log contains the record <$T_i$ **start**>, but does not contain the record <$T_i$ **commit**>.

- Transaction $T_i$ needs to be redone if the log contains both the record <$T_i$ **start**> and the record <$T_i$ **commit**>.

Undo operations are performed first, then redo operations.

**Example**:

Consider the log as it appears at three instances of time.

| (a) | (b) | (c) |
|---|---|---|
| <$T_1$ start> | <$T_1$ start> | <$T_1$ start> |
| <$T_1$, X 10000, 9000> | <$T_1$, X 10000, 9000> | <$T_1$, X 10000, 9000> |
| <$T_1$, Y 8000, 9000> | <$T_1$, Y 8000, 9000> | <$T_1$, Y 8000, 9000> |
| | <$T_1$, Commit> | <$T_1$, Commit> |
| | <$T_2$ start> | <$T_2$ start> |
| | <$T_2$, Z 20000, 19000> | <$T_2$, Z 20000, 19000> |
| | | <$T_2$, Commit> |

Formatted: Centered

and Recovery

Formatted: French France

Recovery actions in each case above are:

(a) undo ($T_1$): Y is restored to 8000 and X to 10000.

(b) undo ($T_2$) and redo ($T_1$): Z is restored to 20000, and then X and Y are set to 9000 and 9000 respectively.

(c) redo ($T_1$) and redo ($T_2$): X and Y are set to 9000 and 9000 respectively. Then *Z* is set to 19000

## Checkpoints

The following problem occurs during recovery procedure:

- searching the entire log is time-consuming as we are not aware of the consistency of the database after restart. Thus, we might unnecessarily redo transactions, which have already output their updates to the database.

Thus, we can streamline recovery procedure by periodically performing **check pointing**. Check pointing involves:

- Output of all the log records currently residing in the non-volatile memory onto stable storage.
- Output all modified buffer blocks to the disk.
- Write a log record < **checkpoint**> on a stable storage.

During recovery we need to consider only the most recent transactions that started before the checkpoint and is not completed till, checkpoint and transactions started after check point. Scan backwards from end of log to find the most recent <**checkpoint**> record. Continue scanning backwards till a record <$T_i$ **start**> is found. Need only consider part of the log following above **star**t record. The earlier part of the log may be ignored during recovery, and can be erased whenever desired. For all transactions (starting from $T_i$ or later) with no <$T_i$ **commit**>, execute **undo** ***(****$T_i$*).* (Done only in case immediate modification scheme is used). Scanning forward in the log, for all transactions starting from $T_i$ or later with a <$T_i$ **commit**>, execute **redo***(*$T_i$*).*

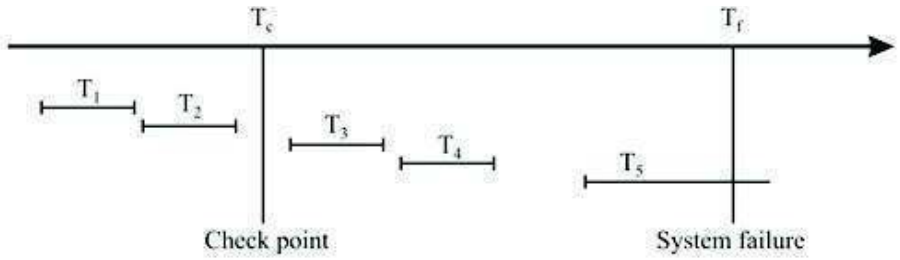



**Figure 7: Checkpoint and failure**

- $T_1$ and $T_2$ can be ignored (updates already output to disk due to checkpoint)
- $T_3$ and $T_4$ are redone.
- $T_5$ is undone.

### 2.9.2 Shadow Paging

**Shadow paging** is an alternative to log-based recovery; this scheme is useful if transactions are executed serially. In this *two* page tables are maintained during the lifetime of a transaction–the **current page table**, and the **shadow page table.** It **s**tores the shadow page table in nonvolatile storage, in such a way that the state of the database prior to transaction execution may be recovered (shadow page table is never modified during execution). To start with, both the page tables are identical. Only the current page table is used for data item accesses during execution of the transaction. Whenever any page is about to be written for the first time a copy of this page is made on an unused page, the current page table is then made to point to the copy and the update is performed on the copy.

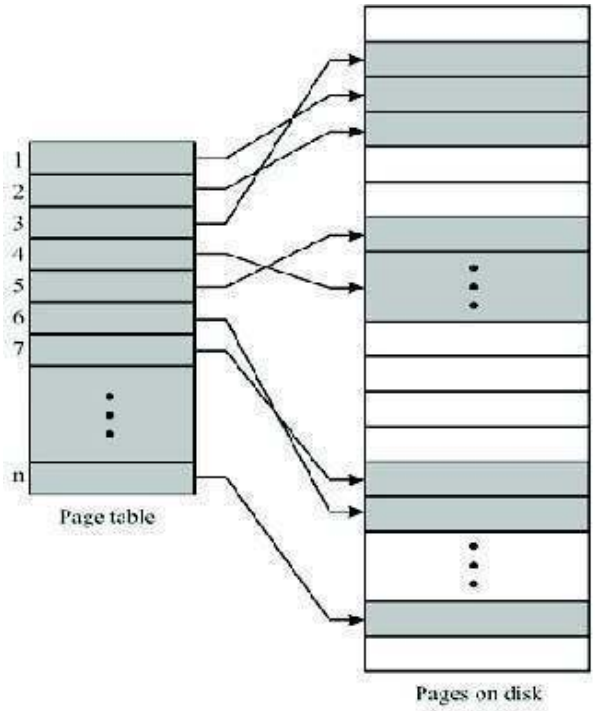



**Figure 8: A Sample page table**

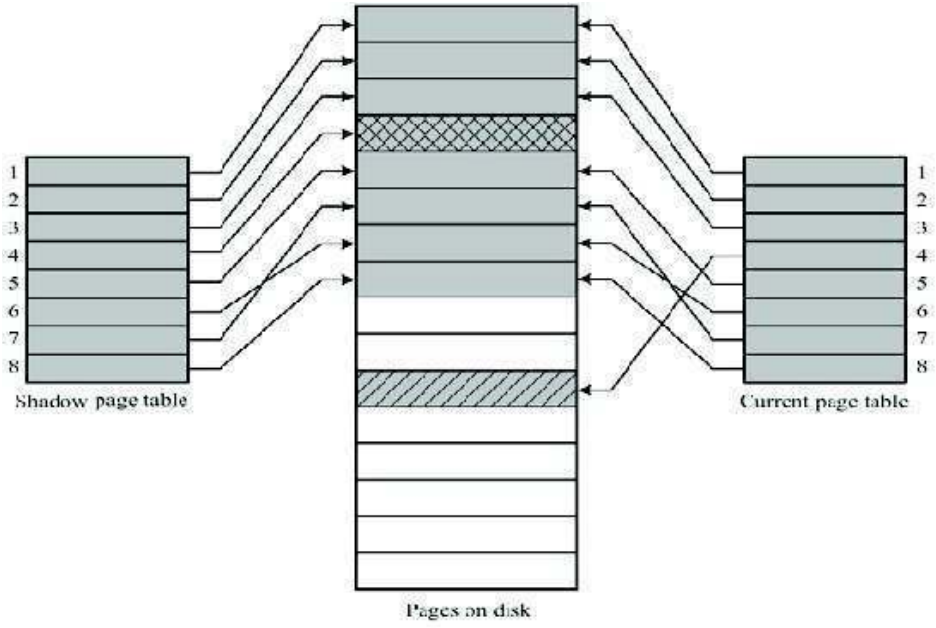



**Figure 9: Shadow page table**

To commit a transaction:

1) Flush all modified pages in main memory to disk
2) Output current page table to disk
3) Make the current page table the new shadow page table, as follows:

- keep a pointer to the shadow page table at a fixed (known) location on disk.
- to make the current page table the new shadow page table, simply update the pointer to point at the current page table on disk.

Once pointer to shadow page table has been written, transaction is committed. No recovery is needed after a crash — new transactions can start right away, using the shadow page table. Pages not pointed to from current/shadow page table should be freed (garbage collected).
Advantages of shadow-paging over log-based schemes:

- It has no overhead of writing log records,
- The recovery is trivial.

**Disadvantages**:

- Copying the entire page table is very expensive, it can be reduced by using a page table structured like a $B^+$-tree (no need to copy entire tree, only need to copy paths in the tree that lead to updated leaf nodes).
- Commit overhead is high even with the above extension (Need to flush every updated page, and page table).
- Data gets fragmented (related pages get separated on disk).
- After every transaction is completed, the database pages containing old versions is completed, of modified data need to be garbage collected/freed.
- Hard to extend algorithm to allow transactions to run concurrently (easier to extend log based schemes).

### 2.9.3 Recovery with Concurrent Transactions

We can modify log-based recovery schemes to allow multiple transactions to execute concurrently. All transactions share a single disk buffer and a single log. A buffer block can have data items updated by one or more transactions. We assume concurrency control using strict two-phase locking; logging is done as described earlier. The checkpointing technique and actions taken on recovery have to be changed since several transactions may be active when a checkpoint is performed.

Checkpoints are performed as before, except that the checkpoint log record is now of the form

< **checkpoint** *L*>

where *L* is the list of transactions active at the time of the checkpoint. We assume no updates are in progress while the checkpoint is carried out. When the system recovers from a crash, it first does the following:

Initialises *undo-list* and *redo-list* to empty
Scans the log backwards from the end, stopping when the first <**checkpoint** *L*> record is found.

For each log record found during the backward scan:

- If the record contains <$T_i$ commit>, add $T_i$ to *redo-list.*
- If the record contains <$T_i$ start>, then if $T_i$ is not in *redo-list*, add $T_i$ to *undo-list*
- For every $T_i$ in *L*, if $T_i$ is not in *redo-list*, add $T_i$ to *undo-list.*

At this point *undo-list* consists of incomplete transactions, which must be undone, and *redo-list* consists of finished transactions that must be redone.

*Recovery now continues as follows:*

Scan log backwards from most recent record, stopping when <$T_i$ **start**> records have been encountered for every $T_i$ in *undo-list*. During the scan, perform **undo**



for each log record that belongs to a transaction in *undo-list*. Locate the most recent <**checkpoint** *L*> record. Scan log forwards from the <**checkpoint** *L*> record till the end of the log. During the scan, perform **redo** for each log record that belongs to a transaction on *redo-list.*

SQL does not have very specific commands for recovery but, it allows explicit COMMIT, ROLLBACK and other related commands.

# 2.10 BUFFER MANAGEMENT

When the database is updated, a lot of records are changed in the buffers allocated to the log records, and database records. Although buffer management is the job of the operating system, however, some times the DBMS prefer buffer management policies of their own. Let us discuss in details buffer management in the subsequent sub-sections.

## 2.10.1 Log Record Buffering

Log records are buffered in the main memory, instead of being output directly to a stable storage media. Log records are output to a stable storage when a block of log records in the buffer is full, or a **log force** operation is executed. Log force is performed to commit a transaction by forcing all its log records (including the commit record) to stable storage. Several log records can thus be output using a single output operation, reducing the I/O cost.

The rules below must be followed if log records are buffered:

- Log records are output to stable storage in the order in which they are created.
- Transaction $T_i$ enters the commit state only when the log record <$T_i$ **commit**> has been output to stable storage.
- Before a block of data in the main memory is output to the database, all log records pertaining to data in that block must be output to a stable storage.

These rules are also called the **write-ahead logging** scheme.

## 2.10.2 Database Buffering

The database maintains an in-memory buffer of data blocks, when a new block is needed, if the buffer is full, an existing block needs to be removed from the buffer. If the block chosen for removal has been updated, even then it must be output to the disk. However, as per write-ahead logging scheme, a block with uncommitted updates is output to disk, log records with undo information for the updates must be output to the log on a stable storage. No updates should be in progress on a block when it is output to disk. This can be ensured as follows:

- Before writing a data item, the transaction acquires exclusive lock on block containing the data item.
- Lock can be released once the write is completed. (Such locks held for short duration are called **latches**).
- Before a block is output to disk, the system acquires an exclusive latch on the block (ensures no update can be in progress on the block).

A database buffer can be implemented either, in an area of real main-memory reserved for the database, or in the virtual memory. Implementing buffer in reserved main-memory has drawbacks. Memory is partitioned before-hand between database buffer and applications, thereby, limiting flexibility. Although the operating system knows how memory should be divided at any time, it cannot change the partitioning of memory.

Database buffers are generally implemented in virtual memory in spite of drawbacks. When an operating system needs to evict a page that has been modified, to make space for another page, the page is written to swap space on disk. When the database



... [15]

decides to write buffer page to disk, buffer page may be in swap space, and may have to be read from swap space on disk and output to the database on disk, resulting in extra I/O, Known as **dual paging** problem. Ideally when swapping out a database buffer page, the operating system should handover the control to the database, which in turn outputs page to database instead of to swap space (making sure to output log records first) dual paging can thus be avoided, but common operating systems do not support such functionality.





## 2.11 ADVANCED RECOVERY TECHNIQUES

Advanced recovery techniques support high-concurrency locking techniques, such as those used for $B^+$-tree concurrency control. Operations like $B^+$-tree insertions and deletions release locks early. They cannot be undone by restoring old values (**physical undo**), since once a lock is released, other transactions may have updated the $B^+$-tree. Instead, insertions/deletions are undone by executing a deletion/insertion operation (known as **logical undo**).

For such operations, undo log records should contain the undo operation to be executed called **logical undo logging**, in contrast to **physical undo logging**. Redo information is logged **physically** (that is, new value for each write). Even for such operations logical redo is very complicated since the databases state on disk may not be "operation consistent".

Operation logging is done as follows:

1) When operation starts, log <$T_i$, $O_j$, **operation-begin**>. Here $O_j$ is a unique identifier of the operation instance.
2) While operation is being executed, normal log records with old and new value information are logged.
3) When operation is completed, <$T_i$, $O_j$, **operation-end**, $U$> is logged, where $U$ contains information needed to perform a logical undo information.

If crash/rollback occurs before the operation is completed, the **operation-end** log record is not found, and the physical undo information is used to undo operation.

If crash/rollback occurs after the operation is completed, the **operation-end** log record is found, and in this case logical undo is performed using $U$; the physical undo information for the operation is ignored. Redo of operation (after crash) still uses physical redo information.

Rollback of transaction $T_i$ is done as follows:

Scan the log backwards.

1) If a log record <$T_i$, $X$, $V_1$, $V_2$> is found, perform the undo and log a special **redo-only log record** <$T_i$, $X$, $V_2$>.
2) If a <$T_i$, $O_j$, **operation-end**, $U$> record is found. Rollback the operation logically using the undo information $U$. Updates performed during roll back are logged just like during the execution of a normal operation.
   At the end of the operation rollback, instead of logging an **operation-end** record, generate a record. <$T_i$, $O_j$, **operation-abort**>. Skip all preceding log records for $T_i$ until the record <$T_i$, $O_j$ **operation-begin**> is found.
3) If a redo-only record is found ignore it.
4) If a <$T_i$, $O_j$, **operation-abort**> record is found:

   - **skip all preceding log records for $T_i$ until the record <$T_i$, $O_j$,** operation-begi**n> is found.**

5) Stop the scan when the record <$T_i$, start> is found
6) Add a <$T_i$, **abort**> record to the log.

**Note:**

- Cases 3 and 4 above can occur only if the database crashes while a transaction is being rolled back.
- Skipping of log records as in case 4 is important to prevent multiple rollback of the same operation.

The following actions are taken when recovering data after the system crash:

Scan log forward from last < **checkpoint** *L*> record

**Repeat history** by physically redoing all updates of all transactions,
Create an undo-list during the scan as follows:

- ***undo-list* is set to *L* initially**
- **Whenever <$T_i$** start**> is found $T_i$ is added to *undo-list***
- **Whenever <$T_i$** commit**> or <$T_i$** abort**> is found, $T_i$ is deleted from *undo-list.***

This brings the database to the state of crash, with committed as well as uncommitted transactions having been redone.

Now *undo-list* contains transactions that are **incomplete**, that is, those neither committed nor been fully rolled back.

Scan log backwards, performing undo on log records of transactions found in *undo-list*.

- Transactions are rolled back as described earlier.
- When <$T_i$ **start**> is found for a transaction $T_i$ in *undo-list*, write a <$T_i$ **abort**> log record.
- Stop scan when <$T_i$ **start**> records have been found for all $T_i$ in *undo-list*.

This undoes the effect of incomplete transactions (those with neither **commit** nor **abort** log records). Recovery is now complete.

**Check pointing** is done as follows:

- Output all log records in memory to stable storage,
- Output to disk all modified buffer blocks,
- Output to log on stable storage a < **checkpoint** *L*> record,
- Transactions are not allowed to perform any action while checkpointing is in progress.

Another checkpoint mechanism called **Fuzzy Check pointing** allows transactions to progress once the checkpoint record is written back to a stable storage, but modified buffer blocks are not yet completely written to the stable storage. **Fuzzy check pointing** is done as follows:

1) Temporarily stop all updates by transactions,
2) Write a <**checkpoint** *L*> log record and force log to stable storage,
3) Note list *M* of modified buffer blocks,
4) Now permit transactions to proceed with their actions,
5) Output to disk all modified buffer blocks in list *M,*
   (a) Blocks should not be updated while being output,
   (b) Follow Write ahead log scheme: all log records pertaining to a block must be output before the block is output,
6) Store a pointer to the **checkpoint** record in a fixed position **lastcheckpoint** on disk.

When recovering the database using a fuzzy checkpoint, start scan from the **checkpoint** record pointed to by **last_checkpoint.**

- Log records before **last_checkpoint** have their updates reflected in database on disk, and need not be redone.
- Incomplete checkpoints, where system had crashed while performing checkpoint, are handled safely.

Formatted: Centered



## 2.12 REMOTE BACKUP SYSTEMS

Remote backup systems provide high availability by allowing transaction processing to continue even if the primary site is destroyed.

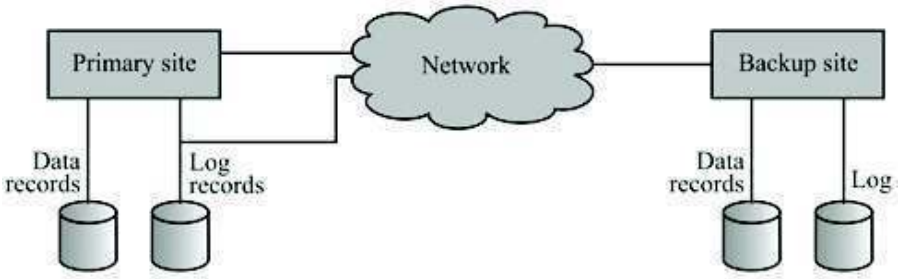



**Figure 10: Remote backup system**

**Detection of Failure**: Backup site must be able to detect when the primary site has failed. To distinguish primary site failure from link failure, we would need to maintain several communication links between the primary and remote backup.

**Transfer of Control:** To take over control, backup site first performs recovery using its copy of the database and all the log records it has received from the primary site. Thus, completed transactions are redone and incomplete transactions are rolled back. When the backup site takes over processing, it becomes the new primary site in order to transfer control back to the old primary site. When it recovers the database, the old primary site receive redo logs from the old backup and apply all updates locally.

**Time to recover**: To reduce delay in takeover, backup site periodically proceses the redo log records (in effect, performing recovery from previous database state), performs a checkpoint, and can then delete earlier parts of the log.

**Hot-Spare** configuration permits very fast takeover, backup continually processes redo log record as they arrive, applying the updates locally. When failure of the primary site is detected, the backup rolls back incomplete transactions, and is ready to process new transactions.

Alternative to remote backup are distributed databases with replicated data. Remote backup is faster and cheaper, but has a lower tolerance level for detecting failure.

The hot-spare configuration ensures durability of updates by delaying transaction committed until update is logged at backup. You can avoid this delay by permitting lower degrees of durability.

**One-safe:** A transaction commits in this scheme as soon as transaction's commit log record is written at the primary site. The problem in such a case is that, updates may not arrive at backup site before it takes over.

**Two-very-safe:** A transaction commits when transaction's commit log record is written at both sites primary and backup. Although this reduces the availability of the data base since transactions cannot commit if either site fails.

**Two-safe:** This commit protocol proceeds if both the primary and backup site are active. However, if only the primary is active, the transaction commits as soon as its commit log record is written at the primary site. This protocol provides better availability than two-very-safe, as it avoids problems of lost transactions in one-safe.

## 2.13 E-TRANSACTIONS

With the Internet boom more and more applications are supporting Internet based transactions. Such transactions may require a sequence of or a combination of the following activities:

- Dissemination of information using product catalogues such as information about trains in a Railway reservation system,
- Price negotiation-sometimes possible in airline reservations, but not in Railway reservation system,
- Actual order for example, booking of a train reservation,
- Payment for the order for example, payment for the booking,
- Electronic delivery if product is electronically deliverable, or else information about product tracking, for example, electronic printing of ticket.
- Communication for after sale services. For example, any other information about trains.

Such e-transactions are slightly lengthier and require a secure transaction processing method. They should be very reliable as there are many possible types of failures. More detailed discussion on these topics is available in the further readings section.

☞ **Check Your Progress 3**

1) Define log based recovery process.

……

……

2) How are remote back up recovery process helpful in transaction processing methods?

……

……

3) Elaborate the dual paging problem.

……

……

4) Write algorithms for various protocols discussed in this unit.

……

……

……

## 2.14 SUMMARY

In this unit we have discussed transaction management and various recovery management methods. We have detailed the features of transactions and log based as well as timestamp based protocols for transaction management. Two phase locking and its variants are very useful mechanisms for implementing concurrent transactions without having any concurrency related problem. But locking often leads to deadlocks. These deadlocks can be avoided if we use timestamp-based protocols. Locking can be done at multiple granularity and the index can also be locked to avoid inconsistent analysis problems. Multi-version scheme although requiring very little rollback has a high overhead. SQL supports weaker level of consistency, which requires programmer support for avoidance of concurrency related problems.

The recovery mechanism is needed in database system to take care of failures. One can use either the log based or page based recovery schemes. Buffer management is an important issue for DBMS as it affects the process of recovery. Remote backup systems allow a copy of complete data.

Formatted: Centered





## 2.15 SOLUTION/ANSWERS

### Check Your Progress 1

1) The transaction processing methods can be:

   Batch – combining jobs in batches
   Online – The effects are known immediately
   Advanced methods may include real time transaction, weak consistency levels etc.

2) Lock based protocols are one of the major techniques to implement concurrent transaction execution. These protocols try to ensure serialisability. The pitfalls of such protocols can be:

   - Overheads due to locking
   - Locking may lead to deadlocks
   - We may either restrict concurrency – strict 2 PL or allow uncommitted data to be seen by other transactions.

3) A timestamp is a label that is allotted to a transaction when it starts. This label is normally generated as per some time unit so that transactions may be identified as per time order. They allow concurrency control but without letting transaction wait for a lock. Thus, problems such as deadlocks may be avoided.

4) The main problems of Insert and Delete are that it may result in inconsistent analysis when concurrent transactions are going on due to phantom records. One such solution to this problem may be locking the index while performing such an analysis.

5) Multiple granularity defines object at various levels like database, file, record, fields, etc.

6) Test them yourself with some data.

### Check Your Progress 2

1) Multi-version technique allows the creation of multi-version of each data items as it gets updated. It also records the timestamp of updating transaction for that data. This protocol tries to make available correct data as per transaction timestamp.

2) Deadlock is a situation when two or more transactions are waiting for each other to release locks held by others. A deadlock can be detected when a cycle is detected in the wait for graph. The basic procedure of deadlock avoidance may be to roll back certain transaction as per some rule such as wound-wait or allow the transaction a maximum time to finish after which it may be rolled back.

3) Weak level of consistency in SQL allows transaction to be executed at a lower level. In such cases the programmer need to ensure that there is no concurrency related problem. SQL provides the following weak consistency levels. Read uncommitted, Read committed, and Repeatable Read.

4) It can only be implemented with more secondary storage, which are placed at different locations to take care of natural disasters or catastrophic events.

## Check Your Progress 3

1) All recovery process requires redundancy. Log based recovery process records a consistent state of database and all the transaction logs after that on a stable storage. In case of any failure the stable log and the database states are used to create the consistent database state.

2) They increase the availability of the system. Also provides a situation through which we may avoid natural disaster or catastrophic failures.

3) When a page buffer under an operating system control is written back to disk swap space by operating system and is required by DBMS, this require two block transfer, hence the name dual paging problem.

4) The algorithms are given in the unit just test them.

| | | |
|---|---|---|
| **Page 1: [1] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

| | | |
|---|---|---|
| **Page 1: [2] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

| | | |
|---|---|---|
| **Page 1: [3] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

| | | |
|---|---|---|
| **Page 1: [4] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

| | | |
|---|---|---|
| **Page 1: [5] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

| | | |
|---|---|---|
| **Page 1: [6] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

| | | |
|---|---|---|
| **Page 1: [7] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

| | | |
|---|---|---|
| **Page 1: [8] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

| | | |
|---|---|---|
| **Page 1: [9] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

| | | |
|---|---|---|
| **Page 1: [10] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

| | | |
|---|---|---|
| **Page 1: [11] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

| | | |
|---|---|---|
| **Page 48: [12] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

French France

| | | |
|---|---|---|
| **Page 48: [13] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

English U.S.

| | | |
|---|---|---|
| **Page 1: [14] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

| | | |
|---|---|---|
| **Page 1: [15] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

| | | |
|---|---|---|
| **Page 1: [16] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

| | | |
|---|---|---|
| **Page 1: [17] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

| | | |
|---|---|---|
| **Page 1: [18] Formatted** | **RAJU** | **2/23/2007 9:50 AM** |

Centered

# UNIT 3 DATABASE SECURITY AND AUTHORISATION

**Structure** **Page Nos.**


| | | |
|---|---|---|
| 3.0 | Introduction | 59 |
| 3.1 | Objectives | 60 |
| 3.2 | Database Security: The Scenario | 60 |
| 3.3 | Levels of Database Security | 60 |
| | 3.3.1 Server Security | |
| | 3.3.2 Database Connections | |
| | 3.3.3 Table Access Control | |
| | 3.3.4 Restricting Database Access | |
| 3.4 | Access Control | 62 |
| | 3.4.1 Granting Permissions | |
| | 3.4.2 Removing Permissions | |
| 3.5 | Statistical Database Security | 66 |
| | 3.5.1 Types of Disclosures | |
| | 3.5.2 Security vs. Decisions | |
| | 3.5.3 Security and Statistical Queries | |
| 3.6 | Multilevel Security | 68 |
| 3.7 | Audit Trails in the Databases | 69 |
| 3.8 | Examples of Vendor-Specific E-Security | 70 |
| 3.9 | Summary | 71 |
| 3.10 | Solutions/Answers | 72 |
| 3.11 | Further Reading | 72 |


## 3.0 INTRODUCTION

As the use of Internet technology is growing for both the Intranet and the Internet, information security is becoming exceedingly crucial for organisations. The World Wide Web provides a convenient, cheap, easily accessible and instantaneous way of information dissemination. It makes the dissemination of very easy but, it is equally important to ensure that information should only be accessible information to rightful users who have access rights to it.

With many organisations using database based dynamic web pages, corporate information security has become extremely important. Earlier, strict database access or specialised client software was required for viewing the data, but today a simple web browser is sufficient to view data in a database that is not properly protected. Thus, information security is at a vulnerable stage. Hence, the more a computing firm moves from the mainframe of the client/server to the Internet the more possibilities of security penetration.

Security database specialists have to rely on network administrators for implementing firewalls or other mechanisms to protect local data since the nature of Intranet/ Internet information access is such; however, the database administrator (DBA) has to perform many security function. This unit will examine the primary security areas that fall within the domain of the DBA, who then has to create database orient solutions.



## 3.1 OBJECTIVES

After going through this unit, you should be able to:

- define the various levels of database security;
- define different access control mechanism;
- explain the importance of security in statistical databases;
- define multi-level security, and
- identify the use of audit trails in database security.

## 3.2 DATABASE SECURITY : THE SCENARIO

You must realise that security is a journey, and not the final destination. We cannot assume a product/technique is absolutely secure, we may not be aware of fresh/new attacks on that product/technique. Many security vulnerabilities are not even published as attackers want to delay a fix, and manufacturers do not want the negative publicity. There is an ongoing and unresolved discussion over whether highlighting security vulnerabilities in the public domain encourages or helps prevention of further attacks.

The most secure database you can think of must be found in a most securely locked bank, or nuclear-proof bunker, installed on a standalone computer without an Internet or network connections, and under guard for 24×7×365. However, that is not a likely scenario with which we would like to work. A database server is to keep up with services, which often contain security problems, and you should be realistic about possible threats. You must assume failure at some point, and never store truly sensitive data in a database that unauthorised users may easily infiltrate/access.

A major point to consider here is that most data loss occurs because of social exploits and not technical ones. Thus, personnel procedures may more than encryption algorithms need to be looked into.

You will be able to develop an effective database security, if you realise that securing data is essential to the market reputation, profitability and business objectives. For example, as personal information such as credit card or bank account numbers are now commonly available in many databases, therefore, there are more opportunities for identity theft. As per an estimate, more than half of all identity theft cases are committed by employees who have access to large financial databases. Banks, companies that take credit cards services externally must place greater emphasis on safeguarding and controlling access to this proprietary database information.

## 3.3 LEVELS OF DATABASE SECURITY

Securing the database is a fundamental tenet for any security personnel while developing his or her security plan. The database is a collection of useful data and can be treated as the most essential component of an organisation and its economic growth. Therefore, any security effort must keep in mind that they need to provide the strongest level of control for the database.

As is true for any other technology, the security of database management systems depends on many other systems. These primarily include the operating system, the applications that use the DBMS, services that interact with the DBMS, the web server that makes the application available to end users, etc. However, please note that most importantly, DBMS security depends on us, the-users.

**Common Database Security Failures**



Database security is of paramount importance for an organisation, but many organisations do not take this fact into consideration, till an eventual problem occurs. The common pitfalls that threaten database security are:

**Weak User Account Settings:** Many of the database user accounts do not contain the user settings that may be found in operating system environments. For example, the user accounts name and passwords, which are commonly known, are not disabled or modified to prevent access.

The user account settings allow limited capabilities for security, without password controls on dictionary checks or account controls supporting expiration of user account.

**Insufficient Segregation of Duties:** No established security administrator role is defined in the database management of the organisation. This results in database administrators (DBAs) performing both the functions of the administrator (for users accounts), as well as the performance and operations expert. This may result in management inefficiencies.

**Inadequate Audit Trails:** The auditing capabilities of databases since it require keeping track of additional requirements, are often ignored for enhanced performance or disk space. Inadequate auditing results in reduced accountability. It also reduces the effectiveness of data history analysis. The audit trails records information about the actions taken on certain critical of data. They log events directly associated with the data, thus, they are essential for monitoring the access and the activities on a database system.

**Unused DBMS Security Features:** The security of an individual application is usually independent of the security of the DBMS. Please note that security measures that are built into an application apply to users of the client software only. The DBMS itself and many other tools or utilities that can connect to the database directly through ODBC or any other protocol, may bypass this application level security completely. Thus, you must try to use security restrictions that are reliable, for instance, try using security mechanism that are defined within the database.

Basically database security can be broken down into the following levels:

- Server Security
- Database Connections
- Table Access Control
- Restricting Database Access.

### 3.3.1 Server Security

Server security is the process of controlling access to the database server. This is the most important aspect of security and should be carefully planned.

The basic idea here is "You cannot access what you do not see".

For security purposes, you should never let your database server be visible to the world. If a database server is supplying information to a web server then it should be configured in such a manner that it is allowed connections from that web server only. Such a connection would require a trusted IP address.

**Trusted IP Addresses**

To connect to a server through a client machine, you would need to configure the server to allow access to only trusted IP addresses. You should know exactly who should be allowed to access your database server. For example, if it is the back end of a web server, then only that web server address should be allowed access to the database server. If the database server is the back end of a local application that is running on the internal network, then it should only talk to addresses from within the internal network.

### 3.3.2 Database Connections



With the ever-increasing number of Dynamic Applications, an application may allow immediate unauthenticated updates to some database. If you are going to allow users make updates to some database via a web page, please ensure that you validate all such updates. This will ensure that all updates are desirable and safe. For example, you may remove any possible SQL code from a user-supplied input. If a normal user does not input SQL code then we need not allow such data to be submitted.

### 3.3.3 Table Access Control

Table access control is probably one of the most overlooked but one of the very strong forms of database security because of the difficulty in applying it. Using a table access control properly would require the collaboration of both the system administrator as well as the database developer. In practise, however such "collaboration" is relatively difficult to find.

### 3.3.4 Restricting Database Access

By now we have defined some of the basic issues of database security, let us now look into the specifics of server security, from the point of view of network access of the system. Internet based databases have been the most recent targets of security attacks. All web-enabled applications listen to a number of ports. Cyber criminals often perform a simple "port scan" to look for ports that are open from the popular default ports used by database systems. How can we address this problem? We can address this problem **"by default",** that is, we can change the default ports a database service would listen into. Thus, this is a very simple way to protect the DBMS from such criminals.

There are many ways of preventing open access from the Internet. Each DBMS and OS has its own set of unique features for the same. Let us discuss some general methods of preventing open access from the Internet.

- **Trusted IP addresses :** UNIX servers are configured to answer ping requests only from a list of trusted hosts. In UNIX, this can be accomplished by configuring the *rhosts* file. Thus, it restricts server access to a list of specific users only.

- **Server Account Disabling :** It may be a good idea to suspend the server ID after three failed password attempts. This may thwart attackers. If such a scheme is not implemented, then an attacker can run a brute force program that generates millions of passwords. Such a program ultimately would break the combination of the user ID and password.

- **Special Tools :** Some customised tools, for example, *Real Secure,* send an alert when an external server is attempting to breach your system security. There are many such similar products available for the protecting of the DBMS from unauthorised Internet access.

## 3.4 ACCESS CONTROL

All relational database management systems provide some sort of intrinsic security mechanisms that are designed to minimise security threats as stated in the previous section. These mechanism range from the simple password protection offered in Microsoft Access to the complex user/role structure supported by advanced relational databases like Oracle and Microsoft SQL Server. But can we define access control for all these DBMS using a single mechanism? SQL provides that interface for access control. Let us discuss the security mechanisms common to all databases using the Structured Query Language (SQL).

An excellent practice is to create individual user accounts for each database user. Although, sharing of user accounts among various users is possible or even one user

account can be created for each type of user, however, such a practice should be discouraged. Why? It could be because of the following reasons:

**It will eliminate individual accountability**: *If any one of the users make a change in the database, we will not be able to trace it back to a specific individual even after going through audit logs. Imagine what would happen when a specific user leaves the organisation and his or her access from the database is to be removed? It will require change in the password and this will cause inconvenience to other users.*

Thus, it is important that we provide separate user accounts for separate users.
Does this mechanism have any drawbacks? If the expected number of database users are small then it is all right to give them individual user name and passwords and all the database access privileges that they need to have on the database items. However, consider a situation when there are a large number of users. Specification of access rights to all these users individually will take a long time. That is still manageable as it may be a one time effort, however, the problem will be compounded if we need to change the access right for a particular type of users. Such an activity would require a huge maintenance cost. This cost can be minimised if we use a specific concept called "Roles". A database may have hundreds of users but their access rights may be categorised in specific roles for example, teachers, student in a university database. Such roles would require specification of access rights only once for the **role.** The users then can be assigned username, password and specific role. Thus, the maintenance of user accounts becomes easier as now we have limited roles to be maintained.

Let us explain SQL related security commands in more details.

### 3.4.1 Granting Permissions

You would need to create the users or roles before you grant them permissions. Then permissions can be granted to a user or a role. This can done with the use of the SQL GRANT statement.

The syntax of this statement is:

**GRANT <permissions>**
**[ON <table>]**
**TO <user/role>**
**[WITH GRANT OPTION]**

Now, let us define this statement line-by-line. The first line, GRANT <permissions>, allows you to specify the specific permissions on a table. These can be either relation-level data manipulation permissions (such as SELECT, INSERT, UPDATE and DELETE) or data definition permissions (such as CREATE TABLE, ALTER DATABASE and GRANT). More than one permission can be granted in a single GRANT statement, but data manipulation permissions and data definition permissions may not be combined in a single statement.

The second line, ON <table>, is used to specify the table on which permissions are being given. This line is not needed if we are granting data definition permissions.

The third line specifies the user or role that are being granted permissions.

Finally, the fourth line, WITH GRANT OPTION, is optional. If this line is included *in the statement, the user* is also permitted to grant the same permissions that s/he has received to other users. Please note that the WITH GRANT OPTION cannot be specified when permissions are assigned to a *role.*

Let us look at a few examples of the use of this statement.



**Example 1:** Assume that you have recently hired a group of 25 data entry operators who will be adding and maintaining student records in a university database system. They need to be able to access information in the STUDENT table, modify this information and add new records to the table. However, they should not be able to entirely delete a record from the database.

**Solution:** First, you should create user accounts for each operator (please refer to MCS-023, Block 2, Unit-1) and then add them all to a new role-Dataentry. Next, we would need to use the following SQL statement to grant them the appropriate permissions:

```
GRANT SELECT, INSERT, UPDATE
ON STUDENT
TO Dataentry
```

And that is all that you need to do. Let us now examine a case where we are assigning data definition permissions.

**Example 2**: We want to allow members of the DBA role to add new tables to our database. Furthermore, we want them to be able to grant other users permission to do the same.

**Solution:** The SQL statement to do so is:

```
GRANT CREATE TABLE
TO DBA
WITH GRANT OPTION
```

Notice that we have included the WITH GRANT OPTION line to ensure that our DBAs can assign this permission to other users.

At this point, we have discussed how to assign permissions to users and roles as necessary. We will now look at the methods for removing permissions from users.

### 3.4.2 Removing Permissions

Once we have granted permissions, it may be necessary to revoke them at a later date. SQL provides us with the REVOKE command to remove granted permissions. The following is the syntax of this command:

```
REVOKE [GRANT OPTION FOR] <permissions>
ON <table>
FROM <user/role>
```

Please notice that the syntax of this command is almost similar to that of the GRANT command. Please also note that the WITH GRANT OPTION is specified on the REVOKE command line and not at the end of the command as was the case in GRANT. As an example, let us imagine we want to revoke a previously granted permission to the user Usha, such that she is not able to remove records from the STUDENT database. The following commands will solve the problem:

```
REVOKE DELETE
ON STUDENT
FROM Usha
```

There is one additional mechanism supported by some commercial DBMS that is worth discussing – the DENY command. This command can be used to *explicitly* deny permission to a user that s/he might otherwise have received because of his/her membership of a role. Here is the syntax:

```
DENY <permission>
ON <table>
TO <user/role>
```

Consider the last problem again, let us imagine that Usha was also a member of the Teachers role that also had access to the STUDENT table. The previous REVOKE

statement would not be sufficient to deny her access to the table. It will remove the permission granted to her through a GRANT statement, but would not affect the permissions gained through her membership in the Teachers role. However, if we use a DENY statement it will block permission for any role. Here is the command:

**DENY DELETE**
**ON STUDENT**
**TO Usha**

Thus DENY command creates a "NOT PERMITTED" statement in the database access controls. If we later want to give Usha permission to remove student records again from the STUDENT table, we cannot simply use the GRANT command. This is because of the fact that the GRANT command permission to DELETE record would be overridden by the existing DENY. Thus, first we use the REVOKE command to remove the DENY Not permission as:

**REVOKE DELETE**
**ON STUDENT**
**FROM Usha**

Please notice that this command is exactly the same as the REVOKE used to remove a granted permission. Thus, the DENY and GRANT commands both work in a similar fashion -- they both create permissions in the database access control mechanism. The REVOKE command removes all such permissions for the specified user. Once this command has been issued, Usha will be able to delete student records from the table if she is a member of a role that possesses that permission. You can also issues a GRANT command to provide the DELETE permission to Usha's account.

The access control mechanisms supported by the Standard Query Language is a good starting point, but you must look into the DBMS documentation to locate the enhanced security measures supported by your system. You will find that many DBMS support more advanced access control mechanisms, such as granting permissions on specific attributes.

### ☞ Check Your Progress 1

1) The most secure database is found in ……

2) On what systems does the security of a Database Management System depend?

……

……

3) ……. is the process of limiting Access to the Database Server.

……

……

4) What are the different ways of preventing open access to the Internet?

……

……

5) Write the syntax for granting permission to alter database.

……

……

……

6) Write the syntax for 'Revoke Statement' that revokes with grant option.



……

……

……

7) What does DENY command do?

……

……

……

## 3.5 STATISTICAL DATABASE SECURITY

By now we have discussed the basic security measures in both DBMS and SQL commands that provide necessary permissions to the users. However, in practice there are many database systems where information can be determined without having the access rights to do so. This is a breach of data confidentiality. In the subsequent subsection we discuss this problem and the way to resolve it.

### 3.5.1 Types of Disclosures

Data of an organisation is a very sensitive resource. Even the characteristics of data are quite sensitive. For example, existence of a piece of data such as "use of health related drugs" is sensitive information and is a form of disclosure. The various type of disclosure may be:

- **Exact Data:** It may be defined as the determination of the value of an item by using a sequence of complex queries.
- **Bounds:** It is defined as finding the value of data item between two values. The bounds in such cases may be lowered using binary search. This may lead to a very narrow range.
- **Negative Result:** Sometimes it may be possible to determine a negative result. This is also a form of disclosure.
- **Existence:** The existence of data value in itself is a sensitive piece of information, regardless of the actual value. For example, existence of a record regarding defence expenses may be a disclosure.

Please note that such disclosure of data can be obtained without making a direct query to the database but rather a set of queries. We will explain it with the help of an example in the next sub-section. Please remember "A good security scheme needs to protect data using access control while avoiding such indirect disclosures".

### 3.5.2 Security vs. Decisions

Disclosure of data as indicated in the previous section is a major problem as disclosure may result in breach of security in some form or the other, and thus, is not acceptable. Thus, the first step in this direction would be to reject any query that directly asks for sensitive information that is hidden. But, how about a sequence of queries that are raised for the purpose of statistics (management information)? For example, we may be able to determine the average marks obtained in a class of 50 student, but if only 2 students have opted for a subject then the first student who knows his/her marks can find the marks of the other student by issuing the average marks query. Thus, statistical queries should be permitted only when some minimum number of records satisfies a condition. Thus, the overall objectives are to make sure that security is not compromised.



Let us discuss some of the queries that may result in the disclosure of sensitive data. Consider the relation in the following *Table*:

| Enrolment No. | Name | Gender | Grade | Fee paid | Age | Hostel |
|---|---|---|---|---|---|---|
| 0601 | Anil | M | C | 10000 | 25 | Ganga |
| 0602 | Dev | M | B | 0 | 20 | Yamuna |
| 0603 | Sunil | M | A | 9000 | 22 | Kaveri |
| 0604 | Esha | F | B | 10000 | 27 | Ganga |
| 0605 | Nagma | F | C | 9000 | 20 | Ganga |
| 0606 | Fiza | F | C | 10000 | 20 | Kaveri |
| 0607 | Ganesh | C | 9000 | 24 | 3 | Kaveri |
| 0608 | Himani | C | 0 | 21 | 0 | Yamuna |

Assume that a student can not only view his/her details in the *Table*, but also the names of his/her colleagues, and that s/he is allowed to make queries as well. Let us see how s/he can attack the database.

**Simple Attacks:** A simple form of direct attack may be by using the following type of query:

```
SELECT name
FROM STUDENT
WHERE (Hostel = 'Yamuna' AND Gender = 'F').
```

The query above will show the result as 'Himani'. Thus, the information that Himani stays in Yamuna hostel has been disclosed. Please note this is a direct query and can be caught, but a person can hide this query in many ways, without changing the basic meaning of the query.

But how do we stop such disclosures? A very simple solution in this case may be: If a query that processes N records, however produces very few records (M) such that N>>M then this query may compromise security and should not be allowed to produce results.

**Inference Based Attacks:** Such an attack is very interesting case for determination of data through mathematical logic. For example, to determine the age of the female candidate, a series of commands can be used for determining the required information. These commands may be:

(i) Determine the sum of age of all the female candidates,

(ii) Determine the sum age of female candidates not residing in Hostel Yamuna.

Since all these queries will result in a sizable number of records they would be answered, however, they can be used to determine the age of 'Himani' who stays in Yamuna and is a sole female student staying there as:

Age of Himani= Result of Query (i) – Result of Query (ii).

Similarly many statistical functions like average, count, sum and their combinations can be used on various combinations of records. This sequence may be used to disclose the sensitive database values.

Thus, the database security has been compromised. In such situation the solution may be – not to allow consecutive queries whose output record set intersection is very small.

But how actually these problems be solved? Let us discuss the solution to such problems in the next subsection.

### 3.5.3 Security and Statistical Queries

The two possible ways of securing database against such attacks may be:

i) Controlling the queries themselves, (such solutions may help in solving the queries that can be identified as causing problems).

ii) The queries that result in limited value can be checked by either rejecting a query. If the result set size is too small, it is also called 'suppression' or if the information that is to be displayed as a result of the query is changed slightly so that it does not match the actual value it is also called concealing.

☞ **Check Your Progress 2**

1)

| Subject Maximum Marks = 100 | Neha | Richa | Neelam | |
|---|---|---|---|---|
| Maths | 50 | 70 | 90 | |
| Science | 60 | 80 | 50 | |
| Total | 110 | 150 | 140 | |

Write a sequence of queries that would disclose the name of the person who scored the highest marks.

2) Write two ways of protecting against inference attacks.

……

……

……

3) With …… sensitive data values are not provided; the query is rejected without response.

## 3.6 MULTILEVEL SECURITY

In certain applications data items can be classified under various levels of security. Some such security levels may be Secret, Classified, etc. The database system that supports such security features are known as Multilevel Sensitive Databases. Such systems may have the following three security features:

- Security of different objects may be different from the other attribute values of that tuple or security may be different from the other values of the attributes.
- Thus, every individual element is the micro item for security.
- In addition, security against statistical queries may also need to be provided.
- A number of security level needs to be defined for such security.

But how do we enforce the multilevel security?

There are many techniques to support multi-level security. Two important methods of these are:

**Partitioning**

In this approach the original database is divided into partitions. Each of the partitions has a different level of security.

However, the major drawback of this approach is that the database looses the advantage of a relation.

**Encryption**

Encryption of critical information may help in maintaining security as a user who accidentally receives them cannot interpret the data. Such a scheme may be used when a user password file is implemented in the database design. However, the simple encryption algorithms are not secure, also since data is available in the database it may also be obtained in response to a query.

There are many more schemes to handle such multi-level database security. These references are given in the further reading.

## 3.7 AUDIT TRAILS IN THE DATABASES

One of the key issues to consider while procuring a database security solution is making sure you have a secure audit-trail. An audit trail tracks and reports activities around confidential data. Many companies have not realised the potential amount of risk associated with sensitive information within databases unless they run an internal audit which details who has access to sensitive data and have assessed it. Consider the situation that a DBA who has complete control of database information may conduct a security breach, with respect to business details and financial information. This will cause tremendous loss to the company. In such a situation database audit helps in locating the source of the problem. The database audit process involves a review of log files to find and examine all reads and writes to database items during a specific time period, to ascertain mischief if any, banking database is one such database which contains very critical data and should have the security feature of auditing. An audit trail is a log that is used for the purpose of security auditing

Database auditing is one of the essential requirements for security especially, for companies in possession of critical data. Such companies should define their auditing strategy based on their knowledge of the application or database activity. Auditing need not be of the type “all or nothing”. One must do intelligent auditing to save time and reduce performance concerns. This also limits the volume of logs and also causes more critical security events to be highlighted.

More often then not, it is the insiders who makes database intrusions as they often have network authorisation, knowledge of database access codes and the idea about the value of data they want to exploit. Sometimes despite having all the access rights and policies in place, database files may be directly accessible (either on the server or from backup media) to such users. Most of the database applications, store information in ‘form text’ that is completely unprotected and viewable.

As huge amounts are at stake, incidents of security breaches will increase and continue to be widespread. For example, a large global investment bank conducted an audit of its proprietary banking data. It was revealed that more than ten DBAs had unrestricted access to their key sensitive databases and over hundred employees had administrative access to the operating systems. The security policy that was in place was that proprietary information in the database should be denied to employees who did not require access to such information to perform their duties. Further, the bank’s database internal audit also reported that the backup data (which is taken once every day) was also cause for concern as tapes could get stolen. Thus the risk to the database was high and real and that the bank needed to protect its data.

However, a word of caution, while considering ways to protect sensitive database information, please ensure that the privacy protection process should not prevent authorised personnel from obtaining the right data at the right time.

The credit card information is the single, most common financially traded information that is desired by database attackers. The positive news is that database misuse or unauthorised access can be prevented with currently available database security products and audit procedures.



# 3.8 EXAMPLES OF VENDOR-SPECIFIC E-SECURITY

Individual vendors largely determine the security schemes that may be implemented to provide the link between the database and its interfaces. Following are some of the security components provided by the major vendors. Like client/server systems, security solutions must be combined together using multiple vendor products.

**Oracle**

Oracle, provides SSL and S-HTTP security. Oracle uses Java as a basic component of its security model. The company created its Oracle Web Server to work most effectively with Oracle clients such as the solutions created with the Developer/2000 or other development tools.

Oracle modified the HTTP protocol to allow a straight/direct connection between the client and the server. This connection defines a session in which the user is identified by a generated ID.

These enhancements are also present in the Secure Network Server (SNS) that is included in the Oracle Universal Database a single login permits access to any Oracle database in an enterprise system.

The Java security classes are used by the oracle development tools to give complete security integration to the client.

**Sybase**

Sybase provides a rather elegant way of protecting data access through the Web. It extends the logon security present in the Web server to the database server for authentication. Thus, it takes advantage of the native security present in the database. Sybase provides a middleware called Web.sql that is used to interface with the Netscape Web servers.

**Informix**

Informix, like Sybase, relies on the logon security present in the Web server. Therefore, any database access is specified through traditional ODBC-type login channels that passes the user and password information through the connectivity middleware. Specific drivers known as Universal Web Connect integrates Informix database security with Microsoft Web servers.

**Microsoft**

Microsoft has included most of the key security technologies with Internet Information Server (IIS). For user authentication, Microsoft provides its tried-and-true challenge/response mechanism. Traditional login on the Web presents the same security as in the basic Windows NT login. Unfortunately, only Microsoft Internet Explorer browser supports this login approach.

For database access, IIS security has been integrated with Microsoft SQL Server through the Internet Database Connector. Although, users must login through an HTML login form, the information may be verified by a SQL Server stored procedure.

Microsoft has also integrated the Kerberos security architecture into Windows NT Server. By releasing the server, Microsoft hopes to integrate the Kerberos native to the NT Server with public key security. Microsoft has already released a Certificate Server API in an attempt to create a Certificate Server standard.

**Netscape**

Netscape Communications intends its suite of servers as a complete system for security on the Internet. Login occurs originally through the browser and then, as in the Novell Directory Services, all certification is unified in this model. Therefore,

once login to the browser occurs, any resources that are permitted to the user are now accessible.

Currently, user authentication occurs by passing information to the data source via the middleware. Most companies, including Sybase, Oracle, and Informix, provide the necessary connectors for this process.

### ☞ Check Your Progress 3

1) What is multilevel Sensitive Database?

   …….

   ……..

   …….

2) Name the different techniques for multilevel security.

   …….

   ……..

   …..…..……..……

3) ……., while providing SSL & S-HTTP security, its using java as a basic component of its security model.

4) …… & …… currently relies on the logon security present in the Web Server.

5) For user Authentication, Microsoft provides its ……. mechanism.

## 3.9 SUMMARY

This unit provides basic information about database security. It introduces various levels of database security. Some of the information that is covered in this unit include the commands to GRANT and REVOKE assess rights. In addition, a command DENY has also been discussed. Please note DENY is not available in all DBMSs. We have then discussed security in databases with statistical queries. We have also suggested certain mechanism to ensure data base security, even through statistical queries may be allowed-definitely with some restriction. We have also defined the concept of audit trail and given few DBMS security support. You must go through any commercial DBMS documents for more details on security implemented in them.

## 3.9 SOLUTIONS/ANSWERS

### Check Your Progress 1

1) Second graded Bunkers.
2) Operating System,
   Application that use the database,
   Service that interact, and
   The Web Server.
3) Server Security
4) Trusted IP address.
   Server Account Disabling
   Special Tools, Example: Real source by ISS.



5) GRANT ALTER DATABASE TO Usha

6) REVOKE GRANT OPTION FOR <permissions>
ON <table>
FROM <user/role>

7) It creates a 'NOT PERMITTED' condition in the database Access control.

### Check Your Progress 2

1) The highest marks can be found as:
Find the total marks
Find the max. marks
Find the names of student whose total is in the set of maximum marks.

2) a) Control to the Queries.
b) Control individual items that are being displayed.

3) Suppression.

### Check Your Progress 3

1) The database with more than one i.e., various levels of security.

2) i) Partitioning
ii) Encryption

3) Oracle

4) Sybase and Informix

5) tried-and-true challenge/response mechanism.

## 3.10 FURTHER READING

1) B. Sudhir Kumar Reddy, "*Aspects of Database and Program Security*" November 2003, available on the internet.

# UNIT 4 DISTRIBUTED DATABASES

**Structure** **Page Nos.**


4.0 Introduction 73
4.1 Objectives 73
4.2 Centralised Versus Non-Centralised Databases 74
4.3 DDBMS and its Functions 74
4.3.1 Heterogeneous Distributed Databases
4.3.2 Homogeneous Distributed Databases
4.3.3 Functions of a DDBMS
4.4 Reference Architecture of DDBMS 76
4.5 Distributed Database Design and Distributed Query Processing 79
4.6 Distributed Concurrency Control 82
4.6.1 Distributed Serialisability
4.6.2 Locking Protocols
4.6.3 Timestamp Protocols
4.6.4 Distributed Deadlock Management
4.7 Distributed Commit Protocols: 2 PC and 3 PC 88
4.7.1 Two-Phase Commit (2PC)
4.7.2 Three-Phase Commit (3PC)
4.8 Concepts of Replication Server 91
4.8.1 Replication Server Components
4.8.2 Benefits of a Replication Server
4.8.3 Functionality
4.8.4 Data Ownership
4.9 Summary 94
4.10 Solution/Answers 94


## 4.0 INTRODUCTION

Relational Database Management Systems are the backbone of most of the commercial database management systems. With the client-server architecture in place they can even be used in local and global networks. However, client-server systems are primarily centralised in nature−that is, the data is under the control of one single DBMS. What about applications that are distributed, where large number of transactions may be related to a local site, and where there are also a few global transactions? In such situations, it may be advisable to use a distributed database system. We have already introduced the concept of distributed databases in the MCS-023, Block 2, in this unit we enhance concepts that were introduced in that course.

In this unit, we will discuss the classification of distributed databases, its functions and architecture, distributed query design and the commit protocol. Towards the end of the unit we will also introduce the concept of replication servers.

## 4.1 OBJECTIVES

After going through this unit, you should be able to:

- define the classification of distributed databases;
- identify the functions and architecture of distributed database management systems;
- state the principle of design of distributed database management system;
- create distributed query processing;
- explain concurrency management in distributed database systems;
- describe distributed commit protocols, and
- explain the concept of replication servers.



## 4.2 CENTRALISED Vs. NON-CENTRALISED DATABASES

If the data of an organisation is stored at a single location under the strict control of the DBMS, then, it is termed as a centralised database system. A centralised system has the advantage of controlling data under a central authority, so, the chances of integrity loss, inaccuracy are very low. In such a system it is easier to enforce business rules and the system can also be made very secure.

However, consider the situation when the organisation is geographically dispersed and the data is created and used at many distributed offices of the same organisation. In such a case the centralised database may act, as a bottleneck as all accesses to data has to be from the central site. The reliability and availability of such a system will not only be dependent on the centralised system but also on the communication links. The load on the central system may also be very high if multiple transactions are going on. Compounding the problem would be the delay that may be caused because of source access through a communication channel.

On the other hand, if we keep the data in databases at local ends that could to an extent solve these problems, as most of the data access now can be local, however, it adds overheads on part of maintaining databases at several sites. The sites must coordinate if a global query is to be handled. Thus, it will increase the overall complexity of the system.

Decision to maintain a centralised or non-centralised data will depend entirely on the nature and structure of an organisation.

## 4.3 DDBMS AND ITS FUNCTIONS

This is the era of distribution where the terms prefixed with “distributed”, “distributed database” and “distributed processing” are attracting a lot of attention. But how are these two different? Let us first define the two terms.

**Distributed Database:** A distributed database defines the distribution of a database application system in many different computer sites supporting their own DBMS environment. However, these databases from the viewpoint of a user are a single application.

**Distributed Processing:** Distributed processing is the distribution of various tasks of an application processing system among different computers on a network.

But how are they related to each other? The relationship can be explained by the fact that a database application may distribute front-end tasks such as display, formatting, etc. to the client site, whereas, the database sever becomes the backend. Many clients may share a backend server. Thus, distributing the tasks of an application that require a database system. Thus, a client-server system is typically a distributed system.

So, a distributed database system may employ a distributed processing architecture.

A distributed database system is managed by a distributed database management system (DDBMS). A DDBMS can be classified as a homogeneous or heterogeneous system. Let us explain these in more detail in this section.

### 4.3.1 Heterogeneous Distributed Databases

In a *heterogeneous* distributed database system various sites that are participating in the distributed database may be using different DBMSs, or different data model, or may not even be a database system (non-system). Thus, the major challenge for the heterogeneous distributed database system would be to make these systems appear as a single application system for the user.

A heterogeneous database system may have to allow data access from various type of data and thus, have to know about such data formats, file formats and management related details for such data. A heterogenous database system may need to develop some standard forms that could be used for bringing the heterogeneous sources of data to a common interface. Therefore, heterogeneous DDBMS are very complex in nature. A detailed discussion on this topic is beyond the scope of this unit. However, let us try to list some of the important issues that need to be addressed by the Heterogeneous Distributed Database Systems:

- *Support for distributed transaction handling:* A heterogeneous DDMS must deal with transactions that are required to be executed on various DBMSs. The main concern here is to implement the commit protocol across the DBMS to ensure proper completion/termination of a transaction.
- *SQL access through DBMSs and Non-DBMSs:* Heterogeneous DDBMS must integrate data that may be available in non-database systems along with the data that is available in the DBMS environment. The key here is that any SQL query should generate the desired result even if, data is available in the non-database system. This action is to be supported transparently, such that the users need not know the details.
- *Common interface for Procedural Access:* A heterogeneous DDMS must support a common interface for any procedural access like messaging and queuing systems. One such implementation interface may be the use of Host language-SQL remote procedure calls.
- *Support for Data Dictionary translation:* One of the key support needed for DDBMS is to homogenise the data dictionaries of the systems as well as the non-database systems. One of the basic requirements here would be to translate a reference to a data dictionary or, a DBMS into a reference and then, to a non-system's data dictionary.
- Heterogeneous DDBMS should allow accessing data of non-systems directly without the use of SQL; on the other hand it should also allow access to stored procedures using remote procedure calls of the host languages.
- It should have support for the various different kinds of character sets that are used across non-systems and DBMS.
- It should support the use of distributed Multi-Threaded Agents. The Multi-threaded agents help in reducing the number of required server processes. It should also provide a graphic interface to access those agents.

### 4.3.2 Homogeneous Distributed Databases

The homogeneous distributed database systems contain identical DBMS across cooperating sites. These sites are aware of each other and need to compromise a little on their individual autonomy. In this unit, we have discussed details regarding these types of distributed database systems.

### 4.3.3 Functions of a DDBMS

A DDBMS primarily is built on a centralised relational database system, thus, it has the basic functionality of a centralised DBMS along with several other functionalities such as:

- It allows access to remote sites using standard SQL interfaces, thus allowing transfer of queries or components of queries as well as data of the database over a network.
- It has complete control over the system catalogue that stores the details of data distribution. Please note that this catalogue itself may however, be distributed.
- It performs distributed query processing, optimisation and evaluation. Please note that distributed query processing involves even access to remote data over the network.



- One of the major functions of the distributed database system is the control of distributed concurrent transactions to maintain consistency of the data that may be replicated accross various sites.
- Access to the databases under the replicated environment should be controlled. Thus, a distributed database management system should follow proper authorisation/access privileges to the distributed data.
- Distributed recovery mechanisms in addition to dealing with the failure of participating sites may also need to deal with the failure of communication links.

Before discussing some specific details of the DBMS, let us first look at the reference architecture of DDBMS.

## 4.4 REFERENCE ARCHITECTURE OF DDBMS

Distributed database systems are very diverse in nature; hence, it may be a good idea to define reference architecture for such database systems. For the DDBMS reference architecture may be looked at as, an extension of ANSI SPARC three level architecture that we have discussed in Unit 1 of MCS 023. The ANSI SPARC architecture has been defined for the centralised system and can be extended for the distributed database systems with various levels defined for data distribution as well.

Reference architecture may also define the following additional schema for distributed database systems:

- A set of global schemas that would define application interface. Thus, a global schema can be further divided into:
  - Global external schema, and
  - Global conceptual schema.
- Fragmentation and allocation schema will determine various fragments and their allocation to various distributed database sites.
- Finally, it would require basic schemas as proposed by ANSI/SPARC architecture for local databases. However, some modification would be. Let us, explain the various schemas of reference architecture in more detail.

**Global External Schema**

The basic objective of providing a global external schema is to define the external application interface for a distributed database application. This level is similar to the external schema as defined for the ANSI SPARC architecture. Thus, this interface must support the concept of data independence. However, in this case, it is logical data independence that is provided to a database application from the distributed data environment.

**Global Conceptual Schema**

A Global conceptual schema defines logical database application as if there is, no distribution of data. Conceptually, this schema is almost similar to, the conceptual level of ANSI/SPARC architecture. Thus, this schema defines the entities, relationships, and constraints including security and integrity constraints for global database application. This level is responsible for ensuring physical data independence between an application and the data distribution environment where it is to be implemented.

