# ¿Qué es una base de datos?

- Una base de datos es una herramienta para recopilar y organizar información.
- En las bases de datos, se puede almacenar información sobre personas, productos, pedidos, o cualquier otra cosa.
- Muchas bases de datos empiezan siendo una lista en un programa de procesamiento de texto o en una hoja de cálculo.
- A medida que crece la lista, empiezan a aparecer repeticiones e inconsistencias en los datos.

# ¿Qué es una base de datos?

- Cada vez resulta más complicado comprender los datos presentados en la lista y existen pocos métodos para buscar o recuperar subconjuntos de datos para revisarlos.
- Cuando empiezan a observarse estos problemas, es aconsejable transferir la información a una base de datos creada mediante un sistema de administración de bases de datos (SGBD)

# ¿Qué es una base de datos?

- Una base de datos informatizada es un contenedor de objetos.
- Una base de datos puede contener más de una tabla.
- Por ejemplo, un sistema de seguimiento de inventario que utiliza tres tablas no es un conjunto de tres bases de datos, sino una sola base de datos que contiene tres tablas.

# Con SGBD, puede:

- Crear o modificar Bases de Datos
- Agregar más datos a una base de datos, por ejemplo, un elemento nuevo en un inventario.
- Modificar datos existentes en la base de datos, por ejemplo, cambiar la ubicación de un elemento.
- Eliminar información, por ejemplo, si se ha vendido o retirado un artículo.
- Organizar y ver los datos de distintas formas.
- Compartir los datos con otros usuarios mediante informes, mensajes de correo electrónico, una intranet o Internet.

# Tablas

- Una tabla de una base de datos es similar en apariencia a una hoja de cálculo, en cuanto a que los datos se almacenan en filas y columnas.
- Como consecuencia, normalmente es bastante fácil importar una hoja de cálculo en una tabla de una base de datos.
- La principal diferencia entre almacenar los datos en una hoja de cálculo y hacerlo en una base de datos es la forma de organizarse los datos.

# Tablas

- Para lograr la máxima flexibilidad para una base de datos, la información tiene que estar organizada en tablas, para que no haya redundancias.
- Por ejemplo, si se almacena información sobre empleados, cada empleado se insertará una sola vez en una tabla que se configurará para contener únicamente datos de los empleados.
- Los datos sobre productos se almacenarán en su propia tabla, y los datos sobre sucursales también tendrán su tabla aparte.
- Este proceso se conoce como ***normalización***.

# Tablas

- Cada fila de una tabla se denomina registro.
- En los registros es donde se almacena cada información individual. Cada registro consta de campos (al menos uno).
- Los campos corresponden a las columnas de la tabla.
- Por ejemplo, puede trabajar con una tabla denominada "Empleados", en la que cada registro (fila) contiene información sobre un empleado distinto y cada campo (columna) contiene un tipo de información diferente, como el nombre, los apellidos, la dirección, o similares.
- Los campos se deben configurar con un determinado tipo de datos, ya sea texto, fecha, hora, numérico, o cualquier otro tipo.

# Tablas

- Otra forma de describir registros y campos es imaginando un catálogo de fichas tradicional de una biblioteca.

- Cada ficha del armario corresponde a un *registro* de la base de datos.

- Cada información contenida en una ficha (autor, título, etc.) corresponde a un *campo* de la base de datos.

# Consultas

- Las consultas son las que verdaderamente hacen el trabajo en una base de datos.
- Su función más común es recuperar datos específicos de las tablas.
- Los datos que desea ver suelen estar distribuidos por varias tablas y, gracias a las consultas, puede verlos en una sola hoja de datos.

# Consultas

- Además, puesto que normalmente no desea ver todos los registros a la vez, las consultas le permiten agregar criterios para "filtrar" los datos hasta obtener solo los registros que desee.

- Las consultas a menudo sirven de origen de registros para informes

# Consultas

- Algunas consultas son “de actualización", lo que significa que es posible editar los datos de las tablas base mediante la hoja de datos de la consulta.

- Si trabaja con una consulta actualizable, recuerde que los cambios se producen también en las tablas, no solo en la hoja de datos de la consulta.

# Consultas

- Hay dos tipos básicos de consultas:
  - las de selección
  - las de acción.

- Una consulta de selección simplemente recupera los datos y hace que estén disponibles para su uso.
- Los resultados de la consulta pueden verse en la pantalla, imprimirse o se pueden utilizar como origen para un informe.

# Consultas

- Una consulta de acción, como su nombre indica, realiza una tarea con los datos.

- Las consultas de acción pueden servir para crear tablas nuevas, agregar datos a tablas existentes, actualizar datos o eliminar datos.

# Conceptos básicos del diseño de una base de datos

- Una base de datos correctamente diseñada permite obtener acceso a información exacta y actualizada.

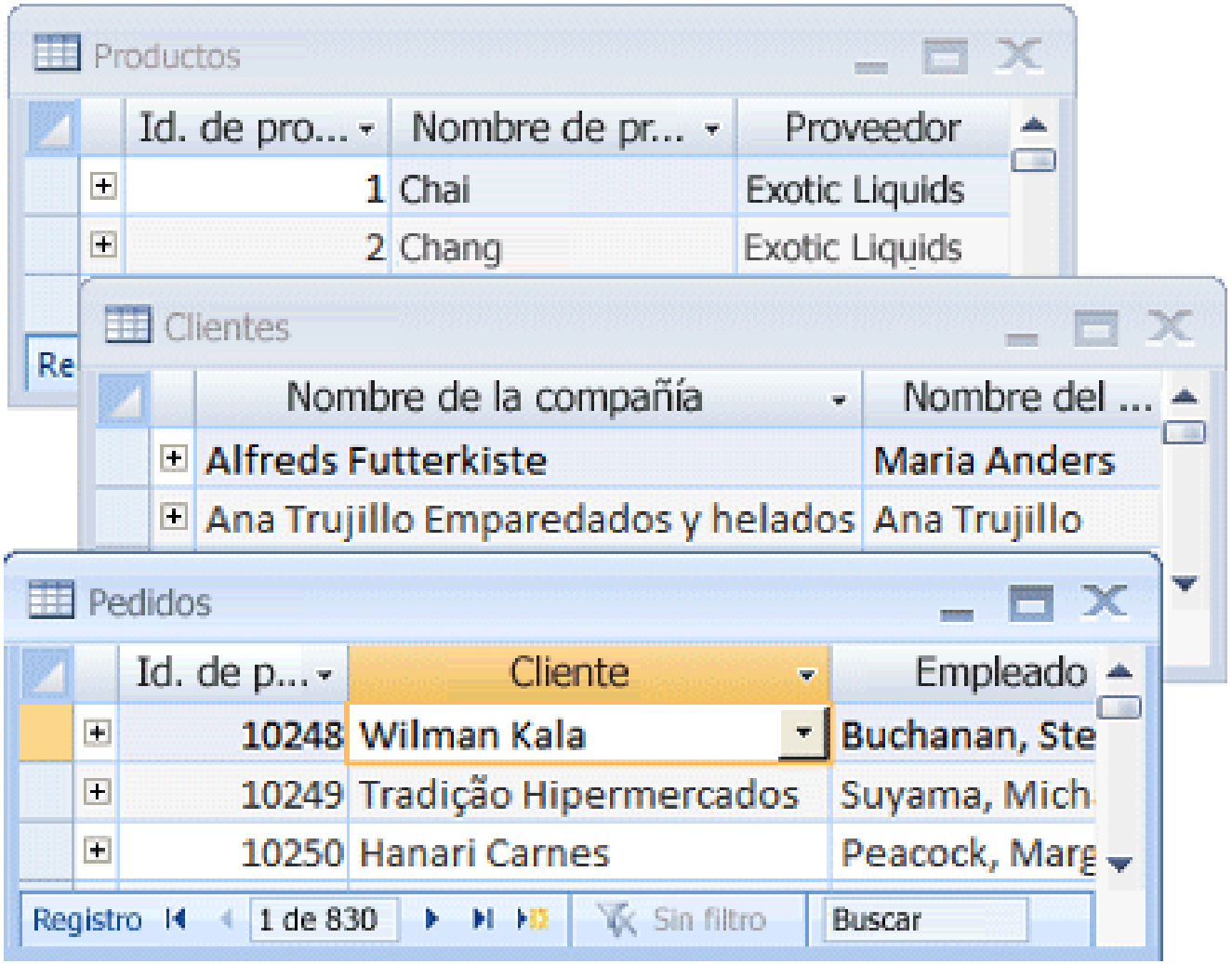
- Puesto que un diseño correcto es esencial para lograr los objetivos fijados para la base de datos, parece lógico emplear el tiempo que sea necesario en aprender los principios de un buen diseño ya que, en ese caso, es mucho más probable que la base de datos termine adaptándose a sus necesidades y pueda modificarse fácilmente.

# Algunos términos sobre bases de datos que debe conocer

- Una base de datos simple puede que sólo contenga una tabla, pero la mayoría de las bases de datos necesitan varias tablas.

- Por ejemplo, podría tener una tabla con información sobre productos, otra con información sobre pedidos y una tercera con información sobre clientes.

# Algunos términos sobre bases de datos que debe conocer



# Algunos términos sobre bases de datos que debe conocer

- Cada fila recibe también el nombre de **registro** y cada columna se denomina también **campo**.
- Un registro es una forma lógica y coherente de combinar información sobre alguna cosa.
- Un campo es un elemento único de información: un tipo de elemento que aparece en cada registro.
- En la tabla Productos, por ejemplo, cada fila o registro contendría información sobre un producto, y cada columna contendría algún dato sobre ese producto, como su nombre o el precio.

# ¿Qué es un buen diseño de base de datos?

- El proceso de diseño de una base de datos se guía por algunos principios:
  - El primero de ellos es que se debe evitar la información duplicada o, lo que es lo mismo, los datos redundantes, porque malgastan el espacio y aumentan la probabilidad de que se produzcan errores e incoherencias.

# ¿Qué es un buen diseño de base de datos?

- El segundo principio es que es importante que la información sea correcta y completa.

- Si la base de datos contiene información incorrecta, los informes que recogen información de la base de datos contendrán también información incorrecta y, por tanto, las decisiones que tome a partir de esos informes estarán mal fundamentadas.

## Un buen diseño de base de datos es, por tanto, aquél que:

- Divide la información en tablas basadas en temas para reducir los datos redundantes.
- Proporciona al SGBD la información necesaria para reunir la información de las tablas cuando así se precise.
- Ayuda a garantizar la exactitud e integridad de la información.
- Satisface las necesidades de procesamiento de los datos y de generación de informes.

# El proceso de diseño

- **Determinar la finalidad de la base de datos**
  - Esto le ayudará a estar preparado para los demás pasos.

- **Buscar y organizar la información necesaria**
  - Reúna todos los tipos de información que desee registrar en la base de datos, como los nombres de productos o los números de pedidos.

# El proceso de diseño

- **Dividir la información en tablas**
  - Divida los elementos de información en entidades o temas principales, como Productos o Pedidos. Cada tema pasará a ser una tabla.

- **Convertir los elementos de información en columnas**
  - Decida qué información desea almacenar en cada tabla. Cada elemento se convertirá en un campo y se mostrará como una columna en la tabla. Por ejemplo, una tabla Empleados podría incluir campos como Apellido y Fecha de contratación.