**Fragmentation and Allocation Schema**

Fragmentation and allocation schema is required to describe the data fragments that have been created for the distributed database, alongwith, their replicas that have been allocated to various database sites. This schema, thus, basically is for the identification of data distribution among the distributed sites.

### Local schemas

The local schema at each site can consists of conceptual and internal schemas. However, to map the fragmentation and allocation schema to local external objects may require local mapping schema. It is this schema (that is local mapping schema), that hides the details of various DBMSs, thus, providing the DBMS independence. This schema mapping, however, would be minimal for a homogeneous distributed database systems. *Figure 1* shows the reference architecture of DDBMS.

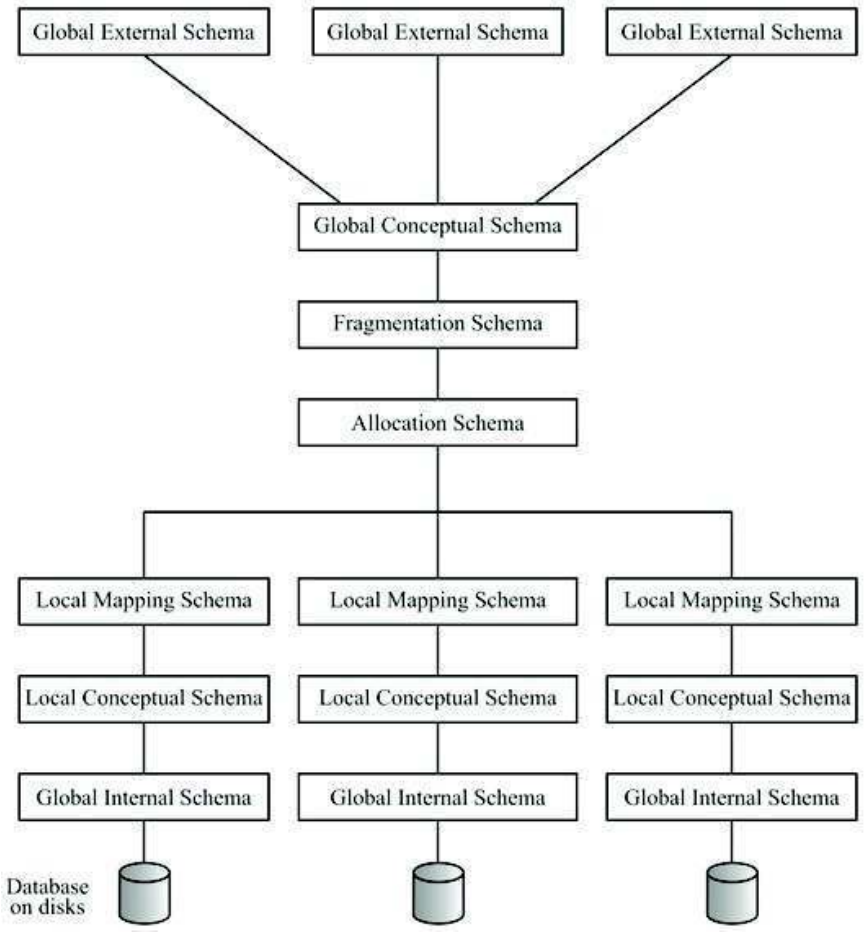



**Figure 1: Reference architecture for DDBMS**

### Component Architecture of DDBMS

From the viewpoint of commercial implementation a DDBMS would require client server implementation with the backbone of a network. Thus, such an architecture would support:

**Local DBMS Component (Clients and Servers):** At a local database site a DDBMS communicates with a local standard DBMS. This site therefore can have its own client server architecture. A computer on such a site (also referred to as a node) may be a client or server depending on its role at that movement of time. For example, in *Figure 2* (this figures is same as that of *Figure 2* unit 4, Block-2 of MCS-023) the computer that has the HQ database will act as a database server if, we request data for local sales, however, it will act as a client if we request data from the Employee table which is not available on this site.

**Data Communications Component:** Although we have the desired client and server components, the question that now arises is related to communication. How will the

client and the server component communicate with each other? This communication should be established with remote clients may not be connected to a server directly. It is the data communication component that enables direct or indirect communication among various sites. For example, in the *Figure 2* the transactions that are shown occurring on Site 1 will have direct connection to Site 1, but they will have an indirect communication to Site 2, to access the local sales data of that site. Thus, a network is necessary for a distributed database system.

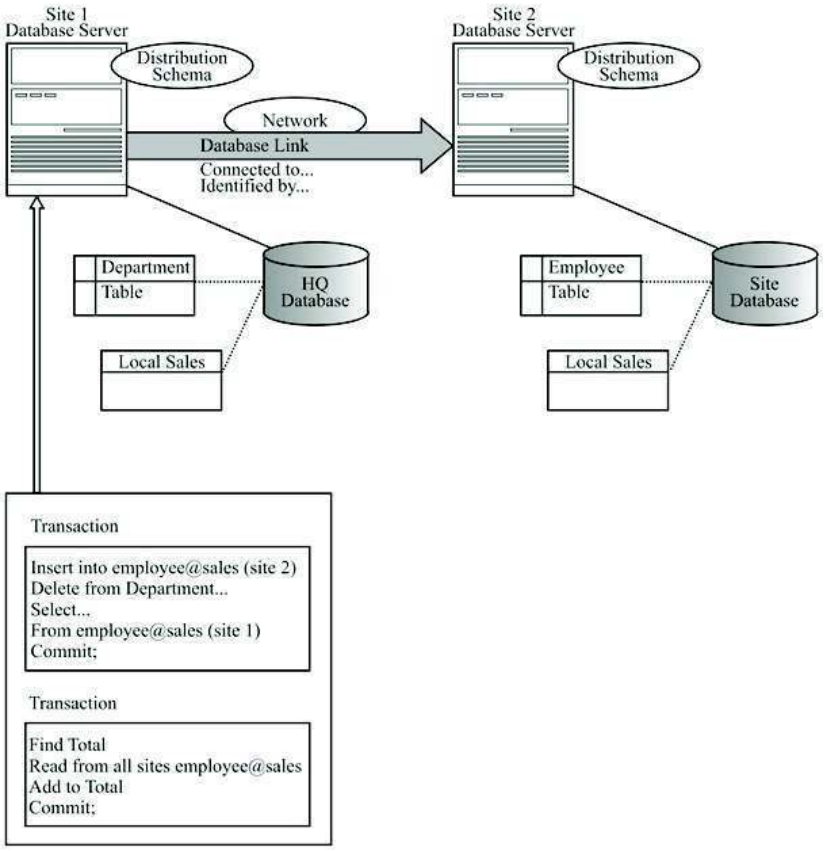



**Figure 2: Component architecture of DDBMS**

Thus, we have clients, servers and the network in the distributed databases. Now the question is how to use names in this network of a DDBMS sites? The names on a DDBMS may be provided using a global directory service. To allow easy access to the configuration of data, a networks name server can be used to provide information on the repositories of the databases and their data distribution.

**Global System Catalogue:** One of the ways of keeping information about data distribution is through the system catalogue of the DBMSs. In a DDBMS we need to keep a global system catalogue. This catalogue itself might be distributed in implementation. In addition to basic information about the databases it should keep information on data fragmentation and replication schema.

**Database Links**: A database in a distributed database is distinct from the other databases in the system. Each of these distributed databases may have their own global database name. All these database components would also have certain local names. Now, the questions are: How reachout from one database in a distributed database to another database and how does one resolve the names of various schema elements in the distributed databases that are available across the network. *Figure 3* represents a hierarchical arrangement of databases components that are available on a network. It shows the hierarchical arrangement of a hypothetical distributed database of some Universities.

As far as the first question is concerned, an application can travel from one database system to another by using database links, which is an unidirectional path from one database to another. Such links must be transparent from a user point of view.

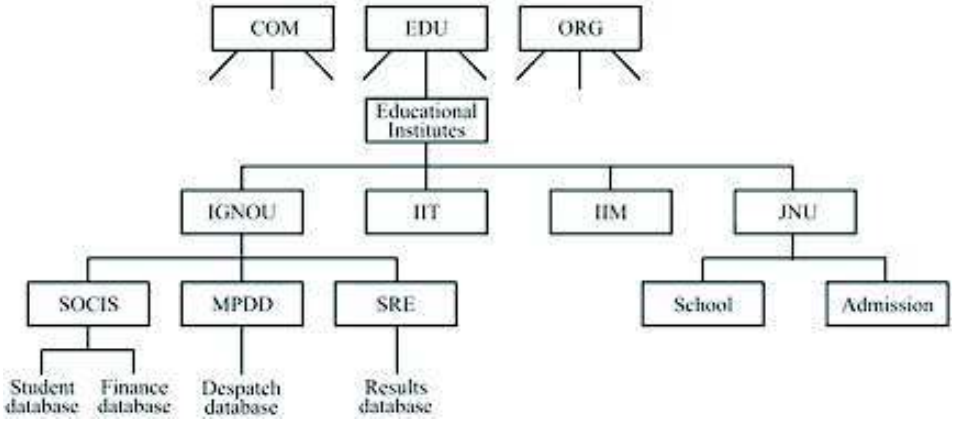



**Figure 3: Network directories and global database naming**

Now, as far as naming issues are concerned, the names of the database may be resolved by specifying the path. For example, in the *Figure 3*, the name for the result database in IGNOU – SRE may be RESULT may also be the same for the JNU, which database for storing the result of the student. However, these databases will be referred to as:

RESULT.SRE.IGNOU.EDU_INST.EDU

(Please note that EDU_INST represents "Educational Institutes" block in *Figure 3*)

and

RESULT.SCHOOL.JNU.EDU_INST.EDU

### ☞ Check Your Progress 1

1) What additional functions does a Distributed Database System have over a centralised database system?

……

……

……

……

2) How does the architecture of DDBMS differ from that of a centralised DBMS?

……

……

……

……

## 4.5 DISTRIBUTED DATABASE DESIGN AND DISTRIBUTED QUERY PROCESSING

Distributed database design has already been covered in MCS-023. Let us recapitulate some of the salient features of the distributed design.



The distributed design is primarily based on the nature of local and global transactions. Some of the key decisions taken for distributed database design, in addition to a centralised database design are:

- *data fragmentation* is concerned with the fragmentation of global data into component relations, and
- *data replication* determining the manner in which fragmented data is to be duplicated at several sites such that it supports ideal performance for data updating and query processing. It also ensures database availability and reliability.

Replication is a very important concept, so let us now talk about data replication in detail, next.

**Data Replication**

A distributed database without replication will have exactly one copy of a data item stored on some site. However, such an implementation would not offer advantages of a distributed database, (that are reliability and availability). A replicated distributed database will result in multiple copies of the data that will be stored on different sites. This replicated data will cause an increase in the overheads of data management, but will improve the availability of the system. For, even if one site with the data is down for some reason, the data will still be available from another site. In addition, data replication improves the performance of the distributed system. How? Well it is primarily because data is replicated with some logic, the chances are that a distributed application may find the remote data replicated on the local site, and thus, need not access a remote server. This in turn, may reduce network traffic and improve performance.

**Distributed Query Optimisation**

The concept of query optimisation in a distributed database system is the enhanced form of query optimisation that exists in the centralised databases. One additional but major objective of the distributed query optimisation is to minimise data transfer among the sites for distributed queries.

To fulfil this objective a distributed query optimiser would require some cost based optimisation that transforms SQL queries such that only the necessary data from a remote site is transferred over the network. It will also require the processing of the data at remote sites. The purpose of this processing would be that instead of a complete table only the data that is required for the query, will be transferred. Thus, reducing the amount of data transfer. Let us explain this with the help of an example:

Consider the following situation where two sites contain information as:

**Employee table at site 1**

| empno | emp-name | Phone | Job |
|---|---|---|---|
| 1001 | Sanjay | 921000001 | Salesperson |
| 1002 | Vijay | 922000001 | Salesperson |
| 1003 | Rahul | 931000001 | Office |
| 1004 | Ajay | ….. | Office |
| 1005 | Kamal | ….. | Clerk |
| … | ……. | …. | ……. |
| ….. | ……. | …. | |
| 10001 | Anurag | | Research |

**Sales table at site 2**

| empno | item-code | quantitysold |
|---|---|---|
| 1001 | A | 5000 |
| 1002 | B | 7000 |
| 1001 | B | 6000 |
| 1009 | A | 7000 |
| 1010 | D | 5000 |

Now consider a query that is submitted at **site 2.** Find the details of the employee who have sold more than 6000 quantity of item B. The SQL query to evaluate it could be:

```
SELECT e.empno, e.emp-name, e.phone, e.job
FROM employee@site1 e, sales@site2  s
WHERE e.empno=s.empno AND s.item-code=B AND quantitysold>6000;
```

Now, to execute this query by simply transferring employee table from site 1 to site 2 may be very expensive, likewise transfer of sales table to site 1, taking the join at site 1 and sending the result back to site 2 may also be very expensive. A possible optimal solution for the query may be:

(i) Find those tuples at site 2 which satisfy the sales conditions at site 2; now project this table on empno.

```
SELECT empno
FROM sales@site2
WHERE item-code=B AND quantitysold >6000
```

This will result in a very small table. Let us store this result in the TEMP table.

(ii) Now transfer this TEMP table to site 1 and take a join there as:

```
SELECT *
FROM employee@site1 e, TEMP
WHERE e.empno = TEMP.empno
```

(iii) Now send the result of step (ii) back to site 2

Thus, this method reduces the cost of communication and hence may enhance performance.

Similarly, an update statement that may effect data at many sites may be partitioned and sent accordingly to different site requirements.

A distributed database system supports may queries, updates, procedure calls and transactions. Let us define them briefly.

A *remote query* accesses information from one remote site. It may access one or more tables at that site. For example, the following query submitted at site 2:

```
SELECT * FROM employee@site1;
```

A *remote update* modifies data at one remote site. It may modify one or more tables at that site. For example the following query issued at site 2:

```
UPDATE employee@site1
SET phone= '29250000'
WHERE empno=10001;
```

A *distributed query* retrieves information from more than one site. Query on sale of an item, as given, while discussing query optimisation is an example of a distributed query.

A *distributed update* modifies data on more than one site. A distributed update program may be written by using an embedded SQL in a host language. You can also write a procedure or trigger. A distributed update would access and update data on more than one site. In such updates program statements are sent to remote sites and execution of a program is treated as a single outcome viz., success or failure.

The *Remote Procedure Calls (RPCs)* is a procedure written in the host language using embedded SQL and are used to perform some tasks at the remote database server site. This call is invoked when a program calls a remote procedure. The local server passes the parameters to such remote procedures.

*Remote Transaction* contains one or more statements that references/modifies data at one remote site.



A *distributed transaction* includes one or more statements that references/ modifies data on two or more distinct sites of a distributed database.

Having discussed queries and the basic definitions of different types of queries and transactions on a distributed database, we are now ready to discuss concurrency control mechanism used for distributed databases.

## 4.6 DISTRIBUTED CONCURRENCY CONTROL

The basic objectives of concurrency control mechanism in a distributed database are:

(1) to ensure that various data items and their replications are in a consistent state, when concurrent transactions are going on at the same time, and
(2) to ensure that all the concurrent transactions get completed in finite time.

The objectives as above should be fulfilled by the concurrency control mechanism under the following:

- failure of a site or communication link,
- strict performance guidelines supporting parallelism of queries, and minimal storage and computational overheads, and
- dealing with the communication delays of the networked environment.

Despite the above considerations it should support atomicity.

A distributed database system has to deal with the following two types of problems that occur due to distributed concurrent transactions:

- Concurrent transactions related problems that occur in a centralised database, (namely, lost updates, read uncommitted and inconsistent analysis).
- In addition, concurrency related problems arise due to data distribution and replication. For example, the need to update a data item that is located at multiple locations. Such data item require to be updated at all the locations or on none at all.

Let us discuss concurrency related issues in distributed databases in more details.

### 4.6.1 Distributed Serialisability

Distributed serialisability is the extension of the concept of serialisability in a centralised database system under data distribution and replication. It is defined as follows:

> If the transaction schedule at each site is serialisable, then the **global schedule** (the union of all such local schedules at various sites) will also be serialisable if and only if the order of execution of transactions in the global schedules does not violate the order of execution of transactions in any of the local schedules.

Like centralised databases, concurrency control in the distributed environment is based on two main approaches−locking and timestamping. We have discussed these two techniques in unit 2 of this block, but just to recollect −locking ensures that the concurrent execution of a set of transactions is equivalent to *some* serial execution of those transactions. The timestamp mechanism ensures that the concurrent execution of a set of transactions is in the same order of the timestamps.

In case, a distributed database has only data fragmentation and no replication, then for remote transactions, we can directly use locking or timestamping protocols that are used for a centralised database, with modifications for handling remote data. However, to deal with distributed transactions and data replications we need to extend these protocols. In addition, for the locking protocol we need to make sure that no deadlock occurs as transactions are processed on more than one site. Let us discuss these mechanisms in more detail.

### 4.6.2 Locking Protocols

The basic locking protocol that is used in the centralised system – the two-phase locking (2PL) has been extended in the distributed system. Let us look at some of the extensions of the two-phase locking protocol.

#### Centralised 2PL

As the name of this protocol suggests that in this protocol a single site known as the central site maintains the entire locking information. Therefore, it has only one lock manager that grants and releases locks. There is only one *lock manager,* for the entire distributed DBMS that can grant and release locks. The following are the sequence of operations in the centralised 2PL protocol of a global transaction initiated at site $S_i$:

(1) A transaction starts at a site $S_i$. This start site is mostly assigned the responsibility of coordinating that transaction. Therefore, it is also known as the transaction coordinator.

(2) A transaction coordinator takes the help of the global system catalogue to find details of the data items needed by the transaction it is coordinating. It then divides the transaction into a number of sub-transactions.

(3) The coordinator makes sure that a transaction is committed as per proper commit protocols as discussed in section 4.7. In addition, an updating transaction needs to obtain locks on the data items it is updating during the life cycle of the transaction. The transaction needs to make sure that all copies of data items are being updated by it, are updated consistently. Thus, the transaction requests an exclusive lock on those data items. However, a transaction that requires only Reading of a data item, can read it from any copy, preferably the coordinator performs read from its own local copy, if the data item already exists.

(4) The sub transactions that are being performed at local sites have their own local transaction manager. The local transaction manager is responsible for making the request for obtaining or releasing a lock (following the rules of 2PL) to the centralised lock manager.

(5) The centralised lock manager grants the locks or asks local transactions to wait by putting them in the wait queue, after checking the status of the item on which lock request is made. In case the lock is granted then this acknowledgement is communicated to the local lock transaction manager.

This scheme can be modified by making the transaction manager responsible for making all the lock requests rather than the sub transaction's local transaction manager. Thus, the centralised lock manager needs to talk to only the transaction coordinator.

One of the major advantage of the centralised 2PL is that this protocol can detect deadlocks very easily as there is only one lock manager. However, this lock manager often, becomes the bottleneck in the distributed database and may make a system less reliable. This is, due to the fact, that in such a case, for the system to function correctly it needs to depend on a central site meant for lock management.

#### Primary Copy 2PL

In this scheme instead of relying on one central site the role of the lock manager is distributed over a number of sites. These lock managers are made responsible for managing the locks on the data items. This responsibility fixing is done by naming one of the copies of the replicated data as the **primary copy.** The other copies are called **slave copies**. Please note that it is not necessary that the site that is made responsible for managing locks on the primary copy is the same site.

In a scheme, a transaction coordinator requests the lock from the site that holds the primary copy so that the request can be sent to the appropriate lock manager. A lock



in this scheme may be granted only on the primary copy as this is the copy that will ensure that changes are propagated to the appropriate slave copies consistently.

This approach may be a good approach for distributed databases in which we do not have frequent updates, and, inconsistent values for some time are acceptable, as this scheme does not always result in consistent values of data in the primary and slave copies. The major disadvantage of this mechanism is that it complicates the process of deadlock detection, as there is many lock managers now.

This locking protocol has lower communication costs and better performance than centralised 2PL as it requires less number of remote locks.

**Distributed 2PL**

The basic objective of this protocol is to overcome the problem of the central site controlling the locking. It distributes the lock managers to every site such that each the lock manager is responsible for managing the locks on the data items of that site. In case there is no replication of data, this protocol, thus, will be identical to the primary copy locking.

Distributed 2PL is a Read-One-Write-All type of protocol. But what does this read one and write all protocol mean. It means that for reading a data item any copy of the replicated data may be read (as all values must be consistent), however, for an update all the replicated copies must be exclusively locked. This scheme manages the locks in a decentralised manner. The disadvantage of this protocol is that deadlock handling is very complex, as it needs to be determined by information from many distributed lock managers. Another disadvantage of this protocol is the high communication cost while updating the data, as this protocol would require locks on all the replicated copies and updating all of them consistently.

### Majority locking

The distributed locking protocol is a natural protocol for the distributed system, but can we avoid the overheads on updates in such a protocol? Well, we need not lock all the replicated copies for updates; it may be alright, if the majority of replicas are locked. The lock manager still needs to be maintained at each site, but now, there would be less number of lock requests for update. However, now, the lock request would be needed for reading as well. On a read or write request from a transaction for a data item the following will be checked:

> Assume the request is for a data item having n replications, then read or exclusive locks are to be obtained for at least ($\lfloor n/2 \rfloor + 1$) replicas. Thus, for each data item more than half of the locks need to be obtained on the replicated data item instances by sending requests to that many lock managers. A transaction needs to obtain all the required locks in a time period, or else it may roll back the lock request. Please note, in this mechanism, a shared lock may be placed on a replicated data item instance by many transactions, but only one exclusive lock can be held on it. Also, the item, cannot be locked exclusively if at least one transaction holds a shared lock. Thus, a data item with all its replication may be locked in read or shared mode by more than one transaction, but only by one transaction in the exclusive mode.

The majority of locking protocols is complex to implement. It may not even be very useful for applications that require more reads and less writes as in this case, pf reading this protocol requires majority locks. Another important thing here is to consider that the number of successful lock requests is equal to the number of unlocking requests too. Thus, reading in this protocol is more expensive than in distributed locking.

## 4.6.3 Timestamp Protocols

Timestamping protocols are basically designed to model the order of execution of transactions in the order of the timestamps. An older transaction has a lower

timestamp and should be given the resources in case of a resource conflict. In a centralised system the timestamp can be generated by a single source. It is therefore, unique. But, how do you generate unique timestamps for a distributed database system? One possible solution here may be to generate unique local and global timestamps for the transactions. This can be done by producing a unique local timestamp and appending it with the information such as a unique site identifier. But where should we put this unique site identifier on the timestamp? Obviously, since the protocol deals with the timestamp order, the site identifier should be at the least significant bits. This would require synchronisation of local timestamps that are being produced at local sites. This synchronisation would require that messages pass between/among the different sites sharing its latest timestamp.

Let us explain the timestamp generation with the help of an example:

Consider two sites of a distributed database management system, producing the timestamps as integer numbers. Also assume that the sites have their unique site identifiers as 1 and 2 respectively. Assume that the site 1 have many transactions, whereas site 2 has a only few transactions. Then some of the timestamps generated by them would be shown in the following *Table*:

| **Site 1** | | | **Site 2** | | |
|---|---|---|---|---|---|
| Local timestamp | Site identifier | Unique Global Timestamp | Local timestamp | Site identifier | Unique Global Timestamp |
| 1<br>2<br>3<br>4<br>5<br>6 | 1<br>1<br>1<br>1<br>1<br>1 | 1,1<br>2,1<br>3,1<br>4,1<br>5,1<br>6,1 | 1<br>2 | 2<br>2 | 1,2<br>2,2 |
| MESSAGE FOR SYNCRONISATION SYNCHRONISES BOTH THE SITES TO HAVE NEXT LOCAL TIMESTAMP AS 7 | | | | | |
| 7<br>8<br>.. | 1<br>1 | 7,1<br>8,1 | 7 | 2 | 7,2 |

### 4.6.4 Distributed Deadlock Management

A deadlock is a situation when some of the transactions are waiting for each other to free up some data items such that none of them is able to continue. The deadlock in a distributed database base management system is even more difficult to detect than that of a centralised database system. Let us demonstrate this with the help of an example.

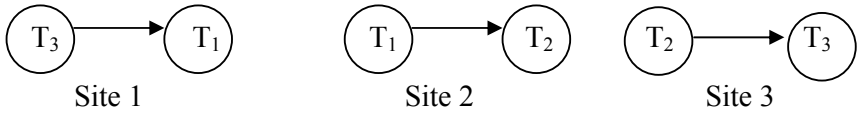
Consider three transactions $T_1$, $T_2$ and $T_3$ with the following locking requests:

| Transaction $T_1$<br>Transaction Coordinator: Site 1 (S1)<br>Sub transaction Site: S2 | Transaction $T_2$<br>Transaction Coordinator: S2<br>Sub transaction Site: S3 | Transaction $T_3$<br>Transaction Coordinator: S3<br>Sub transaction Site: S1 |
|---|---|---|
| (S1) SLOCK (X)<br>(S1) SLOCK (Y)<br>(S2) XLOCK (SUM)<br>(S2) SLOCK (Z)<br>Send X, Y values to S2<br>(S2) SUM = X + Y + Z<br>(S1) UNLOCK (X)<br>(S1) UNLOCK (Y)<br>(S2) UNLOCK (Z)<br>(S2) UNLOCK (SUM) | (S2) XLOCK (Z)<br>(S3) XLOCK (A)<br>(S2) Z = Z - 1000<br>(S3) A = A +1000<br>(S2) UNLOCK (Z)<br>(S3) UNLOCK (A) | (S3) XLOCK (A)<br>(S1) XLOCK (X)<br>(S1) X = X - 1000<br>(S3) A = A +1000<br>(S1) UNLOCK (X)<br>(S3) UNLOCK (A) |

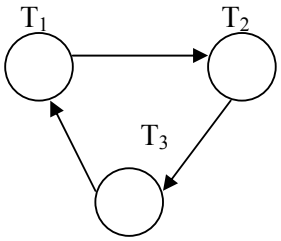
Assume that the three transactions start at the same time, then the lock assignments will be like:

At site S1: $T_1$ has shared lock on X and Y. The sub-transaction of $T_3$ is waiting for lock on item X that is held by $T_1$.
At site S2: $T_2$ has exclusive lock on Z, while the sub-transaction of $T_1$ is waiting for lock on item Z that is held by $T_2$.
At site S3: $T_3$ has shared lock on A. The sub-transaction of $T_2$ is waiting for lock on item A that is held by $T_3$.

Thus, the local wait-for graphs for the three sites are given in *Figure 4* (a). Please note that there is no cycle in the local wait-for graph at each site. However, if we combine the local wait-for graphs to form a global wait-for graph (please refer to *Figure 4* (b)) we detect a cycle of the form: $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$ which indicates the deadlock situation.




**(a) Local wait for graph**




**(b) Global wait for graph: Detects a deadlock**

**Figure 4: Distributed deadlock detection**

Thus, to detect a deadlock situation in a DDBMS, we must make both local and Global wait-for graphs. Let us now discuss the three methods of detecting deadlocks in DDBMSs – Centralised, Hierarchical and Distributed deadlock detection mechanisms.

**Centralised Deadlock Detection:** The following are the steps for the centralised deadlock detection scheme:

- Appoint a single central site as the Deadlock Detection Coordinator (DDC), which constructs and maintains the global WFG.
- Each local lock manager must transmit its local WFG to the DDC.
- The DDC builds the global WFG and checks for cycles in this WFG.
- In case one or more cycles break each cycle by rolling back a transaction this operation will break the entire cycle.
- Restart this rolled back transaction and inform all the sub-transactions involved.

The information flow can be minimised in this approach by transferring the portions of local WFG that have been changed since the last transmission.
The centralised approach is a compromise on reliability in a distributed database system.

**Hierarchical deadlock detection:** This is a scheme based on the tree type architecture. In this approach the Global wait-for graph is constructed by passing local WFGs to the site above in the hierarchy. *Figure 5* shows a hierarchy of eight sites, $S_1$ to $S_8$.

Every level in the hierarchy is a combination of the previous level combined wait-for graphs.

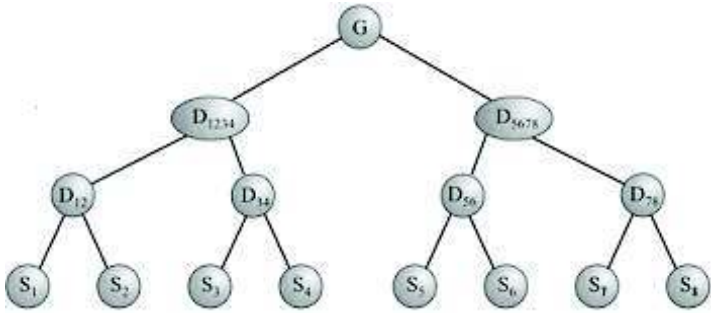



**Figure 5: Hierarchical deadlock detection**

The hierarchical approach is a complex approach but it reduces the dependence on a centralised detection site. It also reduces the local WFGs communication costs.

**Distributed Deadlock Detection:** There are many distributed deadlock detection algorithms. We will briefly explain one of these algorithms.

The distributed deadlock detection algorithm leaves the responsibility of deadlock detection to a local node where transactions are taking place. Each node has its local wait for graph, however, the information about potential deadlock cycles are passed by all other participating sites. Thus, they are able to detect possible deadlock situation. The basic algorithm for it is as follows:

(1) create the local wait for graph,

(2) add possible edges obtained from other sites that may cause deadlock,

(3) the local wait for graph now also contains locks on remote objects and the sub-transactions holding those locks, and

(4) determine the cycles, if it is to be found. There is a deadlock.

### ☞ Check Your Progress 2

1) What are the issues of query processing in a distributed system?

……

……

……

……

2) What are the serialisability requirements for a distributed database?

……

……

……

……

3) Why can we not detect deadlocks in a distributed database with a simple wait for graph?

……

……

……

……



# 4. 7 DISTRIBUTED COMMIT PROTOCOLS: 2 PC AND 3 PC

A transaction is a unit of work as well as the unit of recovery. A transaction has four basic properties – atomicity, consistency, isolation and durability. Thus, it is the responsibility of any DBMS to ensure that the transaction is executed as a single unit, that is, completely or not at all. It should leave the database in a consistent state, the effect of execution of one transaction should not be visible to other transactions till commit, and the transaction once committed cannot be undone. A DDBMS is also expected to preserve these properties of the transaction. However, in order to do so, it needs to do much more than the centralised databases. The key here is to design a commit protocol for distributed transactions. Let us analyse the commit protocol of distributed DDBMS in more details.

### 4.7.1 Two-Phase Commit (2PC)

In a DDBMS since a transaction is being executed at more than one site, the commit protocol may be divided into two phases. The basic objective of this *two-phase commit* protocol is to make sure that a transaction and all its sub-transactions commit together or do not commit at all. Please note that, a two-phase commit protocol ensures integrity and handles remote procedure calls and triggers.

A 2PC consists of two phases:

- voting phase and
- decision phase.

For each transaction, there is a coordinator who controls the 2PC. The sub-transaction sites act as participants. The logical steps of 2PC are:

- coordinator asks its participants whether they are prepared to commit the transaction, or not
- participants may votes to commit, abort or fail to respond within a time-out period,
- the coordinator makes a decision to commit if all participants say commit or abort even if one participant says so or, even if, one of the participant does not respond,
- global decision is communicated to all the participants,
- if a participant votes to abort, then it is free to abort the transaction immediately as the result of this voting will definitely be abort. This is known as a **unilateral** abort,
- if a participant votes to commit, then it waits for the coordinator to send the *global commit* or *global abort* message,
- each participant maintains its local log so that it can commit or rollback the local sub-transaction reliably. In 2PC participants wait for messages from other sites, thus, unnecessarily blocking the participating processes, even though a system of timeouts is used.

The steps for the coordinator in 2PC commit is as follows:

**Coordinator**

*Phase 1*

- Writes a *start-commit* record to its log file.

- Sends a PREPARE message to all the participants.
- Waits for participants to respond within a time period.

*Phase 2*

- If even a single participant returns ABORT vote, then write *abort* record to its log and send a GLOBAL-ABORT message to all the participants. It then waits for participants to acknowledge the global message within a time period.
- If a participant returns a COMMIT vote, then update the list of responses. Check if all participants have voted COMMIT, if so it writes a *commit* record to its log file. It then sends a GLOBAL-COMMIT message to all participants and then waits for acknowledgements in a specific time period.
- If all acknowledgements are received then the coordinator writes an *end-transaction* message to its log file. If a site does not acknowledge then the coordinator re-sends the global decision until an acknowledgement is received.

In Phase1 the coordinator should wait until it has received the vote from all participants. If a site fails to vote, then the coordinator assumes an ABORT vote by default and takes a decision to GLOBAL-ABORT and sends this message to all participants.

The steps at the participant in 2PL is as follows:

**Participant**

*Phase 1*

- On receiving a PREPARE message, a participant:
  - (a) writes a *ready to commit* record to its local log file and send a COMMIT vote message to the coordinator, or
  - (b) writes an *abort* record to its log file and send an ABORT message to the coordinator. It can now abort the transaction unilaterally.
- It then waits for the coordinator to respond within a time period.

*Phase 2*

- If the participant receives a GLOBAL-ABORT message, it writes an *abort* record to its log file. It then aborts the transaction and on completion of this step, it sends an acknowledgement to the coordinator. Please note in case this transaction has sent an abort message in phase 1 then it may have completed the unilateral abort. In such a case only the acknowledgement needs to be sent.
- If the participant receives a GLOBAL-COMMIT message, it writes a *commit* record to its log file. It then commits the transaction, releases all the locks it holds, and sends an acknowledgement to the coordinator.

What happens when a participant fails to receive the GLOBAL – COMMIT or GLOBAL – ABORT from the coordinator or the coordinator fails to receive a response from a participant? In such a situation a termination protocol may be invoked. A termination protocol is followed by only the active sites. The failed sites need to follow those when they recover. In practice, these protocols are quite complex in nature but have been implemented in most of the distributed database management system.

*Figure 6* depicts the communication between the participant and coordinator.

*Figure 7* shows the state of transitions during 2PC.

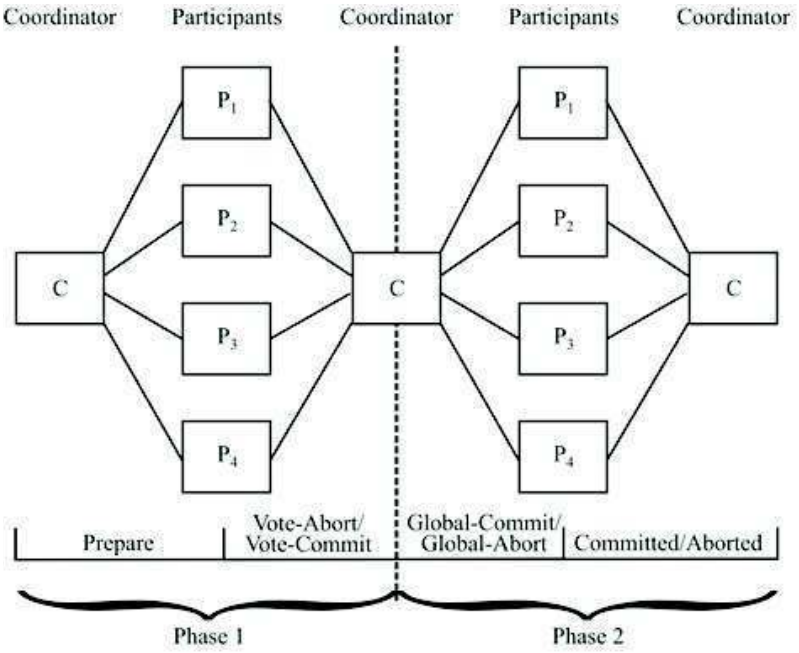



**Figure 6: 2 PC Communication structure**

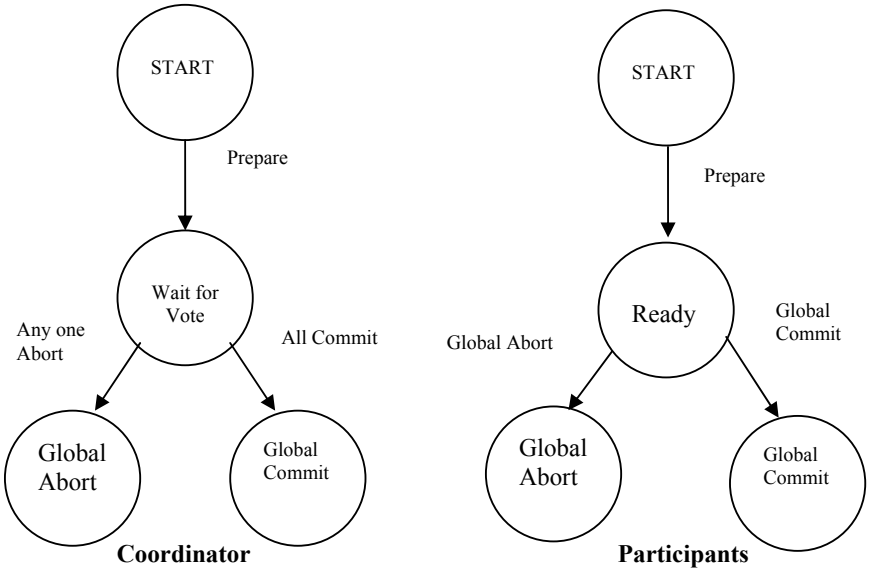



**Figure 7: State Transition diagram in 2PC**

### 4.7.2 Three-Phase Commit (3PC)

The 2PC is a good protocol to be implemented in a DDBMS. However, it has a small disadvantage−it can block participants in certain circumstances. For example, processes that encounter a timeout after voting COMMIT, but before receiving the global commit or abort message from the coordinator, are waiting for the message and doing nothing or in other words are *blocked*. Practically, the probability of blocking is very low for most existing 2PC implementations. Yet an alternative non-blocking protocol, – the **three-phase commit** (3PC) protocol exits. This protocol does not block the participants on site failures, except for the case when all sites fail. Thus, the basic conditions that this protocol requires are:

- no network partitioning,

- at least one available site,
- at most *K* failed sites (called *K-resilient).*

Thus, the basic objective of the 3PC is to remove the blocking period for the participants who have voted COMMIT, and are thus, waiting for the global abort/ commit message from the coordinator. This objective is met by the 3PC by adding another phase – the third phase. This phase called **pre-commit** is introduced between the voting phase and the global decision phase.

If a coordinator receives all the commit votes from the participants, it issues a global PRE-COMMIT message for the participants. A participant on receiving the global pre-commit message knows that this transaction is going to commit definitely.

The coordinator on receiving the acknowledgement of PRE-COMMIT message from all the participants issues the global COMMIT. A Global ABORT is still handled the same way as that of in 2PC.

*Figure 8* shows the state transition diagram for 3PC. You can refer to further readings for more details on this protocol.

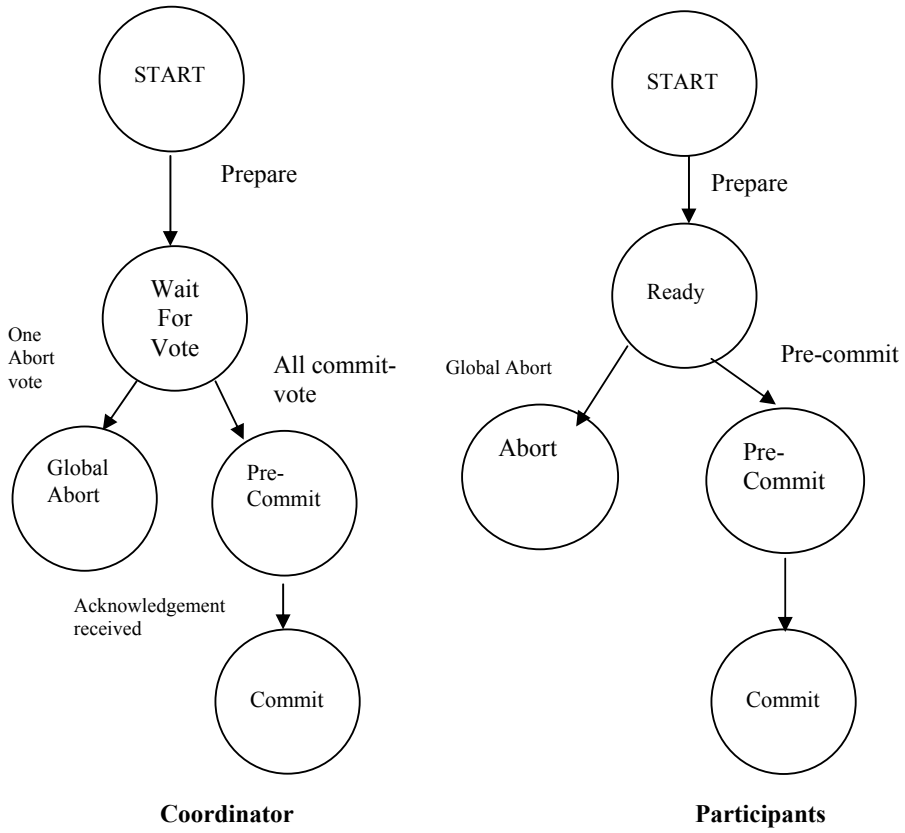



**Figure 8: State transition diagram of 3PC**

Thus, commit protocols ensure that DDBMS are in consistent state after the execution of a transaction. Transaction is also a unit of recovery. The other issue related to distributed databases is security. We have already discussed recovery and security in centralised systems in this block. A detailed discussion on these topics in the context of distributed database system is beyond the scope of this unit. You may refer to the further readings for more details on distributed database systems.

## 4.8 CONCEPTS OF REPLICATION SERVER

In many database applications, a large number of remote users may, need to perform transactions or queries that requires large volume of data transfer. This may block the



server. So what can we do about this situation? The solution to this problem may be provided by the concept of the replication server. Data replication distributes data to certain remote sites so that users can access current data as and when they need the data from the site with less traffic. Replication provides a number of benefits. Some of these benefits are improved performance, increased reliability, data availability and automatic recovery.

*A Replication Server* replicates a database at a remote secondary site. It transmits data from the primary database server (data origination server) to the secondary site on the network. It also handles data transmission to ensure that each replicated database server has an up-to-date copy of the data.

In case the replicated data is updated immediately along with the updation of the source data, then it is referred to as *synchronous replication.* Two possible protocols for this synchronous replication is the 2PC discussed in the previous section. Such replications are useful in applications with critical data such as financial transactions. However, this scheme has some disadvantages as well. For example, a transaction will not be completed if one or more of the sites, holding the replicas, are down. Also the number of messages required to coordinate the synchronisation of data are additional network overheads.

Many commercial distributed DBMSs provide an asynchronous replication in which the target database is updated **after** the source database has been modified. In such a case the delay in regaining consistency may range from a few seconds to several hours or even more. However, the data eventually will synchronise at all replicated sites. This is a practical compromise between data integrity and availability. This form of replication is applicable for organisations that are able to work with replicas that do not necessarily have to be completely synchronised.

Replication Server can do the following operations:

- move transaction to many destinations,
- move data or only a subset of data from one source to another,
- transform data when moving from one source to destination such as merging data from several source databases into one destination database, and
- moving data through a complicated network.

A Replication Server is intended for very close to real-time replication scenarios. The time taken by the data to travel from a source to the destination depends on the number of data items handled by a transaction, the number of concurrent transactions in a particular database, the length of the path (one or more replication servers that the transaction has to pass through to reach the destination), the network bandwidth and traffic etc. Usually, on a LAN, for small transactions, the time taken is about a second.

### 4.8.1 Replication Server Components

The following are the components of a replication server:

**Basic Primary Data Server:** It is the source of data where client applications are connected to enter/delete and modify data.

**Replication Agent/Log Transfer Manager:** This is a separate program/process, which reads transaction log from the source server and transfers them to the replication server.

**Replication Server (s):** The replication server sends the transactions to the target server, which could be another replication server or the target data server.

**Replicate (target) Data server:** This is the destination server in which the replication server repeats the transaction that was performed on the primary data server.

### 4.8.2 Benefits of a Replication Server

The following are the benefits of replication servers:

- Use of Replication Server is efficient, because it only replicates original data that is added, modified, or deleted.
- Replication Servers help in enhancing performance. This is because of the fact that, the Replication Server copies the data to the remote server that a remote user can access over the Local Area Network (LAN).
- Replication server provides an excellent feature for disaster recovery. For example, if the local data server or local network is down and transactions need to be replicated, then the Replication Server will perform all of the necessary synchronisation on the availability of the local data server or the local network.

### 4.8.3 Functionality

The functionality that a replication server must include is:

- *Scalability:* It should handle the replication of small or large volumes of data.
- *Mapping and transformation:* It should handle replication across heterogeneous DBMSs, which will involve mapping and transformation of the data from one data model or data type into a different data model or data type.
- *Object replication*: The replication server system should allow replicating objects such as logs, indexes, etc. in addition to replication of data.
- *Specification of replication schema:* The system should provide a mechanism that allows authorised users define the data and objects that are to be replicated.
- *Subscription mechanism:* The system should provide a mechanism to allow an authorised user to subscribe to the data and the objects that are to be made available for replication.
- *Initialisation mechanism:* The system should provide a mechanism for replicating the data for the first time.
- *Easy administration:* It should be easy administer the system. It should be easy to check the status and monitor the performance of the replication system components.

### 4.8.4 Data Ownership

The site that has the privilege to update the data is known as the owner of the data. The main types of ownership are master/slave, workflow, and update-anywhere (also called peer-to-peer or symmetric replication).

**Master/Slave Ownership:** In such a ownership scheme a site called the Master or Primary site owns the data. Only this site is allowed to modify the data. The other sites can subscribe to the data. They can use the local copy of the data for their purposes. Each site can be a master site for some non-overlapping data sets. The following are some of the examples of the usage of this type of replication:

- *Decision Support System (DSS) Analysis:* Data from one or more distributed databases can be replicated to a separate, local DSS for read-only analysis.
- *Distribution and Dissemination of Centralised Information:* Example of such kinds of application may be a product catalogue and its price. This information could be maintained at the headoffice site and replicated to read-only copies for the remote branch sites.
- *Consolidation of Remote Information:* Data consolidation is a process in which the data that is being updated locally is brought together in a single read-only repository at one location. For example, student details maintained at each



regional office could be replicated to a consolidated read-only copy for the headquarters site.

- *Mobile Computing:* Replication is one of the methods for providing data to a mobile workforce. In this case, the data is downloaded on demand from a local server.

**Workflow Ownership:** Workflow ownership passes the right to update replicated data from site to site. However, at one time only one site may update the particular data set. A typical example of such ownership may be an order processing system, where for each order processing step like order entry, credit approval, invoicing, shipping, the ownership is passed to a different site. Thus, in this ownership model the application tasks and the rights to update the data for those tasks move.

**Update-Anywhere (Symmetric Replication) Ownership:** In both the previous ownership types the data can be modified only by one site. However, in many applications we may wish to update data from other sites too. The update-anywhere model creates a peer-to-peer environment that allows different sites equal rights to updating replicated data. Such sharing of ownership may lead to data inconsistencies. Such inconsistencies could require the implementation of advanced protocols for transactions.

### ☞ Check Your Progress 3

1) What are the steps of two phase commit protocol?

……

……

2) Why do we need replication servers?

……

……

## 4.9 SUMMARY

This unit provided a brief overview of some of the major concepts used in distributed database management systems. The centralised database systems have to give way to distributed systems for organisations that are more complex and autonomous. The homogenous distributed databases may be a good idea for some organisations. The query processing in DDBMS requires multiple sites in the picture and needs optimisation on the part of data communication cost. Deadlocks detection in a distributed transaction is more complex as it requires local and global wait for graphs. Distributed commit protocols require two phase, one voting phase and second decision-making phase. Sometimes a third phase is added between the two phases called pre-commit state, to avoid blocking. The replication server helps in reducing the load of a database server; it also helps in minimising communication costs. You can refer to more details on these topics in the further readings section.

## 4.10 SOLUTIONS/ANSWERS

### Check Your Progress 1

1)

- Access to remote sites, transfer of queries and data through a network,
- A System catalogue for a distributed system having data distribution details,

- Distributed query processing including query optimisation and remote data access,
- Concurrency control to maintain consistency of replicated sites,
- Proper security control for proper authorisation/access privileges of the distributed data, and
- Recovery services to check the failures of individual sites and communication links.

2) Reference architecture shows data distribution, which is an extension of the ANSI/SPARC three level architecture for a centralised DBMS. The reference architecture consists of the following:

- A set of global external schemas to describe the external view for the entire DBMS,
- A global conceptual schema containing the definition of entities, relationships, constraints and security and integrity information for the entire system,
- A fragmentation and allocation schema, and
- A set of schemas for each local DBMS as per ANSI/SPARC architecture.

## Check Your Progress 2

1) A query may be answered by more than one site, in such a case, we would, require that the communication cost among sites should be minimum. You may do data processing at the participant sites to reduce this communication cost.

2) If transactions are distributed at different sites then they should be executed in exactly the same sequence at all the sites.

3) Same set of transactions at different sites may have locked certain resources for which they may be waiting although locally it may seem that $T_1$ is waiting for $T_2$ on site 1 and $T_2$ is waiting for $T_1$ at site 2, but only on combining these two we know that both transactions cannot proceed. Hence, a deadlock occurs.

## Check Your Progress 3

1)

| COORDINATOR | PARTICIPANTS |
|---|---|
| Prepare<br>Wait for vote<br>Collect vote if all COMMIT<br>Send Global COMMIT and ABORT to all participant<br>Waits for acknowledgement from all participant<br>TRANSACTION OVER | Listen to Coordinator<br>Vote<br>Wait for Global Decision<br>Receive decision acts accordingly and sends acknowledgement<br>TRANSACTION OVER |

2) They help in:

- High availability,
- Low communication overheads due to local data availability,
- Disaster recovery, and
- Distribution of functions of a system across organisation, etc.

# UNIT 1 OBJECT ORIENTED DATABASE

**Structure** **Page No.**


1.0 Introduction 5
1.1 Objectives 5
1.2 Why Object Oriented Database? 6
1.2.1 Limitation of Relational Databases
1.2.2 The Need for Object Oriented Databases
1.3 Object Relational Database Systems 8
1.3.1 Complex Data Types
1.3.2 Types and Inheritances in SQL
1.3.3 Additional Data Types of OOP in SQL
1.3.4 Object Identity and Reference Type Using SQL
1.4 Object Oriented Database Systems 15
1.4.1 Object Model
1.4.2 Object Definition Language
1.4.3 Object Query Language
1.5 Implementation of Object Oriented Concepts in Database Systems 22
1.5.1 The Basic Implementation issues for Object-Relational Database Systems
1.5.2 Implementation Issues of OODBMS
1.6 OODBMS Vs Object Relational Database 23
1.7 Summary 24
1.8 Solutions/Answers 24


## 1.0 INTRODUCTION

Object oriented software development methodologies have become very popular in the development of software systems. Database applications are the backbone of most of these commercial business software developments. Therefore, it is but natural that, object technologies also, have their impact on database applications. Database models are being enhanced in computer systems for developing complex applications. For example, a true hierarchical data representation like generalisation hierarchy scheme in a rational database would require a number of tables, but could be a very natural representation for an object oriented system. Thus, object oriented technologies have found their way into database technologies. The present day commercial RDBMS supports the features of object orientation.

This unit provides an introduction to various features of object oriented databases. In this unit, we shall discuss, the need for object oriented databases, the complex types used in object oriented databases, how these may be supported by inheritance etc. In addition, we also define object definition language (ODL) and object manipulation language (OML). We shall discuss object-oriented and object relational databases as well.

## 1.1 OBJECTIVES

After going through this unit, you should be able to:

- define the need for object oriented databases;
- explain the concepts of complex data types;
- use SQL to define object oriented concepts;

- familiarise yourself with object definition and query languages, and
- define object relational and object-oriented databases.

## 1.2 WHY OBJECT ORIENTED DATABASE?

An object oriented database is used for complex databases. Such database applications require complex interrelationships among object hierarchies to be represented in database systems. These interrelationships are difficult to be implement in relational systems. Let us discuss the need for object oriented systems in advanced applications in more details. However, first, let us discuss the weakness of the relational database systems.

### 1.2.1 Limitation of Relational Databases

Relational database technology was not able to handle complex application systems such as Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), and Computer Integrated Manufacturing (CIM), Computer Aided Software Engineering (CASE) etc. The limitation for relational databases is that, they have been designed to represent entities and relationship in the form of two-dimensional tables. Any complex interrelationship like, multi-valued attributes or composite attribute may result in the decomposition of a table into several tables, similarly, complex interrelationships result in a number of tables being created. Thus, the main asset of relational databases viz., its simplicity for such applications, is also one of its weaknesses, in the case of complex applications.

The data domains in a relational system can be represented in relational databases as standard data types defined in the SQL. However, the relational model does not allow extending these data types or creating the user's own data types. Thus, limiting the types of data that may be represented using relational databases.

Another major weakness of the RDMS is that, concepts like inheritance/hierarchy need to be represented with a series of tables with the required referential constraint. Thus they are not very natural for objects requiring inheritance or hierarchy.

However, one must remember that relational databases have proved to be commercially successful for text based applications and have lots of standard features including security, reliability and easy access. Thus, even though they, may not be a very natural choice for certain applications, yet, their advantages are far too many. Thus, many commercial DBMS products are basically relational but also support object oriented concepts.

### 1.2.2 The Need for Object Oriented Databases

As discussed in the earlier section, relational database management systems have certain limitations. But how can we overcome such limitations? Let us discuss some of the basic issues with respect to object oriented databases.

The objects may be complex, or they may consists of low-level object (for example, a window object may consists of many simpler objects like menu bars scroll bar etc.). However, to represent the data of these complex objects through relational database models you would require many tables – at least one each for each inherited class and a table for the base class. In order to ensure that these tables operate correctly we would need to set up referential integrity constraints as well. On the other hand, object

oriented models would represent such a system very naturally through, an inheritance hierarchy. Thus, it is a very natural choice for such complex objects.

Consider a situation where you want to design a class, (let us say a Date class), the advantage of object oriented database management for such situations would be that they allow representation of not only the structure but also the operation on newer user defined database type such as finding the difference of two dates. Thus, object oriented database technologies are ideal for implementing such systems that support complex inherited objects, user defined data types (that require operations in addition to standard operation including the operations that support polymorphism).

Another major reason for the need of object oriented database system would be the seamless integration of this database technology with object-oriented applications. Software design is now, mostly based on object oriented technologies. Thus, object oriented database may provide a seamless interface for combining the two technologies.

The Object oriented databases are also required to manage complex, highly interrelated information. They provide solution in the most natural and easy way that is closer to our understanding of the system. **Michael Brodie** related the object oriented system to human conceptualisation of a problem domain which enhances communication among the system designers, domain experts and the system end users.

The concept of object oriented database was introduced in the late 1970s, however, it became significant only in the early 1980s. The initial commercial product offerings appeared in the late 1980s. Today, many object oriented databases products are available like Objectivity/DB (developed by Objectivity, Inc.), ONTOS DB (developed by ONTOS, Inc.), VERSANT (developed by Versant Object Technology Corp.), ObjectStore (developed by Object Design, Inc.), GemStone (developed by Servio Corp.) and ObjectStore PSE Pro (developed by Object Design, Inc.). An object oriented database is presently being used for various applications in areas such as, e-commerce, engineering product data management; and special purpose databases in areas such as, securities and medicine.

*Figure 1* traces the evolution of object oriented databases. *Figure 2* highlights the strengths of object oriented programming and relational database technologies. An object oriented database system needs to capture the features from both these world. Some of the major concerns of object oriented database technologies include access optimisation, integrity enforcement, archive, backup and recovery operations etc.

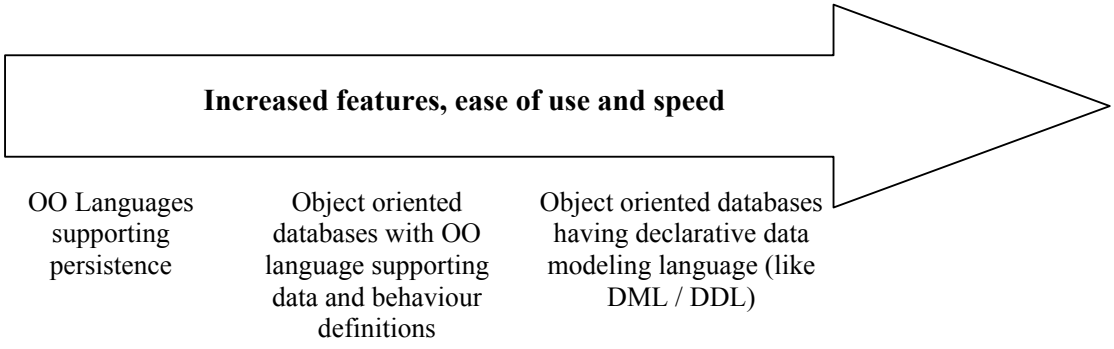



**Figure 1: The evolution of object-oriented databases**

The major standard bodies in this area are Object Management Group (OMG), Object Database Management Group (ODMG) and X3H7.

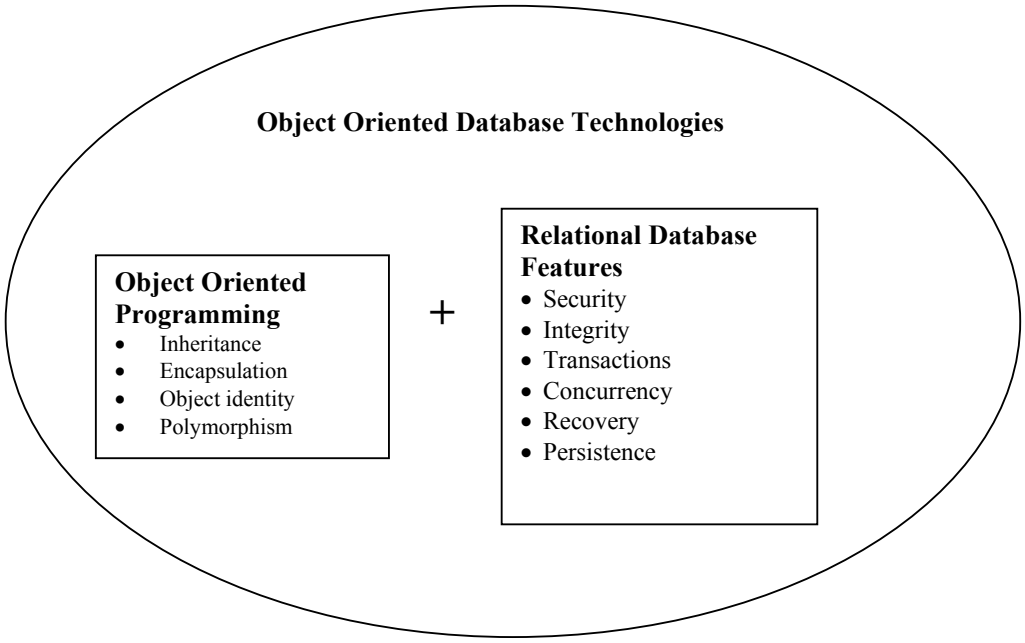



**Figure 2: Makeup of an Object Oriented Database**

Now, the question is, how does one implement an Object oriented database system? As shown in *Figure 2* an object oriented database system needs to include the features of object oriented programming and relational database systems. Thus, the two most natural ways of implementing them will be either to extend the concept of object oriented programming to include database features – OODBMS or extend the relational database technology to include object oriented related features – Object Relational Database Systems. Let us discuss these two viz., the object relational and object oriented databases in more details in the subsequent sections.

## 1.3 OBJECT RELATIONAL DATABASE SYSTEMS

Object Relational Database Systems are the relational database systems that have been enhanced to include the features of object oriented paradigm. This section provides details on how these newer features have been implemented in the SQL. Some of the basic object oriented concepts that have been discussed in this section in the context of their inclusion into SQL standards include, the complex types, inheritance and object identity and reference types.

### 1.3.1 Complex Data Types

In the previous section, we have used the term complex data types without defining it. Let us explain this with the help of a simple example. Consider a composite attribute – *Address*. The address of a person in a RDBMS can be represented as:

House-no and apartment
Locality
City
State
Pin-code

When using RDBMS, such information either needs to be represented as set attributes as shown above, or, as just one string separated by a comma or a semicolon. The second approach is very inflexible, as it would require complex string related operations for extracting information. It also hides the details of an address, thus, it is not suitable.

If we represent the attributes of the address as separate attributes then the problem would be with respect to writing queries. For example, if we need to find the address of a person, we need to specify all the attributes that we have created for the address viz., House-no, Locality…. etc. The question is –Is there any better way of representing such information using a single field? If, there is such a mode of representation, then that representation should permit the distinguishing of each element of the address? The following may be one such possible attempt:

```
CREATE TYPE Address  AS    (
            House              Char(20)
            Locality           Char(20)
            City               Char(12)
            State              Char(15)
            Pincode            Char(6)
            ) ;
```

Thus, *Address* is now a new type that can be used while showing a database system scheme as:

```
CREATE TABLE STUDENT    (
            name               Char(25)
            address            Address
            phone              Char(12)
            programme          Char(5)
            dob                ???
            ) ;
```

* Similarly, complex data types may be extended by including the date of birth field (dob), which is represented in the discussed scheme as??? This complex data type should then, comprise associated fields such as, day, month and year. This data type should also permit the recognition of difference between two dates; the day; and the year of birth. But, how do we represent such operations. This we shall see in the next section.

But, what are the advantages of such definitions?

Consider the following queries:

Find the name and address of the students who are enrolled in MCA programme.

```
SELECT      name, address
FROM        student
WHERE       programme = 'MCA' ;
```

**Please note** that the attribute 'address' although composite, is put only once in the query. But can we also refer to individual components of this attribute?

Find the name and address of all the MCA students of Mumbai.

```
SELECT      name, address
FROM        student
WHERE       programme = 'MCA' AND address.city = 'Mumbai';
```



Thus, such definitions allow us to handle a composite attribute as a single attribute with a user defined type. We can also refer to any of the component of this attribute without any problems so, the data definition of attribute components is still intact.

Complex data types also allow us to model a table with multi-valued attributes which would require a new table in a relational database design. For example, a library database system would require the representation following information for a book.

Book table:

- ISBN number
- Book title
- Authors
- Published by
- Subject areas of the book.

Clearly, in the table above, authors and subject areas are multi-valued attributes. We can represent them using tables (ISBN number, author) and (ISBN number, subject area) tables. (Please note that our database is not considering the author position in the list of authors).

Although this database solves the immediate problem, yet it is a complex design. This problem may be most naturally represented if, we use the object oriented database system. This is explained in the next section.

### 1.3.2 Types and Inheritances in SQL

In the previous sub-section we discussed the data type – Address. It is a good example of a structured type. In this section, let us give more examples for such types, using SQL. Consider the attribute:

- Name – that includes given name, middle name and surname
- Address – that includes address details, city, state and pincode.
- Date – that includes day, month and year and also a method for distinguish one data from another.

SQL uses Persistent Stored Module (PSM)/PSM-96 standards for defining functions and procedures. According to these standards, functions need to be declared both within the definition of type and in a CREATE METHOD statement. Thus, the types such as those given above, can be represented as:

```
CREATE TYPE          Name  AS (
                 given-name Char (20),
                 middle-name        Char(15),
                 sur-name           Char(20)
)
FINAL

CREATE TYPE          Address AS (
                 add-det            Char(20),
                 city               Char(20),
                 state              Char(20),
                 pincode            Char(6)

)
NOT FINAL
```

```
CREATE TYPE     Date  AS  (
            dd              Number(2),
            mm              Number(2),
            yy              Number(4)
)
FINAL
METHOD difference (present Date)
RETURNS INTERVAL days ;
```

This method can be defined separately as:

```
CREATE INSTANCE METHOD difference (present Date)
            RETURNS INTERVAL days FOR Date
BEGIN
// Code to calculate difference of the present date to the date stored in the object. //
// The data of the object will be used with a prefix SELF as: SELF.yy, SELF.mm etc.
//
// The last statement will be RETURN days that would return the number of days//
END
```

These types can now be used to represent class as:

```
CREATE  TYPE    Student AS (
            name        Name,
            address     Address,
            dob         Date
)
```

'FINAL' and 'NOT FINAL' key words have the same meaning as you have learnt in JAVA. That is a final class cannot be inherited further.

There also exists the possibility of using constructors but, a detailed discussion on that is beyond the scope of this unit.

**Type Inheritance**

In the present standard of SQL, you can define inheritance. Let us explain this with the help of an example.

Consider a type University-person defined as:

```
CREATE TYPE     University-person AS (
            name        Name,
            address     Address
)
```

Now, this type can be inherited by the Staff type or the Student type. For example, the Student type if inherited from the class given above would be:

```
CREATE TYPE     Student
            UNDER   University-person  (
            programme Char(10),
            dob             Number(7)
)
```

Similarly, you can create a sub-class for the staff of the University as:

```
CREATE TYPE     Staff
```



```
        UNDER     University-person  (
        designation  Char(10),
        basic-salary Number(7)
)
```

Notice, that, both the inherited types shown above-inherit the name and address attributes from the type University-person. Methods can also be inherited in a similar way, however, they can be overridden if the need arises.

### Table Inheritance

The concept of table inheritance has evolved to incorporate implementation of generalisation/ specialisation hierarchy of an E-R diagram. SQL allows inheritance of tables. Once a new type is declared, it could be used in the process of creation of new tables with the usage of keyword "OF". Let us explain this with the help of an example.

Consider the University-person, Staff and Student as we have defined in the previous sub-section. We can create the table for the type University-person as:

```
CREATE TABLE University-members OF University-person ;
```

Now the table inheritance would allow us to create sub-tables for such tables as:

```
CREATE TABLE student-list OF Student
        UNDER University-members ;
```

Similarly, we can create table for the University-staff as:

```
CREATE TABLE staff OF Staff
        UNDER University-members ;
```

Please note the following points for table inheritance:

- The type that associated with the sub-table must be the sub-type of the type of the parent table. This is a major requirement for table inheritance.
- All the attributes of the parent table – (University-members in our case) should be present in the inherited tables.
- Also, the three tables may be handled separately, however, any record present in the inherited tables are also implicitly present in the base table. For example, any record inserted in the student-list table will be implicitly present in university-members tables.
- A query on the parent table (such as university-members) would find the records from the parent table and all the inherited tables (in our case all the three tables), however, the attributes of the result table would be the same as the attributes of the parent table.
- You can restrict your query to – only the parent table used by using the keyword – ONLY. For example,

```
SELECT NAME FROM university-member ONLY ;
```

### 1.3.3 Additional Data Types of OOP in SQL

The object oriented/relational database must support the data types that allows multi-valued attributes to be represented easily. Two such data types that exit in SQL are:

- Arrays – stores information in an order, and
- Multisets – stores information in an unordered set.

Let us explain this with the help of example of book database as introduced in section 1.3. This database can be represented using SQL as:

```
CREATE TYPE        Book AS  (
            ISBNNO          Char (14),
            TITLE           Char (25),
            AUTHORS Char (25) ARRAY [5],
            PUBLISHER       Char (20),
            KEYWORDS        Char (10) MULTISET
)
```

**Please note,** the use of the type ARRAY. Arrays not only allow authors to be represented but, also allow the sequencing of the name of the authors. Multiset allows a number of keywords without any ordering imposed on them.

But how can we enter data and query such data types? The following SQL commands would help in defining such a situation. But first, we need to create a table:

```
CREATE TABLE       library  OF  Book ;
```

```
INSERT INTO library VALUES.
('008-124476-x', 'Database Systems', ARRAY ['Silberschatz', 'Elmasri' ], 'XYZ
PUBLISHER', multiset [ 'Database', 'Relational', 'Object Oriented']) ;
```

The command above would insert information on a hypothetical book into the database.

Let us now write few queries on this database:

Find the list of books related to area Object Oriented:

```
SELECT ISBNNO, TITLE
FROM library
WHERE 'Object Oriented' IN ( UNNEST ( KEYWORDS)) ;
```

Find the first author of each book:

```
SELECT ISBNNO, TITLE, AUTHORS [1]
FROM  library
```

You can create many such queries, however, a detailed discussion on this, can be found in the SQL 3 standards and is beyond the scope of this unit.

### 1.3.4 Object Identity and Reference Type Using SQL

Till now we have created the tables, but what about the situation when we have attributes that draws a reference to another attribute in the same table. This is a sort of referential constraint. The two basic issues related such a situation may be:

- How do we indicate the referenced object? We need to use some form of identity, and
- How do we establish the link?



Let us explain this concept with the help of an example; consider a book procurement system which provides an accession number to a book:

```
CREATE TABLE        book-purchase-table  (

                ACCESSION-NO    CHAR (10),
                ISBNNO REF (Book) SCOPE (library)
) ;
```

The command above would create the table that would give an accession number of a book and will also refer to it in the library table.

However, now a fresh problem arises how do we insert the books reference into the table? One simple way would be to search for the required ISBN number by using the system generated object identifier and insert that into the required attribute reference. The following example demonstrates this form of insertion:

```
INSERT INTO book-purchase-table VALUES ('912345678', NULL) ;

UPDATE book-table
SET ISBNNO = (SELECT book_id
                    FROM library
                   WHERE ISBNNO = '83-7758-476-6')
WHERE ACCESSION-NO  = '912345678'
```

**Please note** that, in the query given above, the sub-query generates the object identifier for the ISBNNO of the book whose accession number is 912345678. It then sets the reference for the desired record in the book-purchase-table.

This is a long procedure, instead in the example as shown above, since, we have the ISBNNO as the key to the library table, therefore, we can create a user generated object reference by simply using the following set of SQL statements:

```
CREATE TABLE        book-purchase-table  (

                ACCESSION-NO    CHAR (10),
                ISBNNO REF (Book) SCOPE (library) USER GENERATED

) ;

INSERT INTO book-purchase-table VALUES ('912345678', '83-7758-476-6') ;
```

### ☞ Check Your Progress 1

1) What is the need for object-oriented databases?

……..

……

……

2) How will you represent a complex data type?

……..

……

……

3) Represent an address using SQL that has a method for locating pin-code information.

……..

……

……

4) Create a table using the type created in question 3 above.

……..

……

……

5) How can you establish a relationship with multiple tables?

……..

……

……

## 1.4 OBJECT ORIENTED DATABASE SYSTEMS

Object oriented database systems are the application of object oriented concepts into database system model to create an object oriented database model. This section describes the concepts of the object model, followed by a discussion on object definition and object manipulation languages that are derived SQL.

### 1.4.1 Object Model

The ODMG has designed the object model for the object oriented database management system. The Object Definition Language (ODL) and Object Manipulation Language (OML) are based on this object model. Let us briefly define the concepts and terminology related to the object model.

**Objects and Literal**: These are the basic building elements of the object model. An object has the following four characteristics:

- A unique identifier
- A name
- A lifetime defining whether it is persistent or not, and
- A structure that may be created using a type constructor. The structure in OODBMS can be classified as atomic or collection objects (like Set, List, Array, etc.).

A literal does not have an identifier but has a value that may be constant. The structure of a literal does not change. Literals can be atomic, such that they correspond to basic data types like int, short, long, float etc. or structured literals (for example, current date, time etc.) or collection literal defining values for some collection object.

**Interface:** Interfaces defines the operations that can be inherited by a user-defined object. Interfaces are non-instantiable. All objects inherit basic operations (like copy object, delete object) from the interface of Objects. A collection object inherits operations – such as, like an operation to determine empty collection – from the basic collection interface.



**Atomic Objects:** An atomic object is an object that is not of a collection type. They are user defined objects that are specified using *class* keyword. The properties of an atomic object can be defined by its attributes and relationships. An example is the book object given in the next sub-section. **Please note** here that a *class* is instantiable.

**Inheritance:** The interfaces specify the abstract operations that can be inherited by classes. This is called behavioural inheritance and is represented using ": " symbol. Sub-classes can inherit the state and behaviour of super-class(s) using the keyword EXTENDS.

**Extents:** An extent of an object that contains all the persistent objects of that class. A class having an extent can have a key.

In the following section we shall discuss the use of the ODL and OML to implement object models.

### 1.4.2 Object Definition Language

Object Definition Language (ODL) is a standard language on the same lines as the DDL of SQL, that is used to represent the structure of an object-oriented database. It uses unique object identity (OID) for each object such as library item, student, account, fees, inventory etc. In this language objects are treated as records. Any class in the design process has three properties that are attribute, relationship and methods. A class in ODL is described using the following syntax:

```
class <name>
{
      <list of properties>
};
```

Here, class is a key word, and the properties may be attribute method or relationship. The attributes defined in ODL specify the features of an object. It could be simple, enumerated, structure or complex type.

```
class Book
{
      attribute string ISBNNO;
      attribute string TITLE;
      attribute enum CATEGORY
            {text,reference,journal}    BOOKTYPE;
      attribute struct AUTHORS
            {string   fauthor,   string   sauthor,   string
            tauthor}
                                        AUTHORLIST;
};
```

**Please note that**, in this case, we have defined authors as a structure, and a new field on book type as an enum.

These books need to be issued to the students. For that we need to specify a relationship. The relationship defined in ODL specifies the method of connecting one object to another. We specify the relationship by using the keyword "relationship". Thus, to connect a student object with a book object, we need to specify the relationship in the student class as:

```
relationship set <Book> receives
```



Here, for each object of the class student there is a reference to book object and the set of references is called receives.

But if we want to access the student based on the book then the "inverse relationship" could be specified as

```
relationship set <Student> receivedby
```

We specify the connection between the relationship receives and receivedby by, using a keyword "inverse" in each declaration. If the relationship is in a different class, it is referred to by the relationships name followed by a double colon(::) and the name of the other relationship.

The relationship could be specified as:

```
class Book
{
      attribute string ISBNNO;
      attribute string TITLE;
      attribute integer PRICE;
      attribute string PUBLISHER;
      attribute enum CATEGORY
            {text,reference}BOOKTYPE;
      attribute struct AUTHORS
            {string   fauthor,   string   sauthor,   string
            tauthor} AUTHORLIST;
      relationship set <Student> receivedby
            inverse Student::receives;
      relationship set <Supplier> suppliedby
            inverse Supplier::supplies;
};
class Student
{
      attribute string ENROLMENT_NO;
      attribute string NAME;
      attribute integer MARKS;
      attribute string COURSE;
      relationship set <Book> receives
            inverse Book::receivedby;
};
class Supplier
{
      attribute string SUPPLIER_ID;
      attribute string SUPPLIER_NAME;
      attribute string SUPPLIER_ADDRESS;
      attribute string SUPPLIER_CITY;
      relationship set <Book> supplies
            inverse Book::suppliedby;
};
```

Methods could be specified with the classes along with input/output types. These declarations are called "signatures". These method parameters could be in, out or inout. Here, the first parameter is passed by value whereas the next two parameters are passed by reference. Exceptions could also be associated with these methods.

```
class Student
{
      attribute string ENROLMENT_NO;
      attribute string NAME;
      attribute string st_address;
      relationship set <book> receives
```



```
            inverse Book::receivedby;
      void findcity(in set<string>,out set<string>)
            raises(notfoundcity);
};
```

In the method find city, the name of city is passed referenced, in order to find the name of the student who belongs to that specific city. In case blank is passed as parameter for city name then, the exception notfoundcity is raised.

The ODL could be atomic type or class names. The basic type uses many class constructors such as set, bag, list, array, dictionary and structure. We have shown the use of some in the example above. You may wish to refer to the further readings section.

Inheritance is implemented in ODL using subclasses with the keyword “extends”.

```
class Journal extends Book
{
      attribute string VOLUME;
      attribute string emailauthor1;
      attribute string emailauthor2;
};
```

Multiple inheritance is implemented by using extends separated by a colon (:). If there is a class Fee containing fees details then multiple inheritance could be shown as:

```
class StudentFeeDetail extends Student:Fee
{
      void deposit(in set <float>, out set <float>)
                  raises(refundToBeDone)
};
```

Like the difference between relation schema and relation instance, ODL uses the class and its extent (set of existing objects). The objects are declared with the keyword “extent”.

```
class Student (extent firstStudent)
{
      attribute string ENROLMENT_NO;
      attribute string NAME;
      ..........
};
```

It is not necessary in case of ODL to define keys for a class. But if one or more attributes have to be declared, then it may be done with the declaration on key for a class with the keyword “key”.

```
class student (extent firstStudent key ENROLMENT_NO)
{
      attribute string ENROLMENT_NO;
      attribute string NAME;
      ..........
};
```

Assuming that the ENROLMENT_NO and ACCESSION_NO forms a key for the issue table then:

```
class Issue (extent thisMonthIssue key (ENROLMENT_NO,
ACCESSION_NO))
```

```
{
      attribute string ENROLMENT_NO;
      attribute string ACCESSION_NO;
      ..........
};
```

The major considerations while converting ODL designs into relational designs are as follows:

a) It is not essential to declare keys for a class in ODL but in Relational design now attributes have to be created in order for it to work as a key.

b) Attributes in ODL could be declared as non-atomic whereas, in Relational design, they have to be converted into atomic attributes.

c) Methods could be part of design in ODL but, they can not be directly converted into relational schema although, the SQL supports it, as it is not the property of a relational schema.

d) Relationships are defined in inverse pairs for ODL but, in case of relational design, only one pair is defined.

For example, for the book class schema the relation is:

```
Book(ISBNNO,TITLE,CATEGORY,fauthor,sauthor,tauthor)
```

Thus, the ODL has been created with the features required to create an object oriented database in OODBMS. You can refer to the further readings for more details on it.

### 1.4.3 Object Query Language

Object Query Language (OQL) is a standard query language which takes high-level, declarative programming of SQL and object-oriented features of OOPs. Let us explain it with the help of examples.

Find the list of authors for the book titled “The suitable boy”

```
SELECT b.AUTHORS
FROM Book b
WHERE b.TITLE=”The suitable boy”
```

The more complex query to display the title of the book which has been issued to the student whose name is Anand, could be

```
SELECT b.TITLE
FROM Book b, Student s
WHERE s.NAME =”Anand”
```

This query is also written in the form of relationship as

```
SELECT b.TITLE
FROM Book b
WHERE b.receivedby.NAME =”Anand”
```

In the previous case, the query creates a bag of strings, but when the keyword DISTINCT is used, the query returns a set.

```
SELECT DISTINCT b.TITLE
FROM Book b
```



```
WHERE b.receivedby.NAME =”Anand”
```

When we add ORDER BY clause it returns a list.

```
SELECT b.TITLE
FROM Book b
WHERE b.receivedby.NAME =”Anand”
ORDER BY b.CATEGORY
```

In case of complex output the keyword “Struct” is used. If we want to display the pair of titles from the same publishers then the proposed query is:

```
SELECT DISTINCT Struct(book1:b1,book2:b2)
FROM Book b1,Book b2
WHERE b1.PUBLISHER =b2.PUBLISHER
AND b1.ISBNNO < b2.ISBNNO
```

Aggregate operators like SUM, AVG, COUNT, MAX, MIN could be used in OQL. If we want to calculate the maximum marks obtained by any student then the OQL command is

```
Max(SELECT s.MARKS FROM Student s)
```

Group by is used with the set of structures, that are called “immediate collection”.

```
SELECT cour, publ,
                AVG(SELECT p.b.PRICE FROM partition
          p)
FROM Book b
GROUP BY cour:b.receivedby.COURSE, publ:b.PUBLISHER
```

HAVING is used to eliminate some of the groups created by the GROUP by commands.

```
SELECT cour, publ,
                AVG(SELECT p.b.PRICE FROM partition
          p)
FROM Book b
GROUP BY cour:b.receivedby.COURSE, publ:b.PUBLISHER

HAVING AVG(SELECT p.b.PRICE FROM partition p)>=60
```

Union, intersection and difference operators are applied to set or bag type with the keyword UNION, INTERSECT and EXCEPT. If we want to display the details of suppliers from PATNA and SURAT then the OQL is

```
(SELECT DISTINCT su
FROM Supplier su
WHERE su.SUPPLIER_CITY=”PATNA”)
UNION
(SELECT DISTINCT su
FROM Supplier su
WHERE su.SUPPLIER_CITY=”SURAT”)
```

The result of the OQL expression could be assigned to host language variables. If, costlyBooks is a set <book> variable to store the list of books whose price is below Rs.200 then

```
costlyBooks =    SELECT DISTINCT b
```

```
                    FROM Book b
                    WHERE b.PRICE > 200
```

In order to find a single element of the collection, the keyword "ELEMENT" is used. If costlySBook is a variable then

```
costlySBook =    ELEMENT (SELECT DISTINCT b
                          FROM Book b
                          WHERE b.PRICE > 200
                          )
```

The variable could be used to print the details a customised format.

```
bookDetails =    SELECT DISTINCT b
                 FROM Book b
                 ORDER BY b.PUBLISHER,b.TITLE;
bookCount = COUNT(bookDetails);
for (i=0;i<bookCount;i++)
{
      nextBook = bookDetails[i];
      cout<<i<<"\t"<<nextBook.PUBLISHER <<"\t"<<
                              nextBook.TITLE<<"\n";

}
```

☞ **Check Your Progress 2**

1) Create a class staff using ODL that also references the Book class given in section 1.5.

……..

……

…….

2) What modifications would be needed in the Book class because of the table created by the above query?

……..

……

…….

3) Find the list of books that have been issued to "Shashi".

……..

……

…….

# 1.5 IMPLEMENTATION OF OBJECT ORIENTED CONCEPTS IN DATABASE SYSTEMS

Database systems that support object oriented concepts can be implemented in the following ways:

- Extend the existing RDBMSs to include the object orientation; Or



- Create a new DBMS that is exclusively devoted to the Object oriented database.

Let us discuss more about them.

### 1.5.1 The Basic Implementation Issues for Object-Relational Database Systems

The RDBMS technology has been enhanced over the period of last two decades. The RDBMS are based on the theory of relations and thus are developed on the basis of proven mathematical background. Hence, they can be proved to be working correctly. Thus, it may be a good idea to include the concepts of object orientation so that, they are able to support object-oriented technologies too. The first two concepts that were added include the concept of complex types, inheritance, and some newer types such as multisets and arrays. One of the key concerns in object-relational database are the storage of tables that would be needed to represent inherited tables, and representation for the newer types.

One of the ways of representing inherited tables may be to store the inherited primary key attributes along with the locally defined attributes. In such a case, to construct the complete details for the table, you need to take a join between the inherited table and the base class table.

The second possibility here would be, to allow the data to be stored in all the inherited as well as base tables. However, such a case will result in data replication. Also, you may find it difficult at the time of data insertion.

As far as arrays are concerned, since they have a fixed size their implementation is straight forward However, the cases for the multiset would desire to follow the principle of normalisation in order to create a separate table which can be joined with the base table as and when required.

### 1.5.2 Implementation Issues of OODBMS

The database system consists of persistent data. To manipulate that data one must either use data manipulation commands or a host language like C using embedded command. However, a persistent language would require a seamless integration of language and persistent data.

**Please note:** The embedded language requires a lot many steps for the transfer of data from the database to local variables and vice-versa. The question is, can we implement an object oriented language such as C++ and Java to handle persistent data? Well a persistent object-orientation would need to address some of the following issues:

**Object persistence:** A practical approach for declaring a persistent object would be to design a construct that declares an object as persistent. The difficulty with this approach is that it needs to declare object persistence at the time of creation, An alternative of this approach may be to mark a persistent object during run time. An interesting approach here would be that once an object has been marked persistent then all the objects that are reachable from that object should also be persistent automatically.

**Object Identity:** All the objects created during the execution of an object oriented program would be given a system generated object identifier, however, these identifiers become useless once the program terminates. With the persistent objects it is necessary that such objects have meaningful object identifiers. Persistent object identifiers may be implemented using the concept of persistent pointers that remain valid even after the end of a program.

**Storage and access:** The data of each persistent object needs to be stored. One simple approach for this may be to store class member definitions and the implementation of methods as the database schema. The data of each object, however, needs to be stored individually along with the schema. A database of such objects may require the collection of the persistent pointers for all the objects of one database together. Another, more logical way may be to store the objects as collection types such as sets. Some object oriented database technologies also define a special collection as **class extent** that keeps track of the objects of a defined schema.

## 1.6 OODBMS VERSUS OBJECT RELATIONAL DATABASE

An object oriented database management system is created on the basis of persistent programming paradigm whereas, a object relational is built by creating object oriented extensions of a relational system. In fact both the products have clearly defined objectives. The following table shows the difference among them:

| Object Relational DBMS | Object Oriented DBMS |
|---|---|
| The features of these DBMS include:<br>• Support for complex data types<br>• Powerful query languages support through SQL<br>• Good protection of data against programming errors | The features of these DBMS include:<br>• Supports complex data types,<br>• Very high integration of database with the programming language,<br>• Very good performance<br>• But not as powerful at querying as Relational. |
| One of the major assets here is SQL. Although, SQL is not as powerful as a Programming Language, but it is none-the-less essentially a fourth generation language, thus, it provides excellent protection of data from the Programming errors. | It is based on object oriented programming languages, thus, are very strong in programming, however, any error of a data type made by a programmer may effect many users. |
| The relational model has a very rich foundation for query optimisation, which helps in reducing the time taken to execute a query. | These databases are still evolving in this direction. They have reasonable systems in place. |
| These databases make the querying as simple as in relational even, for complex data types and multimedia data. | The querying is possible but somewhat difficult to get. |
| Although the strength of these DBMS is SQL, it is also one of the major weaknesses from the performance point of view in memory applications. | Some applications that are primarily run in the RAM and require a large number of database accesses with high performance may find such DBMS more suitable. This is because of rich programming interface provided by such DBMS. However, such applications may not support very strong query capabilities. A typical example of one such application is databases required for CAD. |

### ☞ Check Your Progress 3

State True or False.

1) Object relational database cannot represent inheritance but can represent complex database types. T ☐ F ☐

2) Persistence of data object is the same as storing them into files. T ☐ F ☐

3) Object- identity is a major issue for object oriented database especially in the context of referencing the objects. T ☐ F ☐



4) The class extent defines the limit of a class. T ☐ F ☐

5) The query language of object oriented DBMS is stronger than object relational databases. T ☐ F ☐

6) SQL commands cannot be optimised. T ☐ F ☐

7) Object oriented DBMS support very high integration of database with OOP. T ☐ F ☐

## 1.7 SUMMARY

Object oriented technologies are one of the most popular technologies in the present era. Object orientation has also found its way into database technologies. The object oriented database systems allow representation of user defined types including operation on these types. They also allow representation of inheritance using both the type inheritance and the table inheritance. The idea here is to represent the whole range of newer types if needed. Such features help in enhancing the performance of a database application that would otherwise have many tables. SQL support these features for object relational database systems.

The object definition languages and object query languages have been designed for the object oriented DBMS on the same lines as that of SQL. These languages tries to simplify various object related representations using OODBMS.

The object relational and object oriented databases do not compete with each other but have different kinds of applications areas. For example, relational and object relational DBMS are most suited for simple transaction management systems, while OODBMS may find applications with e- commerce, CAD and other similar complex applications.

## 1.8 SOLUTIONS/ANSWERS

**Check Your Progress 1**

1) The object oriented databases are need for:

- Representing complex types.
- Representing inheritance, polymorphism
- Representing highly interrelated information
- Providing object oriented solution to databases bringing them closer to OOP.

2) Primarily by representing it as a single attribute. All its components should also be referenced separately.

3)

```
CREATE TYPE Addrtype AS
      (
            houseNo     CHAR(8),
            street      CHAR(10),
            colony      CHAR(10),
            city        CHAR(8),
            state CHAR(8),
            pincode     CHAR(6),
      );
```

```
      METHOD pin() RETURNS CHAR(6);
      CREATE METHOD pin() RETURNS CHAR(6);
      FOR Addrtype
      BEGIN
      . . . . .
      END
```

4)

```
CREATE TABLE address OF Addrtype
(
      REF IS addid SYSTEM GENERATED,
      PRIMARY KEY (houseNo,pincode)
};
```

5) The relationship can be established with multiple tables by specifying the keyword "SCOPE". For example:

```
Create table mylibrary
{
      mybook REF(Book) SCOPE library;
      myStudent REF(Student) SCOPE student;
      mySupplier REF(Supplier) SCOPE supplier;
};
```

## Check Your Progress 2

1)

```
class Staff
{
      attribute string STAFF_ID;
      attribute string STAFF_NAME;
      attribute string DESIGNATION;
      relationship set <Book> issues
            inverse Book::issuedto;
};
```

2) The Book class needs to represent the relationship that is with the Staff class. This would be added to it by using the following commands:

```
RELATIONSHIP SET < Staff > issuedto
      INVERSE :: issues Staff
```

3)

```
SELECT DISTINCT b.TITLE
FROM BOOK b
WHERE b.issuedto.NAME = "Shashi"
```

## Check Your Progress 3

1) False 2) False 3) True 4) False 5) False 6) False 7) True

# " UNIT 2 DATABASE AND XML

**Structure Nos.** **Page**


| | | |
|---|---|---|
| 2.0 | Introduction | 26 |
| 2.1 | Objectives | 27 |
| 2.2 | Structured, Semi Structured and Unstructured Data | 28 |
| 2.3 | XML Hierarchical (Tree) Data Model | 28 |
| 2.4 | XML Tag Sets | 29 |
| 2.5 | Components of XML Document | 29 |
| | 2.5.1 Document Type Declaration (DTD) | |
| | 2.5.2 XML Declaration | |
| | 2.5.3 Document Instance | |
| 2.6 | XML Schema | 34 |
| | 2.6.1 XML Schema Datatypes | |
| | 2.6.2 Schema vs. DTDs | |
| 2.7 | XML Parser | 37 |
| 2.8 | XML Namespaces | 39 |
| 2.9 | XSL Transformations (XSLT) | 39 |
| 2.10 | XPath | 45 |
| 2.11 | XLinks | 46 |
| 2.12 | XQuery | 47 |
| 2.13 | XML and Databases | 49 |
| | 2.13.1 Microsoft's XML Technologies | |
| | 2.13.2 Oracle's XML Technologies | |
| | 2.13.3 XML Databases | |
| 2.14 | Storage of XML Data | 53 |
| 2.15 | XML Database Applications | 53 |
| 2.16 | Summary | 55 |
| 2.17 | Solutions/Answers | 56 |


## 2.0 INTRODUCTION

XML stands for Extensible Markup Language. It is used to describe documents and data in a standardised, text-based format, easily transportable *via* standard Internet protocols. XML, is based on the *mother* of all markup languages–Standard Generalised Markup Language (SGML).

SGML is remarkable inspiration and basis for all modern markup languages. The first popular adaptation of SGML was HTML, primarily designed as a common language for sharing technical documents. The advent of the Internet facilitated document exchange, but not document display. Hypertext Markup Language (HTML) standardises the description of document layout and display, and is an integral part of every Web site today.

Although SGML was a good format for document sharing, and HTML was a good language for describing the document layout in a standardised way, there was no standardised way of describing and sharing data that was stored in the document. For example, an HTML page might have a body that contains a listing of today's share prices. HTML can structure the data using tables, colours etc., once they are rendered as HTML; they no longer are individual pieces of data to extract the top ten shares. You may have to do a lot of processing.

Thus, there was a need for a tag-based markup language standard that could describe data more effectively than HTML, while still using the very popular and standardised HTTP over the Internet. Therefore, in 1998 the World Wide Web Consortium (W3C) came up with the first Extensible Markup Language (XML) Recommendations.

Now, the XML (eXtended Markup Language) has emerged as the standard for structuring and exchanging data over the Web.

XML can be used to provide more details on the structure and meaning of the data pages rather than just specifying the format of the Web pages. The formatting aspects can be specified separately, by using a formatting language such as XSL (eXtended Stylesheet Language). XML can describe data as records of data store or as a single document.

As a language, XML defines both syntax and grammar rules. The rules are called Document Type Definition (DTD), and are one of the major differences between HTML and XML. XML uses metadata for describing data. The metadata of XML is not complex and adds to the readability of the document. XML, like HTML, also uses tags to describe data however, tags, unlike HTML, describes data and not how to present it. To display XML data, you often transform it using XSLT into an HTML page.

HTML is comprised of a defined set of tags, XML on the other hand has very few defined tags. However, it does not mean that XML is powerless, the greatest power of XML is that it is extensible. You can create your own tags with your own semantic meaning. For example, you can create a tag to use for your customer information data such as:

<Customer_First_Name> Manoj </Customer_ First_Name>

This tag has meaning for you and, thus, to your application. This tag has been created by you to designate customer's first name but its tells nothing about its presentation. But how is this tag useful to us? Consider now that data stream contains multiple customers information. If you want to find all customers with first name "Manoj" you can easily search for the <Customer_First_Name> tags. You cannot perform such types of operation in HTML with the same ease and consistency, as HTML was not designed for such purposes.

**Please note**: XML is case sensitive whereas HTML is not. So, you may see that XML and databases have something in common. So, let us discuss more about XML and databases in this unit.

## 2.1 OBJECTIVES

After going through this unit, you should be able to:

- identify XML & XML Document Basics;
- define XML Data Type Definition (DTD);
- identify XML Schema;
- discuss XML Transformation (XSLT) ;
- give overview of XPath, XLink & XQuery;
- give overview of XML Databases & Storage of XML data, and
- discuss a few real life examples of the usage of XML.

## 2.2 STRUCTURED, SEMI STRUCTURED AND UNSTRUCTURED DATA

The data can be categorised in three categories on the basis of its schema: structured, Semi-structured & Unstructured.

Information stored in databases is known as *structured data* because it is represented in a predefined format. The DBMS ensures that all data follows the defined structures and constraints specified in the schema.

In some applications, data is collected in an *ad-hoc* manner, way before you decide on how to store and manage it. This data may have a certain structure, but not all the information collected will have identical structure. This type of data is termed as semi-structured data. In semi-structured data, the schema or format information is mixed with the data values, since each data object can have different attributes that are not known earlier. Thus, this type of data is sometimes referred to as self-describing data.

A third category is known as unstructured data, as there is very limited indication of the type of data. For example, a text document that contains information embedded within it such as web pages in HTML.

## 2.3 XML HIERARCHICAL (TREE) DATA MODEL

The basic object in XML is the XML document. There are two main structuring concepts that construct an XML document:

**Elements and attributes**

Attributes in XML describe elements. Elements are identified in a document by their start tag and end tag. The tag names are enclosed between angular brackets <…>, and end tags are further identified by a backslash </…>. Complex elements are constructed from other elements hierarchically, whereas simple elements contain data values. Thus, there is a correspondence between the XML textual representation and the tree structure. In the tree representation of XML, internal nodes represent complex elements, whereas leaf nodes represent simple elements. That is why the XML model is called a tree model or a hierarchical model.

There are three main types of XML documents:

1) **Data-centric XML documents:** These documents have small data items that follow a specific structure, and hence may be extracted from a structured database. They are formatted as XML documents in order to exchange or display them over the Web.

2) **Document-centric XML documents:** These are documents with large amounts of text, such as articles. There is little or no structured data elements in such documents.

3) **Hybrid XML documents:** These documents may have parts of both that is structured data and textual or unstructured.

## 2.4 XML TAG SETS

The following section presents a closer look at some of the syntactical rules of XML and also looks at why tags are used at all.

Most tags, and all user-defined tags in the XML document instance (i.e. data section), of an XML document follow the convention of a start tag:

< Some_Tag >
Followed by an end tag:
</ Some_Tag >

Some elements in an XML document contain no data – more specifically the data is contained only in one-or-more attributes. In this case, you can reduce the notation to the "empty" form of tag:

< Some_Tag />

**Note**: The white space after the "<" and before the ">" or "/>" is not required but only used here for asthetic purposes. Also, we will use "……." in some examples to show additional options/information may or may not exist but is omitted for brevity.

**XML document declaration:** Every XML document must start with an XML declaration: <?xml ?>. The W3C strongly recommends that at a minimum, you should include the version information to ensure parser compatibility:

<?xml version="1.0" ?>

**XML Comments:** Comments, in XML are same as they are used in programming languages. They are delimited by a special tag: <!-- -->. **Please note:** Two dashes are required for both the start and end tag. For example:

<!-- A comment -->

XML promotes logical structuring of data and document organisation through the hierarchically nested nature of tag sets and it can create tags that have meaning for your application.

<a> ABC Corporation
 K Kumar
</a>

Is certainly less meaningful than:

<CompanyName> ABC Corporation
<ContactName> K Kumar</ContactName>
</CompanyName>

## 2.5 COMPONENTS OF XML DOCUMENT

An XML document have three parts:

- The XML processing Instruction(s), also called the XML declaration;
- The Document Type Declaration;
- The document instance.

### 2.5.1 Document Type Declaration (DTD)

A DTD is used to define the syntax and grammar of a document, that is, it defines the meaning of the document elements. XML defines a set of key words, rules, data types, etc to define the permissible structure of XML documents. In other words, we can say that you use the DTD grammar to define the grammar of your XML documents. The form of DTD is:

<! DOCTYPE name >

Or
<! DOCTYPE name [ a_dtd_definition_or_declaration ]>

The name, while not necessarily the document name, must be the same name as that of the document root node.

The second point of interest with DOCTYPE is that after the name you can declare your Document Type Definition (DTD), the assembling instructions for the document.

You can define them “in-line” or reference external definitions - which is something like an “include” or “import” statement in a language like C. The advantage of creating one-or-more external DTDs is that external DTDs are reusable – more than one XML document may reference the same DTD. DTDs can also reference each other. But how do we define the structure of a XML document?

The structure of the XML document is created by defining its elements, attributes, their relation to one another and the types of data that each may or may not have. So, how do you define these elements and attributes and how are they related to one another in a DTD?

Elements are defined using the <!ELEMENT> keyword. Its attributes are related to the <!ATTLIST> keyword. The following are a few rules for XML DTD notation:

- A * following the element name implies that the element can be repeated zero or more times in the document.
- A + following the element name means that the element can be repeated one or more times. Such elements are required at least once.
- A ? following the element name means that the element can be repeated zero or one times.
- An element appearing without any of the symbols as above must appear exactly once in the document.
- The type of the element is specified using parentheses following the element. If the parentheses include names of other elements, the element that is being defined would be the children of the element in the tree structure. If the parentheses include the keyword #PCDATA or one of the other data types available in XML DTD, the element is at the leaf node of the tree. PCDATA stands for Parsed Character Data, which is roughly similar to a string data type.
- Parentheses can be nested when specifying elements.
- A bar symbol ( e1 | e2 ) specifies that either e1 or e2 can appear in the document.

For example, if your XML document models the components of a house you might define an element, foundation, that contains another element, floor, and has two attributes, material and size. You would write this as follows:

<!ELEMENT foundation (floor) >
<!ELEMENT floor (#PCDATA) >
<!ATTLIST foundationmaterial (#PCDATA) >
<!ATTLIST foundationsize (#PCDATA) >

Another short example that will appeal to anyone dealing with customers is as follows:

```
<?xml version = “1.0” encoding = “UTF-8” ?>
<!DOCTYPE CustomerOrder >

<!ELEMENT CustomerOrder (Customer, Orders*) >

<!ELEMENT Customer (Person, Address+) >
<!ELEMENT Person (FName, LName) >
<!ELEMENT FName (#PCDATA) >
<!ELEMENT LName (#PCDATA) >
<!ELEMENT Address (#PCDATA) >
<!ATTLIST Address
        AddrType ( billing |  home ) “home” >

<!ELEMENT Orders (OrderNo, ProductNo+) >
<!ELEMENT OrderNo (#PCDATA) >
<!ELEMENT ProductNo (#PCDATA) >
]>
```

In the CustomerOrder example, **please note** the following points:

- A CustomerOrder element contains one-and-only-one Customer element and zero-or-more Orders elements (specified by the * in Orders*).

- The Customer element, contains one-and-only-one Person (defined by name) element and one-or-more Address elements (designated by the + in Address+). Thus, showing an emerging hierarchy that defines the structure. **Please note**: In the defined structure, some elements must exist, some may exist once or more and some may or may not.

- The elements FName and LName do not include elements themselves but have something called #PCDATA; the parsed character data.

- Now look at the attribute declaration:

```
<!ATTLIST Address
AddrType (billing | home) “home”>
```

Here an *attribute* AddrType is declared and is associated with the element Address. Furthermore, the attribute is declared to have one of two values (billing or home) and if none were specified then the default would be “home”.

Let us take an example.

**Program 1: A Sample XML Document**

```
<?xml version = “1.0” encoding = “UTF-8” ?>
<! DOCTYPE CustomerOrder
SYSTEM “http://www.mycompany.com/dtd/order.dtd” >
<CustomerOrder>
        <Customer>
                <Person>
                        <FName> Jayesh </FName>
                        <LName> Kumar </LName>
                </Person>
                <Address AddrType = “home”>
                        D-204, Saket, New Delhi 110012 </Address>
                <Address AddrType = “billing”>
                        C-123, Janpath, NewDelhi 110015</Address>
```

```
        </Customer>
        <Orders>
                <OrderNo> 10 </OrderNo>
                <ProductNo> 100 </ProductNo>
                <ProductNo> 200 </ProductNo>
        </Orders>
        <!-- More Customers can be put here ... -->
</CustomerOrder>
```

This is of an example of an XML data stream containing Customer Orders. As you can see, a <CustomerOrder> contains a <Customer> node that, in turn, contains information about a customer. Notice in this example that, a customer can have only one name, but two addresses – "Home" and "Billing".

### 2.5.2 XML Declaration

The XML processing instruction declares, the document to be an XML document. For an application or parser this declaration is important. It may also include: the version of XML, encoding type; whether the document is stand-alone; what namespace, if any, is used etc. and much more. The encoding attribute is used to inform the XML processor of the type of character encoding that is used in the document. UTF-8 and UTF-16 and ISO-10646-UCS-2 are the more common encoding types. The standalone attribute is optional and if, present, has a value of yes or no. The following is an example of an XML declaration:

<?xml version = "1.0" encoding = "UTF-16" standalone="yes" ?>

### 2.5.3 Document Instance

The other components of an XML document provide information on how to interpret actual XML data, whereas the document instance is the actual XML data. There are basically three types of elemental markup that are used in the making of an XML document: i) The document's root element; ii) child elements to the root; and iii) attributes.

**Document:**

**i) Root Element:** There is no difference between the document root element and other elements in an XML document except that the root is the root. The document root element is required if and only if a document type declaration (DOCTYPE) is present. Document root element must have the same name as the name given in the DOCTYPE declaration. If it does not, the XML document will not be valid.

**ii) Child Elements to the Root:** Elements are nodes in the XML hierarchy that may contain other nodes and may or may not have attributes assigned to them. Elements may or may not contain a value.

**iii) Attributes:** Attributes are properties that are assigned to elements. They provide additional information about the element to which they are assigned.

### ☞ Check Your Progress 1

1) What is semi-structured data?

……

……

……

2) What is XML? How does XML compare to SGML and HTML?

……

……

……

3) Why is XML case sensitive, whereas SGML and HTML are not?

……

……

……

4) Is it easier to process XML than HTML?

……

……

……

5) Why is it possible to define your own tags in XML but not in HTML?

……

……

……

6) Discuss the advantages of XML?

……

……

……

7) What is an XML element? What is an XML attribute?

……

……

……

8) Which three attributes can appear in an XML declaration?

……

……

……

## 2.6 XML SCHEMA

The W3C defines XML Schema as a structured framework of XML documents. Schema is a definition language with its own syntax and grammar, It provides a means to structure XML data and does so with semantics. Unlike a DTD, XML Schema are written in XML. Thus, you do not need to learn a second markup language for the purpose of providing document definitions. Schemata are actually composed of two parts: Structure and datatypes.

### 2.6.1 XML Schema Datatypes

There are two kinds of datatypes in XML Schema: Built-in and User-defined.

The built in datatypes include the primitive datatypes and the derived datatypes. The primitive types include, but are not limited to:

- string
- double
- recurringDuration
- decimal
- Boolean

The derived types are derived from primitive datatypes and include:

- integer: Derived from decimal
- nonPositiveInteger: Derived from integer
- CDATA: Derived from string
- time: Derived from recurringDuration

The user-defined datatypes are those types that are derived from either a built in datatype or other user-defined type.

**The Simplest Type Declaration**

The simple types are defined for basic elements, such as a person's first name, in the following manner:

<simpleType name = "FirstName" type = "string" />

The value of the name attribute is the type name you are defining and expect in your XML documents. The type specifies the datatype upon which the type is based defined by you.
The value of the type attribute must be either a primitive type, such as string, or a derived built-in type, such as integer. You cannot define a simpleType based on a user-defined type. When you need to define types, based on a user-defined type you should use the complexType.

Furthermore, simpleType definitions cannot declare sub-elements or attributes; for such cases you need to use the complexType. However, the simpleType can define various constraining properties, known in XML Schema as facets, such as minLength or Length. This is accomplished by applying a restriction as shown in the following example:

```
<simpleType name="FirstName">
        <restriction base =  "string">
                <minLength value = "0" />
                <maxLength value = "25" />
        </restriction>
</simpleType>
```

Lastly, simpleTypes may be used as the basis of complexTypes.

**The Complex Type: <complexType>**

The complexType is used to define types that are not possible with the simpleType declaration. complexTypes may declare sub-elements or element references.

```
<complexType name = "colonialStyleWindow"
        type = "window" >
  <sequence>
    <element name = "frame" type = "frame Type" />
  </sequence>
</complexType>
```

The <sequence> element is used to define a sequence of one or more elements. In the above example colonialStyleWindow has only one sub-element but it could have more, as you will see in Defining a complexType By Example. There are additional control tags, such as <choice>, which you may use. ComplexTypes may also declare attributes or reference attribute groups.

```
<complexType name = "colonialStyleWindow"
        type = "window" >
  <sequence>
    <element name = "frame" type = "frame Type" />
  </sequence>
  <attribute name = "width" type = "integer" />
  <attribute name = "height" type = "integer" />
</complexType>
```

Or

```
<complexType name = "colonialStyleWindow"
        type = "window" >
  <sequence>
    <element name = "frame" type = "frame Type" />
  </sequence>
  <attributeGroup ref = "windowDimensions" />
</complexType>
```

However, the real power and flexibility of complex types lies in their extensibility – that you can define two complexTypes and derive one from the other. A detailed discussion on them is beyond the scope of this unit. However, let us explain them with the help of an example:

```
<!-- Define a complexType -->
<complexType name = "window" type = "someType"/>

<!-- Now lets define a type based on it! -->
<complexType name = "colonialStyleWindow"
        type = "window" … />
```

**Defining a complexType by Example:** Let us look at a more interesting and complete example of defining a complexType. Here, we define a type Customer that declares and must have one Name element and one Address element. The Person element is of the Name type, defined elsewhere, and the Address type is the Address type, the definition of which follows Customer. Examining the definition of the Address type, you see that, it in turn, declares a sequence of elements: Street, City, PostalCode, and Country. A partial schema for the complexType AddrType may be:

```
<complexType name = "Address"
        type = "AddrType" >
        <sequence>
                <element name = "Street" type = "string" />
                        …..
                  </sequence>
</complexType>
```

Given this Schema, the following XML data fragment would be valid:

```
        <Customer>
                <Person>
                        <FName> Jayesh </FName>
                        <LName> Kumar </LName>
                </Person>
```

```
<Address AddrType = “Home”>
        <Street> A-204, Professor’s Colony </Street>
        <City> New Delhi </City>
        <State> DL </State>
        <PostalCode> 110001</PostalCode>
        <Country> INDIA </Country>
</Address>
<Address AddrType = “billing”>
        <Street> B-104, Saket</Street>
        <City> New Delhi </City>
        <State> DL </State>
        <PostalCode> D-102345</PostalCode>
        <Country> INDIA </Country>
</Address>
</Customer>
```

### 2.6.2 Schema vs. DTDs

Both DTDs and Schema are document definition languages. Schemata are written in XML, while DTDs use EBNF (Extended Backus Naur Format) notation. Thus, schemata are extensible as they are written in XML. They are also easy to read, write and define.
DTDs provide the capability for validation the following:

- Element nesting.
- Element occurrence constraints.
- Permitted attributes.
- Attribute types and default values.

However, DTDs do not provide control over the format and data types of element and attribute values. For example, once an element or attribute has been declared to contain character data, no limits may be placed on the length, type, or format of that content. For narrative documents such as, web pages, book chapters, newsletters, etc., this level of control may be all right. But as XML is making inroads into more record-like applications, such as remote procedure calls and object serialisation, it requires more precise control over the text content of elements and attributes. The W3C XML Schema standard includes the following features:

- Simple and complex data types
- Type derivation and inheritance
- Element occurrence constraints
- Namespace-aware element and attribute declarations.

Thus, schema can use simple data types for parsed character data and attribute values, and can also enforce specific rules on the contents of elements and attributes than DTDs can. In addition to built-in simple types (such as string, integer, decimal, and dateTime), the schema language also provides a framework for declaring new data types, deriving new types from old types, and reusing types from other schemas.

## 2.7 XML PARSER

*Figure 1* shows the interaction between application, XML parser, XML documents and DTD or XML Schema.. An XML source document is fed to the parser, that loads a definition document (DTD or XML Schema) and validates the source document from these definition document.

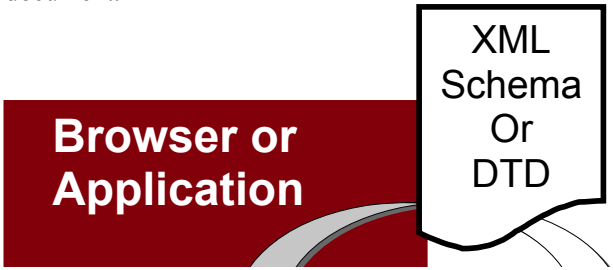


**Figure 1: An XML Parser**

XML parsers know the rules of XML, which obviously includes DTDs or XML Schemata, and how to act on them. The XML parser reads the XML document instance, which may be a file or stream, and parse it according to the specifications of XML. The parser creates an in-memory map of the document creating traversal tree of nodes and node values.

The parser determines whether the document is well-formed. It may also determine if the document instance is valid. But what is a well-formed XML document?

Well-formed XML document contains the required components of an XML document that has a properly nested hierarchy. That is, all tag sets are indeed sets with a begin and end tag, and that intersecting tags do not exist. For example, the following tags are not properly nested because <a> includes <b> but the end tag of <b> is outside the end tag of <a>:

```
<a><b> </a></b>
```

The correct nesting is:

```
<a> <b> </b> </a>
```

A validating parser interprets DTDs or schemata and applies it to the given XML instance. Given below are two popular models for reading an XML document programmatically:

**DOM (Document Object Model)**

This model defines an API for accessing and manipulating XML documents as tree structures. It is defined by a set of W3C recommendations. The most recently completed standard DOM Level 3, provides models for manipulating XML documents, HTML documents, and CSS style sheets. The DOM enables us to:

- Create documents and parts of documents.
- Navigate the documents.
- Move, copy, and remove parts of the document.
- Add or modify attributes.

The Document Object Model is intended to be an operating system- and language-independent, therefore, the interfaces in this model are specified using the Interface Description Language (IDL) notation defined by the Object Management Group.

**Simple API for XML (SAX)**

The Simple API for XML (SAX) is an event-based API for reading XML documents. Many different XML parsers implement the SAX API, including Xerces, Crimson, the Oracle XML Parser for Java, etc. SAX was initially defined as a Java API and is primarily intended for parsers written in Java. However, SAX has been ported to most other object-oriented languages, including C++, Python, Perl, and Eiffel. The SAX API is unusual among XML APIs because it is an event-based push model rather than a tree-based pull model, as the XML parser reads an XML document in real time. Each time the parser sees a start-tag, an end-tag, character data, or a processing instruction, it tells the program. You do not have to wait for the entire document to be read before acting on the data. Thus, the entire document does not have to reside in the memory. This feature makes SAX the API of choice for very large documents that do not fit into available memory.

## 2.8 XML NAMESPACES

Namespaces have two purposes in XML:

1) To distinguish between elements and attributes from different vocabularies with different meanings that happen to share the same name.

2) To group all the related elements and attributes from a single XML application together so that software can easily recognise them.

The first purpose is easier to explain and grasp, but the second purpose is more important in practice.

Namespaces are implemented by attaching a prefix to each element and attribute. Each prefix is mapped to a URI by an xmlns:prefix attribute. Default URIs can also be provided for elements that do not have a prefix. Default namespaces are declared by xmlns attributes. Elements and attributes that are attached to the same URI are in the same namespace. Elements from many XML applications are identified by standard URIs. An example namespace declaration that associates the namespace prefix ‘lib’ with the namespace name http://www.library.com/schema is shown below:

```
<book xmlns:lib = 'http://www.library.com/schema'>
        <!-- the "lib" prefix is now bound to http://ecommerce.org/schema
                for the element "book" and its contents -->
</book>
```

In an XML 1.1 document, an Internationalised Resource Identifier (IRI) can be used instead of a URI. An IRI is just like a URI except it can contain non-ASCII characters such as é: etc. In practice, parsers do not check that namespace names are legal URIs in XML 1.0, so the distinction is mostly academic.

## 2.9 XSL TRANSFORMATIONS (XSLT)

XSLT stands for XML Stylesheet Language Transformations and is yet another widely used and open standard defined by the W3C. Although the W3C defines XSLT as “a language for transforming documents” it is more than that. Unlike XML, XSLT is an active language permitting you to perform Boolean logic on nodes and selected XML sub-trees. Thus, it is closer to a programming language.

It is precisely because of its programmable nature that XSLT enables you to write XSLT transformation documents (a sort of programs). You use these "programs", known as XSL stylesheets (denoted by convention with the file type .XSL) in conjunction with an XSLT processor to transform documents. Although designed for transforming source XML documents into a new target XML documents, XSLT can transform an XML document into another type of text stream, such as an HTML file. A common use of XSL stylesheets is to translate between Schema formats.

### The XSLT Process – Overview

In the case of XSLT, the processor loads a source document and using the already loaded stylesheet, transforms the source into a target document.

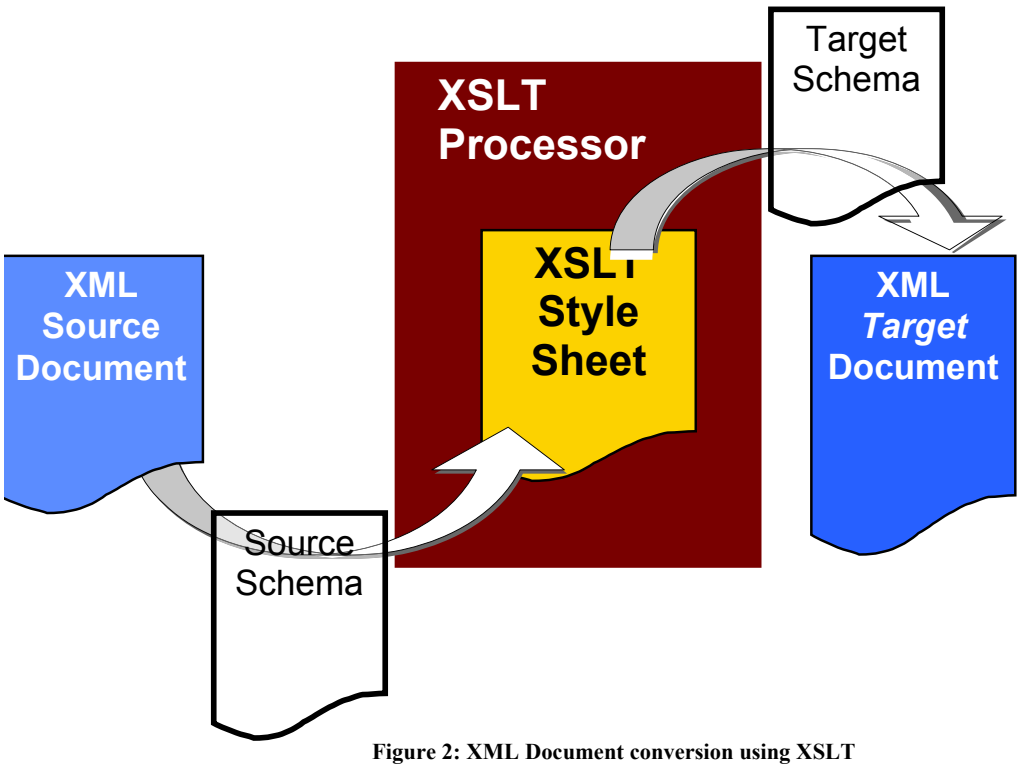



**Figure 2: XML Document conversion using XSLT**

The XSLT process first loads the specified stylesheet. The process then parses the stylesheet and loads the stylesheet templates into memory. It then traverses the source document, node by node, comparing the node values to the directives (or "search conditions") of the stylesheet templates. If there is a match between the current source document node and one of the templates, the process applies the template to the current node. This process continues until the processor has finished traversing the source document node tree and applied all matching templates. The result is a new, transformed document that the XSLT processor then emits as a stream or to file.

In order to perform any of the transformations the right tools; namely, an XSLT processor and a proper XSLT Stylesheet is required. The stylesheet is prefaced with the familiar <?xml?> declaration. But you also need to include the "stylesheet node" which declares the stylesheet namespace. You accomplish this by following your XML processing declaration with:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <!-- The rest of your stylesheet goes here! -->
</ xsl:stylesheet>
```

For example, perhaps your XML source document has the following data:

```
<OrderNo>
  <ProductNo> 100 </ProductNo>
  <ProductNo> 200 </ProductNo>
  <ProductNo> 300 </ProductNo>
</OrderNo>
```

However, you want to display only the ProductNo data and do so as an HTML list: Products in this order:

```
<UL>
  <LI> 100 </LI>
  <LI> 200 </LI>
  <LI> 300 </LI>
</UL>
```

The template will need to contain the <UL> and <LI> HTML codes, but what is needed is the means to select the nodes value you desire to insert between each <LI> and </LI>.

**The Elements - Templates**

The most basic tools at your disposal in XSLT are the <xsl:template> and <xsl:apply-templates/>. The former is used to define a rule and the later to apply it.

**<xsl:template/>**

The <template> tag is used to select a node or node sub-trees. By using the tag's match attribute you can set a pattern to identify the node upon which you wish to apply a rule – or transformation. The pattern value can be a node path, such as Customer/Order/OrderNo, or it can be a predefined pattern value, such as * to select all nodes.

In the previous example, you wanted to select all ProductNo values from a particular Customer node. For the sake of simplicity, we will omit how to iterate over all Customers or Orders for a particular Customer. Let us assume, you have managed to select a particular Customer/Orders/OrderNo and now you wish to display the ProductNo values in that OrderNo.

```
<template match = "Customer/Orders/OrderNo" >
  <! -- apply some rules -->
</template>
```

But this only shows how to select the node or node set on which you wish to work. Now you need to apply some rules – or changes.

```
<xsl:apply-templates />
```

To apply rules you use the <apply-templates> rule. You can apply a template rule to all child nodes of the currently selected node (selected using <template match = "…" />) by using <apply-template /> without a select attribute value pair. Alternatively, you can apply the rule to all child nodes in the set that meet a specific selection criteria by using <apply-template select = "…" /> where the ellipses are some pattern such as ProductNo. The following fragment should help things come together:

```
<xsl:stylesheet ...>
  <html>
    <body>
```

```
    <ul>
    <xsl:template
        match="Customer/Orders/OrderNo">
     <li>
       <xsl:apply-templates select="ProductNo" />
          </li>
         </xsl:template>
         </ul>
         </body>
 </html>
</xsl:stylesheet>
```

Here we have a full example that uses a few XSLT elements not yet covered, such as the <for-each> element that is used to iterate over a selection.

```
<?xml version='1.0'?>
<xsl:stylesheet
      xmlns:xsl="http://www.w3.org/TR/2000/CR-xsl-20001121/">
<xsl:template match="/">
 <HTML>
  <BODY>
    <TABLE border = “3”>
      <xsl:for-each select=“Customer/Orders/OrderNo">
        <xsl:for-each select=“Customer/Orders/ProductNo">
        <TR>
           <TD> <xsl:value-of select=“OrderNo"/></TD>
           <TD> <xsl:value-of select=“ProductNo"/></TD>
        </TR>
        </xsl:for-each>
        <TR></TR>
      </xsl:for-each>
    </TABLE>
  </BODY>
 </HTML>
</xsl:template>
<xsl:stylesheet>
```

Thus, the goal is to create an HTML document that has a table with two columns, one for OrderNo and one for ProductNo, and print out all the OrderNo elements for a Customer. Between each different OrderNo, there will be a blank row.

The first <template> element selects the document, meaning the transformation will apply to the entire source document. Alternatively, you could select a subset of your source document by using different match criteria. Skipping over the HTML code, the first interesting XSLT element is a <for-each> element that iterates over Customer/Orders/OrderNo elements. This means that all the OrderNos for this Customer will be processed.

Recall that each OrderNo can have multiple products, or ProductNo elements. For this reason, the stylesheet has another <for-each> element that iterates over the ProductNo elements. Now, notice that inside the iteration of the ProductNo elements, there are two <value-of> elements. These XSLT elements select the values of the OrderNo and ProductNo respectively and insert them into separate columns.

As you saw in this example, there is more to XSLT than template and apply-templates.

<xsl: value-of select = “...” />

One of the more common XSLT elements to use is the <value-of …>, which creates a new node in the output stream based on the value of the node you select. It also

contains many more elements, however, a discussion on those are beyond the scope of this unit.

### ☞ Check Your Progress 2

1) What are some of the requirements for an XML document to be well-formed?

……..

. ……..

……..

2) What are two XML constructs that let you specify an XML document's syntax so that it can be checked for validity?

……..……..

……..……..

……..……..

3) What is the difference between a well formed XML document and a valid XML Document?

……

……..……..

……

4) Why is the following XML document not well-formed?

```
<?xml version = "1.0" standalone="yes"?>
<employee>
   <name>Jayesh</name>
   <position>Professor</position>
</employee>
<employee>
   <name>Ramesh</name>
   <position>Clerk</position>
</employee>
```

……

……..……..

……

……

5) What's wrong with this XML document?

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document [
<!ELEMENT document (employee)*>
<!ELEMENT employee (hiredate, name)>
]>
<document>
   <employee>
      <hiredate>October 15, 2005</hiredate>
      <name>
         Somya Mittal
      </name>
   </employee>
```

```
</document>
```

……

……..……..

……

……

6) Where do you see a problem with this XML document?