# El proceso de diseño

- **Especificar claves principales**
  - Elija la clave principal de cada tabla. La clave principal es una columna que se utiliza para identificar inequívocamente cada fila, como Id. de producto o Id. de pedido.

- **Definir relaciones entre las tablas**
  - Examine cada tabla y decida cómo se relacionan los datos de una tabla con las demás tablas. Agregue campos a las tablas o cree nuevas tablas para clarificar las relaciones según sea necesario.

# El proceso de diseño

- **Ajustar el diseño**
  - Analice el diseño para detectar errores.
  - Cree las tablas y agregue algunos registros con datos de ejemplo.
  - Compruebe si puede obtener los resultados previstos de las tablas.
  - Realice los ajustes necesarios en el diseño.

- **Aplicar las reglas de normalización**
  - Aplique reglas de normalización de los datos para comprobar si las tablas están estructuradas correctamente.
  - Realice los ajustes necesarios en las tablas.

# Determinar la finalidad de la base de datos

- Es conveniente plasmar en papel el propósito de la base de datos: cómo piensa utilizarla y quién va a utilizarla.
  - Para una pequeña base de datos de un negocio particular, por ejemplo, podría escribir algo tan simple como "La base de datos de clientes contiene una lista de información de los clientes para el envío masivo de correo y la generación de informes".

# Determinar la finalidad de la base de datos

- Si la base de datos es más compleja o la utilizan muchas personas, como ocurre normalmente en un entorno corporativo, la finalidad podría definirse fácilmente en uno o varios párrafos y debería incluir cuándo y cómo va a utilizar cada persona la base de datos.

# Determinar la finalidad de la base de datos

- La idea es desarrollar una declaración de intenciones bien definida que sirva de referencia durante todo el proceso de diseño.
- Esta declaración de intenciones le permitirá centrarse en los objetivos a la hora de tomar decisiones.

# Buscar y organizar la información necesaria

- Para buscar y organizar la información necesaria, empiece con la información existente.
  - Por ejemplo, si registra los pedidos de compra en un libro contable o guarda la información de los clientes en formularios en papel en un archivador, puede reunir esos documentos y enumerar cada tipo de información que contienen (por ejemplo, cada casilla de un formulario).

# Buscar y organizar la información necesaria

- Cuando prepare esta lista, no se preocupe si no es perfecta al principio.
- Simplemente, enumere cada elemento que se le ocurra.
- Si alguien más va a utilizar la base de datos, pídale también su opinión.
- Más tarde podrá ajustar la lista.

# Buscar y organizar la información necesaria

- A continuación, considere los tipos de informes o la correspondencia que desea producir con la base de datos.
- Por ejemplo, tal vez desee crear un informe de ventas de productos que contenga las ventas por región, o un informe de resumen de inventario con los niveles de inventario de los productos.

# Buscar y organizar la información necesaria

- Es posible que también desee generar cartas modelo para enviárselas a los clientes con un anuncio de una actividad de ventas o una oferta.
- Diseñe el informe en su imaginación y piense cómo le gustaría que fuera. ¿Qué información incluiría en el informe? Cree un listado de cada elemento.
- Haga lo mismo para la carta modelo y para cualquier otro informe que tenga pensado crear.

# Buscar y organizar la información necesaria

# Buscar y organizar la información necesaria

- Detenerse a pensar en los informes y en la correspondencia que desea crear le ayudará a identificar los elementos que necesita incluir en la base de datos.

- Suponga, por ejemplo, que ofrece a sus clientes la oportunidad de inscribirse o borrarse de las actualizaciones periódicas de correo electrónico y desea imprimir un listado de los que han decidido inscribirse.

- Para registrar esa información, agrege una columna "Enviar correo electrónico" a la tabla de clientes. Para cada cliente, puede definir el campo en Sí o No.

# Buscar y organizar la información necesaria

- La necesidad de enviar mensajes de correo electrónico a los clientes implica la inclusión de otro elemento.
- Cuando sepa que un cliente desea recibir mensajes de correo electrónico, tendrá que conocer también la dirección de correo electrónico a la que éstos deben enviarse.
- Por tanto, tendrá que registrar una dirección de correo electrónico para cada cliente.

# Buscar y organizar la información necesaria

- Parece lógico crear un prototipo de cada informe o listado de salida y considerar qué elementos necesita para crear el informe.

- Por ejemplo, cuando examine una carta modelo, puede que se le ocurran algunas ideas. Si desea incluir un saludo (por ejemplo, las abreviaturas "Sr." o "Sra." con las que comienza un saludo), tendrá que crear un elemento de saludo.

- Además, tal vez desee comenzar las cartas con el saludo "Estimado Sr. García", en lugar de "Estimado Sr. Miguel Ángel García". Esto implicaría almacenar el apellido independientemente del nombre.

# Buscar y organizar la información necesaria

- Un punto clave que hay que recordar es que debe descomponer cada pieza de información en sus partes lógicas más pequeñas.

- En el caso de un nombre, para poder utilizar el apellido, dividirá el nombre en dos partes: el nombre y el apellido.

- Para ordenar un informe por nombre, por ejemplo, sería útil que el apellido de los clientes estuviera almacenado de forma independiente.

- En general, si desea ordenar, buscar, calcular o generar informes a partir de un elemento de información, debe incluir ese elemento en su propio campo.

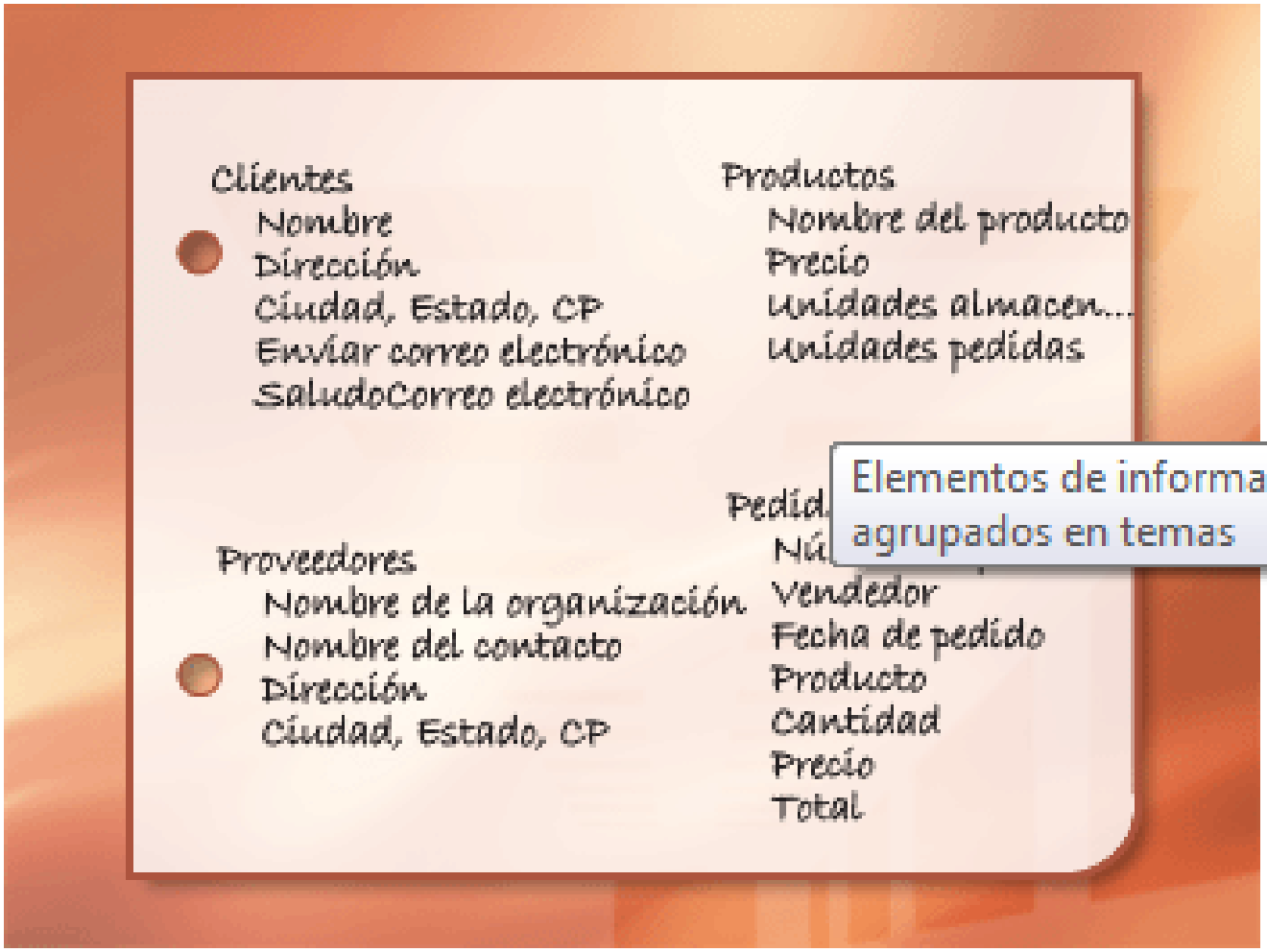
# Buscar y organizar la información necesaria

- Piense en las preguntas que le gustaría que la base de datos contestara.
- Por ejemplo, ¿cuántas ventas de un determinado producto se cerraron el pasado mes? ¿Dónde viven sus mejores clientes? ¿Quién es el proveedor del producto mejor vendido?
- Prever esas preguntas le ayudará a determinar los elementos adicionales que necesita registrar.
- Una vez reunida esta información, ya puede continuar con el paso siguiente.

# Dividir la información en tablas

- Para dividir la información en tablas, elija las entidades o los temas principales.
  - Por ejemplo, después de buscar y organizar la información de una base de datos de ventas de productos, la lista preliminar podría ser similar a la siguiente:

# Dividir la información en tablas



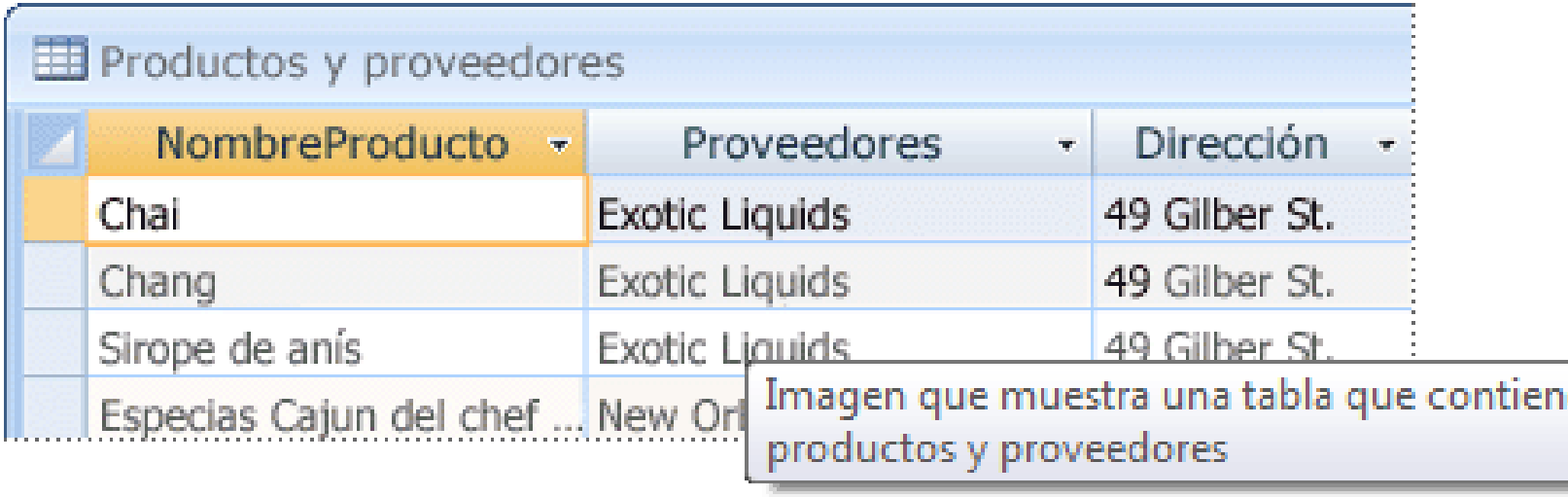
# Dividir la información en tablas

- Las entidades principales mostradas aquí son los productos, los proveedores, los clientes y los pedidos.
- Por tanto, parece lógico empezar con estas cuatro tablas:
  - una para los datos sobre los productos
  - otra para los datos sobre los proveedores
  - otra para los datos sobre los clientes
  - otra para los datos sobre los pedidos.
- Aunque esto no complete la lista, es un buen punto de partida.
- Puede seguir ajustando la lista hasta obtener un diseño correcto.

# Dividir la información en tablas

- Cuando examine por primera vez la lista preliminar de elementos, podría estar tentado a incluirlos todos ellos en una sola tabla en lugar de en las cuatro tablas mostradas en la ilustración anterior.
- A continuación le explicaremos por qué eso no es una buena idea.
- Considere por un momento la tabla que se muestra a continuación:

# Dividir la información en tablas



# Dividir la información en tablas

- En este caso, cada fila contiene información sobre el producto y su proveedor.
- Como hay muchos productos del mismo proveedor, la información del nombre y la dirección del proveedor debe repetirse muchas veces, con lo que se malgasta el espacio en disco.
- Registrar la información del proveedor una sola vez en una tabla Proveedores distinta y luego vincular esa tabla a la tabla Productos es una solución mucho mejor.

# Dividir la información en tablas

- Otro problema de este diseño surge cuando es necesario modificar la información del proveedor.
- Suponga, por ejemplo, que necesita cambiar la dirección de un proveedor.
- Como ésta aparece en muchos lugares, podría sin querer cambiar la dirección en un lugar y olvidarse de cambiarla en los demás lugares.
- Ese problema se resuelve registrando la información del proveedor en un único lugar.

# Dividir la información en tablas

- Cuando diseñe la base de datos, intente registrar siempre cada dato una sola vez.

- Si descubre que está repitiendo la misma información en varios lugares, como la dirección de un determinado proveedor, coloque esa información en una tabla distinta.

# Dividir la información en tablas

- Por último, suponga que el proveedor Bodega Sol sólo suministra un producto y desea eliminar ese producto pero conservar el nombre del proveedor y la información de dirección. ¿Cómo eliminaría el producto sin perder la información del proveedor?

- No puede. Como cada registro contiene datos sobre un producto, además de datos sobre un proveedor, no puede eliminar unos sin eliminar los otros.

- Para mantener estos datos separados, debe dividir la tabla en dos:
  - una tabla para la información sobre los productos
  - otra tabla para la información sobre los proveedores.

- Al eliminar un registro de producto sólo se eliminarían los datos del producto y no los datos del proveedor.

# Dividir la información en tablas

- Una vez seleccionado el tema representado por una tabla, las columnas de esa tabla deben almacenar datos únicamente sobre ese tema.
- Por ejemplo, la tabla de productos sólo debe contener datos de productos.
- Como la dirección del proveedor es un dato del proveedor, pertenece a la tabla de proveedores.

# Convertir los elementos de información en columnas

- Para determinar las columnas de una tabla, decida qué información necesita registrar sobre el tema representado por la tabla.

- Por ejemplo, para la tabla Clientes, una buena lista de columnas iniciales sería:
  - Nombre
  - Dirección
  - Ciudad-Provincia-Código postal
  - Enviar correo electrónico
  - Saludo
  - Correo electrónico.

# Convertir los elementos de información en columnas

- Cada registro de la tabla contiene el mismo número de columnas, por lo que puede almacenar información sobre el nombre, dirección, ciudad-provincia-código postal, envío de correo electrónico, saludo y dirección de correo electrónico para cada registro.

- Por ejemplo, la columna de dirección podría contener las direcciones de los clientes.

- Cada registro contendrá datos sobre un cliente y el campo de dirección, la dirección de ese cliente.

# Convertir los elementos de información en columnas

- Cuando haya determinado el conjunto inicial de columnas para cada tabla, puede ajustar con mayor precisión las columnas.

- Por ejemplo, tiene sentido almacenar los nombres de los clientes en dos columnas distintas (el nombre y el apellido) para poder ordenar, buscar e indizar por esas columnas.

# Convertir los elementos de información en columnas

- De igual forma, la dirección consta en realidad de cinco componentes distintos:
  - dirección
  - ciudad
  - provincia,
  - código postal
  - país o región

- parece lógico también almacenarlos en columnas distintas.

# Convertir los elementos de información en columnas

- Si desea realizar, por ejemplo, una búsqueda o una operación de ordenación o filtrado por provincia, necesita que la información de provincia esté almacenada en una columna distinta.

# Convertir los elementos de información en columnas

- Debe considerar también si la base de datos va a contener información sólo de procedencia nacional o internacional.
  - Por ejemplo, si piensa almacenar direcciones internacionales, es preferible tener una columna Región en lugar de Provincia, ya que esa columna puede incluir provincias del propio país y regiones de otros países o regiones.

# Convertir los elementos de información en columnas

- De igual forma, es más lógico incluir una columna Región en lugar de Comunidad Autónoma si va a almacenar direcciones internacionales.

## En la lista siguiente se incluyen algunas sugerencias para determinar las columnas de la base de datos.

- No incluya datos calculados
- Almacene la información en sus partes lógicas más pequeñas

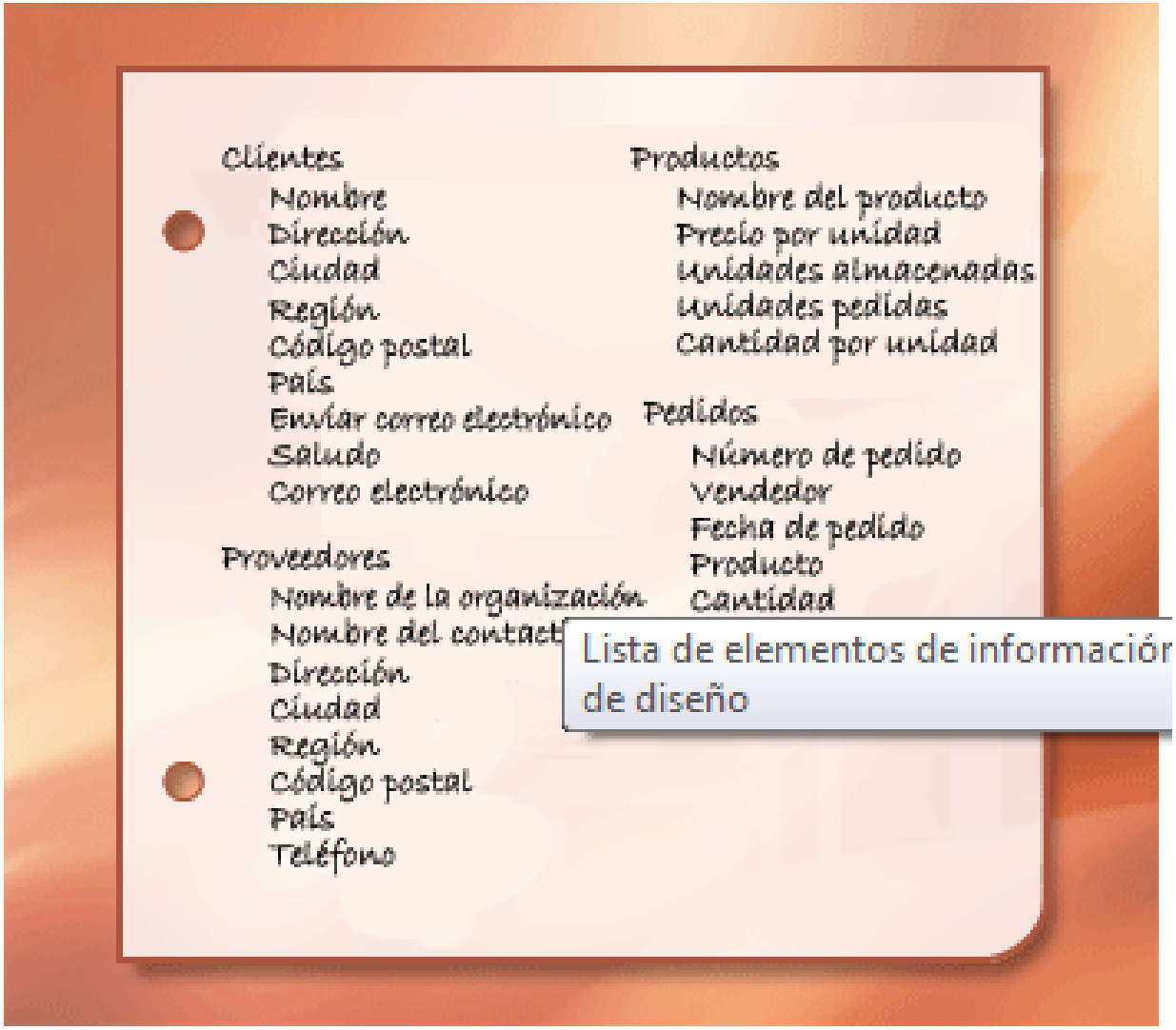
# No incluya datos calculados

- En la mayoría de los casos, no debe almacenar el resultado de los cálculos en las tablas.
- En lugar de ello, puede dejar que SGBD realice los cálculos cuando desee ver el resultado.
  - Por ejemplo, un informe Productos bajo pedido que contiene el subtotal de unidades de un pedido para cada categoría de producto de la base de datos. Sin embargo, no hay ninguna tabla que contenga una columna de subtotal Unidades en pedido. La tabla Productos contiene una columna Unidades en pedido que almacena las unidades incluidas en un pedido de cada producto. Con esos datos, SGBD calcula el subtotal cada vez que se imprime el informe, pero el subtotal propiamente dicho no debe almacenarse en una tabla.

# Almacene la información en sus partes lógicas más pequeñas

- Puede ceder a la tentación de habilitar un único campo para los nombres completos o para los nombres de productos junto con sus descripciones.
- Si combina varios tipos de información en un campo, será difícil recuperar datos individuales más adelante.
- Intente dividir la información en partes lógicas.
  - Por ejemplo, cree campos distintos para el nombre y el apellido, o para el nombre del producto, la categoría y la descripción.

# Almacene la información en sus partes lógicas más pequeñas



# Almacene la información en sus partes lógicas más pequeñas

- Una vez ajustadas las columnas de datos de cada tabla, ya puede seleccionar la **clave principal** de cada tabla.