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document [
<!ELEMENT document (employee)*>
<!ELEMENT employee (name, hiredate)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT hiredate (#PCDATA)>
]>
<document>
   <employee>
      <hiredate>October 15, 2005</hiredate>
      <name>
         Somya Mittal
      </name>
   </employee>
</document>
```

……

..……..

……

……

7) Describe the differences between DTD and the XML Schema?

……

……

……

8) Which namespace uses XML schemas?

……

……

……

9) How can you use an XML schema to declare an element called <name> that holds text content?

……

……

……

10) What is the difference between DOM & SAX APIs?

……

……

……

11) What is XSL Transformation (XSLT)?

……

……

……

## 2.10 XPATH

XPath is a language that permits you to go through the XML document hierarchy. The language also permits the selection of nodes in an XML document based on path and/or selection criteria. Viewing an XML document as a forest of nodes, XPath enables you to navigate and select a specified tree or leaf.

The path selection notation is similar to URL's in that you can use both absolute and relative paths. For example, using the CustomerOrder as a base example, you can navigate to an Order node using the following absolute path:

/CustomerOrder/Orders

The general XPath syntax is axis::node_test[ predicate ]

**Axis**

It defines the area from where you wish to start navigating. For instance, the absolute path example you saw previously has an axis of "/" which denotes the document. Some other possible axis values are: parent, child, ancestor, attribute, etc.

**Node test**

The node test is used to select one or more nodes. It can be done by tag name, using a node selector or by using the wildcard (*). The following are several node selelectors: node( ), text( ), comment( ), etc.

**Predicates**

Optional function or expression enclosed in "[...]". A few of the functions available are: position( ), count( ), etc.

Examples of using predicates:

/Address:: * [@AddrType="billing"]

Or

OrderNo[position=1]/ProductNo[position=3]

Fortunately, you do not always have to write Order No.
[position =1]/ProductNo[position=3] when you wish to select the third ProductNo of the first OrderNo. XPath does provide some means of abbreviating. The following is an equivalent expression.

OrderNo[1]/ProductNo[3]

There are other abbreviations also. Please refer to them in the further readings.

## 2.11 XLINKS

XLinks are an attribute-based syntax for attaching links to XML documents. XLinks can be a simple Point A-to-Point B links. XLinks can also be bidirectional, linking two documents in both directions so you can go from A to B or B to A. XLinks can even be multidirectional, presenting many different paths between any number of XML documents. The documents do not have to be XML documents—XLinks can be placed in an XML document that lists connections between other documents that may or may not be XML documents themselves. At its core, XLink is an XML syntax for describing directed graphs, in which the vertices are documents at particular URIs and the edges are the links between the documents.

Current web browsers at most support simple Xlinks. Many browsers, do not support XLinks at all. A simple link defines a one-way connection between two resources. The source or starting resource of the connection is the link element itself. The target or ending resource of the connection is identified by a Uniform Resource Identifier (URI). The link goes from the starting resource to the ending resource. The starting resource is always an XML element. The ending resource may be an XML document, a particular element in an XML document, a group of elements in an XML document, an MPEG movie or a PDF file which are not the part of XML document. The URI may be something other than a URL, may be a book ISBN number like urn:isbn:1283222229.

A simple XLink is encoded in an XML document as an element of an arbitrary type that has an xlink:type attribute with the simple value and an xlink:href attribute whose value is the URI of the link target. The xlink prefix must be mapped to the http://www.w3.org/1999/xlink namespace URI. As usual, the prefix can change as long as the URI is the same. For example, suppose this novel element appears in a list of children's literature and we want to link it to the actual text of the novel available from the URL ftp://HindiLibrary.org/Premchand/Karmabhumi.txt:

```
<novel>
  <title>Karmabhumi</title>
  <author>Munshi Premchand</author>
  <year>1925</year>
</novel>
```

We give the novel element an xlink:type attribute with the value simple, an xlink:href attribute that contains the URL to which we're linking, and an xmlns:xlink attribute that associates the prefix xlink with the namespace URI http://www.w3.org/1999/xlink like so:

```
<novel xmlns:xlink= "http://www.w3.org/1999/xlink"
      xlink:type = "simple"
      xlink:href = "ftp://HindiLibrary.org/Premchand/Karmabhumi.txt">
  <title>Karmabhumi</title>
  <author>Munshi Premchand</author>
  <year>1925</year>
</novel>
```

This establishes a simple link from this novel element to the plain text file found at ftp://HindiLibrary.org/Premchand/karmabhumi.txt. Browsers are free to interpret this link as they like. However, the most natural interpretation, and the one implemented by a few browsers that do support simple XLinks, is to make this a blue underlined phrase. The user can click on to replace the current page with the file being linked to.

## 2.12 XQUERY

The Xquery as explained by W3C is: "XML is a versatile markup language, capable of labelling the information content of diverse data sources including structured and semi-structured documents, relational databases, and object repositories. A query

language that uses the structure of XML intelligently can express queries across all these kinds of data, whether physically stored in XML or viewed as XML via middleware. This specification describes a query language called XQuery, which is designed to be broadly applicable across many types of XML data sources."

XQuery is a query language something like SQL that we can use with XML documents. XQuery is designed to access data much as if we were working with a database, even though we are working with XML. The creation of such a query language is to provide a way to work with data in XML. XQuery not only gives a data model to interpret XML documents, but also a set of operators and functions to extract data from those documents.

The W3C XQuery specification is divided into several working drafts; the main XQuery 1.0 working drafts you download at http://www.w3.org/TR/xquery.
One major difference between XQuery and other query languages, such as SQL and XPath, is that XQuery will have multiple syntaxes. At the minimum it will have a syntax in XML and another more human-readable syntax. The human-readable syntax is defined in the working draft itself, while the XML syntax, called XQueryX, is defined in a separate working draft, at http://www.w3.org/TR/xqueryx.

A glance at the XQueryX working draft shows that the XML syntax for XQuery will be much more than the human-readable syntax. For example, suppose we have a document that conforms to the following DTD (from the XML Query Use Cases document):

```
<!ELEMENT bib  (book* )>
<!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )>
<!ATTLIST book  year CDATA  #REQUIRED >
<!ELEMENT author  (last, first )>
<!ELEMENT editor  (last, first, affiliation )>
<!ELEMENT title  (#PCDATA )>
<!ELEMENT last  (#PCDATA )>
<!ELEMENT first  (#PCDATA )>
<!ELEMENT affiliation  (#PCDATA )>
<!ELEMENT publisher  (#PCDATA )>
<!ELEMENT price  (#PCDATA )>
```

The XQuery Working draft gives an example query to list each publisher in the XML document, and the average price of its books, which is

```
FOR $p IN distinct(document("bib.xml")//publisher)
LET $a := avg(document("bib.xml")//book[publisher = $p]/price)
RETURN
  <publisher>
    <name> {$p/text()} </name>
    <avgprice> {$a} </avgprice>
  </publisher>
```

Since XQuery is only in Working Draft stage, the final syntax could change, meaning that the query syntax above may not be proper XQuery syntax.

The XQuery as above creates a variable, named **p,** which will contain a list of all of the distinct <publisher> elements from the document bib.xml. That is, if there are multiple <publisher> elements which contain the text "IGNOU Press", the **p** variable will only contain one of them, and ignore the rest. For each of the <publisher> elements in the **p** variable, another variable, called **a** will be created, which will contain the average price of all of the books associated with this publisher. It does this by:

Getting a node-set of all of the <price> elements, which are a child of a <book> element whose <publisher> element has the same value as the current value of p. Passing this node-set to the avg() function, which will return the average value. Once the publisher's name, and the average price of its books, have been discovered, an XML fragment similar to

```
<publisher>
  <name>Publisher's name</name>
  <avgprice>Average price</avgprice>
</publisher>
```

will be returned. This is very similar to the types of queries we can create using SQL.

Thus, the final Xquery generated for the problem above would be:

```
<q:query xmlns:q="http://www.w3.org/2001/06/xqueryx">
  <q:flwr>
    <q:forAssignment variable="$p">
      <q:function name="distinct">
        <q:step axis="SLASHSLASH">
          <q:function name="document">
            <q:constant datatype="CHARSTRING">bib.xml</q:constant>
          </q:function>
          <q:identifier>publisher</q:identifier>
        </q:step>
      </q:function>
    </q:forAssignment>
    <q:letAssignment variable="$a">
      <q:function name="avg">
        <q:step axis="CHILD">
          <q:function name="document">
            <q:constant datatype="CHARSTRING">bib.xml</q:constant>
          </q:function>
          <q:step axis="CHILD">
            <q:predicatedExpr>
              <q:identifier>book</q:identifier>
              <q:predicate>
                <q:function name="EQUALS">
                  <q:identifier>publisher</q:identifier>
                  <q:variable>$p</q:variable>
                </q:function>
              </q:predicate>
            </q:predicatedExpr>
            <q:identifier>price</q:identifier>
          </q:step>
        </q:step>
      </q:function>
    </q:letAssignment>
    <q:return>
      <q:elementConstructor>
        <q:tagName>
          <q:identifier>publisher</q:identifier>
        </q:tagName>
        <q:elementConstructor>
          <q:tagName>
            <q:identifier>name</q:identifier>
          </q:tagName>
          <q:step axis="CHILD">
            <q:variable>$p</q:variable>
            <q:nodeKindTest kind="TEXT" />
```

```
        </q:step>
      </q:elementConstructor>
      <q:elementConstructor>
        <q:tagName>
          <q:identifier>avgprice</q:identifier>
        </q:tagName>
        <q:variable>$a</q:variable>
      </q:elementConstructor>
    </q:elementConstructor>
  </q:return>
 </q:flwr>
</q:query>
```

That is a long text for a query. But remembers the XML syntax is primarily not meant to be read by humans. XQueryX is meant to be generated and processed by, software, thereby, making the query explicit. In the XML syntax will make it easier to be processed by these tools.

## 2.13 XML AND DATABASES

With both XML and databases being data-centric technologies, are they in competition with each other? XML is best used to communicate data, whereas database is best used to store and retrieve data. Thus, both of these are complementary, rather than competitive. XML will never replace the database, however, the two will become more closely integrated with time.

For this reason, database vendors have realised the power and flexibility of XML, and are building support for XML right into their products. This potentially makes the programmer's job easier while writing Data Objects, as there is one less step to perform. Instead of retrieving data from the database, and transforming it to XML, programmers now can retrieve data from the database in XML format.

Let us look at two database vendors who were quick to market XML integration technologies: Oracle, and Microsoft. Both companies offer a suite of tools that can be used for XML development, when communicating with a database and otherwise.

### 2.13.1 Microsoft's XML Technologies

Microsoft has provided an extensive documentation on XML at MSDN (Microsoft Developer Network), the online site devoted to developers working with Microsoft technologies is http://msdn.microsoft.com/xml/.

**MSXML**

The first, and most obvious, form of XML support from Microsoft is that Internet Explorer comes bundled with the MSXML – a COM-based parser. MSXML 3 (that is being shipped with IE 6) provides validating and non-validating modes, as well as support for XML namespaces, SAX 2, and XSLT. MSXML 4 also includes support for the XML Schemas Recommendation.

**.NET**

**.**NET is Microsoft's development model in which software becomes a platform and is device-independent, and data becomes available over the Internet. The .NET Framework is the infrastructure of .NET and XML is the backbone to it. XML is a common communication language for .NET components and servers.

**SQL Server**

The XML support has also been built into SQL Server. SQL Server provides the ability to perform a SQL query through an HTTP request, via an ISAPI filter for

Internet Information Server. So programmers might perform a query like the following (replace servername with the name of your web server, and databasename with the name of the database you are querying):

http://servername/databasename?sql=SELECT+last_name+FROM+Customer+FOR+XML+RAW

If we do not want to be passing our complex SQL statements in a URL, we can also create an XML template file to store the query. It would look something like this:

```
<root>
  <sql:query xmlns:sql="urn:schemas-microsoft-com:xml-sql">
    SELECT last_name FROM Customer FOR XML RAW
  </sql:query>
</root>
```

Notice the words FOR XML RAW added to the end of that SQL statement. This is a language enhancement Microsoft has added to allow SQL queries to natively return XML.

If this template is named lastname.xml, we need to execute the SQL by using the following URL:

http://servername/databasename/lastname.xml

And this query would return XML similar to the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <row last_name="Gupta" />
  <row last_name="Bhushan" />
  <row last_name="Kumar" />
</root>
```

### 2.13.2 Oracle's XML Technologies

Like Microsoft, Oracle provides a number of tools to work with XML, in its XML Developer's Kit (XDK). Even though Microsoft had some XML support built right into the database, it took them much longer to get their database-integration tools to market, whereas Oracle had these tools in the market very quickly indeed. Oracle's Technology Network has a good web site devoted to the XML Developer's Kit, which is located at http://technet.oracle.com/tech/xml/.

#### XML Parsers

The first tool available from Oracle is the XML parser. Oracle provides parsers written in Java, C, C++, and PL/SQL. These parsers provide:

- A DOM interface
- A SAX interface
- Both validating and non-validating support
- Support for namespaces
- Fully compliant support for XSLT.

Like MSXML, the Oracle XML parsers provide extension functions to the DOM, selectSingleNode() and selectNodes(), which function just like the Microsoft methods.

#### Code Generators

Oracle offers Java and C++ class generating applications, just like the Visual Basic class building application. However, these generators work from DTDs, not schemas,

meaning that they are already fully conformant to the W3C specifications. Also, since these tools are part of the XDK, they are fully supported by Oracle, instead of just being sample codes.

**XML SQL Utility for Java**

Along with these basic XML tools, the XDK also provides the XML SQL Utility for Java. This tool can generate an XML document from a SQL query, either in text form or in a DOM. The XML results may differ slightly from the Microsoft SQL Server FOR XML clause, but they are just as easy to use. For example, the following SQL query:

SELECT last_name FROM customer

might return XML like the following:

```
<?xml version="1.0"?>
<ROWSET>
 <ROW id="1">
  <last_name>Gupta</last_name>
 </ROW>
 <ROW id="2">
  <last_name>Bhushan</last_name>
 </ROW>
 <ROW id="3">
  <last_name>Kumar</last_name>
 </ROW>
</ROWSET>
```

So, instead of including the information in attributes of the various <row> elements, Oracle decided to include the information in separate child elements. And, just like Microsoft's enhancements to SQL Server 2000, Oracle's XML SQL Utility for Java can take in XML documents, and use the information to update the database.

**XSQL Servlet**

Microsoft decided to make SQL queries available over HTTP by providing an ISAPI filter for IIS. Oracle, on the other hand, decided to create the Java XSQL Servlet instead. Since it is a Java Servlet, it will run on any web server that supports servlets– which is most of them. This servlet takes in an XML document that contains SQL Queries, like the XML templates used by SQL Server. The servlet can optionally perform an XSLT transformation on the results (if a stylesheet is specified), so the results can potentially be any type of file that can be returned from an XSLT transformation, including XML and HTML. (To accomplish this, the XSQL Servlet makes use of the XML parser mentioned above). Because it is a servlet, it will run on any web server that has a Java Virtual Machine and can host servlets.

### 2.13.3 XML Databases

So far, we have focused on integrating XML with traditional databases. The general idea has been to store the data in the database, and transform it to and from XML when needed.

However, there is a fundamental difference between relational databases and XML documents – relational databases are based on relations, tuples and attributes whereas XML documents are hierarchical in nature, using the notions of parents, children, and descendants. This means that there may be cases where the structure of an XML document will not fit into the relational model.

For such cases, you might want to consider a specialised XML database, instead of transforming back- and-forth from XML, like SQL Server and Oracle. An XML database stores data natively in XML. XML format can offer the following benefits:

- The speed of your application have to be increased as all of the components in your application need to deal with the data in XML format. The XML database will eliminate the need to transform the data back-and-forth – it starts in XML, and stays in XML.

- Many of the XML databases coming out provide functionality to query the database using XPath queries, just as relational databases today use SQL. Since XML is hierarchical in nature, XPath can provide a more natural query language than SQL, which is a more focused relational database.

There are already a number of such XML databases available. The features and focus of each product can vary greatly, meaning that not all of these products are in direct competition with each other. Some of the XML databases that you may encounter include:

- Extensible Information Server, from eXcelon (http://www.exceloncorp.com)

- GoXML, from XML Global (http://www.xmlglobal.com/prod/db/)
- dbXML, an open source product from dbXML Group (http://www.dbxml.com)

- Tamino XML Database, from Software AG (http://www.softwareag.com/tamino/)

## 2.14 STORAGE OF XML DATA

When it comes to storing XML there are a number of options available to us. We can store it in:

- plain flat files (such as plain text files held on the operating system's native file system),
- a relational database (such as Oracle, SQL Server or DB2),
- an object database (which stores DOM representations of the document), and
- a directory service (such as Novell NDS or Microsoft Active Directory).

The first two options are by far the most common. If we use flat files they tend to be accessed from the file system of the operating system we are running. It is just like holding a collection of text documents or word processing documents on our file server. The important thing to do if, we take this approach is to make sure that we have a good naming convention for our files so that they are easily accessed.

Relational databases, however, are still by far the most common data storage format. There are many ways to store XML data in relational databases. One, XML data can be stored as strings in a relational database. Second, relations can represent XML data as tree. Third, XML data can be mapped to relations in the same way that E-R schemes are mapped to relational schemas. Increasingly, relational database vendors are adding more sophisticated XML support to their RDBMS so that users can store data in relational tables, but simply send the database XML for it to insert, update or delete the information it holds; they can also request that contents of the database be retrieved and output in XML format. It is no longer necessary with many databases to convert the XML into another form before the database can accept it – if we are able to just send the database an XML file this will save us doing the conversion ourself.

# 2.15 XML DATABASE APPLICATIONS

XML is platform and language independent, which means it does not matter that one computer may be using, for example, Visual Basic on a Microsoft operating system, and the other is a UNIX/LINUX machine running Java code. Any time one computer program needs to communicate with another program, XML is a potential fit for the exchange format. The following are just a few examples.

**Reducing Server Load**

Web-based applications can use XML to reduce the load on the web servers. This can be done by keeping all information on the client for as long as possible, and then sending the information to those servers in one big XML document.

**Web Site Content**

The W3C uses XML to write its specifications. These XML documents can then be transformed to HTML for display (by XSLT), or transformed to a number of other presentation formats. Some web sites also use XML entirely for their content, where traditionally HTML would have been used. This XML can then be transformed to HTML via XSLT, or displayed directly in browsers via CSS. In fact, the web servers can even determine dynamically the kind of browser that is retrieving the information, and then decide what to do. For example, transform the XML to HTML for older browsers, and just send the XML straight to the client for newer browsers, reducing the load on the server.

In fact, this could be generalised to any content, not just web site content. If your data is in XML, you can use it for any purpose, with presentation on the Web being just one possibility.

**Remote Procedure Calls**

XML is also used as a protocol for Remote Procedure Calls (RPC). RPC is a protocol that allows objects on one computer to call objects on another computer to do work, allowing distributed computing. Using XML and HTTP for these RPC calls, using a technology called the Simple Object Access Protocol (SOAP), allows this to occur even through a firewall, which would normally block such calls, providing greater opportunities for distributed computing.

**Web-Enabling Business Using XML**

XML can be a very effective way to represent business data for transfer between different systems and organisations. With the increasing importance of the Internet and related technologies, XML can be a key factor in helping organisations move their businesses to the Web.

*Using XML in Business to Consumer (B2C) Solutions:* When most people think of using the Internet for business, they think of online retailers selling goods and services through a Web site. Retail e-commerce solutions, such as these usually rely on HTML pages accessed by means of a Web browser. The pages show product data from a database and allow customers to place orders, the details of which are also stored in a database. XML can be an extremely effective way to pass data from the database to the Web application. XSL can then be used to easily transform the XML data into HTML for display in the browser. This approach is usually more efficient than retrieving data as a rowset and writing presentation logic in a Web page script or component to render the data. In addition, as more devices are used to connect to the Internet, the same XML data can be transformed using different style sheets to suit different client devices. For example, an XML product catalogue could be transformed into HTML for display in a browser or into Wireless Markup Language (WML) for display on a Wireless Application Protocol (WAP) – enabled cell phone.

This flexibility makes XML a great choice for developing Web-based applications that will be accessed by multiple client types.

*Using XML in Business to Enterprise (B2E) Solutions:* Of course, Internet technologies such as HTTP are often used to build internal applications. This is particularly helpful in environments where multiple platforms and development languages are used because an Intranet-based solution allows any application that can communicate over TCP/IP to be integrated. For applications that allow users to access and manipulate data, XML-aware browsers such as Microsoft Internet Explorer can be used to download and render the data. Users can then manipulate the data in the browser using XML data islands before sending the updated data back to the server for storage in the database. Existing applications running on platforms such as mainframes or UNIX can use XML as a neutral way to describe data. For example, a mail-order company with a large existing Information Management System (IMS) application running on a mainframe could decide to build a Web-based e-commerce program in this case, the existing telephone sales orders can continue to be entered into the IMS application as before, and new orders placed through the Web site can be represented as XML and passed on to be stored in the IMS application.

*Using XML in Business to Business (B2B) Solutions:* One of the most important aspects of Web development is the integration of business processes across trading partners. Most interbusiness processes involve an exchange of business documents, such as orders, invoices, delivery notes, and so on. XML provides an ideal way to describe these business documents for exchange across the Internet. XML schemas can be used to define the XML representation of the business documents, allowing trading partners to agree on a format for the data being exchanged. Of course, each organisation can represent data differently internally, and, use XSL to transform data for exchange. Because XML is a text-based language, business documents can be exchanged by using any protocol, such as HTTP, SMTP, or FTP, or by using a message queuing solution. This flexibility makes it possible for any business to integrate with its trading partners over the Internet.

### ☞ Check Your Progress 3

1) What is the difference between XPath & XLink?

……

……

……

2) What is XQuery?

……

……

……

3) How can you store an XML document in any relational database?

……

……

……

## 2.16 SUMMARY

XML (eXtensible Markup Language) is a meta language (a language for describing other languages) that enables designers to create their own customised tags to provide

functionality not available within HTML. XML is a restricted form of SGML, designed as a less complex markup language than SGML i.e., which at the same time is network aware.

AN XML document consists of elements, attributes, entity, references, comments and processing instructions. An XML document can optionally have a Document Type Definition (DTD) which defines the valid syntax of an XML document.

An XML schema is the definition of a specific XML structure. XML schema uses the W3C XML Schema language to specify how each type of element in the schema is defined and what data type that element has associated with it. The schema is itself an XML document, so it can be read by the same tools that read the XML it describes.

XML APIs generally fall into two categories: tree based and event based. DOM (Document Object Model) is a tree based API for XML that provides an object oriented view of the data. The API was created by W3C and describes a set of platform and language neutral interfaces that can represent any well formed XML or HTML document. SAX (Simple API for XML) is an event based, serial access API for XML that uses callbacks to report parsing events to the applications. The application handles these events through customised event handlers.

The World Wide Web Consortium (W3C) has recently formed an XML Query Working group to produce a data model for XML documents, a set of query operators on this model, and a query language based on these query operators.

Today, XML has become a *de facto* communication language between cross vendor products, components written in different technologies and running on different operating systems; and cross databases. The key of today's distributed architecture is XML based web service, which is supported by all majors' players of the software industry such as like Microsoft, IBM, Oracle, Sun (i.e., Java).

## 2.17 SOLUTIONS/ANSWERS

### Check Your Progress 1

1) Semi-structured data is data that has some structure, but the structure may not be rigid, regular or complete and generally the data does not conform to a fixed schema. Sometimes the term schema-less or self-describing is used to describe such data.

2) Most documents on Web are currently stored and transmitted in HTML. One strength of HTML is its simplicity. However, it may be one of its weaknesses with the growing needs of users who want HTML documents to be more attractive and dynamic. XML is a restricted version of SGML, designed especially for Web documents. SGML defines the structure of the document (DTD), and text separately. By giving documents a separately defined structure, and by giving web page designers ability to define custom structures, SGML has and provides extremely powerful document management system but has not been widely accepted as it is very complex. XML attempts to provide a similar function to SGML, but is less complex. XML retains the key SGML advantages of extensibility, structure, and validation.
XML cannot replace HTML.

3) XML is designed to work with applications that might not be case sensitive and in which the case folding (the conversion to just one case) cannot be predicted. Thus, to avoid making unsafe assumptions, XML takes the safest route and opts for case sensitivity.

4) Yes, for two reasons. The first is that you normally model your data better with XML than with HTML. You can capture the hierarchical structure giving meaning to your information. The second reason is that XML files are well formed. You have a much better idea what comes in the data stream. HTML files, on the other hand, can take many forms and appearances.

5) In HTML, the tags tell the browser what to do to the text between them. When a tag is encountered, the browser interprets it and displays the text in the proper form. If the browser does not understand a tag, it ignores it. In XML, the interpretation of tags is not the responsibility of the browser. It is the programmer who defines the tags through DTD or Schema.

6)

1. Simplicity
2. Open standard and platform/vendor-independent
3. Extensibility
4. Reuse
5. Separation of content and presentation
6. Improved load balancing
7. Support for integration of data from multiple sources
8. Ability to describe data from a wide variety of applications.

7) An XML element is the basic data-holding construct in an XML document. It starts with an opening tag, can contain text or other elements, and ends with a closing tag, like this: <greeting>hello</greeting>. An attribute gives you more information, and is always assigned a value in XML. Here's how you might add an attribute named language to this element:

<greeting language = "en">hello</greeting>.

8) The attributes you can use in an XML document are: version (required; the XML version), encoding (optional; the character encoding), and standalone (optional; "yes" if the document does not refer to any external documents or entities, "no" otherwise).

## Check Your Progress 2

1)

1. An XML document must contain one or more elements.
2. One element, the root element, must contain all the other elements.
3. Each element must nest inside any enclosing elements correctly.

2) Document Type Definitions (DTDs) and XML schemas.

3) XML document that conforms to structural and notational rules of XML is considered well-formed. XML Validating processor checks that an XML document is well-formed and conforms to a DTD, in which case the XML document is considered valid. A well formed XML document may not be a valid document.

4) In a well-formed XML document, there must be one root element that contains all the others.

5) The <hiredate> and <name> elements are not declared in the DTD.

6) The <hiredate> and <name> elements appear in the wrong order.

7)

1. It is written in a different (non-XML) syntax;
2. It has no support for namespaces;

3.It only offers extremely limited data typing.

8) The namespace that is used by XML schemas is www.w3.org/2001/XMLSchema.

9) You can declare the element like this:

<xsd:element name="name" type="xsd:string"/>

10) Document Object Model (DOM) & Simple API for XML (SAX) are two popular models to interpret an XML document programmatically. DOM (Document Object Model) is a tree-based API that provides object-oriented view of data. It describes a set of platform and language-neutral interfaces that can represent any well-formed XML/HTML document. While SAX is an event-based, serial-access API for XML that uses callbacks to report parsing events to the application. Unlike tree-based APIs, event-based APIs do not build an in-memory tree representation of the XML document.

11) XML Stylesheet Langauge (XSL) is created specifically to define how an XML document's data is rendered and to define how one XML document can be transformed into another document. XSLT, a subset of XSL, is a language in both the markup and programming sense, providing a mechanism to transform XML structure into either another XML structure, HTML, or any number of other text-based formats (such as SQL). XSLT's main ability is to change the underlying structures rather than simply the media representations of those structures, as with Cascading Style Sheet (CSS).

## Check Your Progress 3

1) XPath is a declarative query language for XML that provides a simple syntax for addressing parts of an XML document. It is designed for use with XSLT (for pattern matching). With XPath, collections of elements can be retrieved by specifying a directory-like path, with zero or more conditions placed on the path. While XLink allows elements to be inserted into XML documents to create and describe links between resources. It uses XML syntax to create structures that can describe links similar to simple unidirectional hyperlinks of HTML as well as more sophisticated links.

2) Data extraction, transformation, and integration are well-understood database issues that rely on a query language. SQL does not apply directly to XML because of the irregularity of XML data. W3C recently formed an XML Query Working Group to produce a data model for XML documents, set of query operators on this model, and query language based on query operators. Queries operate on single documents or fixed collections of documents, and can select entire documents or subtrees of documents that match conditions based on document content/structure. Queries can also construct new documents based on what has been selected.

3)
1. XML data can be stored as strings in a relational database.
2. Relations can represent XML data as tree.
3. XML data can be mapped to relations in the same way that E-R schemes are mapped to relational schemas.

# UNIT 3 INTRODUCTION TO DATA WAREHOUSING

**Structure** **Page Nos.**


| | | |
|---|---|---|
| 3.0 | Introduction | 59 |
| 3.1 | Objectives | 59 |
| 3.2 | What is Data Warehousing? | 60 |
| 3.3 | The Data Warehouse: Components and Processes | 62 |
| | 3.3.1 Basic Components of a Data Warehouse | |
| | 3.3.2 Data Extraction, Transformation and Loading (ETL) | |
| 3.4 | Multidimensional Data Modeling for Data Warehouse | 67 |
| 3.5 | Business Intelligence and Data Warehousing | 70 |
| | 3.5.1 Decision Support System (DSS) | |
| | 3.5.2 Online Analytical Processing (OLAP) | |
| 3.6 | Building of Data Warehouse | 73 |
| 3.7 | Data Marts | 75 |
| 3.8 | Data Warehouse and Views | 76 |
| 3.9 | The Future: Open Issues for Data Warehouse | 77 |
| 3.10 | Summary | 77 |
| 3.11 | Solutions/Answers | 78 |


## 3.0 INTRODUCTION

Information Technology (IT) has a major influence on organisational performance and competitive standing. With the ever increasing processing power and availability of sophisticated analytical tools and techniques, it has built a strong foundation for the product - data warehouse. But, why should an organisation consider investing in a data warehouse? One of the prime reasons, for deploying a data warehouse is that, the data warehouse is a kingpin of business intelligence.

The data warehouses provide storage, functionality and responsiveness to queries, that is far superior to the capabilities of today's transaction-oriented databases. In many applications, users only need read-access to data, however, they need to access larger volume of data very rapidly – much more than what can be conveniently handled by traditional database systems. Often, such data is extracted from multiple operational databases. Since, most of these analyses performed do occur periodically, therefore, software developers and software vendors try to design systems to support these functions. Thus, there is a definite need for providing decision makers at middle management level and higher level with information as per the level of details to support decision-making. The data warehousing, online analytical processing (OLAP) and data mining technologies provide this functionality.

This unit covers the basic features of data warehousing and OLAP. Data Mining has been discussed in more details in unit 4 of this Block.

## 3.1 OBJECTIVES

After going through this unit, you should be able to:

- explain the term data warehouse;
- define key concepts surrounding data warehousing systems;



- compare database warehouse with operational information systems;
- discuss data warehousing architecture;
- identify the main stages in the life cycle of data warehousing, and
- discuss the concepts of OLAP, MOLAP, ROLAP.

## 3.2 WHAT IS DATA WAREHOUSING?

Let us first try to answer the question: What is a data warehouse? A simple answer could be: A data warehouse is a tool that manage of data *after and outside* of operational systems. Thus, they are not replacements for the operational systems but are major tools that acquires data from the operational system. Data warehousing technology has evolved in business applications for the process of strategic decision-making. Data warehouses may be sometimes considered as the key components of IT strategy and architecture of an organisation. We will give the more formal definition of data warehouse in the next paragraph.

A data warehouse as defined by **W.H. Inmon** is a *subject-oriented, integrated, nonvolatile, time-variant collection* of data that supports decision-making of the management. Data warehouses provide controlled access to data for complex analysis, knowledge discovery, and decision-making.

*Figure 1* presents some uses of data warehousing in various industries

| S.No. | Industry | Functional Areas of Use | Strategic Uses |
|---|---|---|---|
| 1 | Banking | Creating new schemes for loans and other banking products, helps in operations, identities information for marketing | Finding trends for customer service, service promotions, reduction of expenses. |
| 2 | Airline | Operations, marketing | Crew assignment, aircraft maintenance plans, fare determination, analysis of route profitability, frequent - flyer program design |
| 3 | Hospital | Operation optimisation | Reduction of operational expenses, scheduling of resources |
| 4 | Investment and Insurance | Insurance product development, marketing | Risk management, financial market analysis, customer tendencies analysis, portfolio management |

**Figure 1: Uses of Data Warehousing**

A data warehouse offers the following advantages:

- It provides historical information that can be used in many different forms of comparative and competitive analysis.
- It enhances the quality of the data and tries to make it complete.
- It can help in supporting disaster recovery although not alone but with other back up resources.

One of the major advantages a data warehouse offers is that it allows a large collection of historical data of many operational databases, which may be heterogeneous in nature, that can be analysed through one data warehouse interface, thus, it can be said to be a ONE STOP portal of historical information of an organisation. It can also be used in determining many trends through the use of data mining techniques.

Remember a data warehouse does not create value of its own in an organisation. However, the value can be generated by the users of the data of the data warehouse. For example, an electric billing company, by analysing data of a data warehouse can predict frauds and can reduce the cost of such determinations. In fact, this technology has such great potential that any company possessing proper analysis tools can benefit from it. Thus, a data warehouse supports Business Intelligence (that is), the technology that includes business models with objectives such as reducing operating costs, increasing profitability by improving productivity, sales, services and decision-making. Some of the basic questions that may be asked from a software that supports business intelligence include:

- What would be the income, expenses and profit for a year?
- What would be the sales amount this month?
- Who are the vendors for a product that is to be procured?
- How much of each product is manufactured in each production unit?
- How much is to be manufactured?
- What percentage of the product is defective?
- Are customers satisfied with the quality? etc.

Data warehouse supports various business intelligence applications. Some of these may be - online analytical processing (**OLAP**), decision-support systems (DSS), data mining etc. We shall be discussing these terms in more detail in the later sections.

A data warehouse has many characteristics. Let us define them in this section and explain some of these features in more details in the later sections.

## Characteristics of Data Warehouses

Data warehouses have the following important features:

1) **Multidimensional conceptual view:** A data warehouse contains data of many operational systems, thus, instead of a simple table it can be represented in multidimensional data form. We have discussed this concept in more detail in section 3.3.

2) **Unlimited dimensions and unrestricted cross-dimensional operations:** Since the data is available in multidimensional form, it requires a schema that is different from the relational schema. Two popular schemas for data warehouse are discussed in section 3.3.

3) **Dynamic sparse matrix handling:** This is a feature that is much needed as it contains huge amount of data.

4) **Client/server architecture:** This feature help a data warehouse to be accessed in a controlled environment by multiple users.

5) **Accessibility and transparency, intuitive data manipulation and consistent reporting performance:** This is one of the major features of the data warehouse. A Data warehouse contains, huge amounts of data, however, that should not be the reason for bad performance or bad user interfaces. Since the objectives of data warehouse are clear, therefore, it has to support the following

easy to use interfaces, strong data manipulation, support for applying and reporting of various analyses and user-friendly output.

## 3.3 THE DATA WAREHOUSE: COMPONENTS AND PROCESSES

A data warehouse is defined as *subject-oriented, integrated, nonvolatile, time-variant collection,* but how can we achieve such a collection? To answer this question, let us define the basic architecture that helps a data warehouse achieve the objectives as given/stated above. We shall also discuss the various processes that are performed by these components on the data.

### 3.3.1 The Basic Components of a Data Warehouse

A data warehouse basically consists of three components:

The Data Sources
The ETL and
The Schema of data of data warehouse including meta data.

*Figure 2* defines the basic architecture of a data warehouse. The analytical reports are not a part of the data warehouse but are one of the major business application areas including OLAP and DSS.

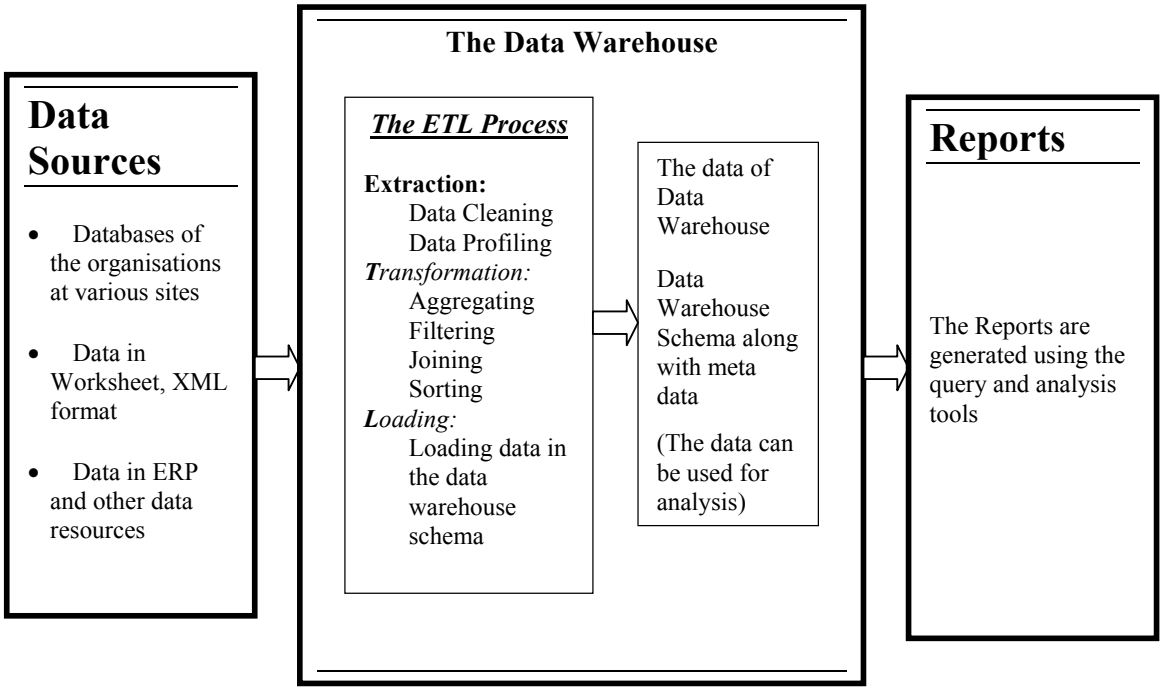



**Figure 2: The Data Warehouse Architecture**

**The Data Sources**

The data of the data warehouse can be obtained from many operational systems. A data warehouse interacts with the environment that provides most of the source data for the data warehouse. By the term environment, we mean, traditionally developed applications. In a large installation, hundreds or even thousands of these database systems or files based system exist with plenty of redundant data.

The warehouse database obtains most of its data from such different forms of legacy systems – files and databases. Data may also be sourced from external sources as well as other organisational systems, for example, an office system. This data needs to be integrated into the warehouse. But how do we integrate the data of these large numbers of operational systems to the data warehouse system? We need the help of ETL tools to do so. These tools capture the data that is required to be put in the data warehouse database. We shall discuss the ETL process in more detail in section 3.3.2.

**Data of Data Warehouse**

A data warehouse has an integrated, "subject-oriented", "time-variant" and "non-volatile" collection of data. The basic characteristics of the data of a data warehouse can be described in the following way:

**i) Integration:** Integration means bringing together data of multiple, dissimilar operational sources on the basis of an enterprise data model. The enterprise data model can be a basic template that identifies and defines the organisation's key data items uniquely. It also identifies the logical relationships between them ensuring organisation wide consistency in terms of:

***Data naming and definition:*** Standardising for example, on the naming of "student enrolment number" across systems.

***Encoding structures:*** Standardising on gender to be represented by "M" for male and "F" for female or that the first two digit of enrolment number would represent the year of admission.

***Measurement of variables:*** A Standard is adopted for data relating to some measurements, for example, all the units will be expressed in metric system or all monetary details will be given in Indian Rupees.

**ii) Subject Orientation:** The second characteristic of the data warehouse's data is that its design and structure can be oriented to important objects of the organisation. These objects such as STUDENT, PROGRAMME, REGIONAL CENTRES etc., are in contrast to its operational systems, which may be designed around applications and functions such as ADMISSION, EXAMINATION and RESULT DELCARATIONS (in the case of a University). Refer to *Figure 3.*

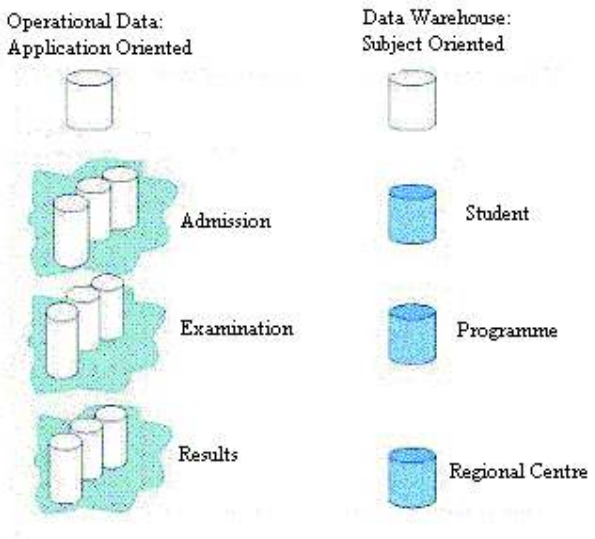



**Figure 3: Operations system data orientation vs. Data warehouse data orientation**

**iii) Time-Variance:** The third defining characteristic of the database of data warehouse is that it is time-variant, or historical in nature. The entire data in the data

warehouse is/was accurate at some point of time. This is, in contrast with operational data that changes over a shorter time period. The data warehouse's data contains data that is date-stamped, and which is historical data. *Figure 4* defines this characteristic of data warehouse.

| OPERATIONAL DATA | DATA WAREHOUSE DATA |
|---|---|
| • It is the current value data | • Contains a snapshot of historical data |
| • Time span of data = 60-90 days | • Time span of data = 5-10 years or more |
| • Data can be updated in most cases | • After making a snapshot the data record cannot be updated |
| • May or may not have a timestamp | • Will always have a timestamp |

**Figure 4: Time variance characteristics of a data of data warehouse and operational data**

**iv) Non-volatility (static nature) of data:** Data warehouse data is loaded on to the data warehouse database and is subsequently scanned and used, but is not updated in the same classical sense as operational system's data which is updated through the transaction processing cycles.

**Decision Support and Analysis Tools**

A data warehouse may support many OLAP and DSS tools. Such decision support applications would typically access the data warehouse database through a standard query language protocol; an example of such a language may be SQL. These applications may be of three categories: simple query and reporting, decision support systems and executive information systems. We will define them in more details in the later sections.

**Meta Data Directory**

The meta data directory component defines the repository of the information stored in the data warehouse. The meta data can be used by the general users as well as data administrators. It contains the following information:

i) the structure of the contents of the data warehouse database,
ii) the source of the data,
iii) the data transformation processing requirements, such that, data can be passed from the legacy systems into the data warehouse database,
iv) the process summarisation of data,
v) the data extraction history, and
vi) how the data needs to be extracted from the data warehouse.

Meta data has several roles to play and uses in the data warehouse system. For an end user, meta data directories also provide some additional information, such as what a particular data item would mean in business terms. It also identifies the information on reports, spreadsheets and queries related to the data of concern. All database management systems (DBMSs) have their own data dictionaries that serve a similar purpose. Information from the data dictionaries of the operational system forms a valuable source of information for the data warehouse's meta data directory.

### 3.3.2 Data Extraction, Transformation and Loading (ETL)



The first step in data warehousing is, to perform data extraction, transformation, and loading of data into the data warehouse. This is called ETL that is Extraction, Transformation, and Loading. ETL refers to the methods involved in accessing and manipulating data available in various sources and loading it into a target data warehouse. Initially the ETL was performed using SQL programs, however, now there are tools available for ETL processes. The manual ETL was complex as it required the creation of a complex code for extracting data from many sources. ETL tools are very powerful and offer many advantages over the manual ETL. ETL is a step-by-step process. As a first step, it maps the data structure of a source system to the structure in the target data warehousing system. In the second step, it cleans up the data using the process of data transformation and finally, it loads the data into the target system.

**What happens during the ETL Process?**

The ETL is three-stage process. During the *Extraction* phase the desired data is identified and extracted from many different sources. These sources may be different databases or non-databases. Sometimes when it is difficult to identify the desirable data then, more data than necessary is extracted. This is followed by the identification of the relevant data from the extracted data. The process of extraction sometimes, may involve some basic transformation. For example, if the data is being extracted from two Sales databases where the sales in one of the databases is in Dollars and in the other in Rupees, then, simple transformation would be required in the data. The size of the extracted data may vary from several hundreds of kilobytes to hundreds of gigabytes, depending on the data sources and business systems. Even the time frame for the extracted data may vary, that is, in some data warehouses, data extraction may take a few days or hours to a real time data update. For example, a situation where the volume of extracted data even in real time may be very high is a web server.

The *extraction* process involves data cleansing and data profiling. Data cleansing can be defined as the process of removal of inconsistencies among the data. For example, the state name may be written in many ways also they can be misspelt too. For example, the state Uttar Pradesh may be written as U.P., UP, Uttar Pradesh, Utter Pradesh etc. The cleansing process may try to correct the spellings as well as resolve such inconsistencies. But how does the cleansing process do that? One simple way may be, to create a Database of the States with some possible fuzzy matching algorithms that may map various variants into one state name. Thus, cleansing the data to a great extent. Data profiling involves creating the necessary data from the point of view of data warehouse application. Another concern here is to eliminate duplicate data. For example, an address list collected from different sources may be merged as well as purged to create an address profile with no duplicate data.

One of the most time-consuming tasks - data *transformation* and *loading* follows the extraction stage. This process includes the following:

- Use of data filters,
- Data validation against the existing data,
- Checking of data duplication, and
- Information aggregation.

*Transformations* are useful for transforming the source data according to the requirements of the data warehouse. The process of transformation should ensure the quality of the data that needs to be loaded into the target data warehouse. Some of the common transformations are:



**Filter Transformation:** Filter transformations are used to filter the rows in a mapping that do not meet specific conditions. For example, the list of employees of the Sales department who made sales above Rs.50,000/- may be filtered out.

**Joiner Transformation:** This transformation is used to join the data of one or more different tables that may be stored on two different locations and could belong to two different sources of data that may be relational or from any other sources like XML data.

**Aggregator Transformation:** Such transformations perform aggregate calculations on the extracted data. Some such calculations may be to find the sum or average.

**Sorting Transformation:** requires creating an order in the required data, based on the application requirements of the data warehouse.

Once the data of the data warehouse is properly extracted and transformed, it is *loaded* into a data warehouse. This process requires the creation and execution of programs that perform this task. One of the key concerns here is to propagate updates. Some times, this problem is equated to the problem of maintenance of the materialised views.

When should we perform the ETL process for data warehouse? ETL process should normally be performed during the night or at such times when the load on the operational systems is low. **Please note** that, the integrity of the extracted data can be ensured by synchronising the different operational applications feeding the data warehouse and the data of the data warehouse.

### ☞ Check Your Progress 1

1) What is a Data Warehouse?

……

……

……

2) What is ETL? What are the different transformations that are needed during the ETL process?

……

……

……

3) What are the important characteristics of Data Warehousing?

……

……

……

4) Name the component that comprise the data warehouse architecture?

……

……

……

# 3.4 MULTIDIMENSIONAL DATA MODELING FOR DATA WAREHOUSING

A data warehouse is a huge collection of data. Such data may involve grouping of data on multiple attributes. For example, the enrolment data of the students of a University may be represented using a student schema such as:

**Student_enrolment (year, programme, region, number)**

Here, some typical data value may be (These values are shown in *Figure 5* also. Although, in an actual situation almost all the values will be filled up):

- In the year 2002, BCA enrolment at Region (Regional Centre Code) RC-07 (Delhi) was 350.
- In year 2003 BCA enrolment at Region RC-07 was 500.
- In year 2002 MCA enrolment at all the regions was 8000.

**Please note that,** to define the student number here, we need to refer to three attributes: the year, programme and the region. Each of these attributes is identified as the dimension attributes. Thus, the data of student_enrolment table can be modeled using dimension attributes (year, programme, region) and a measure attribute (number). Such kind of data is referred to as a Multidimensional data. Thus, a data warehouse may use multidimensional matrices referred to as a data cubes model. The multidimensional data of a corporate data warehouse, for example, would have the fiscal period, product and branch dimensions. If the dimensions of the matrix are greater than three, then it is called a hypercube. Query performance in multidimensional matrices that lend themselves to dimensional formatting can be much better than that of the relational data model.

The following figure represents the Multidimensional data of a University:

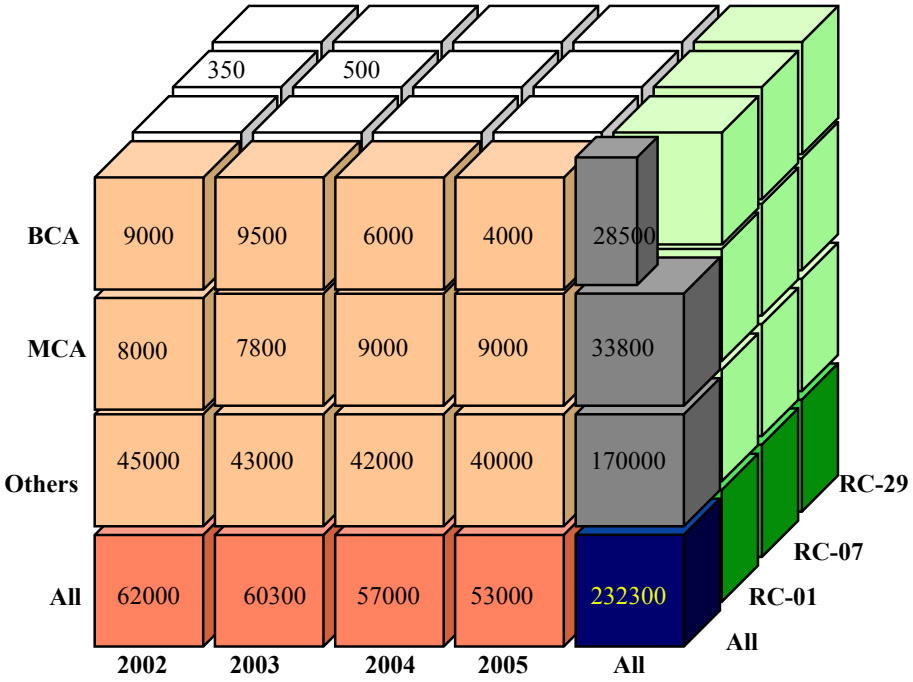



**Figure 5: A sample multidimensional data**



Multidimensional data may be a little difficult to analyse. Therefore, Multidimensional data may be displayed on a certain pivot, for example, consider the following table:

| **Region: ALL THE REGIONS** | | | | |
|---|---|---|---|---|
| | **BCA** | **MCA** | **Others** | **All the Programmes** |
| 2002 | 9000 | 8000 | 45000 | 62000 |
| 2003 | 9500 | 7800 | 43000 | 60300 |
| 2004 | 6000 | 9000 | 42000 | 57000 |
| 2005 | 4000 | 9000 | 40000 | 53000 |
| **ALL the Years** | **28500** | **33800** | **170000** | **232300** |

The table given above, shows, the multidimensional data in *cross-tabulation*. This is also referred to as a *pivot-table*. **Please note that** cross-tabulation is done on any two dimensions keeping the other dimensions fixed as ALL. For example, the table above has two dimensions Year and Programme, the third dimension Region has a fixed value ALL for the given table.

**Please note that,** the cross-tabulation as we have shown in the table above is, different to a relation. The relational representation for the data of the table above may be:

**Table: Relational form for the Cross table as above**

| Year | Programme | Region | Number |
|---|---|---|---|
| 2002 | BCA | All | 9000 |
| 2002 | MCA | All | 8000 |
| 2002 | Others | All | 45000 |
| 2002 | All | All | 62000 |
| 2003 | BCA | All | 9500 |
| 2003 | MCA | All | 7800 |
| 2003 | Others | All | 43000 |
| 2003 | All | All | 60300 |
| 2004 | BCA | All | 6000 |
| 2004 | MCA | All | 9000 |
| 2004 | Others | All | 42000 |
| 2004 | All | All | 57000 |
| 2005 | BCA | All | 4000 |
| 2005 | MCA | All | 9000 |
| 2005 | Others | All | 40000 |
| 2005 | All | All | 53000 |
| All | BCA | All | 28500 |
| All | MCA | All | 33800 |
| All | Others | All | 170000 |
| All | All | All | 232300 |

A cross tabulation can be performed on any two dimensions. The operation of changing the dimensions in a cross tabulation is termed as pivoting. In case a cross tabulation is done for a value other than ALL for the fixed third dimension, then it is called *slicing*. For example, a slice can be created for Region code RC-07 instead of ALL the regions in the cross tabulation of regions. This operation is called *dicing* if values of multiple dimensions are fixed.

Multidimensional data allows data to be displayed at various level of granularity. An operation that converts data with a fine granularity to coarse granularity using aggregation is, termed *rollup* operation. For example, creating the cross tabulation for All regions is a rollup operation. On the other hand an operation that moves from a coarse granularity to fine granularity is known as *drill down* operation. For example, moving from the cross tabulation on All regions back to Multidimensional data is a drill down operation. **Please note:** For the drill down operation, we need, the original data or any finer granular data.

Now, the question is, how can multidimensional data be represented in a data warehouse? or, more formally, what is the schema for multidimensional data?

Two common multidimensional schemas are the star schema and the snowflake schema. Let us, describe these two schemas in more detail. A multidimensional storage model contains two types of tables: the dimension tables and the fact table. The dimension tables have tuples of dimension attributes, whereas the fact tables have one tuple each for a recorded fact. In order to relate a fact to a dimension, we may have to use pointers. Let us demonstrate this with the help of an example. Consider the University data warehouse where one of the data tables is the **Student enrolment table.** The three dimensions in such a case would be:

- Year
- Programme, and
- Region

The star schema for such a data is shown in *Figure* 6.

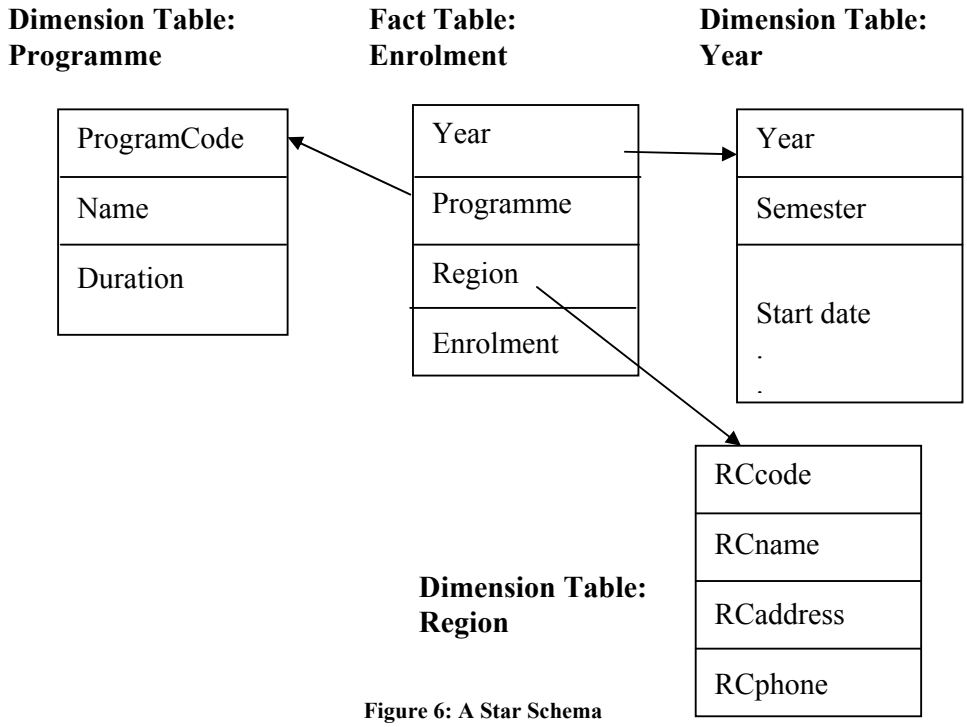



**Figure 6: A Star Schema**

**Please note** that in *Figure 6,* the fact table points to different dimension tables, thus, ensuring the reliability of the data. **Please notice that,** each Dimension table is a table for a single dimension only and that is why this schema is known as a star schema. However, a dimension table may not be normalised. Thus, a new schema named the snowflake schema was created. A snowflake schema has normalised but hierarchical dimensional tables. For example, consider the star schema shown in *Figure* 6, if in the Region dimension table, the value of the field Rcphone is multivalued, then the Region dimension table is not normalised.

Thus, we can create a snowflake schema for such a situation as:

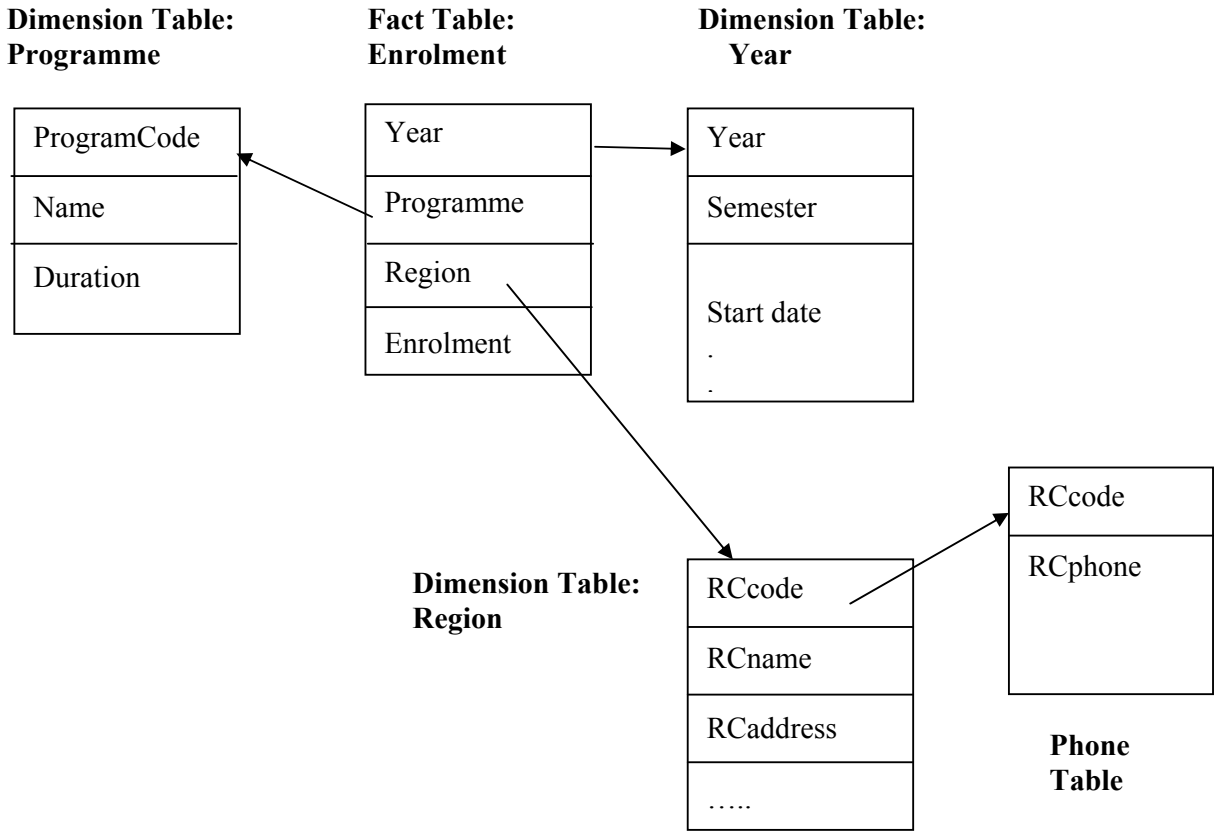



**Figure 7: Snowflake Schema**

Data warehouse storage can also utilise indexing to support high performance access. Dimensional data can be indexed in star schema to tuples in the fact table by using a join index. Data warehouse storage facilitates access to summary data due to the non-volatile nature of the data.

## 3.5 BUSINESS INTELLIGENCE AND DATA WAREHOUSING

A data warehouse is an integrated collection of data and can help the process of making better business decisions. Several tools and methods are available to that enhances advantage of the data of data warehouse to create information and knowledge that supports business decisions. Two such techniques are Decision-support systems and online analytical processing. Let us discuss these two in more details in this section.

### 3.5.1 Decision Support Systems (DSS)



The DSS is a decision support system and NOT a decision-making system. DSS is a specific class of computerised information systems that support the decision-making activities of an organisation. A properly designed DSS is an *interactive* software based system that helps decision makers to compile useful information from raw data, documents, personal knowledge, and/or business models to identify and solve problems and make decisions.

A decision support system may gather or present the following information:

- Accessing current information from data warehouse or legacy databases or other data resources.
- Presenting comparative sales figures of several months for an organisation.
- Creating projected revenue details based on new product sales assumptions.
- Demonstrating the consequences of different decision alternatives based on past experiences.

The DSS assists users in evaluating appropriate analysis or performing different types of studies on the datasets. For example, a spreadsheet can be used to store answers to a series of questionnaires in the form of Excel spreadsheets. This information then, can be passed on to decision makers. More specifically, the feedback data collected on a programme like CIC may be given to subject matter experts for making decisions on the quality, improvement, and revision of that programme. The DSS approach provides a self-assessment weighing tool to facilitate the determining of the value of different types of quality and quantity attributes. Decision support systems are sometimes also referred to as the Executive Information Systems (EIS).

**Executive Information System (EIS):** Executive information systems (EIS) are created for purpose of providing executives with the information they require to run their businesses. An EIS is intended to facilitate and support information and decision-making at senior executives level by, providing easy access to both *internal and external* information. Of course, this information should be relevant and should help them in establishing and meeting the strategic goals of the organisation.

The emphasis of DSS/EIS is mainly on graphical displays and easy-to-use user interfaces as they are there chiefly, to provide help. They offer strong reporting and drill-down capabilities. In general, EIS are enterprise-wide DSS that help top-level executives analyse, compare, and bring to light trends in important market/operational variables so that, they can monitor performance and identify opportunities and future problems. EIS and data warehousing technologies converge are convergent.

The concept of providing information to the executive management is not a new concept except for the ease with which they can get it. Given that top management has succeeded in acquiring the information till date, they can run their business without direct access to computer-based information systems. So why does one need a DSS/EIS? Well, there are a number of factors in support of DSS/EIS. These seem to be more managerial in nature. Some of these factors are:

- The first factor is a strange but true ‘pull’ factor, that is, executives are suggested to be more computer-literate and willing to become direct users of computer systems. For example, a survey suggests that more than twenty percent of senior executives have computers on their desks but rarely 5% use the system, although there are wide variations in the estimates yet, there is a define pull towards this simple easy to use technology.
- The other factor may be the increased use of computers at the executive level. For example, it has been suggested that middle managers who have been directly using computers in their daily work are being promoted to the executive level.



This new breed of executives do not exhibit the fear of computer technology that has characterised executive management up to now and are quite willing to be direct users of computer technology.

- The last factor is more on the side of technology. Technology is gradually becoming extremely simple to use from the end users point of view and it is now finding more users attracted towards it.