# Especificar claves principales

- Cada tabla debe incluir una columna o conjunto de columnas que identifiquen inequívocamente cada fila almacenada en la tabla.
- Ésta suele ser un número de identificación exclusivo, como un número de identificador de empleado o un número de serie.
- En la terminología de bases de datos, esta información recibe el nombre de **clave principal** de la tabla.
- El SGBD puede crear índices de la clave principal de una tabla para mejorar la velocidad de acceso.

# Especificar claves principales

- Si ya tiene un identificador exclusivo para una tabla, como un número de producto que identifica inequívocamente cada producto del catálogo, puede utilizar ese identificador como clave principal de la tabla, pero sólo si los valores de esa columna son siempre diferentes para cada registro.

- No puede tener valores duplicados en una clave principal.
  - Por ejemplo, no utilice los nombres de las personas como clave principal, ya que los nombres no son exclusivos. Es muy fácil que dos personas tengan el mismo nombre en la misma tabla.

# Especificar claves principales

- Una clave principal siempre debe tener un valor.

- Si el valor de una columna puede quedar sin asignar o vacío (porque no se conoce) en algún momento, **no** puede utilizarlo como componente de una clave principal.

# Especificar claves principales

- Debe elegir siempre una clave principal cuyo valor no cambie.
- En una base de datos con varias tablas, la clave principal de una tabla se puede utilizar como referencia en las demás tablas.
- Si la clave principal cambia, el cambio debe aplicarse también a todos los lugares donde se haga referencia a la clave.
- Usar una clave principal que no cambie reduce la posibilidad de que se pierda su sincronización con las otras tablas en las que se hace referencia a ella.

# Especificar claves principales

- A menudo, se utiliza como clave principal un número único arbitrario.
  - Por ejemplo, puede asignar a cada pedido un número de pedido distinto.
  - La única finalidad de este número de pedido es identificar el pedido.
  - Una vez asignado, nunca cambia.

# Especificar claves principales

- Si piensa que no hay ninguna columna o conjunto de columnas que pueda constituir una buena clave principal, considere la posibilidad de utilizar una columna que tenga el tipo de datos **Autonumérico.**

- Cuando se utiliza el tipo de datos Autonumérico, el SGBD asigna automáticamente un valor.

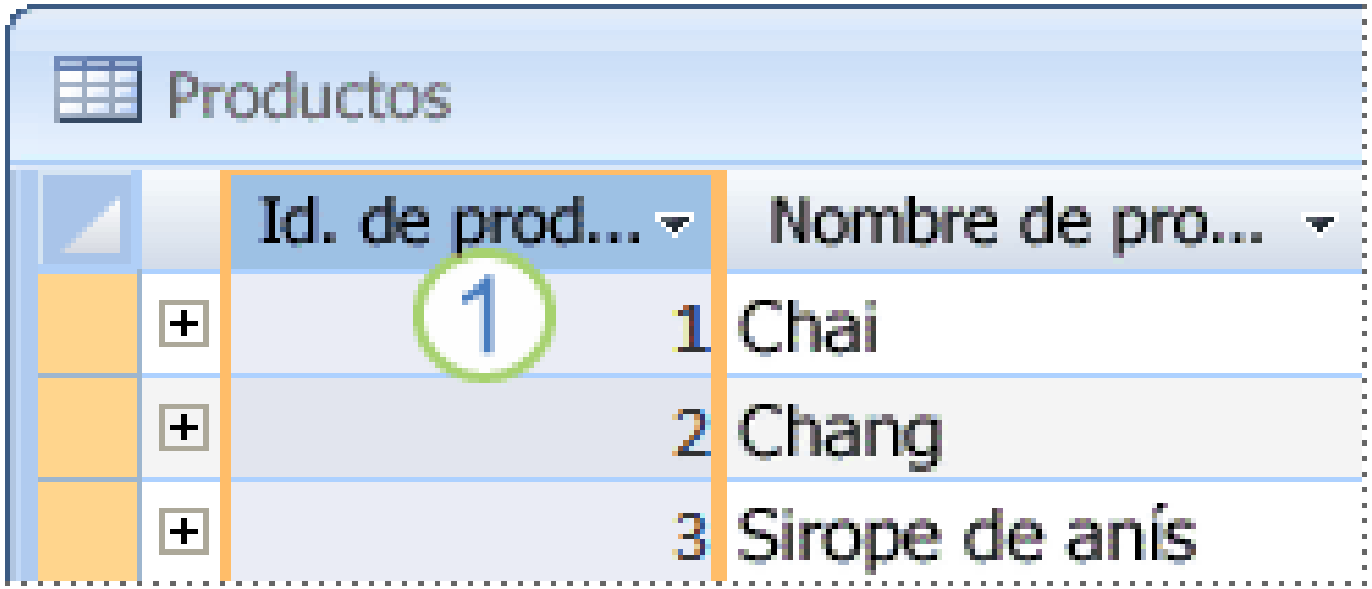
- Este tipo de identificador no es "fáctico", es decir, no contiene información objetiva sobre la fila que representa.

# Especificar claves principales

- Los identificadores de este tipo, **Autonumericos**, son perfectos para usarlos como claves principales, ya que no cambian.

- Una clave principal que contiene datos sobre una fila, como un número de teléfono o el nombre de un cliente, es más probable que cambie, ya que la propia información "fáctica" podría cambiar.

# Especificar claves principales




Una columna establecida en el tipo de datos **Autonumérico** suele constituir una buena clave principal.
No hay dos identificadores de producto iguales.

# Especificar claves principales

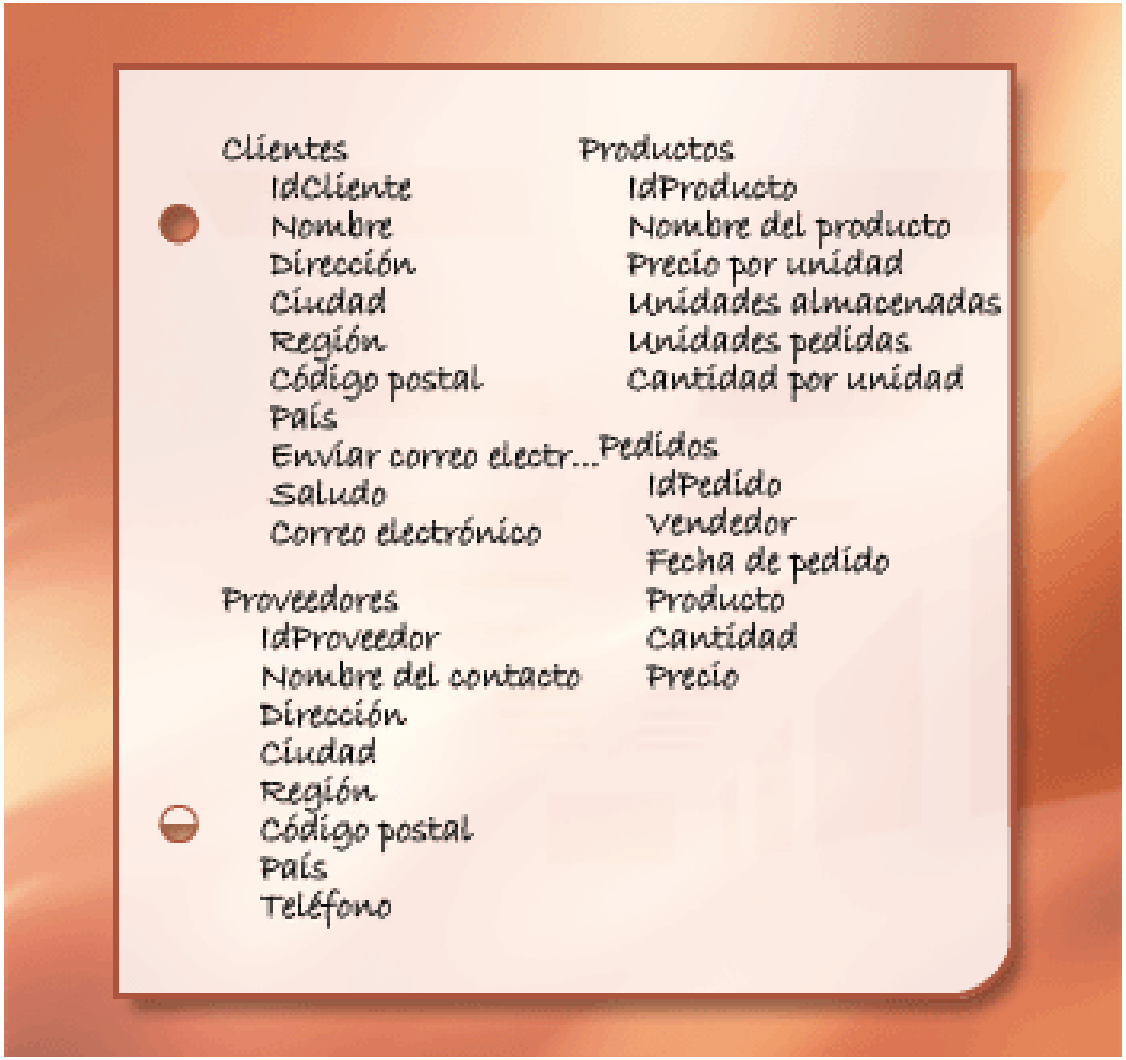
- En algunos casos, tal vez considere conveniente utilizar dos o más campos juntos como clave principal de una tabla.
  - Por ejemplo, una tabla Detalles de pedidos que contenga artículos de línea de pedidos tendría dos columnas en su clave principal:
    - Id. de pedido
    - Id. de producto.
  - Cuando una clave principal está formada por más de una columna se denomina clave compuesta.

# Especificar claves principales

- Para la base de datos de ventas de productos, puede crear una columna autonumérica para cada una de las tablas que funcione como clave principal:

  - IdProducto para la tabla Productos
  - IdPedido para la tabla Pedidos
  - IdCliente para la tabla Clientes
  - IdProveedores para la tabla Proveedores.

# Especificar claves principales

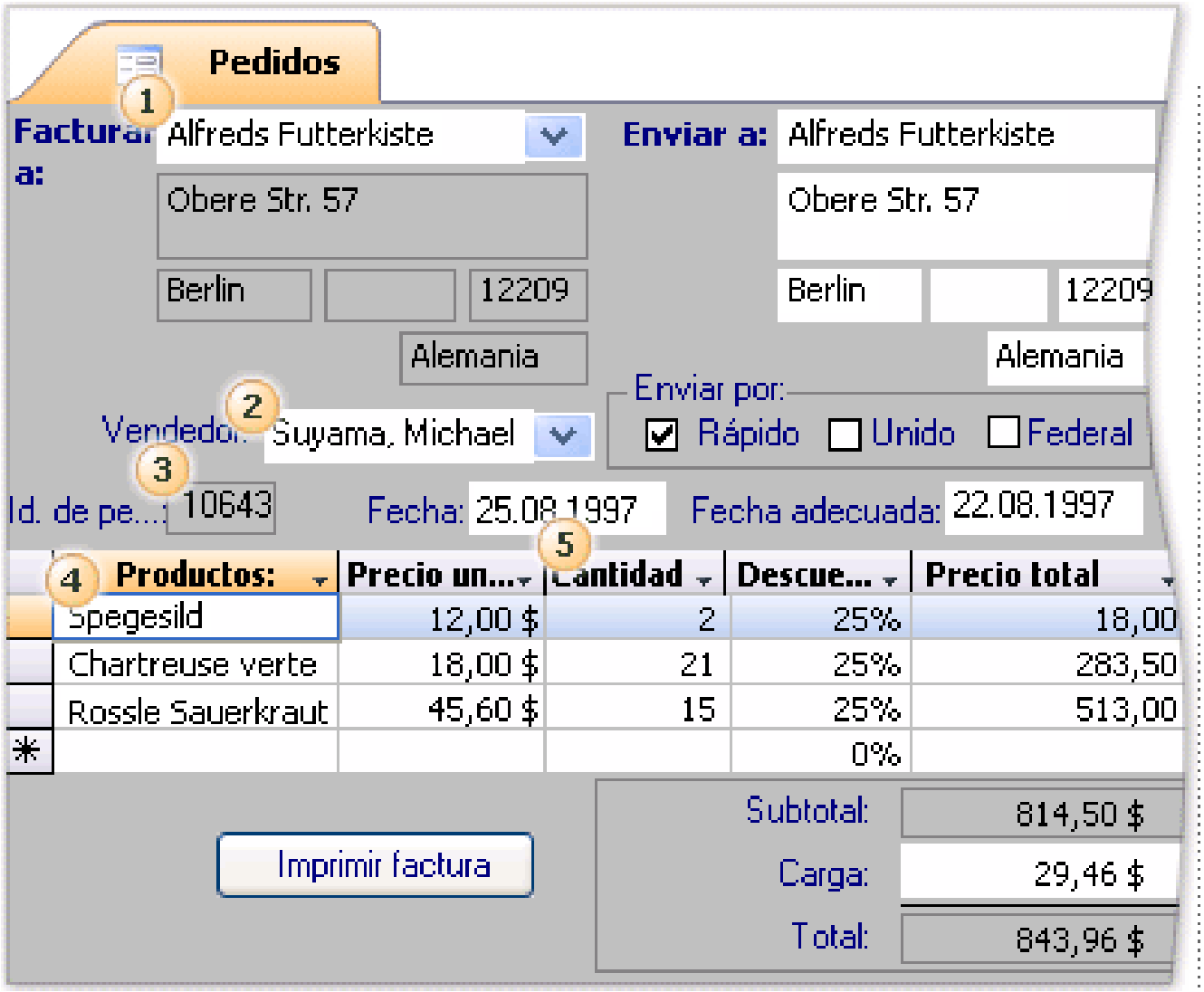


# Crear relaciones entre las tablas

- Ahora que ha dividido la información en tablas necesita un modo de reunir de nuevo la información de forma provechosa.

- Por ejemplo, el siguiente formulario incluye información de varias tablas.

# Crear relaciones entre las tablas



# Crear relaciones entre las tablas

La información de este formulario procede:

- 1 …de la tabla Clientes...
- 2 ...la tabla Empleados...
- 3 ...la tabla Pedidos...
- 4 ...la tabla Productos...
- 5 ...y la tabla Detalles de pedidos.

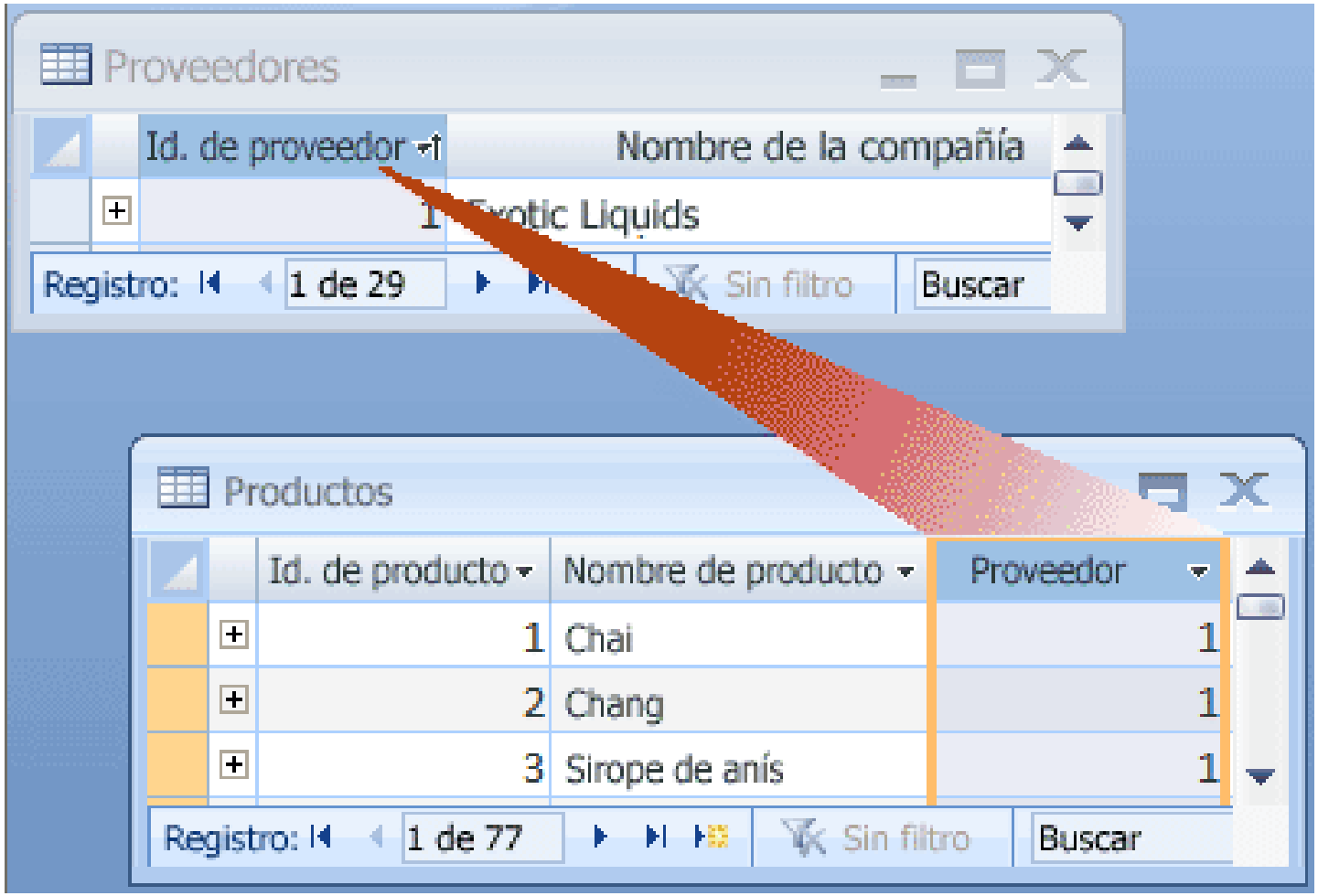
# Crear relaciones entre las tablas

- SGDB es un sistema de administración de bases de datos relacionales.
- En una base de datos relacional, la información se divide en tablas distintas en función del tema.
- A continuación, se utilizan relaciones entre las tablas para reunir la información según se precise.

# Crear una relación de uno a varios

- Considere este ejemplo: las tablas Proveedores y Productos de la base de datos de pedidos de productos.
- Un proveedor puede suministrar cualquier número de productos y, por consiguiente, para cada proveedor representado en la tabla Proveedores, puede haber muchos productos representados en la tabla Productos.
- La relación entre la tabla Proveedores y la tabla Productos es, por tanto, una relación de **uno a varios**.

# Crear una relación de uno a varios



# Crear una relación de uno a varios

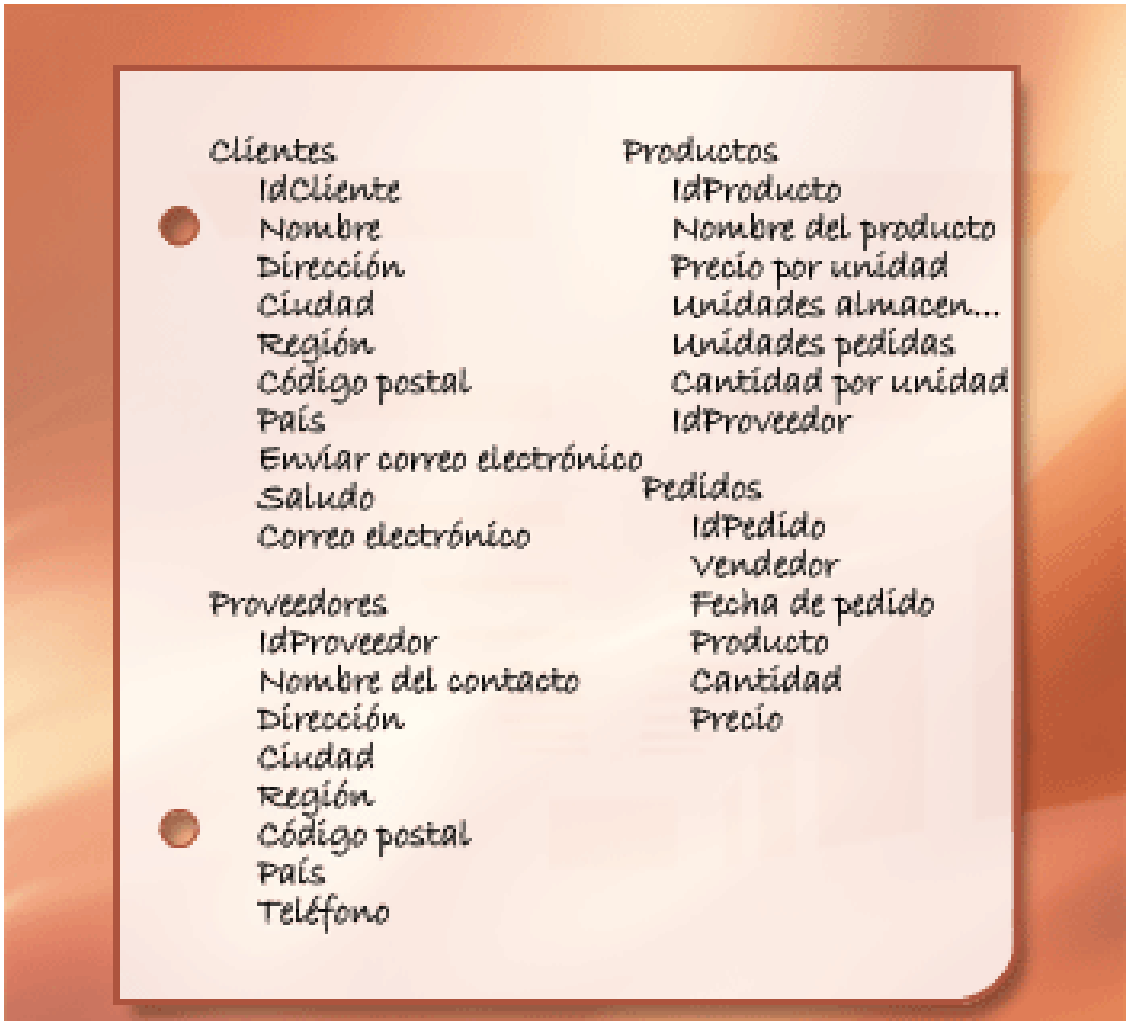
- Para representar una relación de uno a varios en el diseño de la base de datos, tome la clave principal del lado "uno" de la relación y agréguela como columna o columnas adicionales a la tabla en el lado "varios" de la relación.

- En este caso, por ejemplo, agregaría la columna Id. de proveedor de la tabla Proveedores a la tabla Productos. SGBD utilizaría entonces el número de identificador de proveedor de la tabla Productos para localizar el proveedor correcto de cada producto.