### 3.5.2 Online Analytical Processing (OLAP)

Data warehouses are not suitably designed for transaction processing, however, they support increased efficiency in query processing. Therefore, a data warehouse is a very useful support for the analysis of data. But are there any such tools that can utilise the data warehouse to extract useful analytical information?

**On Line Analytical Processing (OLAP)** is an approach for performing analytical queries and statistical analysis of multidimensional data. OLAP tools can be put in the category of business intelligence tools along with data mining. Some of the typical applications of OLAP may include reporting of sales projections, judging the performance of a business, budgeting and forecasting etc.

OLAP tools require multidimensional data and distributed query-processing capabilities. Thus, OLAP has data warehouse as its major source of information and query processing. But how do OLAP tools work?

In an OLAP system a data analyst would like to see different cross tabulations by interactively selecting the required attributes. Thus, the queries in an OLAP are expected to be executed extremely quickly. The basic data model that may be supported by OLAP is the star schema, whereas, the OLAP tool may be compatible to a data warehouse.

Let us, try to give an example on how OLAP is more suitable to a data warehouse rather than to a relational database. An OLAP creates an aggregation of information, for example, the sales figures of a sales person can be grouped (aggregated) for a product and a period. This data can also be grouped for sales projection of the sales person over the regions (North, South) or states or cities. Thus, producing enormous amount of aggregated data. If we use a relational database, we would be generating such data many times. However, this data has many dimensions so it is an ideal candidate for representation through a data warehouse. The OLAP tool thus, can be used directly on the data of the data warehouse to answer many analytical queries in a short time span. The term OLAP is sometimes confused with OLTP. OLTP is online transaction processing. OLTP systems focus on highly concurrent transactions and better commit protocols that support high rate of update transactions. On the other hand, OLAP focuses on good query-evaluation and query-optimisation algorithms.

**OLAP Implementation**

This classical form of OLAP implementation uses multidimensional arrays in the memory to store multidimensional data. Such implementation of OLAP is also referred to as Multidimensional OLAP (MOLAP). MOLAP is faster as it stores data in an already processed aggregated data form using dimension and fact tables. The other important type of OLAP implementation is Relational OLAP (ROLAP), which stores data directly in the relational databases. ROLAP creates multidimensional views upon request rather than in advance as in MOLAP. ROLAP may be used on complex data with a wide number of fields.

## 3.6 BUILDING OF DATA WAREHOUSE

The first basic issue for building a data warehouse is to identify the USE of the data warehouse. It should include information on the expected outcomes of the design. A good data warehouse must support meaningful query facility on the attributes of dimensional and fact tables. A data warehouse design in addition to the design of the schema of the database has to address the following three issues:

- How will the data be acquired?
- How will the data be stored?
- What would be the environment of the data warehouse?

Some of the key concerns of the issues above are:

**Data Acquisition:** A data warehouse must acquire data so that it can fulfil the required objectives. Some of the key issues for data acquisition are:

- Whether the data is to be extracted from multiple, heterogeneous sources? The location of these sources and the kind of data they contain?
- The method of acquiring the data contained in the various sources in a standard data warehouse schema. Remember, you must have consistent data in a data warehouse.
- How will the data be cleaned so that its validity can be ensured?
- How is the data going to be transformed and converted into the data warehouse multidimensional schema model?
- How will the data be loaded in the warehouse. After all, the data is huge and the amount of time the loading will take needs to be ascertained? Here, we need to find the time required for data cleaning, formatting, transmitting, creating additional indexes etc. and also the issues related to data consistency such as, the currency of data, data integrity in multidimensional space, etc.

**Data storage:** The data acquired by the data is also to be stored as per the storage schema. This data should be easily accessible and should fulfil the query needs of the users efficiently. Thus, designers need to ensure that there are appropriate indexes or paths that allow suitable data access. Data storage must be updated as more data is acquired by the data warehouse, but it should still provide access to data during this time. Data storage also needs to address the issue of refreshing a part of the data of the data warehouse and purging data from the data warehouse.

**Environment of the data warehouse:** Data warehouse designers should also keep in mind the data warehouse environment considerations. The designers must find the expected use of the data and predict if it is consistent with the schema design. Another key issue here would be the design of meta data directory component of the data warehouse. The design should be such that it should be maintainable under the environmental changes.

## DATA WAREHOUSING LIFE CYCLE

The data warehouse technologies use very diverse vocabulary. Although the vocabulary of data warehouse may vary for different organisations, the data warehousing industry is in agreement with the fact that the data warehouse lifecycle model fundamentally can be defined as the model consisting of five major phases – design, prototype, deploy, operation and enhancement.

Let us introduce these terms:

1) **Design**: The design of database is to be done for available data inventories, DSS analyst requirements and analytical needs. It needs to produce a robust star schema or snowflake schema. Key activities in the design phase may include



communication with the end users, finding the available catalogues, defining key performance and quality indicators, mapping of decision-making processes as per the information needs at various end user levels, logical and physical schema design etc.

2) **Prototype**: A data warehouse is a high cost project, thus, it may be a good idea to deploy it partially for a select group of decision-makers and database practitioners in the end user communities. This will help in developing a system that will be easy to accept and will be mostly as per the user's requirements.

3) **Deploy**: Once the prototype is approved, then the data warehouse can be put to actual use. A deployed data warehouse comes with the following processes; documentation, training, maintenance.

4) **Operation**: Once deployed the data warehouse is to be used for day-to-day operations. The operation in data warehouse includes extracting data, putting it in database and output of information by DSS.

5) **Enhancement**: These are needed with the updating of technology, operating processes, schema improvements etc. to accommodate the change.

**Please note** you can apply any software life cycle model on the warehouse life cycle.

**Data Warehouse Implementation**

After the design of the data warehouse, the next step for building the data warehouse may be its implementation. Please remember that implementing a data warehouse is a very challenging process. It tests the ability of an organisation to adjust to change. The implementation of the data warehouse may require the following stages:

**Implementation of Data Model:** The data model that is to be implemented should be checked to ensure that it has the key entities and their interrelationships. It also should see that the system records of the data warehouse must be as per the data warehouse data model and should be possible best matches for the operational system data. The physical design should support the schema design.

**Implementation of Data Transformation Programs:** Now, the transformation programs that will extract and transform the data from the system of record should be implemented. They should be tested to load the required data in the data warehouse database.

**Populating Data Warehouse and Maintenance:** Once the data transformation programs are found to be ok, they can be used to populate the data warehouse. Once the data warehouse is operation at it needs to be maintained properly.

**Some general Issues for Warehouse Design and Implementation**

The programs created during the previous phase are executed to populate the data warehouse's database.

**The Development and Implementation Team:** A core team for such implementation may be:

***A Project Leader*** responsible for managing the overall project and the one who helps in obtaining resources and participates in the design sessions.
***Analysts*** documents the end user requirements and creates the enterprise data models for the data warehouse.

***A Data Base Administrator*** is responsible for the physical data base creation, and

***Programmers*** responsible for programming the data extraction and transformation programs and end user access applications.

**Training:** Training will be required not only for end users, once the data warehouse is in place, but also for various team members during the development stages of the data warehouse.

### ☞ Check Your Progress 2

1) What is a dimension, how is it different from a fact table?

……

……

……

2) How is snowflake schema different from other schemes?

……

……

……

3) What are the key concerns when building a data warehouse?

……

……

……

4) What are the major issues related to data warehouse implementation?

……

……

……

5) Define the terms: DSS and ESS.

……

……

……

6) What are OLAP, MOLAP and ROLAP?

……

……

……

## 3.7 DATA MARTS

Data marts can be considered as the database or collection of databases that are designed to help managers in making strategic decisions about business and the organisation. Data marts are usually smaller than data warehouse as they focus on some subject or a department of an organisation (a data warehouses combines databases across an entire enterprise). Some data marts are also called dependent data marts and may be the subsets of larger data warehouses.

A data mart is like a data warehouse and contains operational data that helps in making strategic decisions in an organisation. The only difference between the two is



that data marts are created for a certain limited predefined application. Even in a data mart, the data is huge and from several operational systems, therefore, they also need a multinational data model. In fact, the star schema is also one of the popular schema choices for a data mart.

A dependent data mart can be considered to be a logical subset (view) or a physical subset (extraction) of a large data warehouse. A dependent data mart may be isolated for the following reasons.

(i) For making a separate schema for OLAP or any other similar system.

(ii) To put a portion of the data warehouse or a separate machine to enhance performance.

(iii) To create a highly secure subset of data warehouse.

In fact, to standardise data analysis and usage patterns, data warehouses are generally organised as task specific small units the data marts. The data organisation of a data mart is a very simple star schema. For example, the university data warehouse that we discussed in section 3.4 can actually be a data mart on the problem "The prediction of student enrolments for the next year." A simple data mart may extract its contents directly from operational databases. However, in complex multilevel data warehouse architectures the data mart content may be loaded with the help of the warehouse database and Meta data directories.

## 3.8 DATA WAREHOUSE AND VIEWS

Many database developers classify data warehouse as an extension of a view mechanism. If that is the case, then how do these two mechanisms differ from one another? For, after all even in a database warehouse, a view can be materialised for the purpose of query optimisation. A data warehouse may differ from a view in the following ways:

- A data warehouse has a multi-dimensional schema and tries to integrate data through fact-dimension star schema, whereas views on the other hand are relational in nature.

- Data warehouse extracts and transforms and then stores the data into its schema; however, views are only logical and may not be materialised.

- You can apply mechanisms for data access in an enhanced way in a data warehouse, however, that is not the case for a view.

- Data warehouse data is time-stamped, may be differentiated from older versions, thus, it can represent historical data. Views on the other hand are dependent on the underlying DBMS.

- Data warehouse can provide extended decision support functionality, views normally do not do it automatically unless, an application is designed for it.

## 3.9 THE FUTURE: OPEN ISSUE FOR DATA WAREHOUSE

The administration of a data warehouse is a complex and challenging task. Some of the open issues for data warehouse may be:

- Quality control of data despite having filtration of data.
- Use of heterogeneous data origins is still a major problem for data extraction and transformation.
- During the lifetime of the data warehouse it will change, hence, management is one of the key issues here.
- Data warehouse administration is a very wide area and requires diverse skills, thus, people need to be suitably trained.
- Managing the resources of a data warehouse would require a large distributed team.
- The key research areas in data warehouse is, data cleaning, indexing, view creation, queries optimisation etc.

However, data warehouses are still an expensive solution and are typically found in large firms. The development of a central warehouse is capital intensive with high risks. Thus, at present data marts may be a better choice.

☞ **Check Your Progress 3**

1) How is data mart different from data warehouse?

……

……

……

2) How does data warehouse differ from materialised views?

……

……

……

## 3.10 SUMMARY

This unit provided an introduction to the concepts of data warehousing systems. The data warehouse is a technology that collects operational data from several operational systems, refines it and stores it in its own multidimensional model such as star schema or snowflake schema. The data of a data warehouse can be indexed and can be used for analyses through various DSS and EIS. The architecture of data warehouse supports contains – an interface that interact with operational system, transformation processing, database, middleware and DSS interface at the other end. However, data warehouse architecture is incomplete if, it does not have meta data directory which is extremely useful for each and every step of the data warehouse. The life cycle of a data warehouse has several stages for designing, prototyping, deploying and maintenance. The database warehouse's life cycle, however, can be clubbed with SDLC. Data mart is a smaller version of a data warehouse designed for a specific purpose. Data warehouse is quite different from views. A data warehouse is complex and offers many challenges and open issues, but, in the future data warehouses will be-extremely important technology that will be deployed for DSS. Please go through further readings for more details on data warehouse.



# 3.11 SOLUTIONS/ANSWERS

## Check Your Progress 1

1) A Data Warehouse can be considered to be a "corporate memory". It is a repository of processed but integrated information that can be used for queries and analysis. Data and information are extracted from heterogeneous sources as they are generated. Academically, it is subject – oriented, time-variant, and a collection of operational data.

   Relational databases are designed in general, for on-line transactional processing (OLTP) and do not meet the requirements for effective on-line analytical processing (OLAP). The data warehouses are designed differently from relational databases and are suitable for OLAP.

2). ETL is Extraction, transformation, and loading. ETL refers to the methods involved in accessing and manipulating data available in various sources and loading it into target data warehouse. The following are some of the transformations that may be used during ETL:

   - Filter Transformation
   - Joiner Transformation
   - Aggregator transformation
   - Sorting transformation.

3) Some important characteristics of Data Warehousing are

   i) Multidimensional view
   ii) Unlimited dimensions and aggregation levels and unrestricted cross-dimensional operations.
   iii) Dynamic sparse matrix handling
   iv) Client/server architecture
   v) Accessibility and transparency, intuitive data manipulation and consistent reporting performance.

4) The data warehouse architecture consists of six distinct components that include:

   i) Operational systems
   ii) Transformation processing
   iii) Database
   iv) Middleware
   v) Decision support and presentation processing and
   vi) Meta data directory.

## Check Your Progress 2

1) A dimension may be equated with an object. For example, in a sales organisation, the dimensions may be salesperson, product and period of a quarterly information. Each of these is a dimension. The fact table will represent the fact relating to the dimensions. For the dimensions as above, a fact table may include sale (in rupees) made by a typical sales person for the specific product for a specific period. This will be an actual date, thus is a fact. A fact, thus, represents an aggregation of relational data on the dimensions.

2) The primary difference lies in representing a normalised dimensional table.

3) 
- How will the data be acquired?
- How will it be stored?
- The type of environment in which the data warehouse will be implemented?

4) 
- Creation of proper transformation programs
- Proper training of development team
- Training of data warehouse administrator and end users
- Data warehouse maintenance.

5) The DSS is a decision support system and not a decision-making system. It is a specific class of information system that supports business and organisational decision-making activities. A DSS is an interactive software-based system that helps decision makers compile useful information from raw data or documents, personal knowledge, etc. This information helps these decision makers to identify and solve problems and take decisions.

An Executive Information System (EIS) facilitates the information and decision making needs of senior executives. They provide easy access to relevant information (both internal as well as external), towards meeting the strategic goals of the organisation. These are the same as for the DSS.

6) OLAP refers to the statistical processing of multidimensional such that the results may be used for decision-making. MOLAP and ROLAP are the two implementations of the OLAP. In MOLAP the data is stored in the multidimensional form in the memory whereas in the ROLAP it is stored in the relational database form.

## Check Your Progress 3

1) The basic constructs used to design a data warehouse and a data mart are the same. However, a Data Warehouse is designed for the enterprise level, while Data Marts may be designed for a business division/department level. A data mart contains the required subject specific data for local analysis only.

2) The difference may be:

- Data warehouse has a multi-dimensional schema whereas views are relational in nature.
- Data warehouse extracts and transforms and then stores the data into its schema that is not true for the materialised views.
- Materialised views needs to be upgraded on any update, whereas, a data warehouse does not need updation.
- Data warehouse data is time-stamped, thus, can be differentiated from older versions that is not true for the materialised views.

# UNIT 4 INTRODUCTION TO DATA MINING

**Structure** **Page Nos.**


4.0 Introduction 80
4.1 Objectives 80
4.2 Data Mining Technology 81
- 4.2.1 Data, Information, Knowledge
- 4.2.2 Sample Data Mining Problems
- 4.2.3 Database Processing vs. Data Mining Processing
- 4.2.4 Data Mining vs KDD

4.3 Approaches to Data Mining Problems 84
4.4 Classification 85
- 4.4.1 Classification Approach
- 4.4.2 Classification Using Distance (K-Nearest Neighbours)
- 4.4.3 Decision or Classification Tree
- 4.4.4 Bayesian Classification

4.5 Clustering 93
- 4.5.1 Partitioning Clustering
- 4.5.2 Nearest Neighbours Clustering
- 4.5.2 Hierarchical Clustering

4.6 Association Rule Mining 96
4.7 Applications of Data Mining Problem 99
4.8 Commercial Tools of Data Mining 100
4.9 Summary 102
4.10 Solutions/Answers 102
4.11 Further Readings 103


## 4.0 INTRODUCTION

Data mining is emerging as a rapidly growing interdisciplinary field that takes its approach from different areas like, databases, statistics, artificial intelligence and data structures in order to extract hidden knowledge from large volumes of data. The data mining concept is now a days not only used by the research community but also a lot of companies are using it for predictions so that, they can compete and stay ahead of their competitors.

With rapid computerisation in the past two decades, almost all organisations have collected huge amounts of data in their databases. These organisations need to understand their data and also want to discover useful knowledge as patterns, from their existing data.

This unit aims at giving you some of the fundamental techniques used in data mining. This unit emphasises on a brief overview of data mining as well as the application of data mining techniques to the real world. We will only consider structured data as input in this unit. We will emphasise on three techniques of data mining:

(a) Classification,
(b) Clustering, and
(c) Association rules.

## 4.1 OBJECTIVES

After going through this unit, you should be able to:

- explain what is data mining;
- explain how data mining is applied in the real world;
- define the different approaches to data mining;
- use the classification approach in data mining;

- use the clustering approach;
- explain how association rules are used in data mining, and
- identify some of the leading data mining tools.

## 4.2 DATA MINING TECHNOLOGY

Data is growing at a phenomenal rate today and the users expect more sophisticated information from this data. There is need for new techniques and tools that can automatically generate useful information and knowledge from large volumes of data. Data mining is one such technique of generating hidden information from the data. Data mining can be defined as: "an automatic process of extraction of non-trivial or implicit or previously unknown but potentially useful information or patterns from data in large databases, data warehouses or in flat files".

Data mining is related to data warehouse in this respect that, a data warehouse is well equipped for providing data as input for the data mining process. The advantages of using the data of data warehouse for data mining are or many some of them are listed below:

- Data quality and consistency are essential for data mining, to ensure, the accuracy of the predictive models. In data warehouses, before loading the data, it is first extracted, cleaned and transformed. We will get good results only if we have good quality data.
- Data warehouse consists of data from multiple sources. The data in data warehouses is integrated and subject oriented data. The data mining process performed on this data.
- In data mining, it may be the case that, the required data may be aggregated or summarised data. This is already there in the data warehouse.
- Data warehouse provides the capability of analysing data by using OLAP operations. Thus, the results of a data mining study can be analysed for hirtherto, uncovered patterns.

As defined earlier, data mining generates potentially useful information or patterns from data. In fact, the information generated through data mining can be used to create knowledge. So let us, first, define the three terms data, information and knowledge.

### 4.2.1 Data, Information, Knowledge

Before going into details on data mining, let us, first, try to discuss the differences between data, information and knowledge.

1. **Data (Symbols):** It simply exists. It has no significance beyond its existence. It is raw information. For example, "it is raining".
2. **Information:** Information is the processed data. It provides answer to "who", "what", "where", and "when" questions. For example, "The temperature dropped 12 degrees centigrade and then it started raining" is an example of information.
3. **Knowledge:** Knowledge is the application of data and information and it answers the "how" questions. This is not explicit in the database - it is implicit. For example "If humidity is very high and the temperature drops suddenly, then, the atmosphere is often unlikely to be able to hold the moisture so, it rains', is an example of knowledge.



### 4.2.2 Sample Data Mining Problems

Now that we have defined data, information and knowledge let us define some of the problems that can be solved through the data mining process.

a) Mr Ramniwas Gupta manages a supermarket and the cash counters, he adds transactions into the database. Some of the questions that can come to Mr. Gupta's mind are as follows:

   a) Can you help me visualise my sales?
   b) Can you profile my customers?
   c) Tell me something interesting about sales such as, what time sales will be maximum etc.

   He does not know statistics, and he does not want to hire statisticians.

   The answer of some of the above questions may be answered by data mining.

b) Mr. Avinash Arun is an astronomer and the sky survey has 3 tera-bytes ($10^{12}$) of data, 2 billion objects. Some of the questions that can come to the mind of Mr. Arun are as follows:

   a) Can you help me recognise the objects?
   b) Most of the data is beyond my reach. Can you find new/unusual items in my data?
   c) Can you help me with basic manipulation, so I can focus on the basic science of astronomy?

   He knows the data and statistics, but that is not enough. The answer to some of the above questions may be answered once again, by data mining.

**Please note:** The use of data mining in both the questions given above lies in finding certain patterns and information. Definitely the type of the data in both the database as given above will be quite different.

### 4.2.3 Database Processing Vs. Data Mining Processing

Let us, first, differentiate between database processing and data mining processing: The query language of database processing is well defined and it uses SQL for this, while, the data mining, the query is poorly defined and there is no precise query language. The data used in data processing is operational data, while, in data mining, it is historical data i.e., it is not operational data.

The output of the query of database processing is precise and is the subset of the data, while, in the case of data mining the output is fuzzy and it is not a subset of the data.

Some of the examples of database queries are as follows:

- Find all credit card applicants with the last name Ram.
- Identify customers who have made purchases of more than Rs.10,000/- in the last month.
- Find all customers who have purchased shirt(s).

Some data mining queries may be:

- Find all credit card applicants with poor or good credit risks.
- Identify the profile of customers with similar buying habits.
- Find all items that are frequently purchased with shirt (s).

### 4.2.4 Data Mining Vs. Knowledge Discovery in Databases (KDD)

Knowledge Discovery in Databases (KDD) is the process of finding useful information, knowledge and patterns in data while data mining is the process of using of algorithms to automatically extract desired information and patterns, which are derived by the Knowledge Discovery in Databases process. Let us define KDD in more details.

#### Knowledge Discovery in Databases (KDD) Process

The different steps of KDD are as follows:

- **Extraction:** Obtains data from various data sources.
- **Preprocessing:** It includes cleansing the data which has already been extracted by the above step.
- **Transformation:** The data is converted in to a common format, by applying some technique.
- **Data Mining:** Automatically extracts the information/patterns/knowledge.
- **Interpretation/Evaluation:** Presents the results obtained through data mining to the users, in easily understandable and meaningful format.

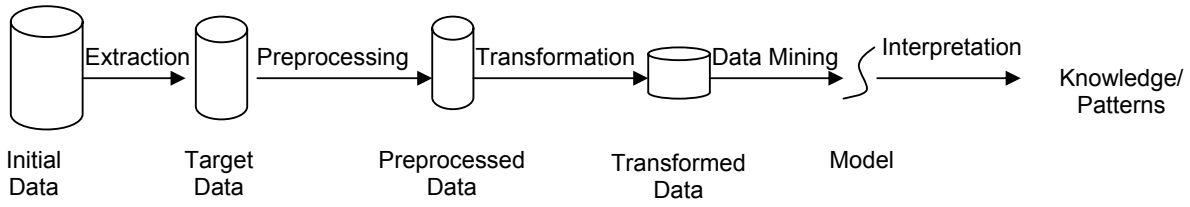



**Figure 1: KDD process**

#### Tasks in Knowledge Discovery in Databases (KDD) Process

The different tasks in KDD are as follows:

- **Obtains information on application domain**: It gathers knowledge from the domain relevant to the user.
- **Extracting data set**: It includes extracting required data which will later, be used for analysis.
- **Data cleansing process**: It involves basic operations such as, the removal of noise, collecting necessary information to from noisy data, such as, deciding on strategies for handling missing data fields.
- **Data reduction and projection**: Using dimensionality reduction or transformation methods it reduces the effective number of dimensions under consideration.
- **Selecting data mining task**: In this stage we decide what the objective of the KDD process is. Whether it is classification, clustering, association rules etc.
- **Selecting data mining method**: In this stage, we decide the methods and the parameter to be used for searching for desired patterns in the data.

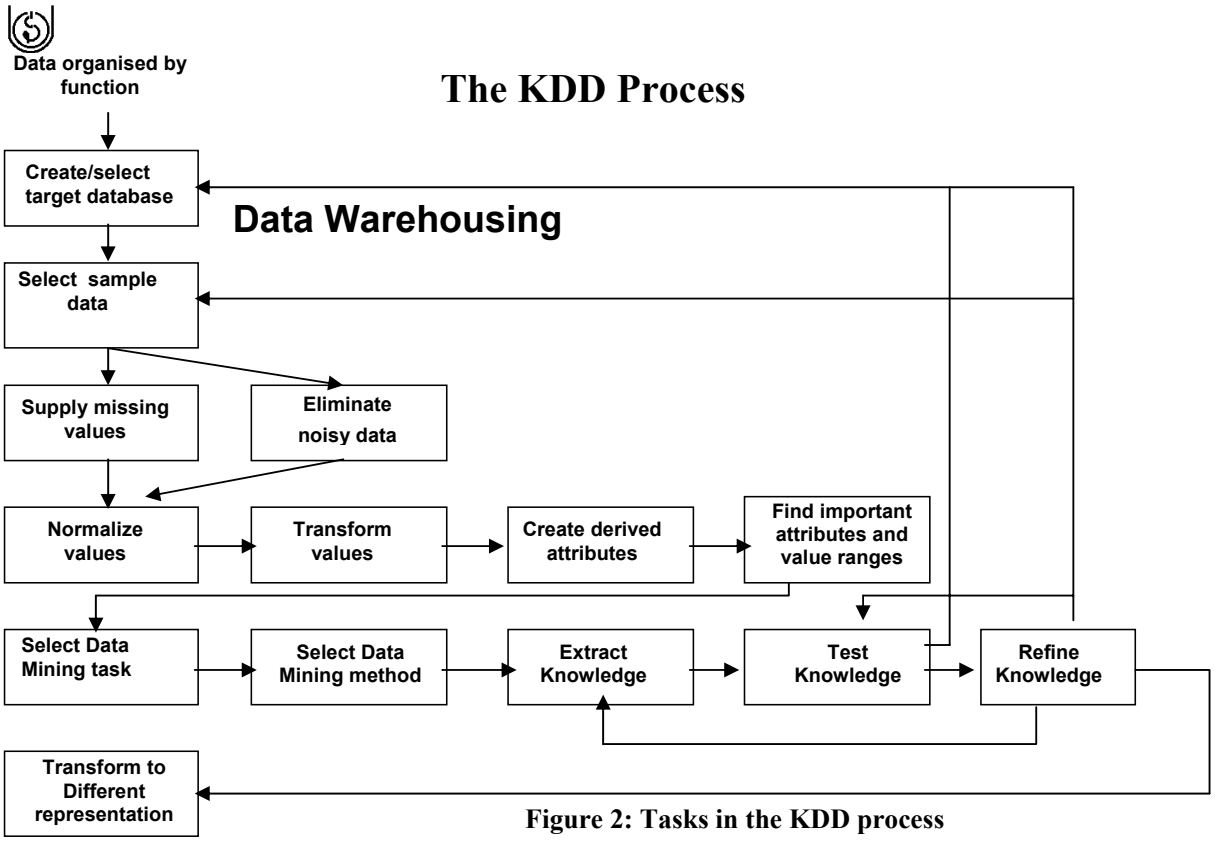



**Figure 2: Tasks in the KDD process**

- **Extraction of patterns:** It includes searching for desired patterns only, because, the data-mining model may generate a lot of patterns.
- Interpretation and presentation of pattern/model.

## 4.3 APPROACHES TO DATA MINING PROBLEMS

The approaches to data mining problems are based on the type of information/ knowledge to be mined. We will emphasis on three different approaches: Classification, Clustering, and Association Rules.

The classification task maps data into predefined groups or classes. The class of a tuple is indicated by the value of a user-specified goal attribute. Tuples consists of a set of predicating attributes and a goal attribute. The task, is to discover, some kind of relationship between the predicating attributes and the goal attribute, so that, the discovered information/ knowledge can be used to predict the class of new tuple(s).

The task of clustering is to group the tuples with similar attribute values into the same class. Given a database of tuples and an integer value k, the Clustering is to define a mapping, such that, tuples are mapped to different cluster.

The principle is to maximise intra-class similarity and minimise the interclass similarity. In clustering, there is no goal attribute. So, classification is supervised by the goal attribute, while clustering is an unsupervised classification.

The task of association rule mining is to search for interesting relationships among items in a given data set. Its original application is on "market basket data". The rule has the form X➔Y, where X and Y are sets of items and they do not intersect. Each

rule has two measurements, support and confidence. Given the user-specified minimum support and minimum confidence, the task is to find, rules with support and confidence above, minimum support and minimum confidence.

The distance measure finds, the distance or dissimilarity between objects the measures that are used in this unit are as follows:

- Euclidean distance: $dis(t_i,t_j)= \sqrt{\sum_{h=1}^{k}(t_{ih}-t_{jh})^2}$
- Manhattan distance: $dis(t_i,t_j)= \sum_{h=1}^{k}|(t_{ih}-t_{jh})|$

where $t_i$ and $t_j$ are tuples and h are the different attributes which can take values from 1 to k

☞ **Check Your Progress 1**

1) What do you mean by data mining?

……

……

……

2) How is data mining different from Knowledge discovery in databases? What are the different steps of KDD?

……

……

……

3) What is the difference between data mining and OLTP?

……

……

……

4) What are different data mining tasks?

……

……

……

# 4.4 CLASSIFICATION

The classification task maps data into predefined groups or classes.
Given a database/dataset $D=\{t_1,t_2,\ldots,t_n\}$ and a set of classes $C=\{C_1,\ldots,C_m\}$, the classification Problem is to define a mapping $f:D\rightarrow C$ where each $t_i$ is assigned to one class, that is, it divides database/dataset D into classes specified in the Set C.

A few very simple examples to elucidate classification could be:

- Teachers classify students' marks data into a set of grades as A, B, C, D, or F.
- Classification of the height of a set of persons into the classes tall, medium or short.

### 4.4.1 Classification Approach



The basic approaches to classification are:

- To create specific models by, evaluating training data, which is basically the old data, that has already been classified by using the domain of the experts' knowledge.
- Now applying the model developed to the new data.

**Please note that** in classification, the classes are predefined.

Some of the most common techniques used for classification may include the use of Decision Trees, Neural Networks etc. Most of these techniques are based on finding the distances or uses statistical methods.

### 4.4.2 Classification Using Distance (K-Nearest Neighbours)

This approach, places items in the class to which they are "closest" to their neighbour.It must determine distance between an item and a class. Classes are represented by centroid (Central value) and the individual points.

One of the algorithms that is used is K-Nearest Neighbors. Some of the basic points to be noted about this algorithm are:

- The training set includes *classes* along with other attributes. (Please refer to the training data given in the *Table* given below).
- The value of the K defines the number of *near items* (items that have less distance to the attributes of concern) that should be used from the given set of training data (just to remind you again, training data is already classified data). This is explained in point (2) of the following example.
- A new item is placed in the class in which the most number of close items are placed. (Please refer to point (3) in the following example).
- The value of K should be <= $\sqrt{Number\_of\_training\_items}$ However, in our example for limiting the size of the sample data, we have not followed this formula.

**Example:** Consider the following data, which tells us the person's class depending upon gender and height

| Name | Gender | Height | Class |
|---|---|---|---|
| Sunita | F | 1.6m | Short |
| Ram | M | 2m | Tall |
| Namita | F | 1.9m | Medium |
| Radha | F | 1.88m | Medium |
| Jully | F | 1.7m | Short |
| Arun | M | 1.85m | Medium |
| Shelly | F | 1.6m | Short |
| Avinash | M | 1.7m | Short |
| Sachin | M | 2.2m | Tall |
| Manoj | M | 2.1m | Tall |
| Sangeeta | F | 1.8m | Medium |
| Anirban | M | 1.95m | Medium |
| Krishna | F | 1.9m | Medium |
| Kavita | F | 1.8m | Medium |
| Pooja | F | 1.75m | Medium |

1) You have to classify the tuple <Ram, M, 1.6> from the training data that is given

to you.

2) Let us take only the **height** attribute for distance calculation and suppose K=5 then the following are the near five tuples to the data that is to be classified (using Manhattan distance as a measure on the height attribute).

| Name | Gender | Height | Class |
|---|---|---|---|
| Sunita | F | 1.6m | Short |
| Jully | F | 1.7m | Short |
| Shelly | F | 1.6m | Short |
| Avinash | M | 1.7m | Short |
| Pooja | F | 1.75m | Medium |

3) On examination of the tuples above, we classify the tuple <Ram, M, 1.6> to *Short* class since most of the tuples above belongs to *Short* class.

### 4.4.3 Decision or Classification Tree

Given a data set D = $\{t_1,t_2 \ldots, t_n\}$ where $t_i=<t_{i1}, \ldots, t_{ih}>$, that is, each tuple is represented by h attributes, assume that, the database schema contains attributes as $\{A_1, A_2, \ldots, A_h\}$. Also, let us suppose that the classes are $C=\{C_1, \ldots., C_m\}$, then:

Decision or Classification Tree is a tree associated with D such that

- Each internal node is labeled with attribute, $A_i$
- Each arc is labeled with the predicate which can be applied to the attribute at the parent node.
- Each leaf node is labeled with a class, $C_j$

Basics steps in the Decision Tree are as follows:

- Building the tree by using the training set dataset/database.
- Applying the tree to the new dataset/database.

Decision Tree Induction is the process of learning about the classification using the inductive approach. During this process, we create a decision tree from the training data. This decision tree can, then be used, for making classifications. To define this we need to define the following.

Let us assume that we are given probabilities $p_1, p_2, .., p_s$ whose sum is 1. Let us also define the term Entropy, which is the measure of the amount of randomness or surprise or uncertainty. Thus our basic goal in the classification process is that, the entropy for a classification should be zero, that, if no surprise then, entropy is equal to zero. Entropy is defined as:

$$H(p_1,p_2,\ldots p_s)= \sum_{i=1}^{s} (p_i * \log(1/p_i)) \qquad \ldots\ldots. \quad (1)$$

### ID3 Algorithm for Classification

This algorithm creates a tree using the algorithm given below and tries to reduce the expected number of comparisons.

**Algorithm: ID3 algorithm for creating decision tree from the given training data.**
**Input:** The *training data* and the *attribute-list*.



**Output**: A decision tree.

**Process**:

Step 1: Create a node N

Step 2: If sample data are all of the same class, C (that is probability is 1 )
then return N as a leaf node labeled class C

Step 3: If *attribute-list* is empty
then return N as a leaf node label it with the most common class in the training data; // majority voting

Step 4: Select *split-attribute*, which is the attribute in the *attribute-list* with the highest information gain;

Step 5: label node N with *split-attribute*;

Step 6: for each known value Ai, of *split-attribute* // partition the samples
Create a branch from node N for the condition: *split-attribute* = $A_i$;
// Now consider a partition and recursively create the decision tree:
Let xi be the set of data from training data that satisfies the condition:
*split-attribute* = $A_i$
if the set xi is empty then
attach a leaf labeled with the most common class in the prior set of training data;
else
attach the node returned after recursive call to the program with training data as xi and
new attribute list = present attribute-list – split-attribute;

End of Algorithm.

**Please note**: The algorithm given above, chooses the split attribute with the highest information gain, that is, calculated as follows:

$$\text{Gain (D,S)} = H(D) - \sum_{i=1}^{s} (P(D_i) * H(D_i)) \quad \text{……..(2)}$$

where S is new states = $\{D_1, D_2, D_3 \ldots D_S\}$ and H(D) finds the amount of order in that state

Consider the following data in which *Position* attribute acts as class

| Department | Age | Salary | Position |
|---|---|---|---|
| Personnel | 31-40 | Medium Range | Boss |
| Personnel | 21-30 | Low Range | Assistant |
| Personnel | 31-40 | Low Range | Assistant |
| MIS | 21-30 | Medium Range | Assistant |
| MIS | 31-40 | High Range | Boss |
| MIS | 21-30 | Medium Range | Assistant |
| MIS | 41-50 | High Range | Boss |
| Administration | 31-40 | Medium Range | Boss |
| Administration | 31-40 | Medium Range | Assistant |
| Security | 41-50 | Medium Range | Boss |
| Security | 21-30 | Low Range | Assistant |

**Figure 3: Sample data for classification**

We are applying ID3 algorithm, on the above dataset as follows:

The initial entropy of the dataset using formula at (1) is

$$H(initial) = (6/11)\log(11/6) + (5/11)\log(11/5) = 0.29923$$
(Assistant) (Boss)

Now let us calculate gain for the departments using the formula at (2)

Gain(Department) = H(initial) – [ P(Personnel) * H(MIS) + P(MIS) * H(Personnel) + P(Administration) * H(Administration) + P(Security) * H(Security) ]

= *0.29923*- { (3/11)[(1/3)log3 +(2/3)log(3/2)] + (4/11)[ (2/4)log 2 + (2/4)log 2] + (2/11)[ (1/2)log 2 + (1/2)log 2 ] + (2/11)[ (1/2)log 2 + (1/2)log 2 ] }
= 0.29923- 0.2943
= 0.0049

Similarly:

Gain(Age) = 0.29923- { (4/11)[(4/4)log(4/4)] + (5/11)[(3/5)log(5/3) + (2/5)log (5/2)] + (2/11)[(2/2) log (2/2)] }
= 0.29923 - 0.1328
= 0.1664

Gain(Salary) = 0.29923 – { (3/11)[(3/3)log 3] + (6/11)[(3/6) log2 + (3/6)log 2] + (2/11) [(2/2 log(2/2) ) }
= 0.29923 – 0.164
= 0.1350

Since age has the maximum gain, so, this attribute is selected as the first splitting attribute. In age range 31-40, class is not defined while for other ranges it is defined.

So, we have to again calculate the spitting attribute for this age range (31-40). Now, the tuples that belong to this range are as follows:

| Department | Salary | Position |
|---|---|---|
| Personnel | Medium Range | Boss |
| Personnel | Low Range | Assistant |
| MIS | High Range | Boss |
| Administration | Medium Range | Boss |
| Administration | Medium Range | Assistant |

Again the initial entropy = $(2/5)\log(5/2) + (3/5)\log(5/3) = 0.29922$
(Assistant) (Boss)

Gain(Department)= 0.29922- { (2/5)[ (1/2)log 2 + (1/2)log 2 ] +1/5[ (1/1)log1] + (2/5)[ (1/2)log 2 + (1/2)log 2 ] }
=0.29922- 0.240
=0.05922

Gain (Salary) = 0.29922- { (1/5) [(1/1)log1 ] +(3/5)[ (1/3)log 3 + (2/3)log(3/2)] +



(1/5) [(1/1)log1 ] }
= 0.29922- 0.1658
=0.13335

The Gain is maximum for salary attribute, so we take salary as the next splitting attribute. In middle range salary, class is not defined while for other ranges it is defined. So, we have to again calculate the spitting attribute for this middle range. Since only department is left, so, department will be the next splitting attribute. Now, the tuples that belong to this salary range are as follows:

| Department | Position |
|---|---|
| Personnel | Boss |
| Administration | Boss |
| Administration | Assistant |

Again in the Personnel department, all persons are Boss, while, in the Administration there is a tie between the classes. So, the person can be either Boss or Assistant in the Administration department.

Now the decision tree will be as follows:

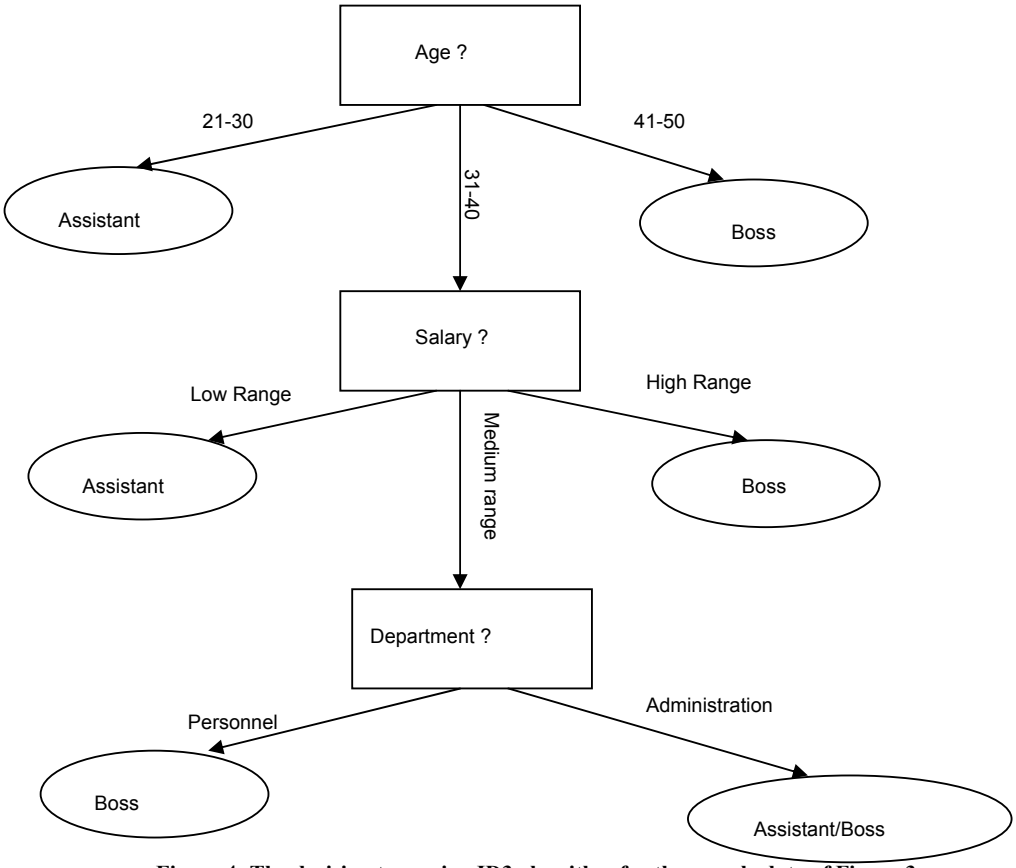



**Figure 4: The decision tree using ID3 algorithm for the sample data of Figure 3.**

Now, we will take a new dataset and we will classify the class of each tuple by applying the decision tree that we have built above.

Let us discuss, another important classification method called Bayesian classification in the next subsection.

### 4.4.4 Bayesian Classification

This is a statistical classification, which predicts the probability that a given sample is a member of a particular class. It is based on the Bayes theorem. The Bayesian classification shows better accuracy and speed when applied to large databases. We will discuss here the simplest form of Bayesian classification.

The basic underlying assumptions (also called class conditional independence) for this simplest form of classification known as the native Bayesian classification is:

"The effect of an attribute value on a given class is independent of the values of other attributes"

Let us discuss naive Bayesian classification in more details. But before that, let us, define the basic theorem on which this classification is based.

Bayes Theorem:

Let us assume the following:

X is a data sample whose class is to be determined
H is the hypothesis such that the data sample X belongs to a class C.
P(H | X) is the probability that hypothesis H holds for data sample X . It is also called the posterior probability that condition H holds for the sample X.
P(H) is the prior probability of H condition on the training data.
P(X | H) is the posterior probability of X sample, given that H is true.
P(X) is the prior probability on the sample X.

**Please note:** We can calculate P(X), P(X | H) and P(H) from the data sample X and training data. It is only P(H | X) which basically defines the probability that X belongs to a class C, and cannot be calculated. Bayes theorem does precisely this function. The Bayer's theorem states:

$$P(H \mid X) = \frac{P(X \mid H)\ P(H)}{P(X)}$$

Now after defining the Bayes theorem, let us explain the Bayesian classification with the help of an example.

i) Consider the sample having an n-dimensional feature vector. For our example, it is a 3-dimensional (Department, Age, Salary) vector with training data as given in the Figure 3.

ii) Assume that there are m classes $C_1$ to $C_m$. And an unknown sample X. The problem is to data mine which class X belongs to. As per Bayesian classification, the sample is assigned to the class, if the following holds:

$$P(C_i|X) > P(C_j|X) \text{ where j is from 1 to m but } j \neq i$$

In other words the class for the data sample X will be the class, which has the maximum probability for the unknown sample. **Please note:** The $P(C_i|X)$ will be found using:

$$P(C_i| X)= \frac{P(X|C_i)\ P(C_i)}{P(X)} \qquad (3)$$

In our example, we are trying to classify the following data:

X = (Department = "Personal", Age = "31 – 40" and Salary = "Medium Range)
into two classes (based on position) $C_1$=BOSS OR $C_2$=Assistant.

iii) The value of P(X) is constant for all the classes, therefore, only $P(X|C_i)\ P(C_i)$ needs to be found to be maximum. Also, if the classes are equally, then,



$P(C_1)=P(C_2)=…..P(C_n)$, then we only need to maximise $P(X|C_i)$.
How is $P(C_i)$ calculated?

$$P(C_i)= \frac{\text{Number of training samples for Class } C_i}{\text{Total Number of Training Samples}}$$

In our example,

$$P(C_1)=\frac{5}{11}$$

and

$$P(C_2)=\frac{6}{11}$$

So $P(C_1) \neq P(C_2)$

iv) $P(X|C_i)$ calculation may be computationally expensive if, there are large numbers of attributes. To simplify the evaluation, in the naïve Bayesian classification, we use the condition of class conditional independence, that is the values of attributes are independent of each other. In such a situation:

$$P(X|C_i)= \prod_{k=1}^{n} P(x_k|C_i) \quad ….(4)$$

where $x_k$ represent the single dimension or attribute.

The $P(x_k|C_i)$ can be calculated using mathematical function if it is continuous, otherwise, if it is categorical then, this probability can be calculated as:

$$P(x_k|C_i)= \frac{\text{Number of training samples of class } C_i \text{ having the value } x_k \text{ for the attribute } A_k}{\text{Number of training samples belonging to } C_i}$$

For our example, we have $x_1$ as Department= “Personnel”
$x_2$ as Age=”31 – 40” and
$x_3$ as Salary= “Medium Range”

P ( Department= “Personnel” | Position = “BOSS”) = 1/5
P ( Department= “Personnel” | Position = “Assistant”) = 2/6
P (Age=”31 – 40” | Position = “BOSS”) = 3/5
P (Age=”31 – 40” | Position = “Assistant”) = 2/6
P (Salary= “Medium Range”| Position = “BOSS”) = 3/5
P (Salary= “Medium Range” | Position = “Assistant”) = 3/6

Using the equation (4) we obtain:
P ( X | Position = “BOSS”) = 1/5 * 3/5 * 3/5
P ( X | Position = “Assistant”) = 2/6 * 2/6 * 3/6

Thus, the probabilities:
P ( X | Position = “BOSS”) P(Position = “BOSS”)
= (1/5 * 3/5 * 3/5) * 5/11
= 0.032727
P ( X | Position = “Assistant”) P( Position = “Assistant”)
= (2/6 * 2/6 * 3/6) * 6/11
= 0.030303

Since, the first probability of the above two is higher, the sample data may be classified into the BOSS position. Kindly check to see that you obtain the same result from the decision tree of *Figure 4*.



# 4.5 CLUSTERING

Clustering is grouping thing with similar attribute values into the same group. Given a database $D=\{t_1,t_2,\ldots,t_n\}$ of tuples and an integer value k, the Clustering problem is to define a mapping where each tuple $t_i$ is assigned to one cluster $K_j$, $1<=j<=k$.

A ***Cluster***, $K_j$, contains precisely those tuples mapped to it. Unlike the classification problem, clusters are not known in advance. The user has to the enter the value of the number of clusters k.

In other words a *cluster* can be defined as the collection of data objects that are similar in nature, as per certain defining property, but these objects are dissimilar to the objects in other clusters.
Some of the clustering examples are as follows:

- To segment the customer database of a departmental store based on similar buying patterns.
- To identify similar Web usage patterns etc.

Clustering is a very useful exercise specially for identifying similar groups from the given data. Such data can be about buying patterns, geographical locations, web information and many more.

Some of the clustering Issues are as follows:

- **Outlier handling:** How will the outlier be handled? (outliers are the objects that do not comply with the general behaviour or model of the data) Whether it is to be considered or it is to be left aside while calculating the clusters?
- **Dynamic data:** How will you handle dynamic data?
- **Interpreting results:** How will the result be interpreted?
- **Evaluating results:** How will the result be calculated?
- **Number of clusters:** How many clusters will you consider for the given data?
- **Data to be used:** whether you are dealing with quality data or the noisy data? If, the data is noisy how is it to be handled?
- **Scalability:** Whether the algorithm that is used is to be scaled for small as well as large data set/database.

There are many different kinds of algorithms for clustering. However, we will discuss only three basic algorithms. You can refer to more details on clustering from the further readings.

### 4.5.1 Partitioning Clustering

The partitioning clustering algorithms constructs k partitions from a given n objects of the data. Here $k \leq n$ and each partition must have at least one data object while one object belongs to **only one** of the partitions. A partitioning clustering algorithm normally requires users to input the desired number of clusters, k.

Some of the partitioning clustering algorithms are as follows:

- Squared Error
- K-Means



Now in this unit, we will briefly discuss these algorithms.

**Squared Error Algorithms**

The most frequently used criterion function in partitioning clustering techniques is the squared error criterion. The method of obtaining clustering by applying this approach is as follows:

**Squared Error Clustering Method:**

(1) Select an initial partition of the patterns with a fixed number of clusters and cluster centers.
(2) Assign each pattern to its closest cluster center and compute the new cluster centers as the centroids of the clusters. Repeat this step until convergence is achieved, i.e., until the cluster membership is stable.
(3) Merge and split clusters based on some heuristic information, optionally repeating step 2.

Some of the parameters that are used in clusters are as follows:

$$\text{Centriod}(C_m) = \sum_{i=1}^{N} t_{mi} / N$$

$$\text{Radius } (R_m) = \sqrt{\sum_{i-1}^{N} (t_{mi} - C_m)^2 / N}$$

$$\text{Diameter } (D_m) = \sqrt{\sum_{i-1}^{N} \sum_{j=1}^{N} (t_{mi} - t_{mj})^2 / (N * (N-1))}$$

A detailed discussion on this algorithm is beyond the scope of this unit. You can refer to more details on clustering from the further readings.

## K-Means clustering

In the K-Means clustering, initially a set of clusters is randomly chosen. Then iteratively, items are moved among sets of clusters until the desired set is reached. A high degree of similarity among elements in a cluster is obtained by using this algorithm. For this algorithm a set of clusters $K_i=\{t_{i1},t_{i2},\ldots,t_{im}\}$ is given , the cluster mean is:

$$m_i = (1/m)(t_{i1} + \ldots + t_{im}) \quad \ldots(5)$$

Where $t_i$ represents the tuples and m represents the mean

**The K-Means algorithm** is as follows:

**Input :**
D= { $t_1,t_2,\ldots t_n$} //Set of elements
A //Adjacency matrix showing distance between elements.
k //Number of desired clusters.

**Output :**
K //Set of Clusters

**K-Means Algorithm:**

Assign initial values for means $m_1,m_2..m_k$;
Repeat
Assign each item $t_i$ to the cluster which has the closest mean;
Calculate new mean for each cluster;
Until convergence criteria is met.

**K-Means Example:**

Let us take the number of clusters as 2 and the following input set is given to us:
Input set={1,2,3, 5,10,12,22,32,16,18}



- Step 1:We randomly assign means: $m_1=3,m_2=5$
- Step 2:$K_1=\{1,2,3\}$, $K_2=\{5,10,12,22,32,16,18\}$, $m_1=2,m_2=16.43$ (calculated mean using the formula (5)).

Now redefine cluster as per the closest mean:

- Step 3:$K_1=\{1,2,3,5\},K_2=\{10,12,22,32,16,18\}$

Calculate the mean once again:

- $m_1=2.75,m_2=18.33$
- Step 4:$K_1=\{1,2,3,5\},K_2=\{10,12,22,32,16,18\}$, $m_1=2.75,m_2=18.33$
- Stop as the clusters with these means are the same.

### 4.5.2 Nearest Neighbour Clustering

In this approach, items are iteratively merged into the existing clusters that are closest. It is an incremental method. The threshold, t, used to determine if items are added to existing clusters or a new cluster is created. This process continues until all patterns are labeled or no additional labeling occurs.

**The Nearest Neighbour algorithm** is as follows:

**Input :**

D= { $t_1,t_2,\ldots t_n$} //Set of elements
A //Adjacency matrix showing distance between elements.
k //Number of desired clusters.

**Output :**

K //Set of Clusters

**Nearest Neighbour algorithm :**

$K_1=\{t_1\}$;
$K=\{K_1\}$;
k=1;
for i=2 to n do
    Find the $t_m$ in some cluster Km in K such that distance($t_i,t_m$) is the smallest;
    If dis($t_i,t_m$)<= t then
        $K_m=K_m \cup t_i$;
    Else
        k=k +1;
        $K_k=\{t_i\}$;

### 4.5.3 Hierarchical Clustering

In this method, the clusters are created in levels and depending upon the threshold value at each level the clusters are again created.

An agglomerative approach begins with each tuple in a distinct cluster and successively merges clusters together until a stopping criterion is satisfied. This is the bottom up approach.

A divisive method begins with all tuples in a single cluster and performs splitting until a stopping criterion is met. This is the top down approach.

A hierarchical algorithm yields a dendrogram representing the nested grouping of tuples and similarity levels at which groupings change. Dendrogram is a tree data structure which illustrates hierarchical clustering techniques. Each level shows clusters for that level. The leaf represents individual clusters while the root represents one cluster.



Most hierarchical clustering algorithms are variants of the single-link, average link and complete-link algorithms. Out of these the single-link and complete-link algorithms are the most popular. These two algorithms differ in the way they characterise the similarity between a pair of clusters.

In the single-link method, the distance between two clusters is the minimum of the distances between all pairs of patterns drawn from the two clusters (one pattern from the first cluster, the other from the second).

In the complete-link algorithm, the distance between two clusters is the maximum of all pair-wise distances between patterns in the two clusters.

In either case, two clusters are merged to form a larger cluster based on minimum distance criteria.

You can refer to more detail on the hierarchical clustering algorithms from the further readings.

### ☞ Check Your Progress 2

1) What is the classification of data? Give some examples of classification.

……

……

……

2) What is clustering?

……

……

……

3) How is clustering different from classification?

……

……

……

## 4.6 ASSOCIATION RULE MINING

The task of association rule mining is to find certain association relationships among a set of items in a dataset/database. The association relationships are described in association rules. In association rule mining there are two measurements, *support* and *confidence*. The confidence measure indicates the rule's strength, while support corresponds to the frequency of the pattern.

A typical example of an association rule created by data mining often termed to as "market basket data" is: " 80% of customers who purchase bread also purchase butter."

Other applications of data mining include cache customisation, advertisement personalisation, store layout and customer segmentation etc. All these applications try to determine the associations between data items, if it exists to optimise performance.

Formal Definition:

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. TID indicates a unique

transaction identifier. An association rule is an implication of the form X ➔ Y, where $X \subset I$, $Y \subset I$, and $X \cap Y = \varnothing$. X is called the antecedent while Y is called the consequence of the rule.

The rule X ➔ Y has support s in the transaction set D if s% of transactions in D contains $X \cup Y$. The rule has confidence c if c% of transactions in D that contains X also contains Y. Support indicates how frequently the pattern occurs, while confidence indicates the strength of the rule.

Given a user specified minimum support and minimum confidence, the problem of mining association rules is to find all the association rules whose support and confidence are larger than the minimum support and minimum confidence. Thus, this approach can be broken into two sub-problems as follows:

(1) Finding the frequent itemsets which have support above the predetermined minimum support.
(2) Deriving all rules, based on each frequent itemset, which have confidence more than the minimum confidence.

There are a lots of ways to find the large itemsets but we will only discuss the Apriori Algorithm.

**Apriori Algorithm: For finding frequent itemsets**

The apriori algorithm applies the concept that if an itemset has minimum support, then all its subsets also have minimum support. An itemset having minimum support is called frequent itemset or large itemset. So any subset of a frequent itemset must also be frequent.

Apriori algorithm generates the candidate itemsets to be counted in the pass, by using only the large item set found in the previous pass – without considering the transactions in the database.

It starts by finding all frequent 1-itemsets (itemsets with 1 item); then consider 2-itemsets from these 1-itemsets, and so forth. During each iteration only candidates found to be frequent in the previous iteration are used to generate a new candidate set during the next iteration. The algorithm terminates when there are no frequent k-itemsets.

Notations that are used in Apriori algorithm are given below:

| k-itemset | An itemset having k items |
|---|---|
| $L_k$ | Set of frequent k-itemset (those with minimum support) |
| $C_k$ | Set of candidate k-itemset (potentially frequent itemsets) |

Apriori algorithm function takes as argument $L_{k-1}$ and returns a superset of the set of all frequent k-itemsets. It consists of a join step and a prune step. The Apriori algorithm is given below :

APRIORI

1. k = 1
2. Find frequent set $L_k$ from $C_k$ of all candidate itemsets
3. Form $C_{k+1}$ from $L_k$; k = k + 1
4. Repeat 2-3 until $C_k$ is empty

Details about steps 2 and 3

**Step 2**: Scan the data set D and count each itemset in $C_k$ , if it is greater than minimum support, it is frequent



**Step 3**:

- For k=1, $C_1$ = all frequent 1-itemsets. (all individual items).
- For k>1, generate $C_k$ from Lk-1 as follows:

The join step

$C_k$ = k-2 way join of $L_{k-1}$ with itself

If both $\{a_1, \ldots, a_{k-2}, a_{k-1}\}$ & $\{a_1, \ldots, a_{k-2}, a_k\}$ are in $L_{k-1}$, then add $\{a_1, \ldots, a_{k-2}, a_{k-1}, a_k\}$ to $C_k$

(We keep items sorted).

The prune step

Remove $\{a_1, \ldots, a_{k-2}, a_{k-1}, a_k\}$ if it contains a non-frequent (k-1) subset.

{In the prune step, delete all itemsets $c \in C_k$ such that some (k-1)-subset of C is not in $L_{k-1}$.}

**Example: Finding frequent itemsets:**

Consider the following transactions with minimum support s=30% for finding the frequent itemsets by applying Apriori algorithm:

| Transaction ID | Item(s) purchased |
|---|---|
| 1 | Shirt, Trouser |
| 2 | Shirt, Trouser, Coat |
| 3 | Coat, Tie, Tiepin |
| 4 | Coat, Shirt, Tie, Trouser |
| 5 | Trouser, Belt |
| 6 | Coat, Tiepin, Trouser |
| 7 | Coat, Tie |
| 8 | Shirt |
| 9 | Shirt, Coat |
| 10 | Shirt, Handkerchief |

Method of finding the frequent itemset is as follows:

| Pass Number | Candidates | Large itemsets ( ≥ 3) |
|---|---|---|
| 1 | $C_1$={Belt 1,<br>Coat 6,<br>Handkerchief 1,<br>Shirt 6,<br>Tie 3,<br>Tiepin 2,<br>Trouser 5 } | $L_1$={Coat 6,<br>Shirt 6,<br>Tie 3,<br>Trouser 5 } |
| 2 | $C_2$= {{Coat, Shirt} 3,<br>{Coat, Tie} 3,<br>{Coat, Trouser} 3,<br>{Shirt, Tie} 1,<br>{Shirt, Trouser} 3,<br>{Tie, Trouser} 1 } | $L_2$={{Coat, Shirt} 3,<br>{Coat, Tie} 3,<br>{Coat, Trouser} 3,<br>{Shirt, Trouser} 3 } |
| 3 | $C_3$= {{Coat, Shirt, Trouser} 2} | $L_3$= Ø |

The calculation of 3-itemsets is mentioned below:

Join operation yields 3 item sets as: {{Coat, Shirt, Tie}, {Coat, Shirt, Trouser}, {Coat, Tie, Trouser}}



However, the Prune operation removes two of these items from the set due to the following reasons:

{Coat, Shirt, Tie} is pruned as {Shirt, Tie} is not in $L_2$
{Coat, Shirt, Trouser} is retained as {Coat, Shirt}, {Coat, Trouser} and {Shirt, Trouser} all three are in $L_2$
{Coat, Tie, Trouser} is pruned as {Tie, Trouser} is not in $L_2$

The set L={ $L_1$ , $L_2$ , $L_3$}

The following algorithm creates the association rules from the set L so created by the Apriori algorithm.

**Algorithm to generate the Association Rules:**

**Input:**

| | |
|---|---|
| D | //Database of transactions |
| I | //Items |
| L | //Large itemsets |
| s | // Support |
| c | // Confidence |

**Output:**

| | |
|---|---|
| R | //Association rules satisfying minimum s and c |

**AR algorithm:**

```
R= Ø
For each l ε L do    // for each large item ( l ) in the set L
        For each x ⊂ I  such that x <> Ø and x <>l do
                if support(l) /support(x)  ≥ c   then
                        R=R ∪  {x ➔ (l - x)};
```

**Apriori Advantages/Disadvantages:**

The following are the advantages and disadvantages of the Apriori algorithm:

- **Advantages:**
  - It uses large itemset property.
  - It easy to implement.
- **Disadvantages:**
  - It assumes transaction database is memory resident.

## 4.7 APPLICATIONS OF DATA MINING PROBLEM

Some of the applications of data mining are as follows:

- **Marketing and sales data analysis:** A company can use customer transactions in their database to segment the customers into various types. Such companies may launch products for specific customer bases.
- **Investment analysis:** Customers can look at the areas where they can get good returns by applying the data mining.
- **Loan approval:** Companies can generate rules depending upon the dataset they have. On that basis they may decide to whom, the loan has to be approved.
- **Fraud detection:** By finding the correlation between faults, new faults can be detected by applying data mining.



- **Network management:** By analysing pattern generated by data mining for the networks and its faults, the faults can be minimised as well as future needs can be predicted.
- **Risk Analysis:** Given a set of customers and an assessment of their risk-worthiness, descriptions for various classes can be developed. Use these descriptions to classify a new customer into one of the risk categories.
- **Brand Loyalty:** Given a customer and the product he/she uses, predict whether the customer will change their products.
- **Housing loan prepayment prediction:** Rule discovery techniques can be used to accurately predict the aggregate number of loan prepayments in a given quarter as a function of prevailing interest rates, borrower characteristics and account data.

## 4.8 COMMERCIAL TOOLS OF DATA MINING

**Commercial Tools:**

1) **AC2**, provides graphical tools for data preparation and building decision trees.
2) **Business Miner**, data mining product positioned for the mainstream business user.
3) **C4.5**, the "classic" decision-tree tool, developed by **J. R. Quinlan**
4) **C5.0/See5**, constructs classifiers in the form of decision trees and rulesets.
5) **CART**, decision-tree software, combines an easy-to-use GUI with advanced features for data mining, data pre-processing and predictive modeling.
6) **Cognos Scenario**, allows you to quickly identify and rank the factors that have a significant impact on your key business measures.
7) **Decisionhouse**, provides data extraction, management, pre-processing and visualisation, plus customer profiling, segmentation and geographical display.
8) **Kernel Miner**, decision-tree-based classifier with fast DB access.
9) **Knowledge Seeker**, high performance interactive decision tree analytical tool.
10) **SPSS AnswerTree**, easy to use package with four decision tree algorithms - two types of CHAID, CART, and QUEST.
11) **XpertRule Miner** (Attar Software), provides graphical decision trees with the ability to embed as ActiveX components.
12) **AIRA**, a rule discovery, data and knowledge visualisation tool. AIRA for Excel extracts rules from MS-Excel spreadsheets.
13) **Datamite**, enables rules and knowledge to be discovered in ODBC-compliant relational databases.
14) **SuperQuery**, business Intelligence tool; works with Microsoft Access and Excel and many other databases.
15) **WizWhy**, automatically finds all the if-then rules in the data and uses them to summarise the data, identify exceptions, and generate predictions for new cases.
16) **XpertRule Miner** (Attar Software) provides association rule discovery from any ODBC data source.
17) **DMSK: Data-Miner Software Kit :Task**: Collection of tools for efficient mining of big data (Classification, Regression, Summarisation, Deviation Detection multi-task tools).

16) **OSHAM** Task: Task (Clustering) interactive-graphic system for discovering concept hierarchies from unsupervised data

17) **DBMiner** is a data mining system for interactive mining of multiple-level knowledge in large relational databases.

**Free Tools:**

1) **EC4.5**, a more efficient version of C4.5, which uses the best among three strategies at each node construction.

2) **IND**, provides CART and C4.5 style decision trees and more. Publicly available from NASA but with export restrictions.

3) **ODBCMINE**, shareware data-mining tool that analyses ODBC databases using the C4.5, and outputs simple IF..ELSE decision rules in ASCII.

4) **OC1**, decision tree system continuous feature values; builds decision trees with linear combinations of attributes at each internal node; these trees then partition the space of examples with both oblique and axis-parallel hyper planes.

5) **PC4.5**, a parallel version of C4.5 built with Persistent Linda system.

6) **SE-Learn**, Set Enumeration (SE) trees generalise decision trees. Rather than splitting by a single attribute, one recursively branches on all (or most) relevant attributes. (LISP)

7) **CBA**, mines association rules and builds accurate classifiers using a subset of association rules.

8) **KINOsuite**-PR extracts rules from trained neural networks.

9) **RIPPER**, a system that learns sets of rules from data

## ☞ Check Your Progress 3

1) What is association rule mining?

……

……

……

2) What is the application of data mining in the banking domain?

……

……

……

3) Apply the Apriori algorithm for generating large itemset on the following dataset:

| Transaction ID | Items purchased |
|---|---|
| T100 | $A_1a_3a_4$ |
| T200 | $A_2a_3a_5$ |
| T300 | $A_1a_2a_3a_5$ |
| T400 | $A_2a_5$ |

……

……

……



## 4.9 SUMMARY

1) Data mining is the process of automatic extraction of interesting (non trivial, implicit, previously unknown and potentially useful) information or pattern from the data in large databases.

2) Data mining is one of the steps in the process of Knowledge Discovery in databases.

3) In data mining tasks are classified as: Classification, Clustering and Association rules.

4) The classification task maps data into predefined classes.

5) Clustering task groups things with similar properties/ behaviour into the same groups.

6) Association rules find the association relationship among a set of objects.

7) Data mining is applied in every field whether it is Games, Marketing, Bioscience, Loan approval, Fraud detection etc.

## 4.10 SOLUTIONS /ANSWERS

### Check Your Progress 1

1) Data mining is the process of automatic extraction of interesting (non trivial, implicit, previously unknown and potentially useful) information or patterns from the data in large databases.

2) Data mining is only one of the many steps involved in knowledge discovery in databases. The various steps in KDD are data extraction, data cleaning and preprocessing, data transformation and reduction, data mining and knowledge interpretation and representation.

3) The query language of OLTP is well defined and it uses SQL for it, while, for data mining the query is poorly defined and there is no precise query language. The data used in OLTP is operational data while in data mining it is historical data. The output of the query of OLTP is precise and is the subset of the data while in the case of data mining the output is fuzzy and it is not a subset of the data.

4) The different data-mining tasks are: Classification, Clustering and Association Rule Mining.

### Check Your Progress 2

1) The classification task maps data into predefined groups or classes. The class of a tuple is indicated by the value of a user-specified goal attribute. Tuples consists of a set of predicating attributes and a goal attribute. The task is to discover some kind of relationship between the predicating attributes and the goal attribute, so that the discovered knowledge can be used to predict the class of new tuple(s).

   Some of the examples of classification are: Classification of students grades depending upon their marks, classification of customers as good or bad customer in a bank.

2) The task of clustering is to group the tuples with similar attribute values into the same class. Given a database of tuples and an integer value k, Clustering defines mapping, such that, tuples are mapped to different clusters.

3) In classification, the classes are predetermined, but, in the case of clustering the groups are not predetermined. The number of clusters has to be given by the user.

**Check Your Progress 3**

1) The task of association rule mining is to search for interesting relationships among items in a given data set. Its original application is on "market basket data". The rule has the form X➔Y, where X and Y are sets of items and they do not intersect.

2) The data mining application in banking are as follows:
   1. Detecting patterns of fraudulent credit card use.
   2. Identifying good customers.
   3. Determining whether to issue a credit card to a person or not.
   4. Finding hidden correlations between different financial indicators.

3) The dataset D given for the problem is:

| Transaction ID | Items purchased |
|---|---|
| T100 | $a_1a_3a_4$ |
| T200 | $a_2a_3a_5$ |
| T300 | $a_1a_2a_3a_5$ |
| T400 | $a_2a_5$ |

Assuming the minimum support as 50% for calculating the large item sets. As we have 4 transaction, at least 2 transaction should have the data item.

1. scan D ➔ $C_1$: $a_1$:2, $a_2$:3, $a_3$:3, $a_4$:1, $a_5$:3
   ➔ $L_1$: $a_1$:2, $a_2$:3, $a_3$:3, $a_5$:3
   ➔ $C_2$: $a_1a_2$, $a_1a_3$, $a_1a_5$, $a_2a_3$, $a_2a_5$, $a_3a_5$
2. scan D ➔ $C_2$: $a_1a_2$:1, $a_1a_3$:2, $a_1a_5$:1, $a_2a_3$:2, $a_2a_5$:3, $a_3a_5$:2
   ➔ $L_2$: $a_1a_3$:2, $a_2a_3$:2, $a_2a_5$:3, $a_3a_5$:2
   ➔ $C_3$: $a_1a_2a_3$, $a_1a_2a_5$, $a_2a_3a_5$
   ➔ Pruned $C_3$: $a_2a_3a_5$
3. scan D ➔ $L_3$: $a_2a_3a_5$:2

Thus L={$L_1$,$L_2$,$L_3$}

## 4.11 FURTHER READINGS

1) *Data Mining Concepts and Techniques*, J Han, M Kamber, Morgan Kaufmann Publishers, 2001.

2) *Data Mining*, A K Pujari, 2004.

# UNIT 1 EMERGING DATABASE MODELS, TECHNOLOGIES AND APPLICATIONS-I

**Structure** **Page Nos.**


1.0 Introduction 5
1.1 Objectives 5
1.2 Multimedia Database 6
1.2.1 Factors Influencing the Growth of Multimedia Data
1.2.2 Applications of Multimedia Database
1.2.3 Contents of MMDB
1.2.4 Designing MMDBs
1.2.5 State of the Art of MMDBMS
1.3 Spatial Database and Geographic Information Systems 10
1.4 Gnome Databases 12
1.4.1 Genomics
1.4.2 Gene Expression
1.4.3 Proteomics
1.5 Knowledge Databases 17
1.5.1 Deductive Databases
1.5.2 Semantic Databases
1.6 Information Visualisation 18
1.7 Summary 19
1.8 Solutions/Answers 20


## 1.0 INTRODUCTION

Database technology has advanced from the relational model to the distributed DBMS and Object Oriented databases. The technology has also advanced to support data formats using XML. In addition, data warehousing and data mining technology has become very popular in the industry from the viewpoint of decision making and planning.

Database technology is also being used in advanced applications and technologies. Some of this new application includes multimedia based database applications, geographic applications, Gnome databases, knowledge and spatial databases and many more such applications. These applications require some additional features from the DBMS as they are special in nature and thus are categorised as emerging database technologies.

This unit provides a brief introduction of database requirements of these newer applications.

## 1.1 OBJECTIVES

After going through this unit, you should be able to:

- define the requirements of a multimedia database systems;
- identify the basic features of geographic databases;
- list the features of Gnome databases;
- differentiate various knowledge databases and their advantages, and
- define the terms information visualisation and spatial databases.



# 1.2 MULTIMEDIA DATABASE

Multimedia and its applications have experienced tremendous growth. Multimedia data is typically defined as containing digital images, audio, video, animation and graphics along with the textual data. In the past decade, with the advances in network technologies, the acquisition, creation, storage and processing of multimedia data and its transmission over networks have grown tremendously.

A Multimedia Database Management System (MMDBMS) provides support for multimedia data types. It also provides the facilities for traditional DBMS functions like database creation, data modeling, data retrieval, data access and organisation, and data independence. With the rapid development of network technology, multimedia database system, multimedia information exchange is becoming very important. Any such application would require the support for a string multimedia database technology. Let us look into some of the factors that influence the growth of multimedia data.

## 1.2.1 Factors Influencing the Growth of Multimedia Data

### (i) Technological Advancements

Some of the technological advances that attributed to the growth of multimedia data are:

- computers, their computational power and availability,
- availability of high-resolution devices for the capture and display of multimedia data (digital cameras, scanners, monitors, and printers),
- development of high-density storage devices, and
- integration of all such technologies through digital means.

### (ii) High Speed Data Communication Networks and Software

Secondly high-speed data communication networks are common these days. These networks not only support high bandwidth but also are more reliable and support digital data transfer. Even the World Wide Web has rapidly grown and software for manipulating multimedia data is now available.

### (iii) Applications

With the rapid growth of computing and communication technologies, many applications have come to the forefront. Thus, any such applications in future will support life with multimedia data. This trend is expected to go on increasing in the days to come.

## 1.2.2 Applications of Multimedia Database

Multimedia data contains some exciting features. They are found to be more effective in dissemination of information in science, engineering, medicine, modern biology, and social sciences. They also facilitate the development of new paradigms in distance learning, and interactive personal and group entertainment.

Some of the typical applications of multimedia databases are:

- media commerce
- medical media databases
- bioinformatics
- ease of use of home media
- news and entertainment
- surveillance

- wearable computing
- management of meeting/presentation recordings
- biometrics (people identification using image, video and/or audio data).

The huge amount of data in different multimedia-related applications needs databases as the basic support mechanism. This is primarily due to the fact that the databases provide consistency, concurrency, integrity, security and availability of data. On the other hand from a user perspective, databases provide ease of use of data manipulation, query and retrieval of meaningful and relevant information from a huge collection of stored data.

Multimedia Databases (MMDBs) must cope with the large volume of multimedia data, being used in various software applications. Some such applications may include digital multimedia libraries, art and entertainment, journalism and so on. Some of these qualities of multimedia data like size, formats etc. have direct and indirect influence on the design and development of a multimedia database.

Thus, a MMDBs needs to provide features of a traditional database as well as some new and enhanced functionalities and features. They must provide a homogenous framework for storing, processing, retrieving, transmitting and presenting a wide variety of multiple media data types available in a large variety of formats.

### 1.2.3 Contents of MMDB

A MMDB needs to manage the following different types of information with respect to the multimedia data:

**Media Data:** It includes the media data in the form of images, audio and video. These are captured, digitised, processes, compressed and stored. Such data is the actual information that is to be stored.

**Media Format Data:** This data defines the format of the media data after the acquisition, processing, and encoding phases. For example, such data may consist of information about sampling rate, resolution, frame rate, encoding scheme etc. of various media data.

**Media Keyword Data:** This contains the keyword related to the description of media data. For example, for a video, this might include the date, time, and place of recording, the person who recorded, the scene description, etc. This is also known as content description data.

**Media Feature Data:** This contains the features derived from the media data. A feature characterises the contents of the media. For example, this could contain information on the distribution of colours, the kinds of textures and the different shapes present in an image. This is also referred to as content dependent data.

The last three types are known as meta data as, they describe several different aspects of the media data. The media keyword data and media feature data are used as indices for searching purpose. The media format data is used to present the retrieved information.

### 1.2.4 Designing MMDBs

The following characteristics of multimedia data have direct and indirect impacts on the design of MMDBs:

- the huge size of MMDBs,
- temporal nature of the data,
- richness of content through media, and



- complexity of representation and subjective interpretation specially from the viewpoint of the meta data.

**Challenges in Designing of Multimedia Databases**

The major challenges in designing multimedia databases are due to the requirements they need to satisfy. Some of these requirements are:

1) The database should be able to manage different types of input, output, and storage devices. For example, the data may be input from a number of devices that could include scanners, digital camera for images, microphone, MIDI devices for audio, video cameras. Typical output devices are high-resolution monitors for images and video, and speakers for audio.

2) The database needs to handle a variety of data compression and storage formats. Please note that data encoding has a variety of formats even within a single application. For example, in a medical application, the MRI image of the brain should be loss less, thus, putting very stringent quality on the coding technique, while the X-ray images of bones can be coded with lossy techniques as the requirements are less stringent. Also, the radiological image data, the ECG data, other patient data, etc. have widely varying formats.

3) The database should be able to support different computing platforms and operating systems. This is due to the fact that multimedia databases are huge and support a large variety of users who may operate computers and devices suited to their needs and tastes. However, all such users need the same kind of user-level view of the database.

4) Such a database must integrate different data models. For example, the textual and numeric data relating to a multimedia database may be best handled using a relational database model, while linking such data with media data as well as handling media data such as video documents are better done using an object-oriented database model. So these two models need to co-exist in MMDBs.

5) These systems need to offer a variety of query systems for different kinds of media. The query system should be easy-to-use, fast and deliver accurate retrieval of information. The query for the same item sometimes is requested in different forms. For example, a portion of interest in a video can be queried by using either:

   (a) a few sample video frames as an example

   (b) a clip of the corresponding audio track or

   (c) a textual description using keywords.

6) One of the main requirements for such a Database would be to handle different kinds of indices. The multimedia data is in exact and subjective in nature, thus, the keyword-based indices and exact range searches used in traditional databases are ineffective in such databases. For example, the retrieval of records of students based on enrolment number is precisely defined, but the retrieval of records of student having certain facial features from a database of facial images, requires, content-based queries and similarity-based retrievals. Thus, the multimedia database may require indices that are content dependent key-word indices.

7) The Multimedia database requires developing measures of data similarity that are closer to perceptual similarity. Such measures of similarity for different media types need to be quantified and should correspond to perceptual similarity. This will also help the search process.

8) Multimedia data is created all over world, so it could have distributed database features that cover the entire world as the geographic area. Thus, the media data may reside in many different distributed storage locations.

9) Multimedia data may have to be delivered over available networks in real-time. Please note, in this context, the audio and video data is temporal in nature. For example, the video frames need to be presented at the rate of about 30 frames/sec for smooth motion.

10) One important consideration with regard to Multimedia is that it needs to synchronise multiple media types relating to one single multimedia object. Such media may be stored in different formats, or different devices, and have different frame transfer rates.

Multimedia data is now being used in many database applications. Thus, multimedia databases are required for efficient management and effective use of enormous amounts of data.

### 1.2.5 State of the Art of MMDBMS

The first multimedia database system **ORION** was developed in 1987. The mid 90s saw several commercial MMDBMS being implemented from scratch. Some of them were **MediaDB**, now **MediaWay**, **JASMINE**, and **ITASCA** (the commercial successor of **ORION**). They were able to handle different kinds of data and support mechanisms for querying, retrieving, inserting, and updating data. However, most of these products are not on offer commercially and only some of them have adapted themselves successfully to hardware, software and application changes.

These software are used to provide support for a wide variety of different media types, specifically different media file formats such as image formats, video etc. These files need to be managed, segmented, linked and searched.

The later commercial systems handle multimedia content by providing complex object types for various kinds of media. In such databases the object orientation provides the facilities to define new data types and operations appropriate for the media, such as video, image and audio. Therefore, broadly MMDBMSs are extensible **Object-Relational DBMS (ORDBMSs)**. The most advanced solutions presently include **Oracle 10g**, **IBM DB2** and **IBM Informix**. These solutions purpose almost similar approaches for extending the search facility for video on similarity-based techniques.

Some of the newer projects address the needs of applications for richer semantic content. Most of them are based on the new MPEG-standards MPEG-7 and MPEG-21.

**MPEG-7**

**MPEG-7** is the ISO/IEC 15938 standard for multimedia descriptions that was issued in 2002. It is XML based multimedia meta-data standard, and describes various elements for multimedia processing cycle from the capture, analysis/filtering, to the delivery and interaction.

**MPEG-21** is the ISO/IEC 21000 standard and is expected to define an open multimedia framework. The intent is that the framework will cover the entire multimedia content delivery chain including content creation, production, delivery, presentation etc.

**Challenges for the Multimedia Database Technologies**: Multimedia technologies need to evolve further. Some of the challenges posed by multimedia database applications are:

- the applications utilising multimedia data are very diverse in nature. There is a need for the standardisation of such database technologies,



- technology is ever changing, thus, creating further hurdles in the way of multimedia databases,
- there is still a need to refine the algorithms to represent multimedia information semantically. This also creates problems with respect to information interpretation and comparison.

☞ **Check Your Progress 1**

1) What are the reasons for the growth of multimedia data?

……

……

……

2) List four application areas of multimedia databases.

……

……

……

3) What are the contents of multimedia database?

……

……

……

4) List the challenges in designing multimedia databases.

……

……

……

## 1.3 SPATIAL DATABASE AND GEOGRAPHIC INFORMATION SYSTEMS

A spatial database keeps track of an object in a multi-dimensional space. A spatial database may be used to represent the map of a country along with the information about railways, roads, irrigation facilities, and so on. Such applications are known as Geographic Information Systems (GIS). Let us discuss GIS in this section.

The idea of a geographic database is to provide geographic information; therefore, they are referred to as the Geographic Information System (GIS). A GIS is basically a collection of the Geographic information of the world. This information is stored and analysed on the basis of data stored in the GIS. The data in GIS normally defines the physical properties of the geographic world, which includes:

- spatial data such as political boundaries, maps, roads, railways, airways, rivers, land elevation, climate etc.
- non-spatial data such as population, economic data etc.

But what are the Applications of the Geographic Databases?

The applications of the geographic databases can be categorised into three broad categories. These are:

- **Cartographic Applications:** These applications revolve around the capture and analysis of cartographic information in a number of layers. Some of the basic applications in this category would be to analyse crop yields, irrigation facility

planning, evaluation of land use, facility and landscape management, traffic monitoring system etc. These applications need to store data as per the required applications. For example, irrigation facility management would require study of the various irrigation sources, the land use patterns, the fertility of the land, soil characteristics, rain pattern etc. some of the kinds of data stored in various layers containing different attributes. This data will also require that any changes in the pattern should also be recorded. Such data may be useful for decision makers to ascertain and plan for the sources and types of and means of irrigation.

- **3-D Digital Modelling Applications:** Such applications store information about the digital representation of the land, and elevations of parts of earth surfaces at sample points. Then, a surface model is fitted in using the interpolation and visualisation techniques. Such models are very useful in earth science oriented studies, air and water pollution studies at various elevations, water resource management etc. This application requires data to be represented as attribute based just as in the case of previous applications.

- The third kind of application of such information systems is using the geographic objects applications. Such applications are required to store additional information about various regions or objects. For example, you can store the information about the changes in buildings, roads, over a period of time in a geographic area. Some such applications may include the economic analysis of various products and services etc.

**Requirements of a GIS**

The data in GIS needs to be represented in graphical form. Such data would require any of the following formats:

- **Vector Data:** In such representations, the data is represented using some geometric objects such as line, square, circle, etc. For example, you can represent a road using a sequence of line segments.
- **Raster Data:** Here, data is represented using an attribute value for each pixel or voxel (a three dimensional point). Raster data can be used to represent three-dimensional elevation using a format termed digital elevation format. For object related applications a GIS may include a temporal structure that records information about some movement related detail such as traffic movement.

A GIS must also support the analysis of data. Some of the sample data analysis operations that may be needed for typical applications are:

- analysing soil erosion
- measurement of gradients
- computing shortest paths
- use of DSS with GIS etc.

One of the key requirements of GIS may be to represent information in an integrated fashion, using both the vector and raster data. In addition it also takes care of data at various temporal structures thus, making it amenable to analysis.

Another question here is the capturing of information in two-dimensional and three-dimensional space in digital form. The source data may be captured by a remote sensing satellite, which can then be, further appended by ground surveys if the need arises. Pattern recognition in this case, is very important for the capture and automating of information input.



Once the data is captured in GIS it may be processed through some special operations. Some such operations are:

- Interpolation for locating elevations at some intermediate points with reference to sample points.
- Some operations may be required for data enhancement, smoothing the data, interpreting the terrain etc.
- Creating a proximity analysis to determine the distances among the areas of interest.
- Performing image enhancement using image processing algorithms for the raster data.
- Performing analysis related to networks of specific type like road network.

The GIS also requires the process of visualisation in order to display the data in a proper visual.

Thus, GIS is not a database that can be implemented using either the relational or object oriented database alone. Much more needs to be done to support them. A detailed discussion on these topics is beyond the scope of this unit.

## 1.4 GNOME DATABASES

One of the major areas of application of information technology is in the field of Genetics. Here, the computer can be used to create models based on the information obtained about genes. This information models can be used to study:

- the transmission of characteristics from one generation to next,
- the chemical structure of genes and the related functions of each portion of structure, and
- the variations of gene information of all organisms.

Biological data by nature is enormous. Bioinformation is one such key area that has emerged in recent years and which, addresses the issues of information management of genetic data related to DNA sequence. A detailed discussion on this topic is beyond the scope of this unit. However, let us identify some of the basic characteristics of the biological data.

**Biological Data – Some Characteristics:**

- Biological data consists of complex structures and relationships.
- The size of the data is very large and the data also has a lot of variations across the same type.
- The schema of the database keeps on evolving once or twice a year moreover, even the version of schema created by different people for the same data may be different.
- Most of the accesses to the database would be read only accesses.
- The context of data defines the data and must be preserved along with the data.
- Old value needs to be kept for future references.
- Complex queries need to be represented here.

The Human Genome Initiative is an international research initiative for the creation of detailed genetic and physical maps for each of the twenty-four different human chromosomes and the finding of the complete deoxyribonucleic acid (DNA) sequence of the human genome. The term Genome is used to define the complete genetic information about a living entity. A genetic map shows the linear arrangement of genes or genetic marker sites on a chromosome. There are two types of genetic maps– genetic linkage maps and physical maps. Genetic linkage maps are created on the

basis of the frequency with which genetic markers are co-inherited. Physical maps are used to determine actual distances between genes on a chromosome.

The Human Genome Initiative has six strong scientific objectives:

- to construct a high-resolution genetic map of the human genome,
- to produce a variety of physical maps of the human genome,
- to determine the complete sequence of human DNA,
- for the parallel analysis of the genomes of a selected number of well-characterised non-human model organisms,
- to create instrumentation technology to automate genetic mapping, physical mapping and DNA sequencing for the large-scale analysis of complete genomes,
- to develop algorithms, software and databases for the collection, interpretation and dissemination of the vast quantities of complex mapping and sequencing data that are being generated by human genome research.

Genome projects generate enormous quantities of data. Such data is stored in a molecular database, which is composed of an annotated collection of all publicly available DNA sequences. One such database is the Genbank of the National Institutes of Health (NIH), USA. But what would be the size of such a database? In February 2000 the Genbank molecular database contained 5,691,000 DNA sequences, which were further composed of approximately 5,805,000,000 deoxyribonucleotides.

One of the major uses of such databases is in computational Genomics, which refers to the applications of computational molecular biology in genome research. On the basis of the principles of the molecular biology, computational genomics has been classified into three successive levels for the management and analysis of genetic data in scientific databases. These are:

- Genomics.
- Gene expression.
- Proteomics.

### 1.4.1 Genomics

Genomics is a scientific discipline that focuses on the systematic investigation of the complete set of chromosomes and genes of an organism. Genomics consists of two component areas:

- **Structural Genomics** which refers to the large-scale determination of DNA sequences and gene mapping, and
- **Functional Genomics**, which refers to the attachment of information concerning functional activity to existing structural knowledge about DNA sequences.

**Genome Databases**

Genome databases are used for the storage and analysis of genetic and physical maps. Chromosome genetic linkage maps represent distances between markers based on meiotic re-combination frequencies. Chromosome physical maps represent distances between markers based on numbers of nucleotides.



Genome databases should define four data types:

- Sequence
- Physical
- Genetic
- Bibliographic

Sequence data should include annotated molecular sequences.

Physical data should include eight data fields:

- Sequence-tagged sites
- Coding regions
- Non-coding regions
- Control regions
- Telomeres
- Centromeres
- Repeats
- Metaphase chromosome bands.

Genetic data should include seven data fields:

- Locus name
- Location
- Recombination distance
- Polymorphisms
- Breakpoints
- Rearrangements
- Disease association
- Bibliographic references should cite primary scientific and medical literature.

**Genome Database Mining**

Genome database mining is an emerging technology. The process of genome database mining is referred to as computational genome annotation. Computational genome annotation is defined as the process by which an uncharacterised DNA sequence is documented by the location along the DNA sequence of all the genes that are involved in genome functionality.

### 1.4.2 Gene Expression

Gene expression is the use of the quantitative messenger RNA (mRNA)-level measurements of gene expression in order to characterise biological processes and explain the mechanisms of gene transcription. The objective of gene expression is the quantitative measurement of mRNA expression particularly under the influence of drugs or disease perturbations.

**Gene Expression Databases**

Gene expression databases provide integrated data management and analysis systems for the transcriptional expression of data generated by large-scale gene expression experiments. Gene expression databases need to include fourteen data fields:

- Gene expression assays
- Database scope
- Gene expression data
- Gene name
- Method or assay
- Temporal information

- Spatial information
- Quantification
- Gene products
- User annotation of existing data
- Linked entries
- Links to other databases
    - Internet access
    - Internet submission.

Gene expression databases have not established defined standards for the collection, storage, retrieval and querying of gene expression data derived from libraries of gene expression experiments.

**Gene Expression Database Mining**

Gene expression database mining is used to identify intrinsic patterns and relationships in gene expression data.

Gene expression data analysis uses two approaches:

- Hypothesis testing and
- Knowledge discovery.

Hypothesis testing makes a hypothesis and uses the results of perturbation of a biological process to match predicted results. The objective of knowledge discovery is to detect the internal structure of the biological data. Knowledge discovery in gene expression data analysis employs two methodologies:

- Statistics functions such as cluster analysis, and
- Visualisation.

Data visualisation is used to display the partial results of cluster analysis generated from large gene expression database cluster.

### 1.4.3 Proteomics

Proteomics is the use of quantitative protein-level measurements of gene expression in order to characterise biological processes and describe the mechanisms of gene translation. The objective of proteomics is the quantitative measurement of protein expression particularly under the influence of drugs or disease perturbations. Gene expression monitors gene transcription whereas proteomics monitors gene translation. Proteomics provides a more direct response to functional genomics than the indirect approach provided by gene expression.

**Proteome Databases**

Proteome databases also provide integrated data management and analysis systems for the translational expression data generated by large-scale proteomics experiments. Proteome databases integrate expression levels and properties of thousands of proteins with the thousands of genes identified on genetic maps and offer a global approach to the study of gene expression.

Proteome databases address five research problems that cannot be resolved by DNA analysis:

- Relative abundance of protein products,
- Post-translational modifications,
- Subcellular localisations,
- Molecular turnover and
- Protein interactions.



The creation of comprehensive databases of genes and gene products will lay the foundation for further construction of comprehensive databases of higher-level mechanisms, e.g., regulation of gene expression, metabolic pathways and signalling cascades.

**Proteome Database Mining**

Proteome database mining is used to identify intrinsic patterns and relationships in proteomics data. Proteome database mining has been performed in areas such as Human Lymphoid Proteins and the evaluation of Toxicity in drug users.

**Some Databases Relating to Genome**

The following table defines some important databases that have been developed for the Genome.

| Database Name | Characteristics | Database Problem Areas |
|---|---|---|
| GenBank | Keeps information on the DNA/RNA sequences and information on proteins | Schema is always evolving. This database requires linking to many other databases |
| GDB | Stores information on genetic map linkages as well as non-human sequence data | It faces the same problem of schema evolution and linking of database. This database also has very complex data objects |
| ACEDB | Stores information on genetic map linkages as well as non-human sequence data. It uses object oriented database technology | It also has the problem of schema evolution and linking of database. This database also has very complex data objects |

A detailed discussion on these databases is beyond the scope of this Unit. You may wish to refer to the further readings for more information.

## ☞ Check Your Progress 2

1) What is GIS? What are its applications?

……

……

……

……

2) List the requirements of a GIS.

……

……

……

……

3) What are the database requirements for Genome?

……

……

……

……

# 1.5 KNOWLEDGE DATABASES

Knowledge databases are the database for knowledge management. But what is knowledge management? Knowledge management is the way to gather, manage and use the knowledge of an organisation. The basic objectives of knowledge management are to achieve improved performance, competitive advantage and higher levels of innovation in various tasks of an organisation.

Knowledge is the key to such systems. Knowledge has several aspects:

- Knowledge can be implicit (called tacit knowledge) which are internalised or can be explicit knowledge.
- Knowledge can be captured before, during, or even after knowledge activity is conducted.
- Knowledge can be represented in logical form, semantic network form or database form.
- Knowledge once properly represented can be used to generate more knowledge using automated deductive reasoning.
- Knowledge may sometimes be incomplete. In fact, one of the most important aspects of the knowledge base is that it should contain upto date and excellent quality of information.

Simple knowledge databases may consist of the explicit knowledge of an organsiation including articles, user manuals, white papers, troubleshooting information etc. Such a knowledge base would provide basic solutions to some of the problems of the less experienced employees.

A good knowledge base should have:

- good quality articles having up to date information,
- a good classification structure,
- a good content format, and
- an excellent search engine for information retrieval.

One of the knowledge base technologies is based on deductive database technology. Let us discuss more about it in the next sub-section.

### 1.5.1 Deductive Databases

A deductive database is a database system that can be used to make deductions from the available rules and facts that are stored in such databases. The following are the key characteristics of the deductive databases:

- the information in such systems is specified using a declarative language in the form of rules and facts,
- an inference engine that is contained within the system is used to deduce new facts from the database of rules and facts,
- these databases use concepts from the relational database domain (relational calculus) and logic programming domain (Prolog Language),
- the variant of Prolog known as Datalog is used in deductive databases. The Datalog has a different way of executing programs than the Prolog and
- the data in such databases is specified with the help of facts and rules. For example, The fact that Rakesh is the manager of Mohan will be represented as:

  Manager(Rakesh, Mohan) having the schema:
  Manager(Mgrname, beingmangaed)



Similarly the following represents a rule:

Manager(Rakesh, Mohan) :- Managedby(Mohan, Rakesh)

- Please note that during the representation of the fact the data is represented using the attribute value only and not the attribute name. The attribute name determination is on the basis of the position of the data. For instance, in the example above Rakesh is the Mgrname.
- The rules in the Datalog do not contain the data. These are evaluated on the basis of the stored data in order to deduce more information.

Deductive databases normally operate in very narrow problem domains. These databases are quite close to expert systems except that deductive databases use, the database to store facts and rules, whereas expert systems store facts and rules in the main memory. Expert systems also find their knowledge through experts whereas deductive database have their knowledge in the data. Deductive databases are applied to knowledge discovery and hypothesis testing.

### 1.5.2 Semantic Databases

Information in most of the database management systems is represented using a simple table with records and fields. However, simple database models fall short of applications that require complex relationships and rich constructs to be represented using the database. So how do we address such a problem? Do we employ object oriented models or a more natural data model that represents the information using semantic models? Semantic modeling provides a far too rich set of data structuring capabilities for database applications. A semantic model contains far too many constructs that may be able to represent structurally complex inter-relations among data in a somewhat more natural way. Please note that such complex inter-relationships typically occur in commercial applications.

Semantic modeling is one of the tools for representing knowledge especially in Artificial Intelligence and object-oriented applications. Thus, it may be a good idea to model some of the knowledge databases using semantic database system.

Some of the features of semantic modeling and semantic databases are:

- these models represent information using high-level modeling abstractions,
- these models reduce the semantic overloading of data type constructors,
- semantic models represent objects explicitly along with their attributes,
- semantic models are very strong in representing relationships among objects, and
- they can also be modeled to represent IS A relationships, derived schema and also complex objects.

Some of the applications that may be supported by such database systems in addition to knowledge databases may be applications such as bio-informatics, that require support for complex relationships, rich constraints, and large-scale data handling.

## 1.6 INFORMATION VISUALISATION

Relational database offers one of the simplest forms of information visualisation in the form of the tables. However, with the complex database technologies and complex database inter-relationship structures, it is important that the information is presented to the user in a simple and understandable form. Information visualisation is the branch of Computer Graphics that deals with the presentation of digital images and interactions to the users in the form that s/he can handle with ease. Information visualisation may result in presentation of information using trees or graph or similar data structures.

Another similar term used in the context of visualisation is knowledge visualisation the main objective of which is to improve transfer of knowledge using visual formats that include images, mind maps, animations etc.

Please note the distinction here. Information visualisation mainly focuses on the tools that are supported by the computer in order to explore and present large amount of data in formats that may be easily understood.

You can refer to more details on this topic in the fifth semester course.

### ☞ Check Your Progress 3

1) What is a good knowledge base?

……

……

……

……

2) What are the features of deductive databases?

……

……

……

……

3) State whether the following are True or False:

(a) Semantic model is the same as an object. ☐

(b) IS A relationship cannot be represented in a semantic model. ☐

(c) Information visualisation is used in GIS. ☐

(d) Knowledge visualisation is the same as information visualisation. ☐

## 1.7 SUMMARY

This unit provides an introduction to some of the later developments in the area of database management systems. Multimedia databases are used to store and deal with multimedia information in a cohesive fashion. Multimedia databases are very large in size and also require support of algorithms for searches based on various media components. Spatial database primarily deals with multi-dimensional data. GIS is a spatial database that can be used for many cartographic applications such as irrigation system planning, vehicle monitoring system etc. This database system may represent information in a multi-dimensional way.

Genome database is another very large database system that is used for the purpose of genomics, gene expression and proteomics. Knowledge database store information either as a set of facts and rules or as semantic models. These databases can be utilised in order to deduce more information from the stored rules using an inference engine. Information visualisation is an important area that may be linked to databases from the point of visual presentation of information for better user interactions.



# 1.8 SOLUTIONS/ANSWERS

## Check Your Progress 1

1) a) Advanced technology in terms of devices that were digital in nature and support capture and display equipment.
   b) High speed data communication network and software support for multimedia data transfer.
   c) Newer application requiring multimedia support.

2) Medical media databases
   Bio-informatics
   Home media
   News etc.

3) Content can be of two basic types:
   a) Media Content
   b) Meta data, which includes media, format data, media keyword data and media feature data.

4) Some of the challenges are:
   a) Support for different types of input/output
   b) Handling many compressions algorithms and formats
   c) Differences in OS and hardware
   d) Integrating to different database models
   e) Support for queries for a variety of media types
   f) Handling different kinds of indices
   g) Data distribution over the world etc.

## Check Your Progress 2

1) GIS is a spatial database application where the spatial and non-spatial data is represented along with the map. Some of the applications of GIS are:
   - Cartographic applications
   - 3-D Digital modeling applications like land elevation records
   - Geographic object applications like traffic control system.

2) A GIS has the following requirements:
   - Data representation through vector and raster
   - Support for analysis of data
   - Representation of information in an integrated fashion
   - Capture of information
   - Visualisation of information
   - Operations on information

3) The data may need to be organised for the following three levels:
   - **Geonomics:** Where four different types of data are represented. The physical data may be represented using eight different fields.
   - **Gene expression:** Where data is represented in fourteen different fields
   - **Proteomics:** Where data is used for five research problems.

## Check Your Progress 3

1) A good knowledge database will have good information, good classification and structure and an excellent search engine.

2) They represent information using facts and rules
   New facts and rules can be deduced
   Used in expert system type of applications.

3) a) False b) False c) True d) False.



# UNIT 2 EMERGING DATABASE MODELS, TECHNOLOGIES AND APPLICATIONS - II

**Structure** **Page Nos.**


2.0 Introduction 21
2.1 Objectives 21
2.2 Mobile and Personal Database 22
2.2.1 The Basic Framework for Mobile Computing
2.2.2 Characteristics of Mobile Databases
2.2.3 Wireless Networks and Databases
2.3 Web Databses 24
2.4 Accessing Databases on different DBMSs 28
2.4.1 Open Database Connectivity (ODBC)
2.4.2 Java Database Connectivity (JDBC)
2.5 Digital Libraries 32
2.6 Data Grid 33
2.7 Summary 35
2.8 Solutions/Answers 36


## 2.0 INTRODUCTION

Database applications have advanced along with the advancement of technology from age old Relational Database Management Systems. Database applications have moved to Mobile applications, Web database applications, Digital libraries and so on. The basic issues to all advanced applications are in making up-to-date data available to the user, in the desired format related anywhere, anytime. Such technology requires advanced communication technologies; advanced database distribution models and advanced hardware. In this unit, we will introduce the concepts of mobile and personal databases, web databases and the issues concerned with such databases. In addition, we will also discuss the concepts and issues related to Digital Libraries, Data Grids and Wireless Communication and its relationship with Databases.

Mobile and personal databases revolve around a mobile computing environment and focuses on the issues that are related to a mobile user. Web databases on the other hand, are very specific to web applications. We will examine some of the basic issues of web databases and also discuss the creation of simple web database development. We will examine some of the concepts associated with ODBC and JDBC – the two standards for connecting to different databases. Digital Libraries are a common source of information and are available through the databases environment. Such libraries are equipped to handle information dissemination, searching and reusability. Data grids allow distributed storage of data anywhere as one major application unit. Thus, all these technologies have a major role to play in the present information society. This unit discusses the issues and concerns of technology with respect to these databases system.

## 2.1 OBJECTIVES

After going through this unit, you should be able to:

- define the requirements of a mobile database systems;
- identify and create simple web database;
- use JDBC and ODBC in databases;
- explain the concept of digital libraries, and
- define the concept of a data grid.

# 2.2 MOBILE AND PERSONAL DATABASE

In recent years, wireless technology has become very useful for the communication of information. Many new applications of wireless technology are already emerging such as electronic valets which allow electronic money anywhere, anytime, mobile cells, mobile reporting, etc. The availability of portable computing devices that supports wireless base a communication helps database users access relevant information from anywhere, at anytime. Let us discuss mobile database systems and the issue related to it in this section.

### 2.2.1 The Basic Framework for Mobile Computing

Mobile Computing is primarily a distributed architecture. This basic architecture is shown in *Figure 1*.

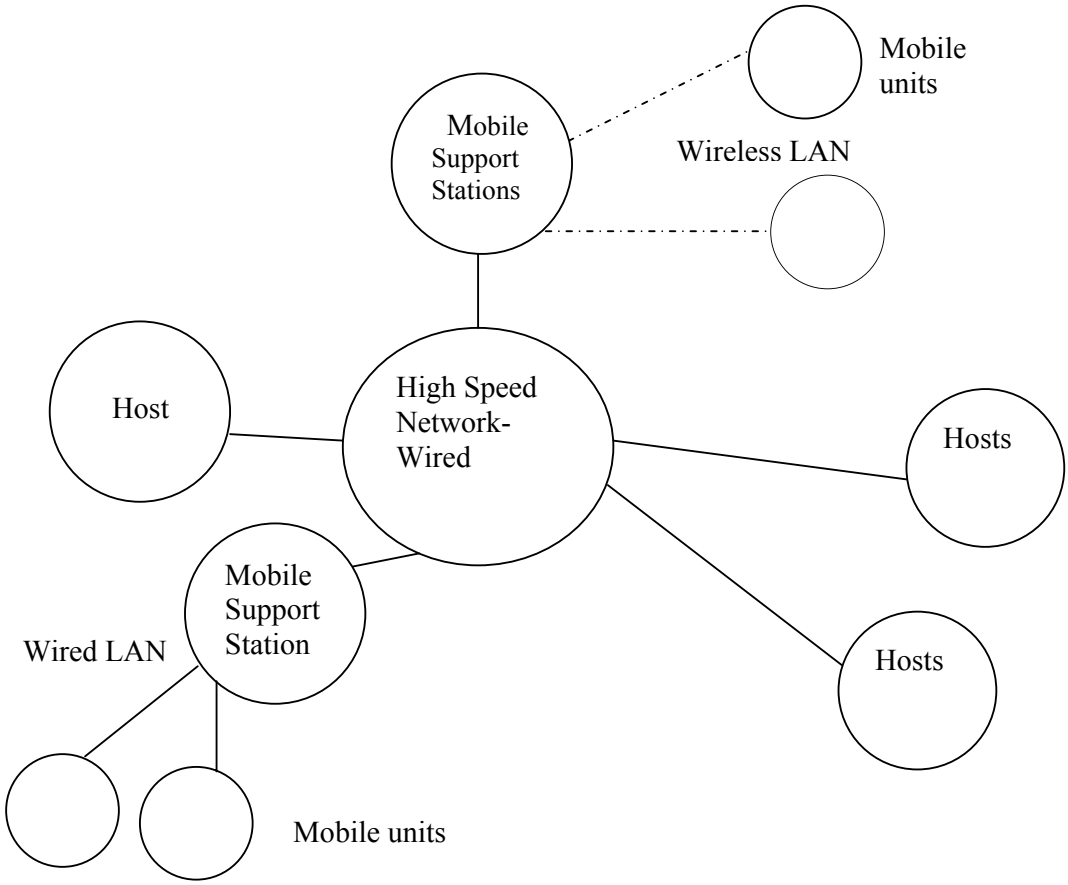



**Figure 1: A typical view for Mobile Computing**

A mobile computing environment consists of:

- host Computers which are fixed,
- mobile support stations,
- mobile client units, and
- networks.

Please note that the basic network may be a wired network but there are possibilities of Wireless LANs as well.

### 2.2.2 Characteristics of Mobile Databases

The mobile environment has the following characteristics:

1) *Communication Latency:* Communication latency results due to wireless transmission between the sources and the receiver. But why does this latency occur? It is primarily due to the following reasons:

   a) due to data conversion/coding into the wireless formats,
   b) tracking and filtering of data on the receiver, and
   c) the transmission time.

2) *Intermittent wireless connectivity:* Mobile stations are not always connected to the base stations. Sometimes they may be disconnected from the network.

3) *Limited battery life:* The size of the battery and its life is limited. Information communication is a major consumer of the life of the battery.

4) *Changing location of the client:* The wireless client is expected to move from a present mobile support station to an other mobile station where the device has been moved. Thus, in general, the topology of such networks will keep on changing and the place where the data is requested also changes. This would require implementation of dynamic routing protocols.

Because of the above characteristics the mobile database systems may have the following features:

Very often mobile databases are designed to work offline by caching replicas of the most recent state of the database that may be broadcast by the mobile support station. The advantages of this scheme are:

- it allows uninterrupted work, and
- reduces power consumption as data communication is being controlled.

However, the disadvantage is the inconsistency of data due to the communication gap between the client and the server.

However, some of the challenges for mobile computing are:

(i) *Scalability:* As the number of stations increase, latency increases. Thus, the time for servicing the client also increases. This results in increase in latency, thus more problems are created for data consistency.

   *The solution:* Do data broadcasting from many mobile stations thus, making the most recent information available to all, thus eliminating the enough latency time.

(ii) *Data Mobile problem:* Client locations keeps on changing in such networks thus, keeping track of the location of the client is important for, the data server and data should be made available to the client from the server which is minimum latency way from the client.

### 2.2.3 Wireless Networks and Databases

A mobile support station covers a large geographic area and will service all the mobile hosts that are available in that wireless geographical area – sometimes referred to as the cell. A mobile host traverses many mobile zones or cells, requiring its information to be passed from one cell to another, not necessarily adjacent, as there is an overlapping of cells. However, with the availability of wireless LANS now, mobile hosts that are in some LAN areas may be connected through the wireless LAN rather than a wide area cellular network. Thus, reducing the cost of communication as well as overheads of the cell movement of the host.



With wireless LAN technology it is now possible that some of the local mobile hosts may communicate directly with each other without the mobile support station. However, please note that such a communication should be based on some standard protocol/technology. Fortunately, the Blue tooth standard is available. This standard allows wireless connectivity on short ranges (10-50 metres) and has nearly 1 megabytes per second speed, thus allowing easy use of PDA mobile phones and intelligent devices. Also there are wireless LAN standards such as 801.11 and 802.16. In addition, wireless technology has improved and packet based cellular networking has moved to the third generation or is even more advanced allowing high-speed digital data transfer and applications. Thus, a combination of all these technologies viz., blue tooth, wireless LANs and 3G cellular networks allows low cost infrastructure for wireless communication of different kinds of mobile hosts.

This has opened up a great potential area for mobile applications where large, real-time, low cost database applications can be created for areas such as, just in time accounting, teaching, monitoring of resources and goods etc. The major advantage here would be that the communication of real time data through these networks would now be possible at low costs.

Major drawback of mobile databases – is the limitation of power and available size of the display unit has found newer technologies like use of flash, power-saving disks, low-powered and power saving displays. However, the mobile devices since are normally smaller in size requires creation of presentation standards. One such protocol in the areas of wireless networking is the Wireless Access Protocol (WAP).

Thus, mobile wireless networks have opened up the potential for new mobile database applications. These applications may be integrated into very strong web-based, intelligent, real time applications.

## 2.3 WEB DATABASES

The term web database can be used in at least two different way:

**Definition 1:** A web database may be defined as the organised listing of web pages for a particular topic. Since the number of web pages may be large for a topic, a web database would require strong indexing and even stronger spider or robot based search techniques.

We are all familiar with the concept of web searching. Web searching is usually text based and gathers hundreds or thousands of web pages together as a result of a search. But how can we sort through these pages solve this problem? How do the web database could help in this regard. However, this is not the basic issue for the discussion in this section. We would like to concentrate on the second definition of a web database.

**Definition 2:** A web database is a database that can be accessed through the web.

This definition actually defines a web application with the database as the backend. Let us discuss more about web database application systems.

**Features of a Web Database Application**

A Web database application should have the following features:

- it should have proper security,
- it should allow multiple users at a time,
- it should allow concurrent online transactions,
- it should produce a response in finite time,

- its response time should be low,
- it should support client – server (2-tier or 3 tier) architecture through the website.
- on the three-tier architecture an additional application server tier operates between the client and server.

**Creating a Web Database Application**

How do we create a web database? Well, we would like to demonstrate this with the help of a very simple example. To implement the example, we use Active Sever Pages (ASP), which is one of the old popular technologies with the backend as Ms-Access. ASP is a very easy technology to use. Although, Microsoft Access does not support websites with a lot of traffic, it is quite suitable for our example. A web database makes it very convenient to build a website.

Let us show the process of creating a web database. We would need to follow the following steps to create a student database and to make it accessible as a web database.

- The first step is to create a database using Ms-Access with the following configuration:

| | | |
|---|---|---|
| Student-id | Text (10) | Primary Key |
| Name | Text (25) | |
| Phone | Text (12) | |

Now, you would need to enter some meaningful data into the database and save it with the name students.mdb.

- Put your database online by using ftp to transfer students.mdb to the web server on which you are allowed access. Do not put the file in the same directory in which your web site files are stored, otherwise, the entire databases may be downloaded by an unauthorised person. In a commercial set up it may be better to keep the data on the Database server. This database then can be connected through a Data Source Name (DSN) to the website. Let us now build the required interface from ASP to the Database. A simple but old method may be with connecting ASP using ActiveX Data Object (ADO) library. This library provides ASP with the necessary functionality for interacting with the database server.
- The first and most basic thing we need to do is to retrieve the contents of the database for display. You can retrieve database records using ADO Recordset, one of the objects of ADO.

  ```
  Dim recordsettest
  Set recordsettest = Server.CreateObject("ADODB.Recordset")
  ```

  The commands given above create a variable (recordsettest) to store new Recordset object using the Server object's CreateObject method.

- Now fill this Recordset with records from the database with its Open method. Open takes two parameters:
  - the table name that contains the records to be fetched and
  - the connection string for the database.

Now, the name of the table is straight forward, as we would obviously, create a table with a name. However, the connection string is slightly more complex. Since the ADO library is capable of connecting to many database servers and other data sources, the string must tell Recordset not only where to find the database (the path and file name) but also how to read the database, by giving the name of its database provider.



A database provider is a software that has a very important role. It allows ADO to communicate with the given type of database in a standard way. ADO the provider for MS-Access, SQL Server, Oracle, ODBC database servers etc. Assuming that we are using the provider Jet.OLEDB to connect to an Access database, the connection string would be:

Provider = Microsoft.Jet.OLEDB.version; Data Source = ~\student\student.mdb

A connection string for MS SQL may be like:

Provider = SQLOLEDB; Data Source = servername; Initial Catalog = database name; User Id = username; Password = password.

So a sample code could be:

```
Dim recordsettest
Dim db_conn 'The database connection string
recordsettest= Server.CreateObject ("ADODB.Recordset")
db_conn = "Provider=Microsoft.Jet.OLEDB.version; Data Source=
                                        ~\student\student.mdb
recordsetest.Open "student", db_conn
```

However, since many ASP pages on the site will require such a string, it is common to place the connection string in an application variable in the global workspace.

```
<SCRIPT LANGUAGE= "VBScript" RUNAT= "Server">
    Sub Application_OnStart ( )
        Dim db_conn
        db_conn= "Provider" = Microsoft.Jet.OLEDB.version;
        Data Source= ~\student\student.mdb Application ("db_conn") = db_conn
    End Sub
</SCRIPT>
```

The code to retrieve the contents of the student table will now include only the code for Recordset.

```
Dim recordsettest
Set recordsettest =  Server.CreateObject("ADODB.Recordset'')
recordsettest.Open "Student", Application("db_conn")
```

Thus, we have established the connection and can get information from the MS-Access database. But another problem remains.  How do we display the results?

- Now, we can access recordsets, like of the database table, the result sets with the data row as one database record. In our example, we have put the contents of the student table into the recordsetrest object. An opened Recordset keeps track of the current record. Initially the first record is the current record. A MoveNext method of the Recordset object moves the pointer to the next record in the set, if any. An EOF property of the Recordset will become true at the end of Recordset. Thus, to display student id and name, you may write the following code:

```
Do While Not recordsettest.EOF
    Response. Write "<li>  "& recordsettest ("students-id") &"
    Response.Write ''<p> "& recordsettest ("name") &" </p> </li>
    recordsetest(''name)
    recordsettest.MoveNext
Loop
If recordsettest.BOF Then
```



Response.Write " No students data in the database. "

Please note that the BOF property checks if the file is empty.

- Once you have completed the task then you must be close the recordset as:
  Recordsettest.Close
  This command sets free the connection to the database. As these connections may not be very large in numbers, therefore, they need not be kept open longer than it is necessary.

Thus, the final code may look like:

```
<html>
<head>
<title> Student Data</title>
</head>
<body>
<o1>
<%
Dim recordsettest
Set recordsettest =  Server.CreateObject("ADODB.Recordset'')
recordsettest.Open "Student", Application("db_conn")
Do While Not recordsettest.EOF
     Response. Write "<li>  "& recordsettest ("students-id") &"
     Response.Write ''<p> "& recordsettest ("name") &" </p> </li>
     recordsetest(''name)
     recordsettest.MoveNext
Loop
If recordsettest.BOF Then
    Response.Write "<p>No students data in the database. </p>"
Recordsettest.Close
```

Save this file on the web server. Now, test this program by storing suitable data in the database. This application should produce the simple list of the student. However, you can create more complex queries to data using SQL. Let us try to explain that with the help of an example.

**Example:** Get the list of those students who have a phone number specified by you. More than one student may be using use this phone number as a contact number. Please note that in actual implementation you would need to create more complex queries.

Please also note that the only change we need to perform in the programming given above is, in the Open statement where we need to insert SQL command, such that on opening the recordset only the required data from the database is transferred to it. Thus, the code for this modified database access may be:

```
<html>
<head>
<title> Student Data</title>
</head>
<body>
<o1>
<%
Dim recordsettest
Set recordsettest =  Server.CreateObject("ADODB.Recordset'')
recordsettest.Open "SELECT student-id, name  FROM Student
                    WHERE phone = " & Request ("phone") ",
                Application("db_conn")
```



```
Do While Not recordsettest.EOF
     Response.Write "<li>  "& recordsettest ("students-id") &"
     Response.Write "<p> "& recordsettest ("name") &" </p> </li>
     recordsetest(''name)
     recordsettest.MoveNext
Loop
If recordsettest.BOF Then
    Response.Write "<p>No students data in the database. </p>"
Recordsettest.Close
```

You can build on more complex queries. You may also refer to more advanced ASP versions and connections. A detailed discussion on these is beyond the scope of the unit.

☞ **Check Your Progress 1**

1) What are the different characteristics of mobile databases?

……

……

……

2) What are the advantages of using wireless LAN?

……

……

……

3) What are the steps required to create a web database?

……

……

……

## 2.4 ACCESSING DATABASES ON DIFFERENT DBMSS

Database access from different interfaces may require some standards. The two standards that are quite useful are ODBC for accessing a database from any DBMS environment that support ODBC and JDBC that allows JAVA program access databases. Let us discuss them in more detail in this unit.

### 2.4.1 Open Database Connectivity (ODBC)

Open Database Connectivity (ODBC) is a standard API (Application Programming Interface) that allows access to as database that is a part of any DBMS that supports this standard. By using ODBC statements in your program you may access any database in MS-Access, SQL Server, Oracle, DB2, even an Excel file, etc. to work with ODBC driver for each of the database that is being accessed from the ODBC. The ODBC allows program to use standard SQL commands for accessing database. Thus, you need not master the typical interface of any specific DBMS.

For implementing ODBC in a system the following components will be required:

- the applications themselves,
- a core ODBC library, and
- the database drives for ODBC.



The library functions as the transitory/ interpreter between the application and the database drivers. Database drivers are written to hide DBMS specific details. Such layered configurations allow you to use standard types and features in your DBMS applications without, knowing details of specific DBMS that the applications may encounter. It also simplifies the implementations of database driver as they need to know only the core library. This makes ODBC modular. Thus, ODBC provides the standard data access using ODBC drivers that exist for a large variety of data sources including drivers for non-relational data such as spreadsheet, XML files.

ODBC also has certain disadvantages. These are:

- if a large number of client machines require different drivers, and DLLs are to be connected through ODBC then it has a complex and large administration overhead. Thus, large organisations are on the lookout for server side ODBC technology.

- the layered architecture of ODBC may introduce minor performance penalty.

The Call Level Interface (CLI) specifications of SQL are used by the ODBC as its base. The ODBC and its applications are becoming stronger. For example, Common Request Broker Architecture (CORBA) that is distributed object architecture, and the Persistent Object Service (POS) – an API, are the superset of both the Call-Level Interface and ODBC. If you need to write database application program in JAVA, then you need Java database Connectivity (JDBC) application program interface. From JDBC you may use a JDBC-ODBC "bridge" program to reach ODBC-accessible database.

***The designers of ODBC wanted to make it independent of any programming language, Operating System or DBMS.***

**Connecting Database to ODBC**

A database application needs to have the following types of statements for connecting to ODBC:

- linkage of core ODBC Library to client program,
- call results in communication to the request of the client library to the server, and
- results fetched at the client end.

But how do we actually write a code using ODBC? The following steps are needed to write a code using ODBC:

- establish the connection,
- run a SQL query that may be written by using embedded SQL, and
- release the connection.

But how do we establish the connection? Some of the ways of establishing a connection are:

*An ODBC/DSN connection to MySqL using ASP:*

```
Dim db_conn
set db_conn = server.createobject("adodb.connection")
db_conn.open("dsn=DSNmysql; uid = username; pwd = password; database =
student")
```



What is a DSN in the connection above? It is a Data Source Name that has been given to a specific user connection. DSN are more secure as you have to be the user as defined in DSN otherwise, you will not be allowed to use the data.

*ODBC setting for Oracle:*

To set a connection you may need to edit the odbc.ini file in certain installations of Oracle. The basic objective here is to allow you to connection to the relational data source of Oracle. Let us assume that we are using Oracle Version 8 and trying to connect to an Oracle data source oradata using an ODBC driver. You need to write the following:

```
[ODBC data Sources]
oradata=Oracle8 Source Data
…
[oradata]
Driver = / . . . /odbc/drivername
Description=my oracle source
ServerName=OracleSID
```

The last line defines the name of an Oracle database defined in the environment file of Oracle.

Once the connection is established through the required ODBC driver to the database through, the user-id and password, you can write the appropriate queries using SQL. Finally, you need to close the connection using close( ) method.

### 2.4.2 Java Database Connectivity (JDBC)

Accessing a database in Java requires, Java Database Connectivity (JDBC). JDBC allows you to accesses a database in your application and applets using a set of JDBC drivers.

**What is JDBC?**

Java Database Connectivity (JDBC) provides a standard API that is used to access databases, regardless of the DBMS, through JAVA. There are many drivers for JDBC that support popular DBMSs. However, if no such driver exits for the DBMS that you have selected then, you can use a driver provided by Sun Microsystems to connect to any ODBC compliant database. This is called JDBC to ODBC Bridge. For such an application, you may need to create, an ODBC data source for the database before, you can access it from the Java application.

**Connecting to a Database**

In order to connect to a database, let us say an oracle database, the related JDBC driver has to be loaded by the Java Virtual Machine class loader successfully.

```
// Try loading the oracle database driver
 try
{
      Class.forName(“oracle.jdbc.driver.OracleDriver”).newInstance( ) ;
}
  catch  (ClassNotFoundException ce) // driver not found
{
          System.err.println (“Drivernotfound”);
          // Code to handle error
}
```

Now, you can connect to the database using the driver manager class that selects the appropriate driver for the database. In more complex applications, we may use different drivers to connect to multiple databases. We may identify our database using a URL, which helps in identifying the database. A JDBC URL starts with "jdbc:" that indicates the use of JDBC protocol.

A sample database URL may be

jdbc:oracle@db.ignou.ac.in:2000:student

To connect to the database, we need to connect with a username and password. Assuming it to be "username" and "password", the connection string would be:

```
// Try creating a database connection
Connection db_conn =
        DriverManager.getConnection
            (jdbc:oracle@db.ignou.ac.in.2000:student, "username","password")
```

Thus, you will now be connected. Now the next step is to execute a query.

You can create a query using the following lines:

```
Statement   stmt = db_conn.CreateStatement( )
try {
        stmt.executeQuery (
                // Write your SQL query using SQL and host language
        )
}  catch (……) {……}
stmt.close ( )
db_conn.close ( )
```

Thus, the JDBC standard allows you to handle databases through JAVA as the host language. Please note that you can connect to any database that is ODBC compliant through JAVA either through the specialised driver or through the JDBC – ODBC bridge if no appropriate driver exists.

### ☞ Check Your Progress 2

1) Why is there a need for ODBC?

……..……..……

……..……..……

……

2) What are the components required for implementing ODBC?

……

……

……

3) Why do you need JDBC? What happens when a DBMS does not have a JDBC driver?

……

……

……



## 2.5 DIGITAL LIBRARIES

Let us now, define a digital library.

" A digital library is a library that allows almost the same functionality as that of a traditional library, however, having most of its information resources in digital form that are stored using multimedia repositories. The information to the digital library may be through the web or intranet."

The following may be some of the objectives of the digital library:

- to allow online catalog we access,
- to search for articles and books,
- to permit search on multiple resources at a time,
- to refine the search to get better results,
- to save the search results for future use, and
- to access search results by just clicking on the result.

Please note that most of the objectives above can easily be fulfilled due to the fact that a digital library allows web access.

But what are the advantages of the digital library?

A digital library has the following advantages:

- very large storage space,
- low cost of maintenance,
- information is made available in rich media form,
- round the clock operation,
- multi-user access, and
- can become a part of the world knowledge base/knowledge society.

Now the next question is what is the cost involved in creating and maintaining Digital Libraries?

The following are the cost factors for the creation and maintenance of digital libraries:

- cost of conversion of material into digital form,
- cost of maintaining digital services including cost of online access, and
- cost of maintains archival information.

So far, we have seen the advantages of digital libraries, but do digital libraries have any disadvantages?

Following are some of the disadvantages of digital libraries:

- only information available public domain can be made available,
- a digital library does not produce the same environment as a physical library, and
- the person needs to be technology friendly in order to use the library.

At present many Universities offer digital library facility online.

**Functionality of Digital library**

A digital library supports the following functions:

- Searching for information: A digital library needs a very strong search facility. It should allow search on various indexes and keyword. It should have a distributed search mechanism to provide answers to individual users search.

- Content management: A digital library must have facility for the continuous updating of source information. Older a information in such cases may need to be archived.
- Licenses and rights management: Only appropriate/authorised users are given access to protected information. A library needs to protect the copyright.
- All the links to further information should be thoroughly checked
- The library should store the Meta data in a proper format. One such structure for storing the meta data is the Dublin core.
- Library information should be represented in standard data formats. One such format may be XML. The contents may be represented in XML, HTML, PDF, JPEG, GIF, TIFF etc.

**Technology Required for the Digital Library**

A digital library would require expensive technology. Some of these requirements may be:

- Very large, reliable storage technologies supporting tera or even peta bytes of information. You may use Storage Access Networks (SAN).
- The transfer rate of information from the storage point to the computer should be high in order to fulfil the request of many users.
- High Internet bandwidth to store many users at a time.
- A distributed array of powerful servers that may process large access requests at the same time.
- Very reliable library software. Such software are developed on top of RDBMS to support features of full text indexing, meta-data indexing etc.