# Crear una relación de uno a varios

- La columna Id. de proveedor de la tabla Productos se denomina clave externa.
- Una clave externa es la clave principal de otra tabla.
- La columna Id. de proveedor de la tabla Productos en una clave externa porque también es la clave principal en la tabla Proveedores.

# Crear una relación de uno a varios



# Crear una relación de uno a varios

- El punto de partida para la unión de tablas relacionadas se proporciona estableciendo parejas de claves principales y claves externas.

# Crear una relación de varios a varios

- Considere la relación entre la tabla Productos y la tabla Pedidos.

# Crear una relación de varios a varios

- Un solo pedido puede incluir varios productos.
- Por otro lado, un único producto puede aparecer en muchos pedidos.
- Por tanto, para cada registro de la tabla Pedidos puede haber varios registros en la tabla Productos. Y para cada registro de la tabla Productos puede haber varios registros en la tabla Pedidos.
- Este tipo de relación se denomina **relación de varios a varios** porque para un producto puede haber varios pedidos, y para un pedido puede haber varios productos.

# Crear una relación de varios a varios

- Los temas de las dos tablas (pedidos y productos) tienen una relación de varios a varios.
- Esto presenta un problema.
- Para comprender el problema, imagine qué sucedería si intenta crear la relación entre las dos tablas agregando el campo Id. de producto a la tabla Pedidos.

# Crear una relación de varios a varios

- Para que haya más de un producto por pedido, necesita más de un registro en la tabla Pedidos para cada pedido y, en ese caso, tendría que repetir la información de pedido para cada fila relacionada con un único pedido, lo que daría lugar a un diseño ineficaz que podría producir datos inexactos.

# Crear una relación de varios a varios

- El mismo problema aparece si coloca el campo Id. de pedido en la tabla Productos:
  - tendría varios registros en la tabla Productos para cada producto.

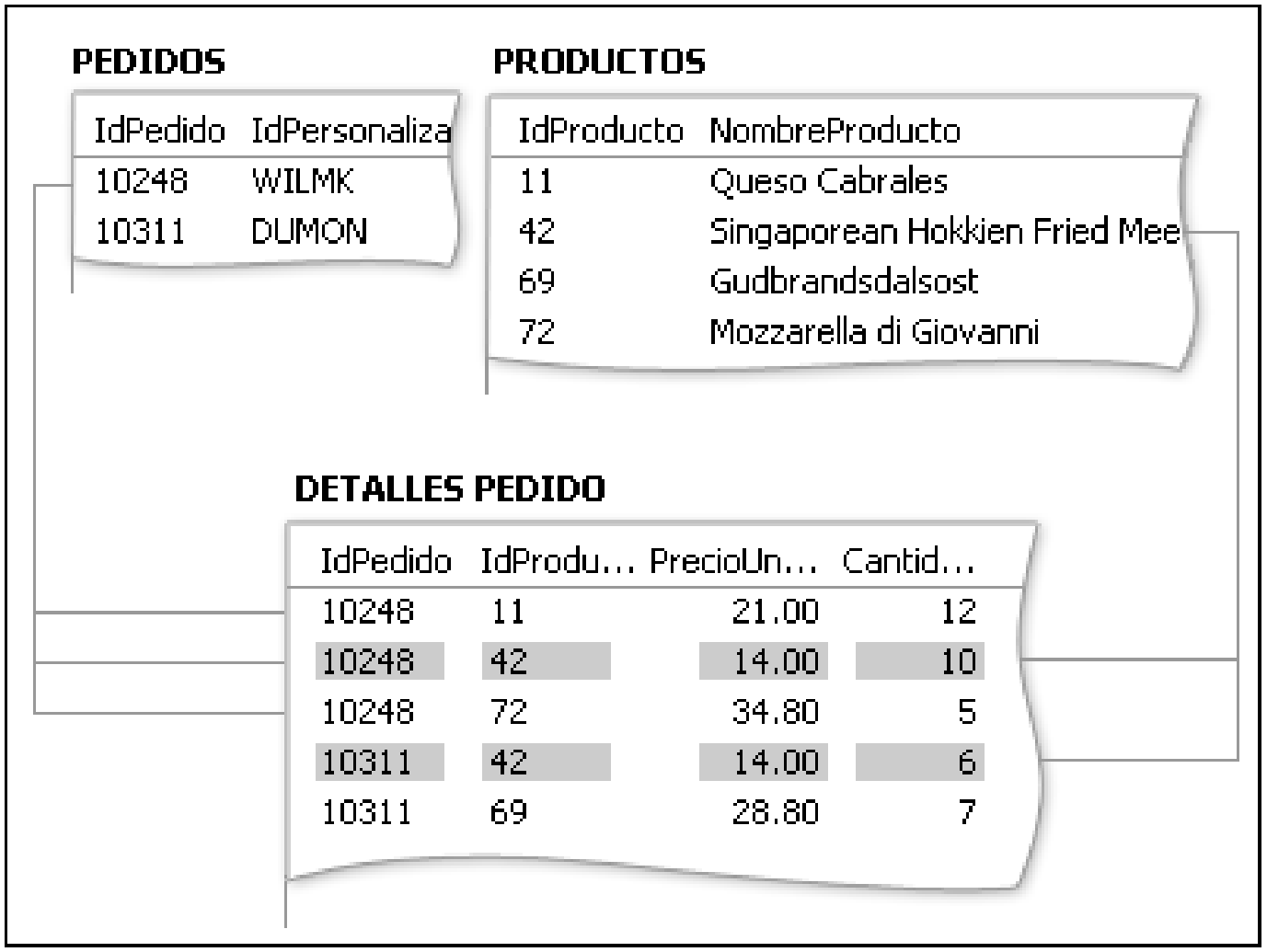
- ¿Cómo se soluciona este problema?

# Crear una relación de varios a varios

- La solución a este problema consiste en crear una tercera tabla que descomponga la relación de varios a varios en dos relaciones de uno a varios.

- Insertaría la clave principal de cada una de las dos tablas en la tercera tabla y, por consiguiente, la tercera tabla registraría todas las apariciones o instancias de la relación.

# Crear una relación de varios a varios



# Crear una relación de varios a varios

- Cada registro de la tabla Detalles de pedidos representa un artículo de línea de un pedido.
- La clave principal de la tabla Detalles de pedidos consta de dos campos:
  - las claves externas de las tablas Pedidos y Productos.
- El campo Id. de pedido no se puede utilizar en solitario como clave principal, ya que un pedido puede tener varios artículos de línea.

# Crear una relación de varios a varios

- El identificador de pedido se repite para cada artículo de línea del pedido, por lo que el campo no contiene valores únicos.
- Tampoco serviría utilizar solamente el campo Id. de producto, porque un producto puede aparecer en varios pedidos.
- Pero los dos campos juntos producen un valor exclusivo para cada registro.

# Crear una relación de varios a varios

- En la base de datos de ventas de productos, la tabla Pedidos y la tabla Productos no se relacionan directamente entre sí, sino indirectamente a través de la tabla Detalles de pedidos.

- La relación de varios a varios entre los pedidos y los productos se representa en la base de datos mediante dos relaciones de uno a varios:

# Crear una relación de varios a varios

- La tabla Pedidos y la tabla Detalles de pedidos tienen una relación de uno a varios.
  - Cada pedido tiene varios artículos de línea, pero cada artículo está asociado a un único pedido.

- La tabla Productos y la tabla Detalles de pedidos tienen una relación de uno a varios.
  - Cada producto puede tener varios artículos asociados, pero cada artículo de línea hace referencia únicamente a un producto.

# Crear una relación de varios a varios

- Desde la tabla Detalles de pedidos puede determinar todos los productos de un determinado pedido, así como todos los pedidos de un determinado producto.

# Crear una relación de varios a varios

Después de incorporar la tabla Detalles de pedidos, la lista de tablas y campos sería similar a la siguiente:

# Crear una relación de uno a uno

- Otro tipo de relación es la relación de uno a uno.
  - Suponga, por ejemplo, que necesita registrar información complementaria sobre productos que apenas va a necesitar o que sólo se aplica a unos pocos productos.

# Crear una relación de uno a uno

- Como no necesita la información con frecuencia, y como almacenar la información en la tabla Productos crearía un espacio vacío para todos los productos que no necesitan esa información, la coloca en una tabla distinta.

- Al igual que en la tabla Productos, utiliza el identificador de producto como clave principal.

# Crear una relación de uno a uno

- La relación entre esta tabla complementaria y la tabla Productos es una relación de uno a uno.
- Para cada registro de la tabla Productos hay un único registro coincidente en la tabla complementaria.
- Cuando identifique esta relación, ambas tablas deben compartir un campo común.

# Crear una relación de uno a uno

- Cuando necesite crear una relación de uno a uno en la base de datos, considere si puede incluir la información de las dos tablas en una tabla.

- Si no desea hacer eso por algún motivo (quizás porque se crearía una gran cantidad de espacio vacío), puede representar esa relación en su diseño guiándose por las pautas siguientes:

# Crear una relación de uno a uno

- Si las dos tablas tienen el mismo tema, probablemente podrá definir la relación utilizando la misma clave principal en ambas tablas.

- Si las dos tablas tienen temas diferentes con claves principales distintas, elija una de las tablas (cualquiera de ellas) e inserte su clave principal en la otra tabla como clave externa.

# Crear una relación de uno a uno

- Determinar las relaciones entre las tablas le ayudará a asegurarse de que tiene las tablas y columnas correctas.
- Cuando existe una relación de uno a uno o de uno a varios, las tablas implicadas deben compartir una o varias columnas comunes.
- Cuando la relación es de varios a varios, se necesita una tercera tabla para representar la relación.

# Ajustar el diseño

- Cuando tenga las tablas, los campos y las relaciones necesarias, debe crear y rellenar las tablas con datos de ejemplo y probar que funcionan con la información:
  - creando consultas, agregando nuevos registros, etc.

- Esto le permitirá encontrar posibles problemas, como la necesidad de agregar una columna que olvidó insertar durante la fase de diseño, o dividir una tabla en dos tablas para eliminar datos duplicados.

# Ajustar el diseño

- Compruebe si puede usar la base de datos para obtener las respuestas que desea.
- Cree formularios e informes provisionales y compruebe si muestran los datos según lo previsto.
- Compruebe si existen datos duplicados innecesarios y, si encuentra alguno, modifique el diseño para eliminar la duplicación.

# Ajustar el diseño

- Cuando pruebe la base de datos inicial, probablemente se dará cuenta de que se puede mejorar. Éstas son algunas comprobaciones que puede hacer:

# Ajustar el diseño

- ¿Olvidó incluir alguna columna? Y, en ese caso, ¿pertenece la información a alguna de las tablas existentes? Si se trata de información sobre otro tema, tal vez necesite crear otra tabla.

  - Cree una columna para cada elemento de información que desee registrar. Si la información no se puede calcular a partir de otras columnas, es probable que necesite una nueva columna para esa información.

# Ajustar el diseño

- ¿Hay alguna columna innecesaria porque se puede calcular con los campos existentes?
  - Si un elemento de información se puede calcular a partir de otras columnas existentes (como un descuento calculado a partir del precio de venta al público), normalmente es preferible que se calcule en lugar de crear una nueva columna

# Ajustar el diseño

- ¿Ha proporcionada información duplicada en alguna de las tablas?
  - Si es así, probablemente tendrá que dividir la tabla en dos tablas que tengan una relación de uno a varios.

# Ajustar el diseño

- ¿Tiene tablas con muchos campos, un número limitado de registros y muchos campos vacíos en cada registro?
  - En ese caso, considere la posibilidad de volver a diseñar la tabla de forma que tenga menos campos y más registros.

# Ajustar el diseño

- ¿Ha dividido cada elemento de información en sus partes lógicas más pequeñas?
  - Si necesita generar informes, ordenar, buscar o calcular a partir de un elemento de información, incluya ese elemento en su propia columna.

# Ajustar el diseño

- ¿Contiene cada columna datos sobre el tema de la tabla?
  - Si una columna no contiene información sobre el tema de la tabla, pertenece a una tabla distinta.

# Ajustar el diseño

- ¿Están representadas todas las relaciones entres las tablas mediante campos comunes o mediante una tercera tabla?

  - Las relaciones de uno a uno y de uno a varios requieren columnas comunes. Las relaciones de varios a varios requieren una tercera tabla.

# Ajustar la tabla Productos

- Suponga que cada producto de la base de datos de ventas de productos pertenece a una categoría general, como bebidas, condimentos o marisco.
  - La tabla Productos podría incluir un campo que mostrara la categoría de cada producto.

# Ajustar la tabla Productos

- Suponga que después de examinar y ajustar el diseño de la base de datos, decide almacenar una descripción de la categoría junto con su nombre.
  - Si agrega un campo Descripción de categoría a la tabla Productos, tendría que repetir la descripción de cada categoría para cada producto perteneciente a dicha categoría, lo cual no es una buena solución.

# Ajustar la tabla Productos

- Una solución mejor es convertir las categorías en un nuevo tema de la base de datos, con su propia tabla y su propia clave principal.

- A continuación, puede agregar la clave principal de la tabla Categorías a la tabla Productos como clave externa.

# Ajustar la tabla Productos

- Las tablas Categorías y Productos tienen una relación de uno a varios:
  - una categoría puede incluir varios productos, pero un producto pertenece únicamente a una categoría.

# Ajustar la tabla Productos

- Cuando examine las estructuras de las tablas, compruebe si existen grupos extensibles. Por ejemplo, considere una tabla con las siguientes columnas:

  - Id. de producto
  - Nombre
  - Id1 de producto
  - Nombre1
  - Id2 de producto
  - Nombre2
  - Id3 de producto
  - Nombre3

# Ajustar la tabla Productos

- Aquí, cada producto es un grupo extensible de columnas que se diferencia de los demás solamente por el número agregado al final del nombre de columna. Si tiene columnas numeradas de esta forma, debe revisar el diseño.

# Ajustar la tabla Productos

- Un diseño como éste tiene varios defectos. Para empezar, obliga a crear un límite máximo de número de productos. En cuanto supere ese límite, deberá agregar un nuevo grupo de columnas a la estructura de la tabla, lo que supone una tarea administrativa importante.

# Ajustar la tabla Productos

- Otro problema es que se malgastará el espacio en aquellos proveedores que tengan menos que el número máximo de productos, ya que las columnas adicionales quedarán en blanco.

- El defecto más grave de este diseño es que muchas tareas son difíciles de realizar, como ordenar o indizar la tabla por identificador de producto o nombre.

# Ajustar la tabla Productos

- Siempre que aparezcan grupos extensibles, revise atentamente el diseño con vistas a dividir la tabla en dos.

- En el ejemplo anterior, sería conveniente usar dos tablas, una para proveedores y otra para productos, vinculadas por el identificador de proveedor.

# Aplicar las reglas de normalización

- En el siguiente paso del diseño, puede aplicar las reglas de normalización de datos (denominadas a veces simplemente reglas de normalización).

- Estas reglas sirven para comprobar si las tablas están estructuradas correctamente.

# Aplicar las reglas de normalización

- La normalización es más útil una vez representados todos los elementos de información y después de haber definido un diseño preliminar.
- La idea es asegurarse de que se han dividido los elementos de información en las tablas adecuadas.
- Lo que la normalización no puede hacer es garantizar que se dispone de los elementos de datos correctos para empezar a trabajar.

# Aplicar las reglas de normalización

- Las reglas se aplican consecutivamente en cada paso para garantizar que el diseño adopta lo que se conoce como "forma normal".
- Hay cinco formas normales ampliamente aceptadas: de la primera forma normal a la quinta forma normal.
  - Aquí se abordan las tres primeras, porque todas ellas son necesarias para la mayoría de los diseños de base de datos.

# Primera forma normal

- La primera forma normal establece que en cada intersección de fila y columna de la tabla existe un valor y nunca una lista de valores.
- Por ejemplo, no puede haber un campo denominado Precio en el que se incluya más de un precio.
- Si considera cada intersección de filas y columnas como una celda, cada celda sólo puede contener un valor.

# Segunda forma normal

- La segunda forma normal exige que cada columna que no sea clave dependa por completo de toda la clave principal y no sólo de parte de la clave.

- Esta regla se aplica cuando existe una clave principal formada por varias columnas.

# Segunda forma normal

- Suponga, por ejemplo, que existe una tabla con las siguientes columnas, de las cuales Id. de pedido e Id. de producto forman la clave principal:
  - Id. de pedido (clave principal)
  - Id. de producto (clave principal)
  - Nombre de producto

# Segunda forma normal

- Este diseño infringe los requisitos de la segunda forma normal, porque Nombre de producto depende de Id. de producto, pero no de Id. de pedido, por lo que no depende de toda la clave principal.

- Debe quitar Nombre de producto de la tabla, ya que pertenece a una tabla diferente (a la tabla Productos).

# Tercera forma normal

- La tercera forma normal exige no sólo que cada columna que no sea clave dependa de toda la clave principal, sino también que las columnas que no sean clave sean independientes unas de otras.

# Tercera forma normal

- O dicho de otra forma: cada columna que no sea clave debe depender de la clave principal y nada más que de la clave principal.
- Por ejemplo, considere una tabla con las siguientes columnas:
  - IdProducto (clave principal)
  - Nombre
  - PVP
  - Descuento

# Tercera forma normal

- Suponga que la columna Descuento depende del precio de venta al público (PVP) sugerido.

- Esta tabla infringe los requisitos de la tercera forma normal porque una columna que no es clave, la columna Descuento, depende de otra columna que no es clave, la columna PVP.

# Tercera forma normal

- La independencia de las columnas implica que debe poder cambiar cualquier columna que no sea clave sin que ninguna otra columna resulte afectada.
- Si cambia un valor en el campo PVP, la columna Descuento cambiaría en consecuencia e infringiría esa regla.
- En este caso, la columna Descuento debe moverse a otra tabla cuya clave sea PVP.

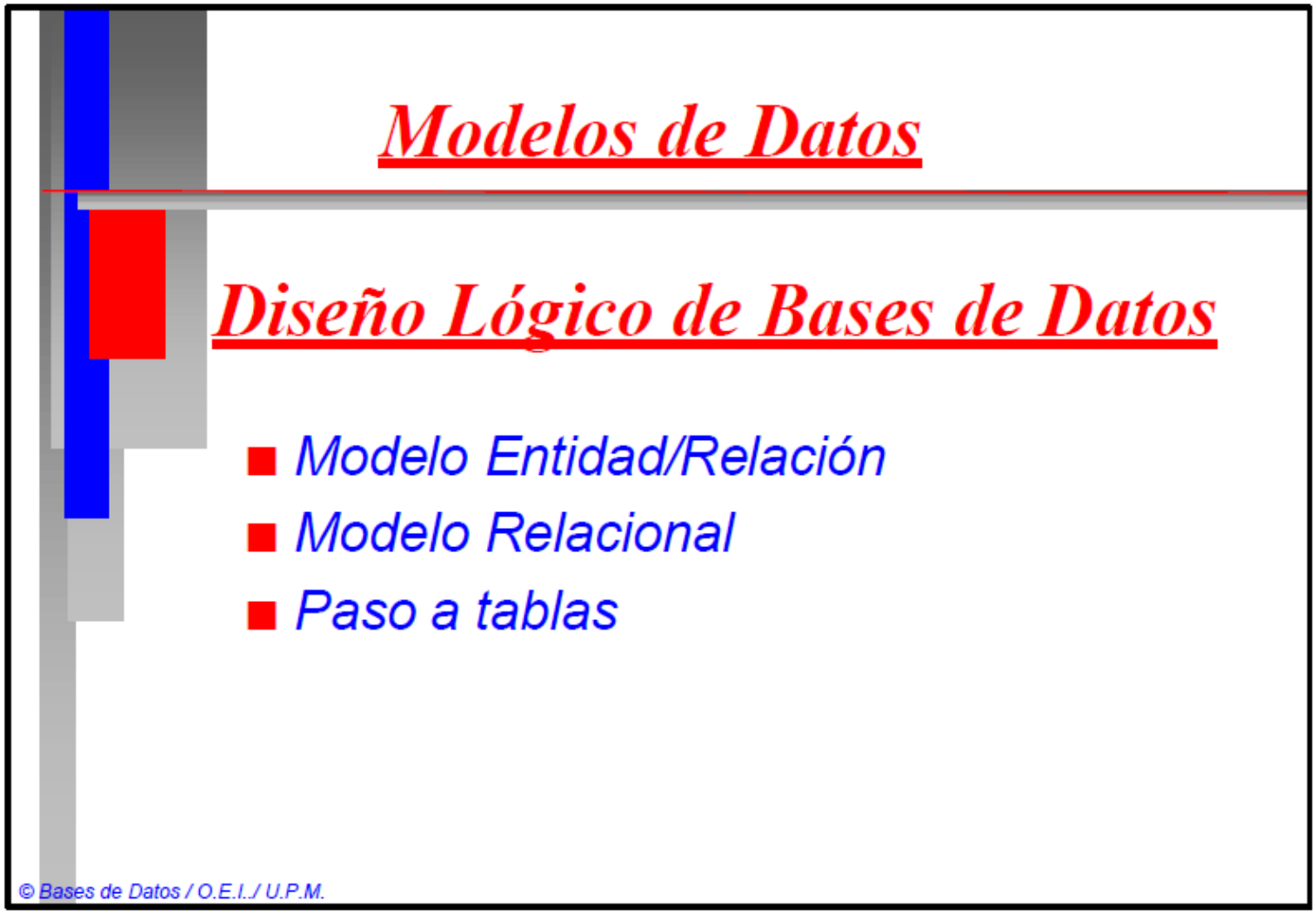
# Para obtener más información


- Hernandez, Michael J. *Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design, Second Edition*. Addison-Wesley Professional. 2003.

- Fleming, Candace C. von Halle, Barbara. *Handbook of Relational Database Design*. Addison-Wesley Professional. 1989.