## 2.6 DATA GRID

Let us first define the term data grid. The concept of a data grid is somewhat difficult to define. A data grid can be seen as the process of creating a virtual database across hardware of almost the entire data that exists in some form. Thus, the key concept is Virtual Database. The basic tenet behind a data grid is that an application need not know either the place or the DBMS where the data is stored; rather the application is only interested in getting the correct results. *Figure 2* shows a typical structure for a data grid.

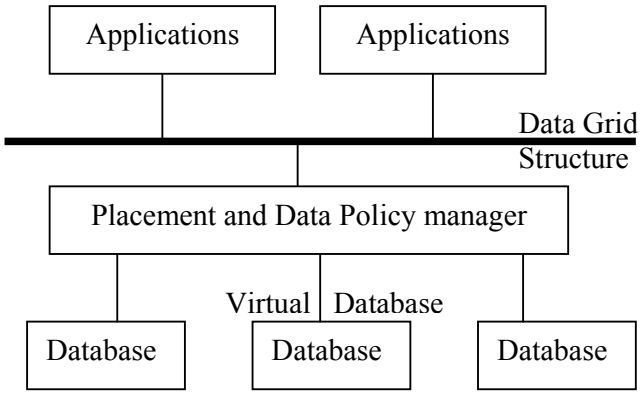



**Figure 2: A Data Grid**



A data grid should address the following issues:

- It should allow data security and domain specific description of data. Domain specific data helps in the identification of correct data in response to a query.
- It should have a standard way of representing information. [The possible candidates in this category may be XML].
- It should allow simple query language like SQL for accessing information.
- The data requirements should be fulfilled with a level of confidence, (that is there has to be a minimum quality of service in place).
- There needs to be a different role assigned to the data administrator. The data administrators should not be concerned with what the data represent. Data domain specification should be the sole responsibility of the data owner. Unfortunately, this does not happen in the present database implementations.
- The separation of the data administration and the data manager will help the database administrator to concentrate on data replication and query performance issues based on the DBMS statistics.

Thus, a data grid virtualises data. Many DBMSs today, have the ability to separate the roles of the database administrator and the data owner from the application access. This needs to be extended to the grid across a heterogeneous infrastructure.

**What are the implications of using a data grid?**

An application written in a data grid, references a virtual data source with a specified quality of service. Such an application need not be compiled again even if the data moves from one server to another or, there is a change in the infrastructure or due to changes in the data access methods. This happens as in the data grid, data sources are transparent to the application. Thus, it moves the concept of data independence further up the ladder.

A data grid permits a data provider with facilities that need not explicitly mention the location and structure of current and future applications. This information can be published in the data dictionary or Registry. A grid middleware then, is needed, to query the registry in order to locate this information for the applications. An administrator in a data grid is primarily concerned with the infrastructure and its optimum uses and providing the required qualities of service. Some of the things that the administrator is allowed to do based on the statistics stored in the data dictionary includes, - enabling replication, failure recovery, partitioning, changing the infrastructure etc.

**Data Grid Applications**

A data grid includes most of the relational database capabilities including schema integration, format conversion of data, distributed query support etc. In addition, a data grid should be scalable, reliable and should support efficient file access. These things are not easy to do. However, in this sub-section let us try to define some of the applications that a data grid may have.

**A Medical Grid:** Consider a situation in which a patient is admitted to a hospital where s/he has no record. By querying the data grid on a key field (may be a voter's id number) all the information about the previous medical history of the patient can be obtained using the data grid of hospitals. The data grid of hospitals may have their independent database systems, which is a part of the data grid. A hospital need not hand over all the confidential data as part of the data grid. Please note that the main

feature here is that the hospital is in complete control of its data. It can change, hide, and secure any part of its own database while, participating in the data grid federation. Thus, a virtual medical database that is partitioned across hospitals can be made. Please note that a query can be run across the whole set of hospitals and can retrieve consolidated results. A query may not need to retrieve data from all the hospitals participating in the grid to get significant information, for example, a query about the symptoms of a disease that has been answered by 70 percent of the hospitals can return meaningful results. However, on the other hand some queries would require all the data grid members to participate. For instance, a query to find the patient's complete medical record wherever its parts are stored, would require all the hospitals to answer.

Some of the major requirements of a data grid are:

- *Handling of failure of a data source*: A grid may have replicas and caches, but grid applications tend to access a data resource. But needs to be happens when this data source fails? A data grid flexible and it should have a middleware that automatically moves the operations to either another data resource with a similar data set or to multiple data resources each with a subset of the data.
- *Parallel access for local participant*: Some organisations may have very large data, thus, a query on that would take longer and such a query could be beyond the performance criteria set for the grid. Therefore, it may be a good idea to use a "virtual distributed" database to fetch data in parallels and keep processing it.
- *Global access through SQL*: A data grid would require dynamic selection of data sources. Therefore, it requires a complete SQL query transformation and optimisation capability.
- A data grid application may need to access data sources such as content management systems, Excel files, or databases not yet supported.

☞ **Check Your Progress 3**

1) What are the basic functions of a digital library?

……

……

……

2) What are the advantages of Data Grid?

……

……

……

3) What are different types of hardware and software required for digital libraries?

……

……

……

## 2.7 SUMMARY

This unit introduced you to several concepts related to emerging database applications. This unit also provides insight into some of the practical issues on connecting to databases from any DBMS or using JAVA, as well as analysed simple applications related to web database.



The unit introduces the requirements of mobile databases. Although most of the mobile applications in the past work on data broadcasting, however, this may change in the era when new wireless LAN technologies are available. This may give rise to new real time database applications. The ODBC and JDBC are two standards that can be used to connect to any database of any DBMS on any operating system and JAVA respectively. A web database may be the backend for the browser front end or the browser – application server tiers. However, it basically provides access to the data through the web. Digital libraries are a very useful application that has emerged in recent years. Digital libraries follow some of the meta-data and storage standards. The digital libraries also support distributed search. A Data grid is a virtual database created for the purpose of information sharing. The grid is loosely controlled.

## 2.8 SOLUTIONS/ANSWERS

### Check Your Progress 1

1) The following are the characteristics of mobile databases:
   - Mobile database relies on the broadcast of data
   - Mobile stations may be working in standalone mode most of the time
   - The data on a mobile workstation may be inconsistent
   - Mobile database may be made scalable
   - Mobile database are closer to distributed database technology
   - A mobile unit may be changing its physical location. It is to be reached at all locations.

2) Wireless LANs may allow low cost communication between two mobile units that are located in the same LAN area. Thus, it may result in reduced cost of operation.

3) The following steps are required to create a web database:
   - Create a database and put it on a database server.
   - Create a connection string to connect to the server through a valid username and password
   - Open the connection to bring in the required data based on the suitable query interactions
   - Format and display the data at the client site.

### Check Your Progress 2

1) ODBC allows using standard SQL commands to be used on any database on any DBMS. It gives application designer freedom from learning the features of individual DBMS, or OS etc. Thus, it simplifies the task of database programmers.

2) The following three components are required to implement ODBC:
   - The application
   - The core ODBC library
   - The database driver for ODBC

3) JDBC is an API that allows Java programmers to access any database through the set of this standard API. In case a JDBC driver is not available for a DBMS then the ODBC-JDBC bridge can be used to access data.



## Check Your Progress 3

1) A digital library supports

   - Content management
   - Search
   - License management
   - Link management
   - Meta data storage

2) Data Grid is helpful in the sharing of large amount of information on a particular topic. Thus, allowing a worldwide repository of information, while, on the other hand, still giving full control of information to the creator.

3) Digital libraries require:

   - Very large secondary storage
   - Distributed array of powerful servers
   - Large bandwidth and data transfer rate
   - A reliable library software.

# UNIT 3 POSTGRESQL

**Structure** **Page Nos.**


| | | |
|---|---|---|
| 3.0 | Introduction | 38 |
| 3.1 | Objectives | 38 |
| 3.2 | Important Features | 38 |
| 3.3 | PostgreSQL Architectural Concepts | 39 |
| 3.4 | User Interfaces | 41 |
| 3.5 | SQL Variation and Extensions | 43 |
| 3.6 | Transaction Management | 44 |
| 3.7 | Storage and Indexing | 46 |
| 3.8 | Query Processing and Evaluation | 47 |
| 3.9 | Summary | 49 |
| 3.10 | Solutions/Answers | 49 |


## 3.0 INTRODUCTION

PostgreSQL is an open-source object relational DBMS (ORDBMS). This DBMS was developed by the academic world, thus it has roots in Academia. It was first developed as a database called Postgres (developed at UC Berkley in the early 80s). It was officially called PostgreSQL around 1996 mostly, to reflect the added ANSI SQL compliant translator. It is one of the most feature-rich robust open-source database. In this unit we will discuss the features of this DBMS. Some of the topics that are covered in this unit include its architecture, user interface, SQL variation, transactions, indexes etc.

## 3.1 OBJECTIVES

After going through this unit, you should be able to:

- define the basic features of PostgreSQL;
- discuss the user interfaces in this RDBMS;
- use the SQL version of PostgreSQL, and
- identify the indexes and query processing used in PostgreSQL.

## 3.2 IMPORTANT FEATURES

PostgreSQL is an object relational database management system. It supports basic object oriented features including inheritance and complex data types along with special functions to deal with these data types. But, basically most of it is relational. In fact, most users of PostgreSQL do not take advantage of its extensive object oriented functionality. Following are the features of PostgreSQL that make it a very good DBMS:

- Full ANSI-SQL 92 compliance: It supports:
    - most of ANSI 99 compliance as well,
    - extensive support for the transactions,
    - BEFORE and AFTER triggers, and
    - implementation of stored procedures, constraints, referential integrity with cascade update/delete.

- Many high level languages and native interfaces can be used for creating user-defined database functions.
- You can use native SQL, PgSQL (postgres counterpart to Oracle PL/SQL or MS SQL Server's/Sybase TransactSQL), Java, C, C++, and Perl.
- Inheritance of table structures - this is probably one of the rarely used useful features.
- Built-in complex data types such as IP Address, Geometries (Points, lines, circles), arrays as a database field type and ability to define your own data types with properties, operators and functions on these user-defined data types.
- Ability to define Aggregate functions.
- Concept of collections and sequences.
- Support for multiple operating systems like Linux, Windows, Unix, Mac.
- It may be considered to be one of the important databases for implementing Web based applications. This is because of the fact that it is fast and feature rich. PostgreSQL is a reasonably fast database with proper support for web languages such as PHP, Perl. It also supports the ODBC and JDBC drivers making it easily usable in other languages such as ASP, ASP.Net and Java. It is often compared with MySQL - one of the fastest databases on the web (open source or non). Its querying speed is in line with MySQL. In terms of features though PostgreSQL is definitely a database to take a second look.

## 3.3 POSTGRESQL ARCHITECTURAL CONCEPTS

Before we dig deeper into the features of PostgreSQL, let's take a brief look at some of the basic concepts of Postgres system architecture. In this context, let us define how the parts of Postgres interact. This will make the understanding of the concepts simpler. *Figure 1* shows the basic architecture of the PostGreSQL on the Unix operating system.

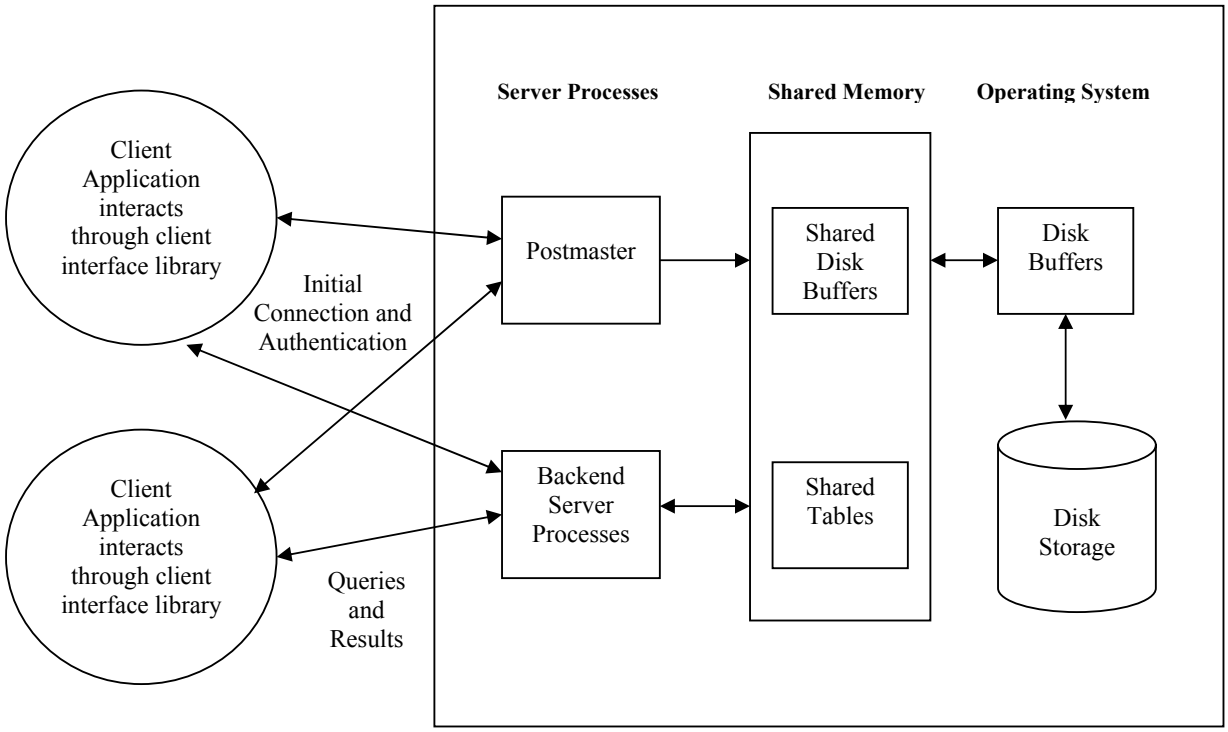



**Figure 1: The Architecture of PostGresSQL**

The Postgres uses a simple *process per-user* client/server model.



A session on PostgresSQL database consists of the following co-operating processes:

- A supervisory daemon process (also referred to as postmaster),
- The front-end user application process (e.g., the psql program), and
- One or more backend database server processes (the PostGres process itself).

The single postmaster process manages the collection of a database (also called an installation or site) on a single host machine. The client applications that want access to a database stored at a particular installation make calls to the client interface library. The library sends user requests over the network to the postmaster, in turn starts a new backend server process. The postmaster then connects the client process to the new server. Exam this point onwards, the client process and the backend server process communicate with each other without any intervention on the part of the postmaster. Thus, the postmaster process is always running – waiting for requests from the new clients. Please note, the client and server processes will be created and destroyed over a period of time as the need arises.

Can a client process, make multiple connections to a backend server process? The libpq library allows a single client to make multiple connections to backend server processes. However, please note, that these client processes are not multi-threaded processes. At present the multithreaded front-end/backend connections are not supported by libpq. This client server or font-end/back-end combination of processes on different machines files that can be accessed on a client machine that permits may not be accessible (or may only be accessed using a different filename) on the database server machine.

Please, also note that the postmaster and postgres servers run with the user-id of the Postgres *superuser* or the administrator. Please also note, that the Postgres superuser does not have to be a special user and also that the Postgres superuser should definitely not be the UNIX superuser (called root). All the files related to database belong to this Postgres superuser.

*Figure 2* shows the establishing of connection in PostgresSQL.

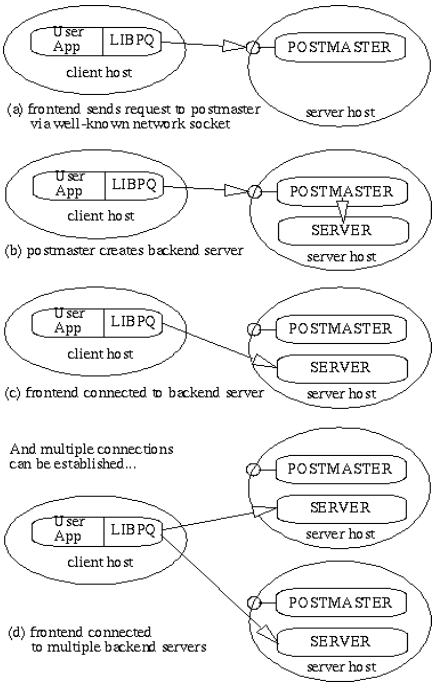



**Figure 2: Establishing connection in PostgresSQL**

# 3.4 USER INTERFACES

Having discussed the basic architecture of the PostgreSQL, the question that now arises is, how does one access databases? PostgreSQL have the following interfaces for the access of information:

- **Postgres terminal monitor programs (e.g. psql):** It is a SQL command level interface that allows you to enter, edit, and execute SQL commands interactively.
- **Programming Interface:** You can write a C program using the LIBPQ subroutine library. This allows you to submit SQL commands from the host language - C and get responses and status messages back to your program.

But how would you be referring to this interface? To do so, you need to install PostgreSQL on your machine. Let us briefly point out some facts about the installation of PostgreSQL.

*Installing Postgres on Your machine*: Since, Postgres is a client/server DBMS, therefore, as a user, you need the client's portion of the installation (an example of a client application interface is the interactive monitor psql). One of the common directory where PostGres may be installed on Unix machines is /usr/local/pgsql. Therefore, we will assume that Postgres has been installed in the directory/usr/local/pgsql. If you have installed PostGres in a different directory then you should substitute this directory name with the name of that directory. All PostGres commands are installed in the directory /usr/local/pgsql/bin. Therefore, you need to add this directory to your shell command path in Unix.

For example, on the Berkeley C shell or its variants such as csh or tcsh, you need to add:

```
% set path = ( /usr/local/pgsql/bin path )
```

in the .login file in the home directory.

On the Bourne shell or its variants such as sh, ksh, or bash, you need to add:

```
% PATH=/usr/local/pgsql/bin PATH
% export PATH
```

to the profile file in your home directory.

**Other Interfaces**

Some other user interfaces that are available for the PostGres are:

**pgAdmin 3** from http://www.pgadmin.org for Windows/Linux/*BSD*/nix (Experimental Mac OS-X port). This interface was released under the Artistic License is a complete PostgreSQL administration interface. It is somewhat similar to Microsoft's Enterprise Manager and written in C++ and wxWindows. It allows administration of almost all database objects and ad-hoc queries.

**PGAccess** from http://www.pgaccess.org for most platforms is the original PostgreSQL GUI. It is a MS Access-style database browser that has been written in Tcl/Tk. It allows browsing, adding and editing tables, views, functions, sequences, databases, and users, as well as graphically-assisted queries. A form and report designer are also under development.

Many similar open source tools are available on the Internet.
Let us now describe the most commonly used interface for PostgreSQL i.e., psql in more details.



**Starting the Interactive Monitor (psql)**

You can process an application from a client if:

- the site administrator has properly started the postmaster process, and
- you are authorised to use the database with the proper user id and password.

As of Postgres v6.3, two different styles of connections are supported. These are:

- TCP/IP network connections or
- Restricted database access to local (same-machine) socket connections only.

These choices are significant in case, you encounter problems in connecting to a database. For example, in case, you get the following error message from a Postgres command (such as psql or createdb):

*% psql template1*
*Connection to database 'postgres' failed.*
*connectDB() failed: Is the postmaster running and accepting connections at 'UNIX Socket' on port '5432'?*

or

*% psql -h localhost template1*
*Connection to database 'postgres' failed.*
*connectDB() failed: Is the postmaster running and accepting TCP/IP (with -i) connections at 'localhost' on port '5432'?*

It is because of the fact that either the postmaster is not running, or you are attempting to connect to the wrong server host. Similarly, the following error message means that the site administrator has started the postmaster as the wrong user.

*FATAL 1:Feb 17 23:19:55:process userid (2360) != database owner (268)*

**Accessing a Database**

Once you have a valid account then the next thing is to start accessing the database. To access the database with PostGres mydb database you can use the command:

*% psql mydb*

You may get the following message:

*Welcome to the POSTGRESQL interactive sql monitor:*
*Please read the file COPYRIGHT for copyright terms of POSTGRESQL*

*type \? for help on slash commands*
*type \q to quit*
*type \g or terminate with semicolon to execute query*
*You are currently connected to the database: template1*

*mydb=>*

The prompt indicates that the terminal monitor is ready for your SQL queries. These queries need to be input into a workspace maintained by the terminal monitor. The psql program also responds to escape codes (you must have used them while programming in C) that begin with the backslash character, "\". For example, you can get help on the syntax of PostGres SQL commands by typing:

*mydb=> \h*

Once you have completed the query you can pass the contents of the workspace to the Postgres server by typing:

*mydb=> \g*

This indicates to the server that it may process the query. In case you terminate the query with a semicolon, the "\g" is not needed. psql automatically processes the queries that are terminated by a semicolon.

You can store your queries in a file. To read your queries from such a file you may type:

*mydb=> \i filename*

To exit psql and return to UNIX, type

*mydb=> \q*

White space (i.e., spaces, tabs and new line characters) may be used in SQL queries. You can also enter comments. Single-line comments are denoted by "--". Multiple-line comments, and comments within a line, are denoted by "/* ... */".

### ☞ Check Your Progress 1

1) What are the basic features of PostgreSQL?

……

……

……

2) What are the basic processes in PostgreSQL?

……

……

……

3) What are the different types of interfaces in PostgreSQL?

……

……

……

## 3.5 SQL VARIATION AND EXTENSIONS

The SQL was standardised by American National Standards Institute (ANSI) in 1986. The International Standards Organisation (ISO) standardised it in 1987. The United States Government's Federal Information Processing Standard (*FIPS*) adopted the ANSI/ISO standard in 1989, a revised standard known commonly as *SQL89* or *SQL1*, was published.

The SQL89 standard was intentionally left incomplete to accommodate commercial DBMS developer interests. However, the standard was strengthened by the ANSI committee, with the *SQL92* standard that was ratified in 1992 (also called *SQL2*). This standard addressed several weaknesses in SQL89 and set forth conceptual SQL features which at that time exceeded the capabilities of the RDBMSs of that time. In fact, the SQL92 standard was approximately six times the length of its predecessor. Since SQL 92 was large, therefore, the authors of the standards defined three levels of SQL92 compliance: *Entry-level conformance* (only the barest improvements to SQL89), *Intermediate-level conformance* (a generally achievable set of major advancements), and *Full conformance* (total compliance with the SQL92 features).



More recently, in 1999, the ANSI/ISO released the *SQL99* standard (also called *SQL3*). This standard addresses some of the more advanced and previously ignored areas of modern SQL systems, such as object-relational database concepts, call level interfaces, and integrity management. SQL99 replaces the SQL92 levels of compliance with its own degrees of conformance: *Core SQL99* and *Enhanced SQL99*. PostgreSQL presently conforms to most of the Entry-level SQL92 standard, as well as many of the Intermediate- and Full-level features. Additionally, many of the features new in SQL99 are quite similar to the object-relational concepts pioneered by PostgreSQL (arrays, functions, and inheritance).

PostgreSQL also provide several extenstions to standard SQL. Some of these extensions are:

- PostgreSQL supports many non-standard types. These include abstract data types like complex, domains, cstring, record, trigger, void etc. It also includes polymorphic types like any array.

- It supports triggers. It also allows creation of functions which can be stored and executed on the server.

- It supports many procedural programming languages like PL/pqSQL, PL/Tcl, PL/Python etc.

## 3.6 TRANSACTION MANAGEMENT

Every SQL query is executed as a transaction. This results in some desirable properties, while processing such queries are all-or-nothing types when they are making modifications. This ensures the integrity and recoverability of queries. For example, consider the query:

```
UPDATE students SET marks = marks +10;
```

Assume that the query above has modified first 200 records and is in the process of modifying 201$^{st}$ record out of the 2000 records. Suppose a user terminates the query at this moment by resetting the computer, then, on the restart of the database, the recovery mechanism will make sure that none of the records of the student is modified. The query is required to be run again to make the desired update of marks. Thus, the PostGres has made sure that the query causes no recovery or integrity related problems.

This is a very useful feature of this DBMS. Suppose you were executing a query to increase the salary of employees of your organsiation by Rs.500 and there is a power failure during the update procedure. Without transactions support, the query may have updated records of some of the persons, but not all. It would be difficult to know where the UPDATE failed. You would like to know: “Which records were updated, and which ones were not?” You cannot simply re-execute the query, because some people who may have already received their Rs. 500 increment would also get another increase by Rs. 500/-. With the transactions mechanism in place, you need not bother about it for when the DBMS starts again, first it will recover from the failure thus, undoing any update to the data. Thus, you can simply re-execute the query.

### Multistatement Transactions

By default in PostGres each SQL query runs in its own transaction. For example, consider two identical queries:

```
mydb=> INSERT INTO table1 VALUES (1);
INSERT 1000 1
OR
mydb=> BEGIN WORK;
BEGIN
mydb=> INSERT INTO table1 VALUES (1);
INSERT 1000 1
mydb=> COMMIT WORK;
COMMIT
```

The former is a typical INSERT query. Before PostgreSQL starts the INSERT it automatically begins a transaction. It performs the INSERT, and then commits the transaction. This step occurs automatically for any query with no explicit transaction. However, in the second version, the INSERT uses explicit transaction statements. BEGIN WORK starts the transaction, and COMMIT WORK commits the transaction. Both the queries results in same database state, the only difference being the implied BEGIN WORK...COMMIT WORK statements. However, the real utility of these transactions related statements can be seen in the ability to club multiple queries into a single transaction. In such a case, either all the queries will execute to completion or none at all. For example, in the following transaction either both INSERTs will succeed or neither.

```
mydb=> BEGIN WORK;
BEGIN
mydb=> INSERT INTO table1 VALUES (1);
INSERT 1000 1
mydb=> INSERT INTO table1 VALUES (2);
INSERT 2000 1
mydb=> COMMIT WORK;
COMMIT
```

The PostgreSQL has implemented both two-phase locking and multi-version concurrency control protocols. The multi-version concurrency control supports all the isolation levels of SQL standards. These levels are:

- Read uncommitted
- Read committed
- Repeatable Read, and
- Serialisable.

### ☞ Check Your Progress 2

1) List the add-on non-standard types in PostgreSQL?

……

……

……

2) How are the transactions supported in PostgreSQL?

……

……

……



3) State True or False

a) PostgreSQL is fully compliant with SQL 99 standard. ☐

b) PostgreSQL supports server-based triggers. ☐

c) PostgreSQL supports all levels of isolation as defined by SQL standard ☐

d) COMMIT is not a keyword in PostgrSQL. ☐

## 3.7 STORAGE AND INDEXING

After discussing the basics of transactions let us discuss some of the important concepts used in PostgreSQL from the point of view of storage and indexing of tables. An interesting point here is that PostgreSQL defines a number of *system columns* in all tables. These system columns are normally invisible to the user, however, explicit queries can report these entries. These columns, in general, contains *meta-data* i.e., data about data contained in the records of a table.

Thus, any record would have attribute values for the system-defined columns as well as the user-defined columns of a table. The following table lists the system columns.

| Column Name | Description |
|---|---|
| oid (object identifier) | The unique object identifier of a record.<br>It is automatically added to all records.<br>It is a 4-byte number.<br>It is never re-used within the same table. |
| tableoid (table object identifier) | The oid of the table that contains a row.<br>The pg_class system table relates the name and oid of a table. |
| xmin (transaction minimum) | The transaction identifier of the inserting transaction of a tuple. |
| Cmin (command minimum) | The command identifier, starting at 0, is associated with the inserting transaction of a tuple. |
| xmax (transaction maximum) | The transaction identifier of a tuple's deleting transaction.<br>If a tuple has not been deleted then this is set to zero. |
| cmax (command maximum) | The command identifier is associated with the deleting transaction of a tuple. Like xmax, if a tuple has not been deleted then this is set to zero. |
| ctid (tuple identifier) | This identifier describes the physical location of the tuple within the database.<br>A pair of numbers are represented by the ctid: the block number, and tuple index within that block. |

**Figure 3: System Columns**

If the database creator does not create a primary key explicitly, it would become difficult to distinguish between two records with identical column values. To avoid such a situation PostgreSQL appends every record with its own *object identifier* number, or *OID*, which is unique to that table. Thus, no two records in the same table will ever have the same OID, which, also mean that no two records are identical in a table. The oid makes sure of this.

Internally, PostgreSQL stores data in operating system files. Each table has its own file, and data records are stored in a sequence in the file. You can create an index on the database. An index is stored as a separate file that is sorted on one or more columns as desired by the user. Let us discuss indexes in more details.

**Indexes**

Indexes allow fast retrieval of specific rows from a table. For a large table using an index, finding a specific row takes fractions of a second while non-indexed entries will require more time to process the same information. PostgreSQL does not create indexes automatically. Indexes are user defined for attributes or columns that are frequently used for retrieving information.

For example, you can create an index such as:

```
mydb=> CREATE INDEX stu_name ON Student (first_name);
```

Although you can create many indexes they should justify the benefits they provide for retrieval of data from the database. Please note that an index adds on overhead in terms of disk space, and performance as a record update may also require an index update. You can also create an index on multiple columns. Such multi-column indexes are sorted by the first indexed column and then the second indexed column.

PostgreSQL supports many types of index implementations. These are:

- **B-Tree Indexes:** These are the default type index. These are useful for comparison and range queries.
- **Hash Indexes:** This index uses linear hashing. Such indexes are not preferred in comparison with B-tree indexes.
- **R-Tree indexes:** Such index are created on built-in spatial data types such as box, circle for determining operations like overlap etc.
- **GiST Indexes:** These indexes are created using Generalised search trees. Such indexes are useful for full text indexing, and are thus useful for retrieving information.

## 3.8 QUERY PROCESSING AND EVALUATION

This section provides a brief introduction to the query processing operations of PostgreSQL. It will define the basic steps involved in query processing and evaluation in the PostgreSQL. The query once submitted to PostgreSQL undergoes the following steps, (in sequential order) for solving the query:

- A connection from the client application program to the PostgreSQL server is established. The application program transmits the query to the server and waits for the server to process the query and return the results.

- The *parser* at the server checks the syntax of the query received from the client and creates a *query tree.*

- The *rewrite system* takes the query tree created by the parser as the input, and selects the *rules* stored in the *system catalogues* that may apply to the query tree. It then performs the transformation given as per the *rules*. It also rewrites any query made against a view to a query accessing the base tables.

- The *planner/optimiser* takes the (rewritten) query tree and creates a *query plan* that forms the input to the *executor*. It creates a structured list of all the possible *paths* leading to the same result. Finally, the cost for the execution of each path is estimated and the cheapest path is chosen. This cheapest path is expanded into a complete query evaluation plan that the executor can use.



- The executor recursively steps through the query evaluation *plan tree* supplied by the planner and creates the desired output.

A detailed description of the process is given below:

A query is sent to the backend (it may be the query processor) via data packets that may result from database access request through TCP/IP on a remote database access or local Unix Domain sockets. The query is then loaded into a string, and passed to the parser, where the lexical scanner, **scan.l**, tokenises the words of the query string. The parser then uses another component **gram.y** and the tokens to identify the query type, such as, Create or Select queries. Now the proper query-specific structure is loaded.

The statement is then identified as complex (*SELECT / INSERT / UPDATE / DELETE*) or as simple, e.g., *CREATE USER, ANALYSE* etc. Simple utility commands are processed by statement-specific functions, however, for handling complex statements it need further detailing and processing.

Complex queries could be, *SELECT,* return columns of data or specify columns that need to be modified like *INSERT* and *UPDATE.* The references of these columns are converted to TargetEntry entries that may, be linked together to create the *target list* of the query. The target list is stored in **Query.targetList.**

Now, the Query is modified for the desired *VIEWS* or implementation of *RULES* that may apply to the query.

The optimiser then, creates the query execution plan based on the Query structure and the operations to be performed in order to execute the query. The Plan is passed to the executor for execution, and the results are returned to the client.

**Query Optimisation**

The task of the *planner/optimizer* is to create an optimal execution plan out of the available alternatives. The query tree of a given SQL query can be actually executed in a wide variety of different ways, each of which essentially producing the same set of results. It is not possible for the query optimiser to examine each of these possible execution plans to choose the execution plan that is expected to run the fastest. Thus, the optimiser must find a reasonable (non optimal) query plan in the predefined time and space complexity. Here a Genetic query optimiser is used by the PostgreSQL.

After the cheapest path is determined, a full-fledged *plan tree* is built in order to pass it to the executor. This represents the desired execution plan in sufficient detail for the executor to run it.

### ☞ Check Your Progress 3

1) What are system columns?

……

……

……

2) What are the different types of indexes in PostgreSQL?

……

……

……

3) What are the different steps for query evaluation?

……

……

……

## 3.9 SUMMARY

This unit provided an introduction to some of the basic features of the PostgreSQL DBMS. This is very suitable example of a DBMS available as Open Source software. This database provides most of the basic features of the relational database management systems. It also supports some of the features of object oriented programming like inheritance, complex data types, declarations of functions and polymorphism. Thus, it is in the category of object relational DBMS. PostgreSQL has features such as, the client server architecture with the Postmaster, server and client as the main processes. It supports all the basic features of SQL 92 and many more features recommended by SQL 99. PostgreSQL treats even single SQL queries as transaction implicitly. This helps in maintaining the integrity of the database at all times. PostgreSQL places many system related attributes in the tables defined by the users. Such attributes help the database with tasks that may be useful for indexing and linking of records. It also supports different types of indexes, which includes B-Tree, Hash, R-Tree and GiST types of indexes. Thus, making it suitable for many different types of application like spatial, multi-dimensional database applications. It has a standard process for query evaluation and optimisation.

## 3.10 SOLUTIONS/ANSWERS

### Check Your Progress 1

1) PostgreSQL supports the following features:

- ANSI SQL 2 compliance,
- Support for transactions, triggers, referential integrity and constraints,
- High level language support,
- Inheritance, complex data types and polymorphism,
- Built in complex data types like IP address,
- Aggregate functions, collections and sequences,
- Portability, and
- ODBC and JDBC drivers.

2) PostgreSQL has the following three processes:

- Postmaster
- Server process
- Client processes

3) PostgreSQL supports both the terminal monitor interfaces and program driven interfaces.

### Check Your Progress 2

1) Some of these types are: complex, domains, cstring, record, trigger, void etc.

2) Each SQL statement is treated as a transaction. It also has provision for multi-statement transactions.

3) (a) False (b) True (c) True (d) False



## Check Your Progress 3

1) System columns are added by PostgrSQL to all the tables. These are oid (object identifier), tableoid, xmin, xmax, cmax, ctid.

2) PostgreSQL supports the following four types of indexes: B-Tree, Hash, R-Tree and GiST.

3) The steps are:

- Query submission, the
- Parsing by the parser to query tree,
- Transformation of query by rewrite system
- Creation of query evaluation plan by the optimiser, and
- Execution by executor.

# UNIT 4 ORACLE

**Structure** **Page Nos.**


4.0 Introduction 51
4.1 Objectives 52
4.2 Database Application Development Features 52
4.2.1 Database Programming
4.2.2 Database Extensibility
4.3 Database Design and Querying Tools 57
4.4 Overview of Oracle Architecture 50
4.4.1 Physical Database Structures
4.4.2 Logical Database Structures
4.4.3 Schemas and Common Schema Objects
4.4.4 Oracle Data Dictionary
4.4.5 Oracle Instance
4.4.6 Oracle Background Processes
4.4.7 How Oracle Works?
4.5 Query Processing and Optimisation 71
4.6 Distributed Oracle 75
4.6.1 Distributed Queries and Transactions
4.6.2 Heterogenous Services
4.7 Data Movement in Oracle 76
4.7.1 Basic Replication
4.7.2 Advanced Replication
4.7.3 Transportable Tablespaces
4.7.4 Advanced Queuing and Streams
4.7.5 Extraction, Transformation and Loading
4.8 Database Administration Tools 77
4.9 Backup and Recovery in Oracle 78
4.10 Oracle Lite 79
4.11 Scalability and Performance Features of Oracle 79
4.11.1 Concurrency
4.11.2 Read Consistency
4.11.3 Locking Mechanisms
4.11.4 Real Application Clusters
4.11.5 Portability
4.12 Oracle DataWarehousing 83
4.12.1 Extraction, Transformation and Loading (ETL)
4.12.2 Materialised Views
4.12.3 Bitmap Indexes
4.12.4 Table Compression
4.12.5 Parallel Execution
4.12.6 Analytic SQL
4.12.7 OLAP Capabilities
4.12.8 Data Mining
4.12.9 Partitioning
4.13 Security Features of Oracle 85
4.14 Data Integrity and Triggers in Oracle 86
4.15 Transactions in Oracle 87
4.16 SQL Variations and Extensions in Oracle 88
4.17 Summary 90
4.18 Solutions/Answers 91


## 4.0 INTRODUCTION

The relational database concept was first described by Dr. Edgar F. Codd in an IBM research publication titled “System R4 Relational” which in 1970. Initially, it was unclear whether any system based on this concept could achieve commercial success. However, if we look back there have been many products, which support most of the features of relational database models and much more. Oracle is one such product



that was created by Relational Software Incorporated (RSI) in 1977. They released Oracle V.2 as the world's first relational database within a couple of years. In 1983, RSI was renamed Oracle Corporation to avoid confusion with a competitor named RTI. During this time, Oracle developers made a critical decision to create a portable version of Oracle (Version 3) that could run not only on Digital VAX/VMS systems, but also on Unix and other platforms. Since the mid-1980s, the database deployment model has evolved from dedicated database application servers to client/servers to Internet computing implemented with PCs and thin clients accessing database applications via browsers – and, to the grid with Oracle Database 10g.

Oracle introduced many innovative technical features to the database as computing and deployment models changed (from offering the first distributed database to the first Java Virtual Machine in the core database engine). Oracle also continues to support emerging standards such as XML and .NET.

We will discuss some of the important characteristics and features of Oracle in this unit.

## 4.1 OBJECTIVES

After going through this unit, you should be able to:

- define the features of application development in Oracle;
- identify tools for database design and query;
- describe the Oracle architecture;
- discuss the query processing and optimisation in Oracle;
- define the distributed Oracle database features;
- identify tools for database administration including backup and recovery;
- discuss the features of Oracle including scalability, performance, security etc., and
- define newer applications supported by Oracle.

## 4.2 DATABASE APPLICATION DEVELOPMENT FEATURES

The main use of the Oracle database system is to store and retrieve data for applications. Oracle has evolved over the past 20 years. The following table traces the historical facts about Oracle.

| Year | Feature |
|---|---|
| 1979 | Oracle Release 2—the first commercially available relational database to use SQL. |
| 1980-1990 | Single code base for Oracle across multiple platforms, Portable toolset, Client/server Oracle relational database, CASE and 4GL toolset, Oracle Financial Applications built on relational database. |
| 1991-2000 | Oracle Parallel Server on massively parallel platform, cost-based optimiser, parallel operations including query, load, and create index, Universal database with extended SQL via cartridges, thin client, and application server, Oracle 8 with inclusion of object-relational and Very Large Database (VLDB) features, Oracle8i added a new twist to the Oracle database – a combination of enhancements that made the Oracle8i database the focal point of the world of Internet (the i in 8i) computing.<br><br>Java Virtual Machine (JVM) was added into the database and Oracle |

| Year | Feature |
|---|---|
| | tools were integrated in the middle tier. |
| 2001 | Oracle9i Database Server is generally available: Real Application Clusters; OLAP and data mining API in the database. |
| 2003 | Oracle Database 10g enables grid (the g in 10g) computing. A grid is simply a pool of computers that provides needed resources for applications on an as-needed basis. The goal is to provide computing resources that can scale transparently to the user community, much as an electrical utility company can deliver power, to meet peak demand, by accessing energy from other power providers' plants via a power grid. Oracle Database 10g further reduces the time, cost, and complexity of database management through the introduction of self-managing features such as the Automated Database Diagnostic Monitor, Automated Shared Memory Tuning, Automated Storage Management, and Automated Disk Based Backup and Recovery. One important key to Oracle Database 10g's usefulness in grid computing is the ability to have a provision for CPUs and data. |

However, in this section we will concentrate on the tools that are used to create applications. We have divided the discussion in this section into two categories: database programming and database extensibility options. Later in this unit, we will describe the Oracle Developer Suite, a set of optional tools used in Oracle Database Server and Oracle Application Server development.

### 4.2.1 Database Programming

All flavors of the Oracle database include different languages and interfaces that allow programmers to access and manipulate data in the database. The following are the languages and interfaces supported by Oracle.

**SQL**

All operations on the information in an Oracle database are performed using SQL statements. A statement must be the equivalent of a complete SQL sentence, as in:

SELECT last_name, department_id FROM employees;

Only a complete SQL statement can run successfully. A SQL statement can be thought of as a very simple, but powerful, computer instruction. SQL statements of Oracle are divided into the following categories.

**Data Definition Language (DDL) Statements:** These statements create, alter, maintain, and drop schema objects. DDL statements also include statements that permit a user the right to grant other users the privilege of accessing the database and specific objects within the database.

**Data Manipulation Language (DML) Statements:** These statements manipulate data. For example, querying, inserting, updating, and deleting rows of a table are all DML operations. Locking a table and examining the execution plan of a SQL statement are also DML operations.

**Transaction Control Statements:** These statements manage the changes made by DML statements. They enable a user group changes into logical transactions. Examples include COMMIT, ROLLBACK, and SAVEPOINT.



**Session Control Statements:** These statements let a user control the properties of the current session, including enabling and disabling roles and changing language settings. The two session control statements are ALTER SESSION and SET ROLE.

**System Control Statements:** These statements change the properties of the Oracle database instance. The only system control statement is ALTER SYSTEM. It lets users change settings, such as the minimum number of shared servers, kill a session, and perform other tasks.

**Embedded SQL Statements:** These statements incorporate DDL, DML, and transaction control statements in a procedural language program. Examples include OPEN, CLOSE, FETCH, and EXECUTE.

**Datatypes**

Each attribute and constant in a SQL statement has a datatype, which is associated with a specific storage format, constraints, and a valid range of values. When you create a table, you must specify a datatype for each of its columns.

Oracle provides the following built-in datatypes:

- Character datatypes
- Numeric datatypes
- DATE datatype
- LOB datatypes
- RAW and LONG RAW datatypes
- ROWID and UROWID datatypes.

New object types can be created from any in-built database types or any previously created object types, object references, and collection types. Metadata for user-defined types is stored in a schema available to SQL, PL/SQL, Java, and other published interfaces.

An object type differs from native SQL datatypes in that it is user-defined, and it specifies both the underlying persistent data (attributes) and the related behaviours (methods). Object types are abstractions of real-world entities, for example, purchase orders.

Object types and related object-oriented features, such as variable-length arrays and nested tables, provide higher-level ways to organise and access data in the database. Underneath the object layer, data is still stored in columns and tables, but you can work with the data in terms of real-world entities – customers and purchase orders, that make the data meaningful. Instead of thinking in terms of columns and tables when you query the database, you can simply select a customer.

**PL/SQL**

PL/SQL is Oracle's procedural language extension to SQL. PL/SQL combines the ease and flexibility of SQL with the procedural functionality of a structured programming language, such as IF, THEN, WHILE, and LOOP.

When designing a database application, consider the following advantages of using stored PL/SQL:

- PL/SQL code can be stored centrally in a database. Network traffic between applications and the database is reduced, hence, both application and system performance increases. Even when PL/SQL is not stored in the database, applications can send blocks of PL/SQL to the database rather than individual SQL statements, thereby reducing network traffic.

- Data access can be controlled by stored PL/SQL code. In this case, PL/SQL users can access data only as intended by application developers, unless another access route is granted.

- PL/SQL blocks can be sent by an application to a database, running complex operations without excessive network traffic.

- Oracle supports PL/SQL Server Pages, so the application logic can be invoked directly from your Web pages.

The PL/SQL program units can be defined and stored centrally in a database. Program units are stored procedures, functions, packages, triggers, and anonymous transactions.

Procedures and functions are sets of SQL and PL/SQL statements grouped together as a unit to solve a specific problem or to perform a set of related tasks. They are created and stored in compiled form in the database and can be run by a user or a database application. Procedures and functions are identical, except that all functions always return a single value to the user. Procedures do not return values.

Packages encapsulate and store related procedures, functions, variables, and other constructs together as a unit in the database. They offer increased functionality (for example, global package variables can be declared and used by any procedure in the package). They also improve performance (for example, all objects of the package are parsed, compiled, and loaded into memory once).

**Java Features and Options**

Oracle8i introduced the use of Java as a procedural language with a Java Virtual Machine (JVM) in the database (originally called JServer). JVM includes support for Java stored procedures, methods, triggers, Enterprise JavaBeans (EJBs), CORBA, and HTTP. The Accelerator is used for project generation, translation, and compilation, and can also be used to deploy/install-shared libraries. The inclusion of Java within the Oracle database allows Java developers to level their skills as Oracle application developers as well. Java applications can be deployed in the client, Application Server, or database, depending on what is most appropriate. Oracle data warehousing options for OLAP and data mining provide a Java API. These applications are typically custom built using Oracle's JDeveloper.

**Large Objects**

Interest in the use of large objects (LOBs) continues to grow, particularly for storing non-traditional data types such as images. The Oracle database has been able to store large objects for some time. Oracle8 added the capability to store multiple LOB columns in each table. Oracle Database 10g essentially removes the space limitation on large objects.

**Object-Oriented Programming**

Support of object structures has been included since Oracle8i to allow an object-oriented approach to programming. For example, programmers can create user-defined data types, complete with their own methods and attributes. Oracle's object support includes a feature called Object Views through which object-oriented programs can make use of relational data already stored in the database. You can also store objects in the database as varying arrays (VARRAYs), nested tables, or index organised tables (IOTs).



### Third-Generation Languages (3GLs)

Programmers can interact with the Oracle database from C, C++, Java, COBOL, or FORTRAN applications by embedding SQL in those applications. Prior to compiling applications using a platform's native compilers, you must run the embedded SQL code through a precompiler. The precompiler replaces SQL statements with library calls the native compiler can accept. Oracle provides support for this capability through optional "programmer" precompilers for languages such as C and C++ (Pro*C) and COBOL (Pro*COBOL). More recently, Oracle added SQLJ, a precompiler for Java that replaces SQL statements embedded in Java with calls to a SQLJ runtime library, also written in Java.

### Database Drivers

All versions of Oracle include database drivers that allow applications to access Oracle via ODBC (the Open DataBase Connectivity standard) or JDBC (the Java DataBase Connectivity open standard). Also available are data providers for OLE DB and for .NET.

### The Oracle Call Interface

This interface is available for the experienced programmer seeking optimum performance. They may choose to define SQL statements within host-language character strings and then explicitly parse the statements, bind variables for them, and execute them using the Oracle Call Interface (OCI). OCI is a much more detailed interface that requires more programmer time and effort to create and debug. Developing an application that uses OCI can be time-consuming, but the added functionality and incremental performance gains often make spending the extra time worthwhile.

### National Language Support

National Language Support (NLS) provides character sets and associated functionality, such as date and numeric formats, for a variety of languages. Oracle9i featured full Unicode 3.0 support. All data may be stored as Unicode, or select columns may be incrementally stored as Unicode. UTF-8 encoding and UTF-16 encoding provides support for more than 57 languages and 200 character sets. Oracle Database 10g adds support for Unicode 3.2. Extensive localisation is provided (for example, for data formats) and customised localisation can be added through the Oracle Locale Builder. Oracle Database 10g includes a Globalisation Toolkit for creating applications that will be used in multiple languages.

## 4.2.2 Database Extensibility

The Internet and corporate intranets have created a growing demand for storage and manipulation of non-traditional data types within the database. There is a need for its extension to the standard functionality of a database for storing and manipulating image, audio, video, spatial, and time series information. These capabilities are enabled through extensions to standard SQL.

### Oracle Text and InterMedia

Oracle Text can identify the gist of a document by searching for themes and key phrases in the document.

Oracle interMedia bundles additional image, audio, video, and locator functions and is included in the database license. Oracle interMedia offer the following capabilities:

- The image portion of interMedia can store and retrieve images.
- The audio and video portions of interMedia can store and retrieve audio and video clips, respectively.
- The locator portion of interMedia can retrieve data that includes spatial coordinate information.

**Oracle Spatial Option**

The Spatial option is available for Oracle Enterprise Edition. It can optimise the display and retrieval of data linked to coordinates and is used in the development of spatial information systems. Several vendors of Geographic Information Systems (GIS) products now bundle this option and leverage it as their search and retrieval engine.

**XML**

Oracle added native XML data type support to the Oracle9i database and XML and SQL interchangeability for searching. The structured XML object is held natively in object relational storage meeting the W3C DOM specification. The XPath syntax for searching in SQL is based on the SQLX group specifications.

## 4.3 DATABASE DESIGN AND QUERYING TOOLS

Many Oracle tools are available to developers to help them present data and build more sophisticated Oracle database applications. This section briefly describes the main Oracle tools for application development: Oracle Forms Developer, Oracle Reports Developer, Oracle Designer, Oracle JDeveloper, Oracle Discoverer Administrative Edition and Oracle Portal. Oracle Developer Suite was known as Oracle Internet Developer Suite with Oracle9i.

**SQL*Plus**

SQL*Plus is an interactive and batch query tool that is installed with every Oracle Database Server or Client installation. It has a command-line user interface, a Windows Graphical User Interface (GUI) and the *i*SQL*Plus web-based user interface.

SQL*Plus has its own commands and environment, and it provides access to the Oracle Database. It enables you to enter and execute SQL, PL/SQL, SQL*Plus and operating system commands to perform the following:

- format, perform calculations on, store, and print from query results,
- examine table and object definitions,
- develop and run batch scripts, and
- perform database administration.

You can use SQL*Plus to generate reports interactively, to generate reports as batch processes, and to output the results to text file, to screen, or to HTML file for browsing on the Internet. You can generate reports dynamically using the HTML output facility of SQL*Plus, or using the dynamic reporting capability of *i*SQL*Plus to run a script from a web page.

**Oracle Forms Developer**



Oracle Forms Developer provides a powerful tool for building forms-based applications and charts for deployment as traditional client/server applications or as three-tier browser-based applications via Oracle Application Server. Developer is a fourth-generation language (4GL). With a 4GL, you define applications by defining values for properties, rather than by writing procedural code. Developer supports a wide variety of clients, including traditional client/server PCs and Java-based clients. The Forms Builder includes an in-built JVM for previewing web applications.

**Oracle Reports Developer**

Oracle Reports Developer provides a development and deployment environment for rapidly building and publishing web-based reports via Reports for Oracle's Application Server. Data can be formatted in tables, matrices, group reports, graphs, and combinations. High-quality presentation is possible using the HTML extension Cascading Style Sheets (CSS).

**Oracle JDeveloper**

Oracle JDeveloper was introduced by Oracle in 1998 to develop basic Java applications without writing code. JDeveloper includes a Data Form wizard, a Beans Express wizard for creating JavaBeans and BeanInfo classes, and a Deployment wizard. JDeveloper includes database development features such as various Oracle drivers, a Connection Editor to hide the JDBC API complexity, database components to bind visual controls, and a SQLJ precompiler for embedding SQL in Java code, which you can then use with Oracle. You can also deploy applications developed with JDeveloper using the Oracle Application Server. Although JDeveloper user wizards to allow programmers create Java objects without writing code, the end result is a generated Java code. This Java implementation makes the code highly flexible, but it is typically a less productive development environment than a true 4GL.

**Oracle Designer**

Oracle Designer provides a graphical interface for Rapid Application Development (RAD) for the entire database development process—from building the business model to schema design, generation, and deployment. Designs and changes are stored in a multi-user repository. The tool can reverse-engineer existing tables and database schemas for re-use and re-design from Oracle and non-Oracle relational databases.

The designer also include generators for creating applications for Oracle Developer, HTML clients using Oracle's Application Server, and C++. Designer can generate applications and reverse-engineer existing applications or applications that have been modified by developers. This capability enables a process called round-trip engineering, in which a developer uses Designer to generate an application, modifies the generated application, and reverse-engineers the changes back into the Designer repository.

**Oracle Discoverer**

Oracle Discoverer Administration Edition enables administrators to set up and maintain the Discoverer End User Layer (EUL). The purpose of this layer is to shield business analysts using Discoverer as an ad hoc query or ROLAP tool from SQL complexity. Wizards guide the administrator through the process of building the EUL. In addition, administrators can put limits on resources available to analysts monitored by the Discoverer query governor.

**Oracle Portal**

Oracle Portal, introduced as WebDB in 1999, provides an HTML-based tool for developing web-enabled applications and content-driven web sites. Portal application systems are developed and deployed in a simple browser environment. Portal includes wizards for developing application components incorporating "servlets" and access to other HTTP web sites. For example, Oracle Reports and Discoverer may be accessed as servlets. Portals can be designed to be user-customisable. They are deployed to the middle-tier Oracle Application Server.

Oracle Portal has enhanced the WebDB, with the ability to create and use portlets, which allow a single web page to be divided into different areas that can independently display information and interact with the user.

### ☞ Check Your Progress 1

1) What are Data Definition statements in Oracle?

……

……

……

……

……

2) …… manage the changes made by DML statements.

3) What are the in- built data types of ORACLE?

……

……

……

……

……

4) Name the main ORACLE tools for Application Development.

……

……

……

……

……

## 4.4 OVERVIEW OF ORACLE ARCHITECTURE

The following section presents the basic architecture of the Oracle database. A schematic diagram of Oracle database is given below:

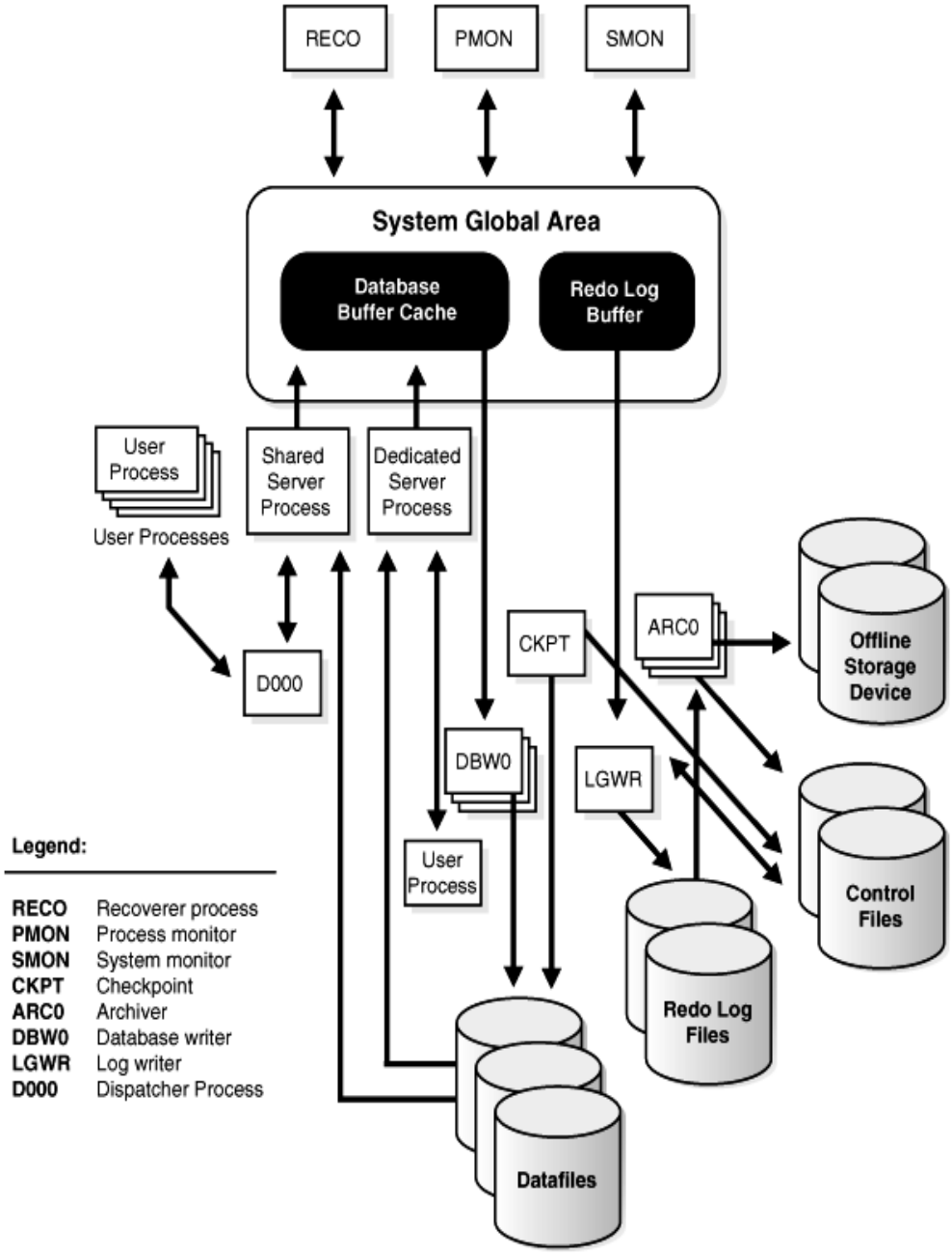



**Figure 1: Oracle 10g Architecture (Source: www.oracle.com)**

### 4.4.1 Physical Database Structures

The following sections explain the physical database structures of an Oracle database, including datafiles, redo log files, and control files.

**Datafiles**

Every Oracle database has one or more physical datafiles. The datafiles contain all the database data. The data of logical database structures, such as tables and indexes, is physically stored in the datafiles allocated for a database.
The characteristics of datafiles are:

- a datafile can be associated with only one database,

- datafiles can have certain characteristics set to let them automatically extend when the database runs out of space,
- one or more datafiles form a logical unit of database storage called a tablespace.

Data in a datafile is read, as needed, during normal database operation and stored in the memory cache of Oracle. For example, assume that a user wants to access some data in a table of a database. If the requested information is not already in the memory cache for the database, then it is read from the appropriate datafiles and stored in the memory.

Modified or new data is not necessarily written to a datafile immediately. To reduce the amount of disk access and to increase performance, data is pooled in memory and written to the appropriate datafiles all at once, as determined by the database writer process (DBW*n*) background process.

**Control Files**

Every Oracle database has a control file. A control file contains entries that specify the physical structure of the database. For example, it contains the following information:

- database name,
- names and locations of datafiles and redo log files, and
- time stamp of database creation.

Oracle can multiplex the control file, that is, simultaneously maintain a number of identical control file copies, to protect against failure involving the control file.

Every time an instance of an Oracle database is started, its control file identifies the database and redo log files that must be opened for database operation to proceed. If the physical makeup of the database is altered (for example, if a new datafile or redo log file is created), then the control file is automatically modified by Oracle to reflect the change. A control file is also used in database recovery.

**Redo Log Files**

Every Oracle database has a set of two or more redo log files. The set of redo log files is collectively known as the redo log for the database. A redo log is made up of redo entries (also called redo records).

The primary function of the redo log is to record all changes made to the data. If a failure prevents modified data from being permanently written to the datafiles, then the changes can be obtained from the redo log, so work is never lost. To protect against a failure involving the redo log itself, Oracle allows a multiplexed redo log so that two or more copies of the redo log can be maintained on different disks.

The information in a redo log file is used only to recover the database from a system or media failure that prevents database data from being written to the datafiles. For example, if an unexpected power shortage terminates database operation, then the data in the memory cannot be written to the datafiles, and the data is lost. However, lost data can be recovered when the database is opened, after power is restored. By applying the information in the most recent redo log files to the database datafiles, Oracle restores the database to the time when the power failure had occurred. The process of applying the redo log during a recovery operation is called rolling forward.

**Archive Log Files**

You can also enable automatic archiving of the redo log. Oracle automatically archives log files when the database is in ARCHIVELOG mode.

**Parameter Files**



Parameter files contain a list of configuration parameters for that instance and database.

Oracle recommends that you create a server parameter file (SPFILE) as a dynamic means of maintaining initialisation parameters. A server parameter file lets you store and manage your initialisation parameters persistently in a server-side disk file.

**Alert and Trace Log Files**

Each server and background process can write to an associated trace file. When an internal error is detected by a process, it dumps information about the error to its trace file. Some of the information written to a trace file is intended for the database administrator, while other information is for Oracle Support Services. Trace file information is also used to tune applications and instances.

The alert file, or alert log, is a special trace file. The alert file of a database is a chronological log of messages and errors.

**Backup Files**

To restore a file is to replace it with a backup file. Typically, you restore a file when a media failure or user error has damaged or deleted the original file.

User-managed backup and recovery requires you to actually restore backup files before a trial recovery of the backups can be attempted/performed.

Server-managed backup and recovery manages the backup process, such as scheduling of backups, as well as recovery processes such as applying the correct backup file when recovery is needed.

### 4.4.2 Logical Database Structures

The logical storage structures, including data blocks, extents, and segments, enable Oracles fine-grained control of disk space use.

**Tablespaces**

A database is divided into logical storage units called tablespaces, which group related logical structures together. For example, tablespaces commonly group together all application objects to simplify administrative operations.

Each database is logically divided into one or more tablespaces. One or more datafiles are explicitly created for each tablespace to physically store the data of all logical structures in a tablespace. The combined size of the datafiles in a tablespace is the total storage capacity of the tablespace. Every Oracle database contains a SYSTEM tablespace and a SYSAUX tablespace. Oracle creates them automatically when the database is created. The system default is to create a smallfile tablespace, which is the traditional type of Oracle tablespace. The SYSTEM and SYSAUX tablespaces are created as smallfile tablespaces.

Oracle also lets you create bigfile tablespaces up to 8 exabytes (8 million terabytes) in size. With Oracle-managed files, bigfile tablespaces make datafiles completely transparent for users. In other words, you can perform operations on tablespaces, rather than on the underlying datafiles.

**Online and Offline Tablespaces**

A tablespace can be online (accessible) or offline (not accessible). A tablespace is generally online, so that users can access information in the tablespace. However,

sometimes a tablespace is offline, in order to make a portion of the database unavailable while allowing normal access to the remainder of the database. This makes many administrative tasks easier to perform.

**Oracle Data Blocks**

At the finest level of granularity, Oracle database data is stored in data blocks. One data block corresponds to a specific number of bytes of physical database space on disk. The standard block size is specified by the DB_BLOCK_SIZE initialization parameter. In addition, you can specify up to five other block sizes. A database uses and allocates free database space in Oracle data blocks.

**Extents**

The next level of logical database space is an extent. An extent is a specific number of contiguous data blocks, obtained in a single allocation, used to store a specific type of information.

**Segments**

Next, is the segment, or the level of logical database storage. A segment is a set of extents allocated for a certain logical structure. The following table describes the different types of segments.

| Segment | Description |
|---|---|
| Data segment | Each non-clustered table has a data segment. All table data is stored in the extents of the data segment.<br>For a partitioned table, each partition has a data segment.<br>Each cluster has a data segment. The data of every table in the cluster is stored in the cluster's data segment. |
| Index segment | Each index has an index segment that stores all of its data.<br>For a partitioned index, each partition has an index segment. |
| Temporary segment | Temporary segments are created by Oracle when a SQL statement needs a temporary database area to complete execution. When the statement has been executed, the extents in the temporary segment are returned to the system for future use. |
| Rollback segment | If you are operating in automatic undo management mode, then the database server manages undo space-using tablespaces. Oracle recommends that you use automatic undo management.<br>Earlier releases of Oracle used rollback segments to store undo information. The information in a rollback segment was used during database recovery for generating read-consistent database information and for rolling back uncommitted transactions for users.<br>Space management for these rollback segments was complex, and not done that way. Oracle uses the undo tablespace method of managing undo; this eliminates the complexities of managing rollback segment space.<br>Oracle does use a SYSTEM rollback segment for performing system |



| Segment | Description |
|---|---|
| | transactions. There is only one SYSTEM rollback segment and it is created automatically at CREATE DATABASE time and is always brought online at instance startup. You are not required to perform any operation to manage the SYSTEM rollback segment. |

Oracle dynamically allocates space when the existing extents of a segment become full. In other words, when the extents of a segment are full, Oracle allocates another extent for that segment. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on a disk.

### 4.4.3 Schemas and Common Schema Objects

A schema is a collection of database objects. A schema is owned by a database user and has the same name as that user. Schema objects are the logical structures that refer directly to the database's data. Schema objects include structures like tables, views, and indexes. (There is no relationship between a tablespace and a schema. Objects in the same schema can be in different tablespaces, and a tablespace can hold objects from different schemas). Some of the most common schema objects are defined in the following section.

**Tables**

Tables are the basic unit of data storage in an Oracle database. Database tables hold all user-accessible data. Each table has columns and rows.

**Indexes**

Indexes are optional structures associated with tables. Indexes can be created to increase the performance of data retrieval. An index provides an access path to table data.

When processing a request, Oracle may use some or all of the available indexes in order to locate, the requested rows efficiently. Indexes are useful when applications frequently query a table for a range of rows (for example, all employees with salaries greater than 1000 dollars) or a specific row.

An index is automatically maintained and used by the DBMS. Changes to table data (such as adding new rows, updating rows, or deleting rows) are automatically incorporated into all relevant indexes with complete transparency to the users. Oracle uses B-trees to store indexes in order to speed up data access.

An index in Oracle can be considered as an ordered list of the values divided into block-wide ranges (leaf blocks). The end points of the ranges along with pointers to the blocks can be stored in a search tree and a value in log($n$) time for $n$ entries could be found. This is the basic principle behind Oracle indexes.
The following *Figure 2* illustrates the structure of a B-tree index.

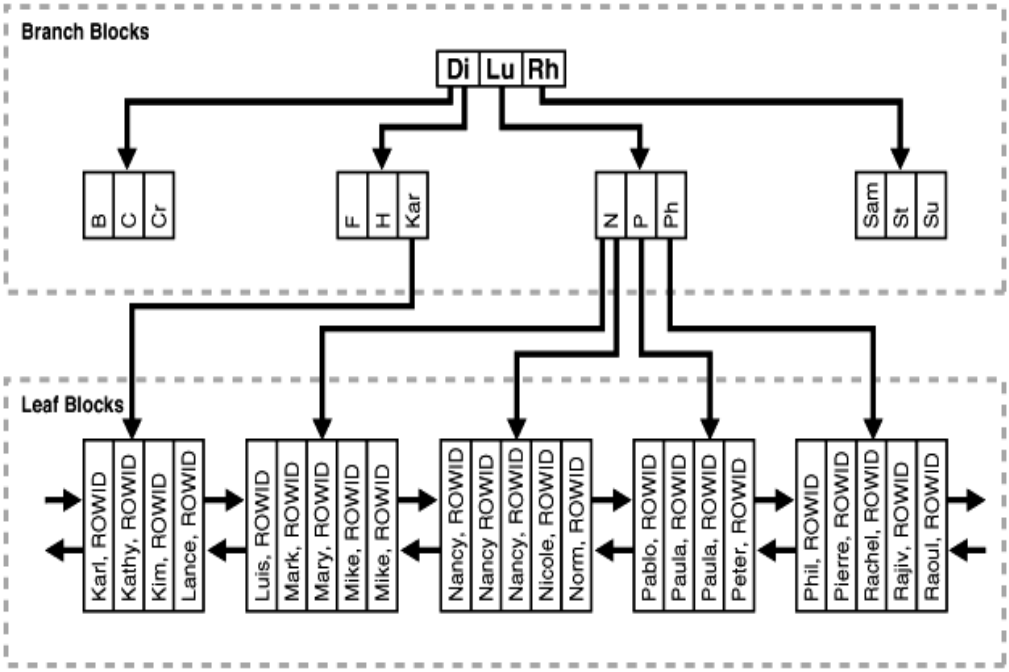



**Figure 2: Internal Structure of an Oracle Index (Source: www.oracle.com)**

The upper blocks (branch blocks) of a B-tree index contain index data that points to lower-level index blocks. The lowest level index blocks (leaf blocks) contains every indexed data value and a corresponding rowid used to locate the actual row. The leaf blocks are doubly linked. Indexes in columns containing character data are based on the binary values of the characters in the database character set.

Index contains two kinds of blocks:

- Branch blocks for searching, and
- Leaf blocks that store values.

Branch Blocks: Branch blocks store the following:

- the minimum key prefix needed to make a branching decision between two keys, and
- the pointer to the child block containing the key.

If the blocks have *n* keys then they have *n*+1 pointers. The number of keys and pointers is limited by the block size.

Leaf Blocks: All leaf blocks are at the same depth from the root branch block. Leaf blocks store the following:

- the complete key value for every row, and
- ROWIDs of the table rows

All key and ROWID pairs are linked to their left and right siblings. They are sorted by (key, ROWID).

**Views**

Views are customised presentations of data in one or more tables or other views. A view can also be considered as a stored query. Views do not actually contain data. Rather, they derive their data from the tables on which they are based, (referred to as the base tables of the views). Views can be queried, updated, inserted into, and deleted from, with some restrictions.



Views provide an additional level of table security by restricting access to a pre-determined set of rows and columns of a table. They also hide data complexity and store complex queries.

**Clusters**

Clusters are groups of one or more tables physically stored together because they share common columns and are often used together. Since related rows are physically stored together, disk access time improves.

Like indexes, clusters do not affect application design. Whether a table is part of a cluster is transparent to users and to applications. Data stored in a clustered table is accessed by SQL in the same way as data stored in a non-clustered table.

**Synonyms**

A synonym is an alias for any table, view, materialised view, sequence, procedure, function, package, type, Java class schema object, user-defined object type, or another synonym. Because a synonym is simply an alias, it requires no storage other than its definition in the data dictionary.

### 4.4.4 Oracle Data Dictionary

Each Oracle database has a data dictionary. An Oracle data dictionary is a set of tables and views that are used as a read-only reference on the database. For example, a data dictionary stores information about the logical and physical structure of the database. A data dictionary also stores the following information:

- the valid users of an Oracle database,
- information about integrity constraints defined for tables in the database, and
- the amount of space allocated for a schema object and how much of it is in use.

A data dictionary is created when a database is created. To accurately reflect the status of the database at all times, the data dictionary is automatically updated by Oracle in response to specific actions, (such as, when the structure of the database is altered). The database relies on the data dictionary to record, verify, and conduct on-going work. For example, during database operation, Oracle reads the data dictionary to verify that schema objects exist and that users have proper access to them.

### 4.4.5 Oracle Instance

An Oracle database server consists of an Oracle database and an Oracle instance. Every time a database is started, a system global area (SGA) (please refer to *Figure1)* is allocated and Oracle background processes are started. The combination of the background processes and memory buffers is known as an Oracle instance.

**Real Application Clusters: Multiple Instance Systems**

Some hardware architectures (for example, shared disk systems) enable multiple computers to share access to data, software, or peripheral devices. Real Application Clusters (RAC) take advantage of such architecture by running multiple instances that share a single physical database. In most applications, RAC enables access to a single database by users on multiple machines with increased performance.

An Oracle database server uses memory structures and processes to manage and access the database. All memory structures exist in the main memory of computers that constitute the database system. Processes are jobs that work in the memory of these computers.

**Instance Memory Structures**

Oracle creates and uses memory structures to complete several jobs. For example, memory stores program code being run and data shared among users. Two basic memory structures associated with Oracle are: the system global area and the program global area. The following subsections explain each in detail.

**System Global Area**

The System Global Area (SGA) is a shared memory region that contains data and control information for one Oracle instance. Oracle allocates the SGA when an instance starts and deallocates it when the instance shuts down. Each instance has its own SGA.

Users currently connected to an Oracle database share the data in the SGA. For optimal performance, the entire SGA should be as large as possible (while still fitting in to the real memory) to store as much data in memory as possible and to minimise disk I/O.

The information stored in the SGA is divided into several types of memory structures, including the database buffers, redo log buffer, and the shared pool.

**Database Buffer Cache of the SGA**

Database buffers store the most recently used blocks of data. The set of database buffers in an instance is the database buffer cache. The buffer cache contains modified as well as unmodified blocks. Because the most recently (and often, the most frequently) used data is kept in the memory, less disk I/O is required, and the performance is improved.

**Redo Log Buffer of the SGA**

The redo log buffer stores redo entries – a log of changes made to the database. The redo entries stored in the redo log buffers are written to an online redo log, which is used if database recovery is necessary. The size of the redo log is static.

**Shared Pool of the SGA**

The shared pool contains shared memory constructs, such as shared SQL areas. A shared SQL area is required to process every unique SQL statement submitted to a database. A shared SQL area contains information such as the parse tree and execution plan for the corresponding statement. A single shared SQL area is used by multiple applications that issue the same statement, leaving more shared memory for other uses.

**Statement Handles or Cursors**

A cursor is a handle or name for a private SQL area in which a parsed statement and other information for processing the statement are kept. (Oracle Call Interface, OCI, refers to these as statement handles). Although most Oracle users rely on automatic cursor handling of Oracle utilities, the programmatic interfaces offer application designers more control over cursors.

For example, in precompiler application development, a cursor is a named resource available to a program and can be used specifically to parse SQL statements embedded within the application. Application developers can code an application so that it controls the phases of SQL statement execution and thus improves application performance.

**Program Global Area**



The Program Global Area (PGA) is a memory buffer that contains data and control information for a server process. A PGA is created by Oracle when a server process is started. The information in a PGA depends on the configuration of Oracle.

### 4.4.6 Oracle Background Processes

An Oracle database uses memory structures and processes to manage and access the database. All memory structures exist in the main memory of computers that constitute the database system. Processes are jobs that work in the memory of these computers.

The architectural features discussed in this section enables the Oracle database to support:

- many users concurrently accessing a single database, and
- the high performance required by concurrent multiuser, multiapplication database systems.

Oracle creates a set of background processes for each instance. The background processes consolidate functions that would otherwise be handled by multiple Oracle programs running for each user process. They asynchronously perform I/O and monitor other Oracle processes to provide increased parallelism for better performance and reliability. There are numerous background processes, and each Oracle instance can use several background processes.

**Process Architecture**

A process is a “thread of control” or a mechanism in an operating system that can run a series of steps. Some operating systems use the terms job or task. A process generally has its own private memory area in which it runs. An Oracle database server has two general types of processes: user processes and Oracle processes.

**User (Client) Processes**

User processes are created and maintained to run the software code of an application program or an Oracle tool (such as Enterprise Manager). User processes also manage communication with the server process through the program interface, which is described in a later section.

**Oracle Processes**

Oracle processes are invoked by other processes to perform functions on behalf of the invoking process. Oracle creates server processes to handle requests from connected user processes. A server process communicates with the user process and interacts with Oracle to carry out requests from the associated user process. For example, if a user queries some data not already in the database buffers of the SGA, then the associated server process reads the proper data blocks from the datafiles into the SGA.

Oracle can be configured to vary the number of user processes for each server process. In a dedicated server configuration, a server process handles requests for a single user process. A shared server configuration lets many user processes share a small number of server processes, minimising the number of server processes and maximising the use of available system resources.

On some systems, the user and server processes are separate, while on others they are combined into a single process. If a system uses the shared server or if the user and server processes run on different machines, then the user and server processes must be separate. Client/server systems separate the user and server processes and run them on different machines.

Let us discuss a few important process briefly next.

**Database Writer (DBWR)**

The Database Writer process writes database blocks from the database buffer cache in the SGA to the datafiles on disk. An Oracle instance can have up to 10 DBWR processes, from DBW0 to DBW9, to handle the I/O load to multiple datafiles. Most instances run one DBWR. DBWR writes blocks out of the cache for two main reasons:

- If Oracle needs to perform a checkpoint (i.e., to update the blocks of the datafiles so that they "catch up" to the redo logs). Oracle writes the redo for a transaction when it's committed, and later writes the actual blocks. Periodically, Oracle performs a checkpoint to bring the datafile contents in line with the redo that was written out for the committed transactions.

- If Oracle needs to read blocks requested by users into the cache and there is no free space in the buffer cache. The blocks written out are the least recently used blocks. Writing blocks in this order minimises the performance impact of losing them from the buffer cache.

**Log Writer (LGWR)**

The Log Writer process writes the redo information from the log buffer in the SGA to all copies of the current redo log file on disk. As transactions proceed, the associated redo information is stored in the redo log buffer in the SGA. When a transaction is committed, Oracle makes the redo information permanent by invoking the Log Writer to write it to disk.

**System Monitor (SMON)**

The System Monitor process maintains the overall health and safety of an Oracle instance. SMON performs crash recovery when the instance is started after a failure and coordinates and performs recovery for a failed instance when there is more than one instance accessing the same database, as with Oracle Parallel Server/Real Application Clusters. SMON also cleans up adjacent pieces of free space in the datafiles by merging them into one piece and gets rid of space used for sorting rows when that space is no longer needed.

**Process Monitor (PMON)**

The Process Monitor process watches over the user processes that access the database. If a user process terminates abnormally, PMON is responsible for cleaning up any of the resources left behind (such as memory) and for releasing any locks held by the failed process.

**Archiver (ARC)**

The Archiver process reads the redo log files once Oracle has filled them and writes a copy of the used redo log files to the specified archive log destination(s).

An Oracle instance can have up to 10 processes, numbered as described for DBWR above. LGWR will start additional Archivers as needed, based on the load, up to the limit specified by the initialisation parameter LOG_ARCHIVE_MAX_PROCESSES.

**Checkpoint (CKPT)**



The Checkpoint process works with DBWR to perform checkpoints. CKPT updates the control file and database file headers to update the checkpoint data when the checkpoint is complete.

**Recover (RECO)**

The Recover process automatically cleans up failed or suspended distributed transactions.

### 4.4.7 How Oracle Works?

The following example describes the most basic level of operations that Oracle performs. This illustrates an Oracle configuration where the user and associated server process are on separate machines (connected through a network).

1) An instance has started on the computer running Oracle (often called the host or database server).

2) A computer running an application (a local machine or client workstation) runs the application in a user process. The client application attempts to establish a connection to the server using the proper Oracle Net Services driver.

3) The server is running the proper Oracle Net Services driver. The server detects the connection request from the application and creates a dedicated server process on behalf of the user process.

4) The user runs a SQL statement and commits the transaction. For example, the user changes a name in a row of a table.

5) The server process receives the statement and checks the shared pool for any shared SQL area that contains a similar SQL statement. If a shared SQL area is found, then the server process checks the user's access privileges to the requested data, and the previously existing shared SQL area is used to process the statement. If not, then a new shared SQL area is allocated for the statement, so it can be parsed and processed.

6) The server process retrieves any necessary data values from the actual datafile (table) or those stored in the SGA.

7) The server process modifies data in the system global area. The DBW*n* process writes modified blocks permanently to the disk when doing so is efficient. Since, the transaction is committed, the LGWR process immediately records the transaction in the redo log file.

8) If the transaction is successful, then the server process sends a message across the network to the application. If it is not successful, then an error message is transmitted.

9) Throughout this entire procedure, the other background processes run, watching for conditions that require intervention. In addition, the database server manages other users' transactions and prevents contention between transactions that request the same data.

☞ **Check Your Progress 2**

1) What are the different files used in ORACLE?

……

……

……

……

2) ……Storage Structure enables Oracle to have fine – gained control of disk space use.

3) ……..are the basic unit of data storage in an ORACLE database.

4) What are the advantages of B+ tree structure?

……

……

……

……

5) What does a Data Dictionary store?

……

……

……

……

6) ……is a handle for a private SQL area in which a parsed statement and other information for processing the statement are kept.

7) ……is a memory buffer that contains data and control information for a server process.

8) What is a process in Oracle?

……

……

……

……

9) Expand the following:

a) LGWR
b) ARC
c) RECO.

## 4.5 QUERY PROCESSING AND OPTIMISATION

This section provides the various stages of the execution of a SQL statement in each stage of DML statement processing. The following stages are necessary for each type of statement processing:

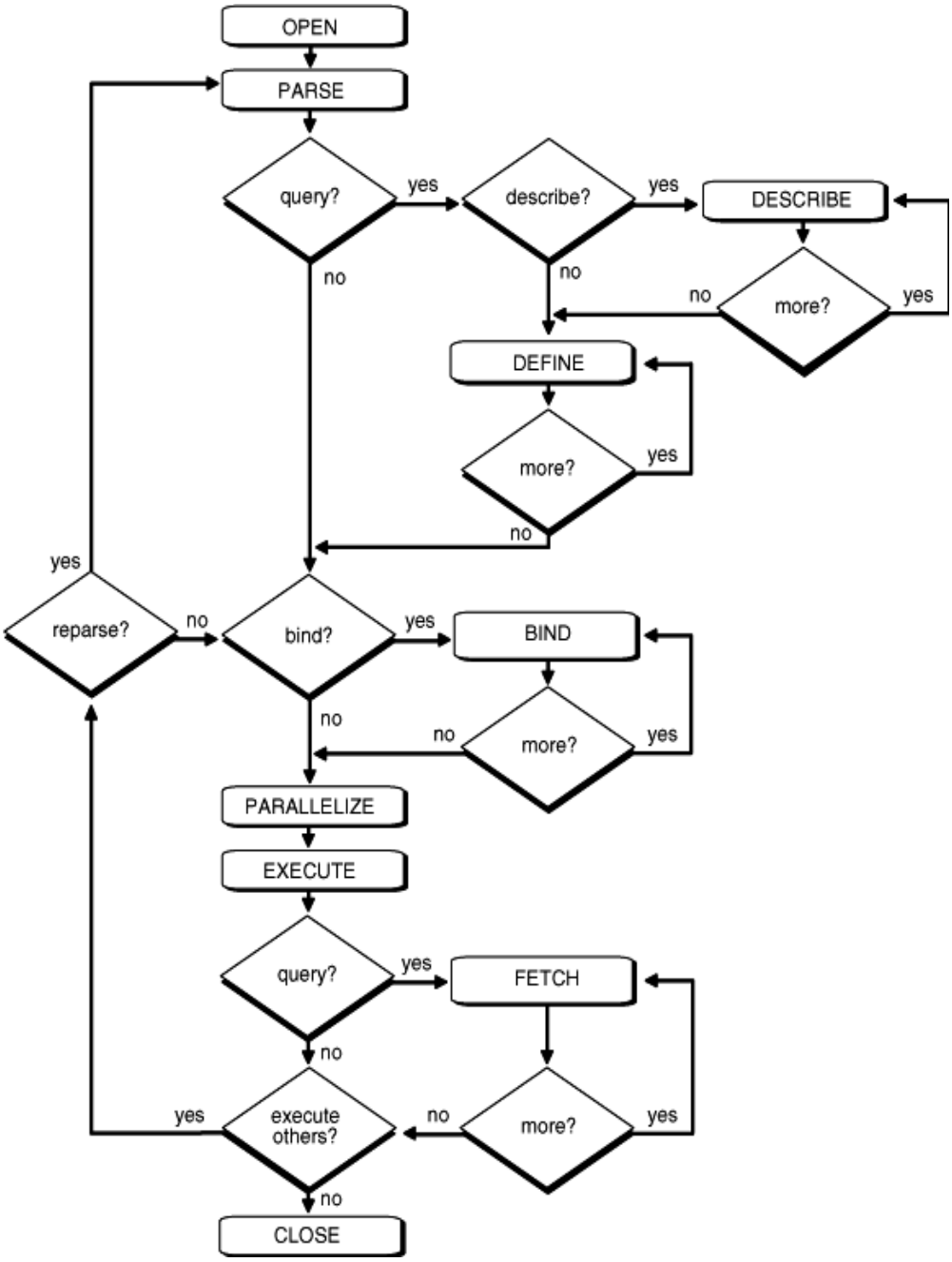



**Figure 3: Stages in Query Processing (Source: www.oracle.com)**

### Stage 1: Create a Cursor

A program interface call creates a cursor. This cursor is not the Oracle PL/SQL cursor but the reference to memory space that is used to keep and manipulate the data in the primary memory. The cursor is created independent of any SQL statement: it is created in expectation of any SQL statement. In most applications, the cursor creation is automatic. However, in pre-compiler programs, cursor creation can either occur implicitly or be explicitly declared.

### Stage 2: Parse the Statement

During parsing, the SQL statement is passed from the user process to Oracle, and a parsed representation of the SQL statement is loaded into a shared SQL area. Many errors may be detected during this stage of statement processing.
Parsing is the process of:

- translating a SQL statement, verifying it to be a valid statement,
- performing data dictionary lookups to check table and column definitions,
- acquiring parse locks on required objects so that their definitions do not change during the statement's parsing,
- checking privileges to access referenced schema objects,
- determining the optimal execution plan for the statement,
- loading it into a shared SQL area, and
- routing all or part of distributed statements to remote nodes that contain referenced data.

Oracle parses a SQL statement only if a shared SQL area for a similar SQL statement does not exist in the shared pool. In this case, a new-shared SQL area is allocated, and the statement is parsed.

The parse stage includes processing requirements that need to be done only once, no matter, how many times the statement is executed. Oracle translates each SQL statement only once, re-executing that parsed statement during subsequent references to the statement.

Although parsing a SQL statement validates that statement, parsing only identifies errors that can be found before the execution of the statement. Thus, some errors cannot be caught by parsing. For example, errors in data conversion or errors in data (such as an attempt to enter duplicate values in a primary key) and deadlocks are all errors or situations that can be encountered and reported only during the execution stage.

**Note:** Queries are different from other types of SQL statements because, if successful, they return data as results. Whereas, other statements simply return success or failure, for instance, a query can return one row or thousands of rows. The results of a query are always in tabular format, and the rows of the result are fetched (retrieved), either a row at a time or in groups.

Several issues are related only to query processing. Queries include not only explicit SELECT statements but also the implicit queries (subqueries) in other SQL statements. For example, each of the following statements, require a query as a part of its execution:

INSERT INTO table SELECT...

UPDATE table SET x = y WHERE...

DELETE FROM table WHERE...

CREATE table AS SELECT...

In particular, queries:

- require read consistency,
- can use temporary segments for intermediate processing, and
- can require the describe, define, and fetch stages of SQL statement processing.

**Stage 3: Describe Results of a Query**

The describe stage is necessary only if the characteristics of a query's result are not known; for example, when a query is entered interactively by a user. In this case, the describe stage determines the characteristics (datatypes, lengths, and names) of a query's result.

**Stage 4: Define Output of a Query**



In the define stage of a query, you specify the location, size, and datatype of variables defined to receive each fetched value. Oracle performs datatype conversion if necessary.

**Stage 5: Bind any Variables**

At this point, Oracle knows the meaning of the SQL statement but still does not have enough information to execute the statement. Oracle needs values for any variables listed in the statement; for example, Oracle needs a value for DEPT_NUMBER. The process of obtaining these values is called binding variables.

A program must specify the location (memory address) of the value. End users of applications may be unaware that they are specifying bind variables, because the Oracle utility can simply prompt them for a new value.

Because you specify the location (binding by reference), you need not rebind the variable before re-execution. You can change its value and Oracle will look up the value on each execution, using the memory address.

You must also specify a datatype and length for each value (unless they are implied or defaulted) in order for Oracle to perform datatype conversions.

**Stage 6: Parallelise the Statement**

Oracle can parallelise queries (SELECTs, INSERTs, UPDATEs, MERGEs, DELETEs), and some DDL operations such as index creation, creating a table with a subquery, and operations on partitions. Parallelisation causes multiple server processes to perform the work of the SQL statement so it can be completed faster.

**Stage 7: Execute the Statement**

At this point, Oracle has all the necessary information and resources, so the statement is executed. If the statement is a query or an INSERT statement, no rows need to be locked because no data is being changed. If the statement is an UPDATE or DELETE statement, however, all rows that the statement affects are locked from use by other users of the database until the next COMMIT, ROLLBACK, or SAVEPOINT for the transaction. This ensures data integrity.

For some statements you can specify a number of executions to be performed. This is called array processing. Given *n* number of executions, the bind and define locations are assumed to be the beginning of an array of size *n*.

**Stage 8: Fetch Rows of a Query**

In the fetch stage, rows are selected and ordered (if requested by the query), and each successive fetch retrieves another row of the result until the last row has been fetched.

**Stage 9: Close the Cursor**

The final stage of processing a SQL statement is closing the cursor.

## Query Optimisation

An important facet of database system performance tuning is the tuning of SQL statements. SQL tuning involves three basic steps:

- Identifying high load or top SQL statements that are responsible for a large share of the application workload and system resources, by reviewing past SQL execution history available in the system.
- Verifying that the execution plans produced by the query optimiser for these statements perform reasonably.

- Implementing corrective actions to generate better execution plans for poorly performing SQL statements.

These three steps are repeated until the system performance reaches a satisfactory level or no more statements can be tuned.

A SQL statement can be executed in many different ways, such as full table scans, index scans, nested loops, and hash joins. The Oracle query optimiser determines the most efficient way to execute a SQL statement after considering many factors related to the objects referenced and the conditions specified in the query. This determination is an important step in the processing of any SQL statement and can greatly affect execution time.

The query optimiser determines most efficient execution plan by considering available access paths and by factoring in information based on statistics for the schema objects (tables or indexes) accessed by the SQL statement. The query optimiser also considers hints, which are optimisation suggestions placed in a comment in the statement.

The query optimiser performs the following steps:

1) The optimiser generates a set of potential plans for the SQL statement based on available access paths and hints.

2) The optimiser estimates the cost of each plan based on statistics in the data dictionary for data distribution and storage characteristics of the tables, indexes, and partitions accessed by the statement.

   The cost is an estimated value proportional to the expected resource use needed to execute the statement with a particular plan. The optimiser calculates the cost of access paths and join orders based on the estimated computer resources, which includes I/O, CPU, and memory.

   Serial plans with higher costs take more time to execute than those with lesser costs. When using a parallel plan, however, resource use is not directly related to elapsed time.

3) The optimiser compares the costs of the plans and chooses the one with the lowest cost.

## 4.6 DISTRIBUTED ORACLE

One of the strongest features of the Oracle database is its ability to scale up to handling extremely large volumes of data and users. Oracle scales not only by running on more and more powerful platforms, but also by running in a distributed configuration. Oracle databases on separate platforms can be combined to act as a single logical distributed database. This section describes some of the basic ways that Oracle handles database interactions in a distributed database system.

### 4.6.1 Distributed Queries and Transactions

Data within an organisation is often spread among multiple databases for reasons of both capacity and organisational responsibility. Users may want to query this distributed data or update it as if it existed within a single database. Oracle, first, introduced distributed databases in response to the requirements for accessing data on multiple platforms in the early 1980s. Distributed queries can retrieve data from multiple databases. Distributed transactions can insert, update, or delete data on distributed databases. Oracle's two-phase commit mechanism guarantees that all the database servers that are part of a transaction will either commit or roll back the



transaction. Background recovery processes can ensure database consistency in the event of system interruption during distributed transactions. Once the failed system comes back online, the same process will complete the distributed transactions.

Distributed transactions can also be implemented using popular transaction monitors (TPs) that interact with Oracle via XA, an industry standard (X/Open) interface. Oracle8i added native transaction coordination with the Microsoft Transaction Server (MTS), so you can implement a distributed transaction initiated under the control of MTS through an Oracle database.

### 4.6.2 Heterogeneous Services

Heterogeneous Services allow non-Oracle data and services to be accessed from an Oracle database through generic connectivity via ODBC and OLE-DB included with the database. Optional Transparent Gateways use agents specifically tailored for a variety of target systems. Transparent Gateways allow users to submit Oracle SQL statements to a non-Oracle distributed database source and have them automatically translated into the SQL dialect of the non-Oracle source system, which remains transparent to the user. In addition to providing underlying SQL services, Heterogeneous Services provide transaction services utilising Oracle's two-phase commit with non-Oracle databases and procedural services that call for third-generation language routines on non-Oracle systems. Users interact with the Oracle database as if all objects are stored in the Oracle database, and Heterogeneous Services handle the transparent interaction with the foreign database on the user's behalf.

## 4.7 DATA MOVEMENT IN ORACLE

Moving data from one Oracle database to another is often a requirement when using distributed databases, or when a user wants to implement multiple copies of the same database in multiple locations to reduce network traffic or increase data availability. You can export data and data dictionaries (metadata) from one database and import them into another. Oracle Database 10g introduces a new high speed data pump for the import and export of data. Oracle also offers many other advanced features in this category, including replication, transportable tablespaces, and Advanced Queuing.

The ensuing sections describe the technology used to move data from one Oracle database to another automatically.

### 4.7.1 Basic Replication

You can use basic replication to move recently, added and updated data from an Oracle "master" database to databases on which duplicate sets of data reside. In basic replication, only the single master is updated. You can manage replication through the Oracle Enterprise Manager (OEM or EM). While replication has been a part of all recent Oracle releases, replication based on logs is a more recent addition, appearing for the first time in Oracle9i Release 2.

### 4.7.2 Advanced Replication

You can use advanced replication in multi-master systems in which any of the databases involved can be updated and in which conflict-resolution features are required to resolve inconsistencies in the data. Because, there is more than one master database, the same data may be updated on multiple systems at the same time. Conflict resolution is necessary to determine the "true" version of the data. Oracle's advanced replication includes a number of conflict-resolution scenarios and also allows programmers to write their own conflict-resolution scenarios.

### 4.7.3 Transportable Tablespaces

Transportable tablespaces were introduced in Oracle8i. Instead of using the export/import process, which dumps data and the structures that contain it into an intermediate files for loading, you simply put the tablespaces in read-only mode, move or copy them from one database to another, and mount them. You must export the data dictionary (metadata) for the tablespace from the source and import it at the target. This feature can save a lot of time during maintenance, because it simplifies the process. Oracle Database 10g allows you to move data with transportable tablespaces between different platforms or operating systems.

### 4.7.4 Advanced Queuing and Streams

Advanced Queuing (AQ), first introduced in Oracle8, provides the means to asynchronously send messages from one Oracle database to another. Because messages are stored in a queue in the database and sent asynchronously when a connection is made, the amount of overhead and network traffic is much lower than it would be using traditional guaranteed delivery through the two-phase commit protocol between source and target. By storing the messages in the database, AQ provides a solution with greater recoverability than other queuing solutions that store messages in file systems.

Oracle messaging adds the capability to develop and deploy a content-based publish and subscribe solution using the rules engine to determine relevant subscribing applications. As new content is published to a subscriber list, the rules on the list determine which subscribers should receive the content. This approach means that a single list can efficiently serve the needs of different subscriber communities.

In the first release of Oracle9i, AQ added XML support and Oracle Internet Directory (OID) integration. This technology is leveraged in Oracle Application Interconnect (OAI), which includes adapters to non-Oracle applications, messaging products, and databases.

The second release of Oracle9i introduced Streams. Streams have three major components: log-based replication for data capture, queuing for data staging, and user-defined rules for data consumption. Oracle Database 10g includes support for change data capture and file transfer solutions via Streams.

### 4.7.5 Extraction, Transformation and Loading

Oracle Warehouse Builder is a tool for the design of target data stores including data warehouses and a metadata repository, but it also provides a front-end to building source-to-target maps and for generating extraction, transformation, and loading (ETL) scripts. OWB leverages key embedded ETL features first made available in the Oracle9i database.

## 4.8 DATABASE ADMINISTRATION TOOLS

Oracle includes many features that make the database easier to manage. We will discuss two types of tools: Oracle Enterprise Manager, and add-on packs in this section. The tools for backup and recovery, and database availability will be explained in the next section.

**Oracle Enterprise Manager**

As part of every Database Server, Oracle provides the Oracle Enterprise Manager (EM), a database management tool framework with a graphical interface to manage database users, instances, and features (such as replication) that can provide additional



information about the Oracle environment. EM can also manage Oracle's Application Server, Collaboration Suite, and E-Business Suite.

Prior to the Oracle8i database, the EM software was installed on Windows-based systems. Each repository was accessible only by a single database manager at a time. EM evolved to a Java release providing access from a browser or Windows-based system. Multiple database administrators could then access the EM repository at the same time.

More recently, an EM HTML console was released with Oracle9iAS. This console has important new application performance management and configuration management features. The HTML version supplemented the Java-based Enterprise Manager earlier available. Enterprise Manager 10g, released with Oracle Database 10g, also comes in Java and HTML versions. EM can be deployed in several ways: as a central console for monitoring multiple databases leveraging agents, as a "product console" (easily installed with each individual database), or through remote access, also known as "studio mode". The HTML-based console includes advanced management capabilities for rapid installation, deployment across grids of computers, provisioning, upgrades, and automated patching.

Oracle Enterprise Manager 10g has several additional options (sometimes called packs) for managing the Oracle Enterprise Edition database. These options, which are available for the HTML-based console, the Java-based console, or both, include:

- Database Diagnostics Option
- Application Server Diagnostics Option
- Database Tuning Option
- Database Change Management Option
- Database Configuration Management Option
- Application Server Configuration Management Option.

Standard management pack functionality for managing the Standard Edition is now also available for the HTML-based console.

## 4.9 BACKUP AND RECOVERY IN ORACLE

As every database administrator knows, backing up a database is a rather mundane but necessary task. An improper backup makes recovery difficult, if not impossible. Unfortunately, people often realise the extreme importance of this everyday task only when it is too late – usually after losing critical business data due to the failure of a related system.

**Recovery Manager**

Typical backups include complete database backups (the most common type), tablespace backups, datafile backups, control file backups, and archived redo log backups. Oracle8 introduced the Recovery Manager (RMAN) for server-managed backup and recovery of the database. Previously, Oracle's Enterprise Backup Utility (EBU) provided a similar solution on some platforms. However, RMAN, with its Recovery Catalogue we stored in an Oracle database, provides a much more complete solution. RMAN can automatically locate, back up, restore, and recover datafiles, control files, and archived redo logs. RMAN, since Oracle9i, can restart backups, restore and implement recovery window policies when backups expire. The Oracle Enterprise Manager Backup Manager provides a GUI-based interface to RMAN. Oracle Enterprise Manager 10g introduces a new improved job scheduler that can be used with RMAN and other scripts, and that can manage automatic backups to disk.

**Incremental Backup and Recovery**

RMAN can perform incremental backups of Enterprise Edition databases. Incremental backups, back up, only the blocks modified since the last backup of a datafile, tablespace, or database; thus, they are smaller and faster than complete backups. RMAN can also perform point-in-time recovery, which allows the recovery of data until just prior to an undesirable event (such as the mistaken dropping of a table).

**Oracle Storage Manager and Automated Disk Based Backup and Recovery**

Various media-management software vendors support RMAN. Since Oracle8i, a Storage Manager has been developed with Oracle to provide media-management services, including the tracking of tape volumes, for up to four devices. RMAN interfaces automatically with the media-management software to request the mounting of tapes as needed for backup and recovery operations.

Oracle Database 10g introduces Automated Disk Based Backup and Recovery. The disk acts as a cache, and archives and backups can then be copied to tape. The disk "cache" can also serve as a staging area for recovery.

## 4.10 ORACLE LITE

Oracle Lite is Oracle's suite of products for enabling mobile use of database-centric applications. The key components of Oracle Lite include the Oracle Lite Database, Mobile Development Kit, and Mobile Server (an extension of the Oracle Application Server).

Although the Oracle Lite Database engine runs on a much smaller platform than other Oracle implementations (it requires a 50K to 1 MB footprint depending on the platform), Mobile SQL, C++, and Java-based applications can run against the database. ODBC is therefore, supported. Java support includes Java stored procedures and JDBC. The database is self-tuning and self-administering. In addition to Windows-based laptops, Oracle Lite also supports for handheld devices running on WindowsCE, Palm's Computing Platform, and Symbian EPOC.

## 4.11 SCALABILITY AND PERFORMANCE FEATURES OF ORACLE

Oracle includes several software mechanisms to fulfil the following important requirements of an information management system:

- Data concurrency of a multiuser system must be maximised.
- Data must be read and modified in a consistent fashion. The data a user is viewing or changing is not changed (by other users) until the user is finished with the data.
- High performance is required for maximum productivity from the many users of the database system.

### 4.11.1 Concurrency

A primary concern of a multiuser database management system is controlling concurrency, which is the simultaneous access of the same data by many users. Without adequate concurrency controls, data could be updated or changed improperly, compromising data integrity.



One way to manage data concurrency is to make each user wait for a turn. The goal of a database management system is to reduce that wait so it is either nonexistent or negligible to each user. All data manipulation language statements should proceed with as little interference as possible, and undesirable interactions among concurrent transactions must be prevented. Neither performance nor data integrity can be compromised.

Oracle resolves such issues by using various types of locks and a multiversion consistency model. These features are based on the concept of a transaction. It is the application designer's responsibility to ensure that transactions fully exploit these concurrency and consistency features.

### 4.11.2 Read Consistency

Read consistency, as supported by Oracle, does the following:

- Guarantees that the set of data seen by a statement is consistent with respect to a single point in time and does not change during statement execution (statement-level read consistency).
- Ensures that readers of database data do not wait for writers or other readers of the same data.
- Ensures that writers of database data do not wait for readers of the same data.
- Ensures that writers only wait for other writers if they attempt to update identical rows in concurrent transactions.

The simplest way to think of Oracle's implementation of read consistency is to imagine each user operating a private copy of the database, hence the multiversion consistency model.

To manage the multiversion consistency model, Oracle must create a read-consistent set of data when a table is queried (read) and simultaneously updated (written). When an update occurs, the original data values changed by the update are recorded in the database undo records. As long as this update remains part of an uncommitted transaction, any user that later queries the modified data views the original data values. Oracle uses current information in the system global area and information in the undo records to construct a read-consistent view of a table's data for a query.

Only when a transaction is committed are the changes of the transaction made permanent. Statements that start *after* the user's transaction is committed only see the changes made by the committed transaction.

Transaction is the key to Oracle's strategy for providing read consistency. This unit of committed (or uncommitted) SQL statements dictates the start point for read-consistent views generated on behalf of readers and controls modified data can be seen by other transactions of the time span when, database for reading or updating.

By default, Oracle guarantees statement-level read consistency. The set of data returned by a single query is consistent with respect to a single point in time. However, in some situations, you might also require transaction-level read consistency. This is the ability to run multiple queries within a single transaction, all of which are read-consistent with respect to the same point in time, so that queries in this transaction do not see the effects of intervening committed transactions. If you want to run a number of queries against multiple tables and if you are not doing any updating, you would prefer a read-only transaction.

### 4.11.3 Locking Mechanisms

Oracle also uses locks to control concurrent access to data. When updating information, the data server holds that information with a lock, until, the update is submitted or committed. Until that happens, no one else can make changes to the locked information. This ensures the data integrity of the system.

Oracle provides unique non-escalating row-level locking. Unlike other data servers that "escalate" locks to cover entire groups of rows or even the entire table, Oracle always locks only the row of information being updated. Because Oracle includes the locking information with the actual rows themselves, Oracle can lock an unlimited number of rows so users can work concurrently without unnecessary delays.

**Automatic Locking**

Oracle locking is performed automatically and requires no user action. Implicit locking occurs for SQL statements as necessary, depending on the action requested. Oracle's lock manager automatically locks table data at the row level. By locking table data at the row level, contention for the same data is minimised.

Oracle's lock manager maintains several different types of row locks, depending on the type of operation that established the lock. The two general types of locks are: exclusive locks and share locks. Only one exclusive lock can be placed on a resource (such as a row or a table); however, many share locks can be placed on a single resource. Both exclusive and share locks always allow queries on the locked resource but prohibit other activity on the resource (such as updates and deletes).

**Manual Locking**

Under some circumstances, a user might want to override default locking. Oracle allows manual override of automatic locking features at both the row level (by first querying for the rows that will be updated in a subsequent statement) and the table level.

### 4.11.4 Real Application Clusters

Real Application Clusters (RAC) comprises several Oracle instances running on multiple clustered machines, which communicate with each other by means of an interconnect. RAC uses cluster software to access a shared database that resides on a shared disk. RAC combines the processing power of these multiple interconnected computers to provide system redundancy, near linear scalability, and high availability. RAC also offers significant advantages for both OLTP and data warehouse systems and all systems and applications can efficiently exploit clustered environments.

You can scale applications in RAC environments to meet increasing data processing demands without changing the application code. As you add resources such as nodes or storage, RAC extends the processing powers of these resources beyond the limits of the individual components.

### 4.11.5 Portability

Oracle provides unique portability across all major platforms and ensures that your applications run without modification after changing platforms. This is because the Oracle code base is identical across platforms, so you have identical feature functionality across all platforms, for complete application transparency. Because of this portability, you can easily upgrade to a more powerful server as your requirements change.

### ☞ Check Your Progress 3

1) What is Parsing?



……

……

……

……

2) What are the three basic steps involved in SQL tuning?

……

……

……

……

3) The oracle ……determines the most efficient way to execute a SQL statement after considering many factors related to the object references and the conditions specified in the Query.

4) What are heterogeneous services?

……

……

……

5) Name the basic technology used to move data from are ORACLE database to another.

……

……

……

……

6) What is Oracle Enterprise manager?

……

……

……

7) What are the different types of backup?

……

……

……

8) …….. is Oracle's suite of products for enabling mobile use of database – centric Applications.

9) …….. comprises several ORACLE instances running on multiple clustered machines.

## 4.12 ORACLE DATA WAREHOUSING

A data warehouse is a relational database designed for query and analysis rather than for transaction processing. It usually contains historical data derived from transaction data, but it can include data from other sources. It separates analysis workload from transaction workload and enables an organisation to consolidate data from several sources.

In addition to a relational database, a data warehouse environment includes an extraction, transportation, transformation, and loading (ETL) solution, an online analytical processing (OLAP) engine, client analysis tools, and other applications that manage the process of gathering data and delivering it to business users.

### 4.12.1 Extraction, Transformation and Loading (ETL)

You must load your data warehouse regularly so that it can serve its purpose of facilitating business analysis. To do this, data from one or more operational systems must be extracted and copied into the warehouse. The process of extracting data from source systems and bringing it into the data warehouse is commonly called ETL, which stands for extraction, transformation, and loading.

### 4.12.2 Materialised Views

A materialised view provides indirect access to table data by storing the results of a query in a separate schema object. Unlike an ordinary view, which does not take up any storage space or contain any data, a materialised view contains the rows resulting from a query against one or more base tables or views. A materialised view can be stored in the same database as its base tables or in a different database.

Materialised views are stored in the same database as their base tables can improve query performance through query rewrites. Query rewrites are particularly useful in a data warehouse environment.

### 4.12.3 Bitmap Indexes

Data warehousing environments typically have large amounts of data and *ad hoc* queries, but a low level of concurrent database manipulation language (DML) transactions. For such applications, bitmap indexing provides:

- reduced response time for large classes of ad hoc queries,
- reduced storage requirements compared to other indexing techniques,
- dramatic performance gains even on hardware with a relatively small number of CPUs or a small amount of memory, and
- efficient maintenance during parallel DML and loads.

Fully indexing a large table with a traditional B-tree index can be prohibitively expensive in terms of space because the indexes can be several times larger than the data in the table. Bitmap indexes are typically only a fraction of the size of the indexed data in the table.

### 4.12.4 Table Compression

To reduce disk use and memory use (specifically, the buffer cache), you can store tables and partitioned tables in a compressed format inside the database. This often leads to better scaleup for read-only operations. Table compression can also speed up query execution. There is, however, a slight cost in CPU overhead.

### 4.12.5 Parallel Execution

When Oracle runs SQL statements in parallels, multiple processes work together simultaneously to run a single SQL statement. By dividing the work, necessary to run a statement among multiple processes, Oracle can run the statement more quickly than if only a single process ran it. This is called parallel execution or parallel processing.

Parallel execution dramatically reduces response time for data-intensive operations on large databases, because statement processing can be split up among many CPUs on a single Oracle system.



### 4.12.6 Analytic SQL

Oracle has many SQL operations for performing analytic operations in the database. These include ranking, moving averages, cumulative sums, ratio-to-reports, and period-over-period comparisons.

### 4.12.7 OLAP Capabilities

Application developers can use SQL online analytical processing (OLAP) functions for standard and ad-hoc reporting. For additional analytic functionality, Oracle OLAP provides multidimensional calculations, forecasting, modeling, and what-if scenarios. This enables developers to build sophisticated analytic and planning applications such as sales and marketing analysis, enterprise budgeting and financial analysis, and demand planning systems. The data can also be stored in either relational tables or multidimensional objects.

Oracle OLAP provides the query performance and calculation capability, previously found only in multidimensional databases to Oracle's relational platform. In addition, it provides a Java OLAP API that is appropriate for the development of internet-ready analytical applications. Unlike other combinations of OLAP and RDBMS technology, Oracle OLAP is not a multidimensional database using bridges to move data from the relational data store to a multidimensional data store. Instead, it is truly an OLAP-enabled relational database. As a result, Oracle provides the benefits of a multidimensional database along with the scalability, accessibility, security, manageability, and high availability of the Oracle database. The Java OLAP API, which is specifically designed for internet-based analytical applications, offers productive data access.

### 4.12.8 Data Mining

With Oracle Data Mining, data never leaves the database — the data, data preparation, model building, and model scoring results all remain in the database. This enables Oracle to provide an infrastructure for application developers to integrate data mining seamlessly with database applications. Some typical examples of the applications that data mining are used in are call centers, ATMs, ERM, and business planning applications. Data mining functions such as model building, testing, and scoring are provided through a Java API.

### 4.12.9 Partitioning

Partitioning addresses key issues in supporting very large tables and indexes by letting you decompose them into smaller and more manageable pieces called partitions. SQL queries and DML statements do not need to be modified in order to access partitioned tables. However, after partitions are defined, DDL statements can access and manipulate individual partitions rather than entire tables or indexes. This is how partitioning can simplify the manageability of large database objects. Also, partitioning is entirely transparent to applications.

Partitioning is useful for many different types of applications, particularly applications that manage large volumes of data. OLTP systems often benefit from improvements in manageability and availability, while data warehousing systems benefit from performance and manageability.

## 4.13 SECURITY FEATURES OF ORACLE

Oracle includes security features that control the accesssing and using of a database. For example, security mechanisms in Oracle:

- prevent unauthorised database access,
- prevent unauthorised access to schema objects, and
- audit user actions.

Associated with each database user is a schema by the same name. By default, each database user creates and has access to all objects in the corresponding schema.

Database security can be classified into two categories: system security and data security.

System security includes mechanisms that control the access and use of the database at the system level. For example, system security includes:

- valid user name/password combinations,
- the amount of disk space available to a user's schema objects, and
- the resource limits for a user.

System security mechanisms check whether a user is authorised to connect to the database, whether database auditing is active, and the system operations that a user has been permitted to perform.

Data security includes mechanisms that control the access and use of the database at the schema object level. For example, data security includes:

- users with access to a specific schema object and the specific types of actions permitted to each user on the schema object (for example, user MOHAN can issue SELECT and INSERT statements but not DELETE statements using the employees table),
- the actions, if any, that are audited for each schema object, and
- data encryption to prevent unauthorised users from bypassing Oracle and accessing the data.

**Security Mechanisms**

The Oracle database provides discretionary access control, which is a means of restricting access to information based on privileges. The appropriate privilege must be assigned to a user in order for that user to access a schema object. Appropriately privileged users can grant other users privileges at their discretion.

Oracle manages database security using several different facilities:

- authentication to validate the identity of the entities using your networks, databases, and applications,
- authorisation processes to limit access and actions, limits that are linked to user's identities and roles,
- access restrictions on objects, like tables or rows,
- security policies, and
- database auditing.

## 4.14 DATA INTEGRITY AND TRIGGERS IN ORACLE

Data must adhere to certain business rules, as determined by the database administrator or application developer. For example, assume that a business rule says that no row in the inventory table can contain a numeric value greater than nine in the sale_discount column. If an INSERT or UPDATE statement attempts to violate this integrity rule, then Oracle must undo the invalid statement and return an error to the



application. Oracle provides integrity constraints and database triggers to manage data integrity rules.

Database triggers let you define and enforce integrity rules, but a database trigger is not the same as an integrity constraint. Among other things, a database trigger does not check data already loaded into a table. Therefore, it is strongly recommended that you use database triggers only when the integrity rule cannot be enforced by integrity constraints.

**Integrity Constraints**

An integrity constraint is a declarative way to define a business rule for a column of a table. An integrity constraint is a statement about table data that is always true and that follows these rules:

- If an integrity constraint is created for a table and some existing table data does not satisfy the constraint, then the constraint cannot be enforced.

- After a constraint is defined, if any of the results of a DML statement violate the integrity constraint, then the statement is rolled back, and an error is returned.

Integrity constraints are defined with a table and are stored as part of the table's definition in the data dictionary, so that all database applications adhere to the same set of rules. When a rule changes, it only needs be changed once at the database level and not for each application.

The following integrity constraints are supported by Oracle:

- NOT NULL: Disallows nulls (empty entries) in a table's column.
- UNIQUE KEY: Disallows duplicate values in a column or set of columns.
- PRIMARY KEY: Disallows duplicate values and nulls in a column or set of columns.
- FOREIGN KEY: Requires each value in a column or set of columns to match a value in a related table's UNIQUE or PRIMARY KEY. FOREIGN KEY integrity constraints also define referential integrity actions that dictate what Oracle should do with dependent data if the data references is altered.
- CHECK: Disallows values that do not satisfy the logical expression of the constraint.

**Keys**

Key is used to define several types of integrity constraints. A key is the column or set of columns included in the definition of certain types of integrity constraints. Keys describe the relationships between the different tables and columns of a relational database. Individual values in a key are called key values.
The different types of keys include:

- **Primary key:** The column or set of columns included in the definition of a table's PRIMARY KEY constraint. A primary key's value uniquely identifies the rows in a table. Only one primary key can be defined for each table.

- **Unique key:** The column or set of columns included in the definition of a UNIQUE constraint.

- **Foreign key**: The column or set of columns included in the definition of a referential integrity constraint.

- **Referenced key:** The unique key or primary key of the same or a different table referenced by a foreign key.

**Triggers**

Triggers are procedures written in PL/SQL, Java, or C that run (fire) implicitly, whenever a table or view is modified or when some user actions or database system action occurs.

Triggers supplement the standard capabilities of Oracle to provide a highly customised database management system. For example, a trigger can restrict DML operations against a table to those issued during regular business hours.

## 4.15 TRANSACTIONS IN ORACLE

A transaction is a logical unit of work that comprises one or more SQL statements run by a single user. According to the ANSI/ISO SQL standard, with which Oracle is compatible, a transaction begins with the user's first executable SQL statement. A transaction ends when it is explicitly committed or rolled back by that user.

Transactions let users guarantee consistent changes to data, as long as the SQL statements within a transaction are grouped logically. A transaction should consist of all necessary parts for one logical unit of work – no more and no less. Data in all referenced tables are in a consistent state before the transaction begins and after it ends. Transactions should consist of only SQL statements that make one consistent change to the data.

Consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction can consist of three separate operations: decrease in the savings account, increase in the checking account, and recording the transaction in the transaction journal.

The transfer of funds (the transaction) includes increasing one account (one SQL statement), decreasing another account (one SQL statement), and recording the transaction in the journal (one SQL statement). All actions should either fail or succeed together; the credit should not be committed without the debit. Other non-related actions, such as a new deposit to one account, should not be included in the transfer of funds transaction. Such statements should be in other transactions.

Oracle must guarantee that all three SQL statements are performed to maintain accounts accurately. When something prevents one of the statements in the transaction from running (such as a hardware failure), then the other statements of the transaction must be undone. This is called rolling back. If an error occurs in making any of the updates, then no updates are made.

**Commit and Undo Transactions**

The changes made by the SQL statements that constitute a transaction can be either committed or rolled back. After a transaction is committed or rolled back, the next transaction begins with the next SQL statement.

To commit, a transaction makes permanent the changes resulting from all SQL statements in the transaction. The changes made by the SQL statements of a transaction become visible to other user sessions' transactions that start only after the transaction is committed.

To undo a transaction, retracts, any of the changes resulting from the SQL statements in the transaction. After a transaction is rolled back, the affected data is left unchanged, as if the SQL statements in the transaction were never run.

**Savepoints**



Savepoints divide a long transaction with many SQL statements into smaller parts. With savepoints, you can arbitrarily mark your work at any point within a long transaction. This gives you the option of later rolling back all work performed from the current point in the transaction to a declared savepoint within the transaction.

## 4.16 SQL VARIATIONS AND EXTENSIONS IN ORACLE

Oracle is compliant to industry-accepted standards and participates actively in SQL standards committees. Industry-accepted committees are the American National Standards Institute (ANSI) and the International Standards Organisation (ISO), affiliated to the International Electrotechnical Commission (IEC). Both ANSI and the ISO/IEC have accepted SQL as the standard language for relational databases. Oracle 10g conforms Core SQL: 2003 for most of the features except the following partially supported and not supported features.

Oracle partially supports these sub-features:

- CHARACTER VARYING data type (Oracle does not distinguish a zero-length VARCHAR string from NULL).
- Character literals (Oracle regards the zero-length literal “as being null”).
- Correlation names in FROM clause (Oracle supports correlation names, but not the optional AS keyword).
- WITH HOLD cursors (in the standard, a cursor is not held through a ROLLBACK, but Oracle does hold through ROLLBACK).
- ALTER TABLE statement: ADD COLUMN clause (Oracle does not support the optional keyword COLUMN in this syntax).
- User-defined functions with no overloading.
- User-defined procedures with no overloading.
- In the standard, the mode of a parameter (IN, OUT or INOUT) comes before the parameter name, whereas in Oracle it comes after the parameter name.
- The standard uses INOUT, whereas Oracle uses IN OUT.
- Oracle requires either IS or AS after the return type and before the definition of the routine body, while the standard lacks these keywords.

Oracle does not support the following sub-features:

- Rename columns in the FROM clause.
- DROP TABLE statement: RESTRICT clause.
- DROP VIEW statement: RESTRICT clause.
- REVOKE statement: RESTRICT clause.
- Features and conformance views.
- Distinct data types.

Oracle supports the following subfeatures in PL/SQL but not in Oracle SQL:

- RETURN statement.

Oracle’s triggers differ from the standard as follows:

- Oracle does not provide the optional syntax FOR EACH STATEMENT for the default case, the statement trigger.
- Oracle does not support OLD TABLE and NEW TABLE; the transition tables specified in the standard (the multiset of before and after images of affected rows) are not available.
- The trigger body is written in PL/SQL, which is functionally equivalent to the standard’s procedural language PSM, but not the same.

- In the trigger body, the new and old transition variables are referenced beginning with a colon.
- Oracle's row triggers are executed as the row is processed, instead of buffering them and executing all of them after processing all rows. The standard's semantics are deterministic, but Oracle's in-flight row triggers are more performant.
- Oracle's before row and before-statement triggers may perform DML statements, which is forbidden in the standard. On the other hand, Oracle's after-row statements may not perform DML, while it is permitted in the standard.
- When multiple triggers apply, the standard says they are executed in order of definition; in Oracle the execution order is non-deterministic.

In addition to traditional structured data, Oracle is capable of storing, retrieving, and processing more complex data.

- Object types, collection types, and REF types provide support for complex structured data. A number of standard-compliant multiset operators are now supported by the nested table collection type.
- Large objects (LOBs) provide support for both character and binary unstructured data. A single LOB reach a size of 8 to 128 terabytes, depending on the size of the database block.
- The XML datatype provides support for semistructured XML data.

### ☞ Check Your Progress 4

1) What is ETL?

……

……

2) What is Parallel Execution?

……

……

3) With Oracle …….data never leaves the database.

4) What does system security include?

……

……

5) How does Oracle manage database security?

……

……

6) …….is a declarative way to defined a business rule for a column of a table.

7) What are the different integrity constraints supported by oracle?

……

……

8) Name the different types of keys.

……

……

9) What is a trigger?



……

……

10) ……divides a long transaction with many SQL statements into smaller parts.

## 4.17 SUMMARY

This unit provided an introduction to Oracle, one of the commercial DBMS in the market. Some other products include MS SQL server, DB2 by IBM and so on. Oracle being a commecial DBMS supports the features of the Relational Model and also some of the Object oriented models. It has a very powerful Database Application Development Feature that is used for database programming. Oracle supports the Standard Query Language (SQL) and the use of embedded languages including C, JAVA etc.

The Oracle Architecture defines the physical and logical database components of the database that is stored in the Oracle environment. All database objects are defined using a schema definition. This information is stored in the data dictionary. This data dictionary is used actively to find the schema objects, integrity and security constraints on those objects etc. Oracle defines an instance as the present database state. There are many Oracle backend processes that support various operations on oracle database. The database is stored in the SGA area of the memory.

Oracle is relational as the basic technology supports query optimisation. Query optimisation is needed for commercial databases, as the size of such databases may be very high. Oracle supports indexing using the B-tree structure, in addition the bit wise index makes indexing even faster. Oracle also supports distributed database technology. It supports replication, transportability of these replicas and advanced queuing and streams.

Oracle has many tools for system administration. They support basic used management to backup and recovery tools. Oracle Lite is the version of Oracle used for mobile database. Oracle uses standard implementation methods for concurrency control. Oracle has Data Warehousing capabilities. It contains Extraction, Transformation and Loading (ETL) tools, materialised views, bitmap indexes, table compression and Data mining and OLAP in support to a Data Warehouse. Oracle supports both system and database security features.

The unit also explained data integrity, transactions, SQL Variations and Extensions in Oracle.

## 4.18 SOLUTIONS/ANSWERS

### Check Your Progress 1

1) These statements create, alter, maintain and drop schemes objects.

2) Transaction Control Statements.

3) (a) Character
   (b) Numeric
   (c) DATE data type
   (d) LOB data type
   (e) RAW and LONGRAW
   (f) ROWID and UNROWID data type

4) ORACLE forms Developer
   ORACLE Reports Developer
   ORACLE Designer
   ORACLE J Developer
   ORACLE Discoverer Administrative Edition

ORACLE Portal.




## Check Your Progress 2

1) (a) Data files
   (b) Redo log files
   (c) Control files

2) Logical.

3) Tables

4)
   (a) All Leaf Block have same depth,
   (b) B-Tree indexes are balanced,
   (c) Blocks of the B+ tree are 3 quarters full on the average,
   (d) B+ tree have excellent retrieval performance,
   (e) Inserts, updates and deletes are all efficient, and
   (f) Good Performance.

5)
   (1) Valid users of Oracle data types,
   (2) Information about integrity Constraints defined for the tables, and
   (3) The Amount of space Allocated for a scheme Object.

6) Cursor

7) Program Global Area

8) A process is a thread of control or a mechanism in an operating system that can run a series of steps.

9)
   (a) Log writers,
   (b) Archiver,
   (c) Recover.

## Check Your Progress 3

1) Translating a SQL statement and verifying it to be a valid statement.

2)
   (1) Identifying high load on top of SQL Statements.
   (2) Verifying that the execution plans perform reasonably, and
   (3) Implementing corrective actions to generate better execution plans.

3) Query optimiser

4) It allows non – oracle data and services to be accessed from an Oracle database through generic connectivity.

5)
   (1) Basic replication,
   (2) Advanced replication,
   (3) Transportable replication,
   (4) Advanced queuing and streams, and
   (5) Extraction, transformation, loading.

6) A complete interface for enterprise wide application development

7)
   (a) Complete database,
   (b) Tablespace,
   (c) Data file,
   (d) Control file, and
   (e) Achieved Redo Log.



8) Oracle LITE.

9) Real application clusters.

## Check Your Progress 4

1) ETL means extraction, transformation and loading. It is basically the process of extracting data from source systems and transferring it to the data warehouse.

2) The process of making multiple processes run together.

3) Data Mining.

4) It includes the mechanisms that control access and use of the database at the system level.

5) Oracle manages database security using:
   - Authentication,
   - Authorisation,
   - Access restrictions,
   - Security policies, and
   - Database Auditing.

6) Integrity constraints.

7)
   - NOT NULL
   - UNIQUE KEY
   - PRIMARY KEY
   - FOREIGN KEY
   - CHECK

8)
   - Primary
   - Unique
   - Foreign
   - Referenced

9) It is a event driven procedure.

10) Save points.