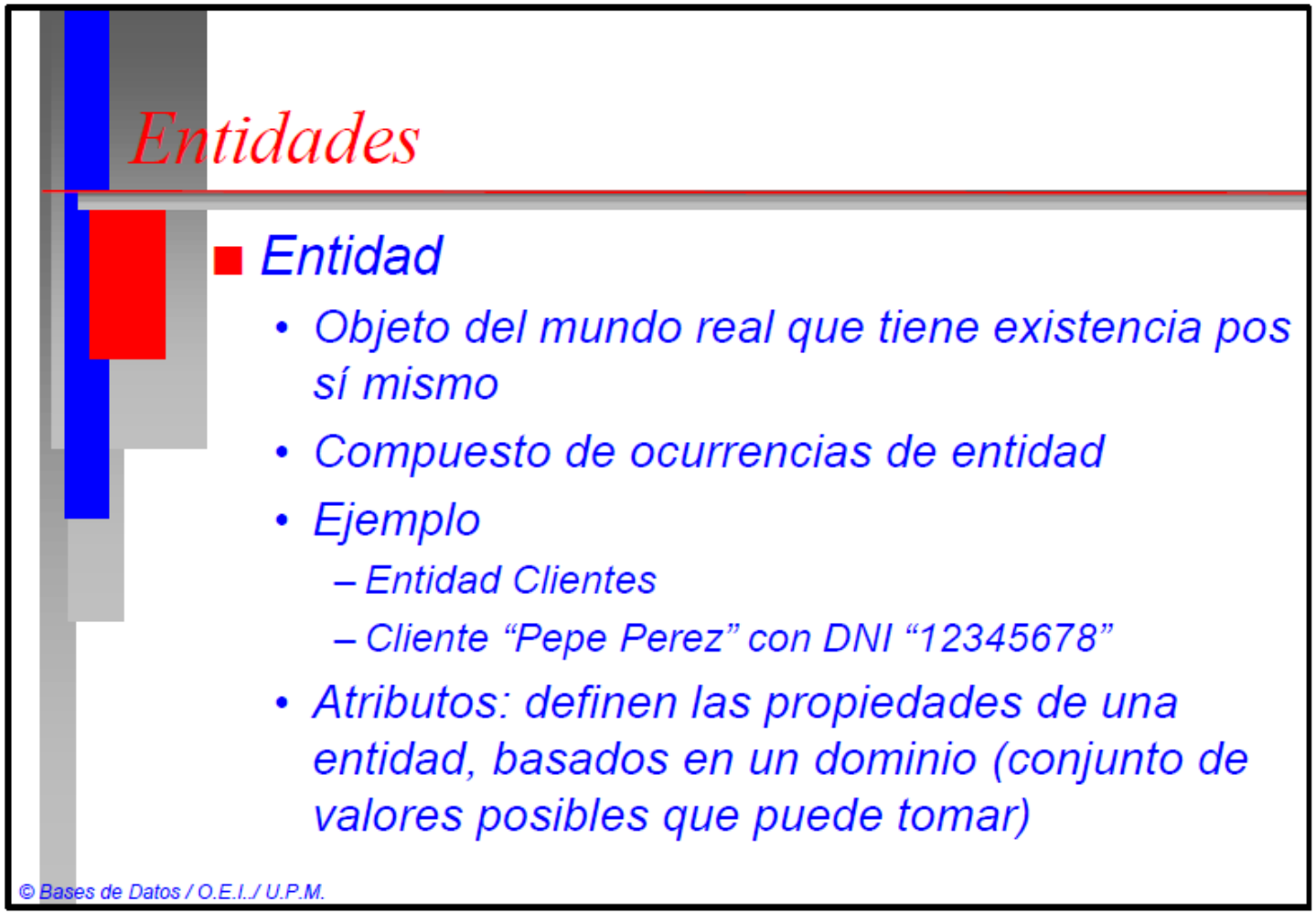
- Riordan, Rebecca M. *Designing Effective Database Systems*. Addison-Wesley Professional. 2005.

# El modelo Entidad Relación



# El modelo Entidad Relación

# El modelo Entidad Relación

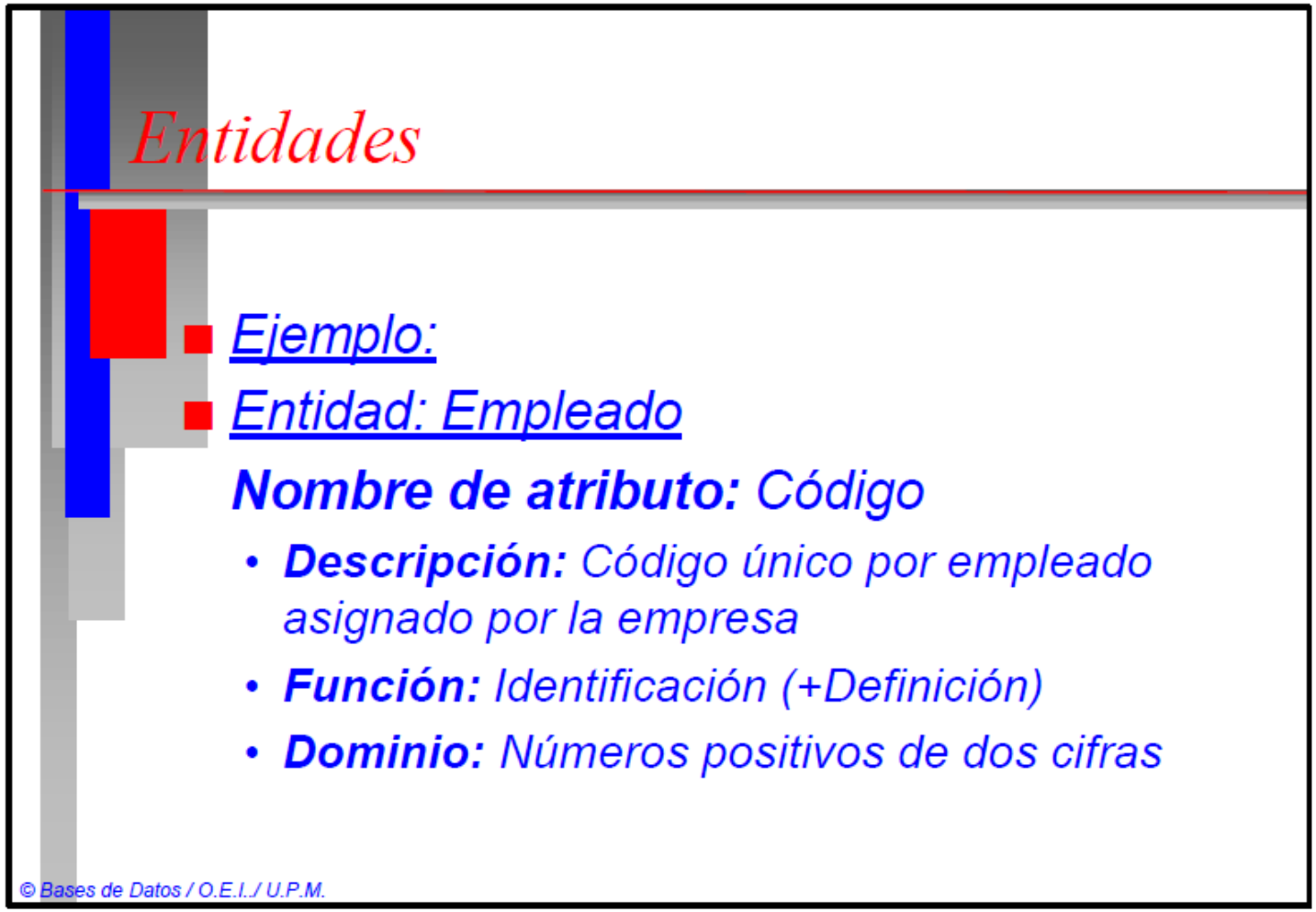


# El modelo Entidad Relación

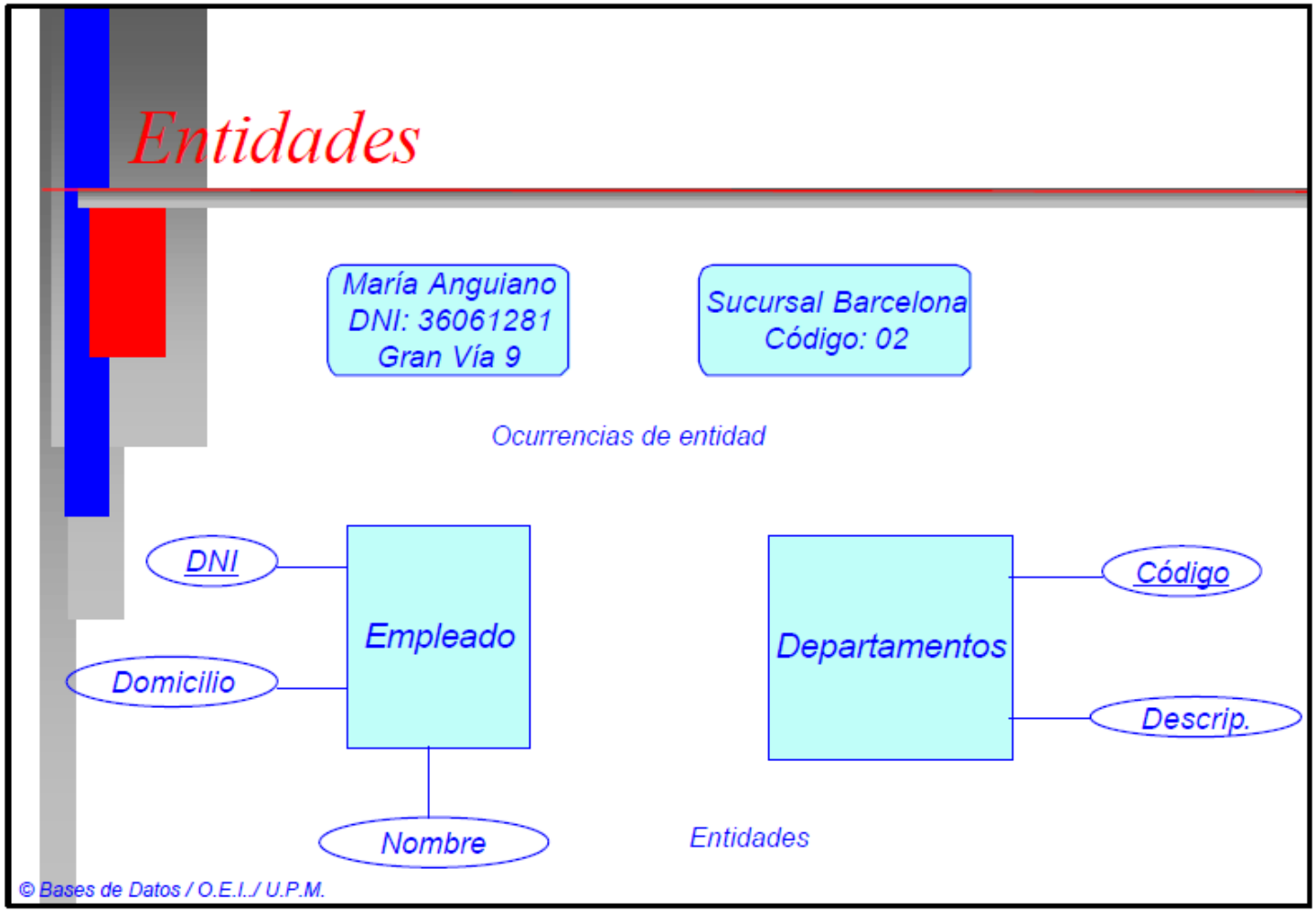
## Entidades

- *Atributo - Característica propia de una entidad, común para todas las ocurrencias del mismo tipo*
- *Dominio - Conjunto de valores permitidos para un atributo*
- *Para cada atributo hay que definir:*
  - *Nombre Descripción Dominio Función (identificación o definición)*

# El modelo Entidad Relación

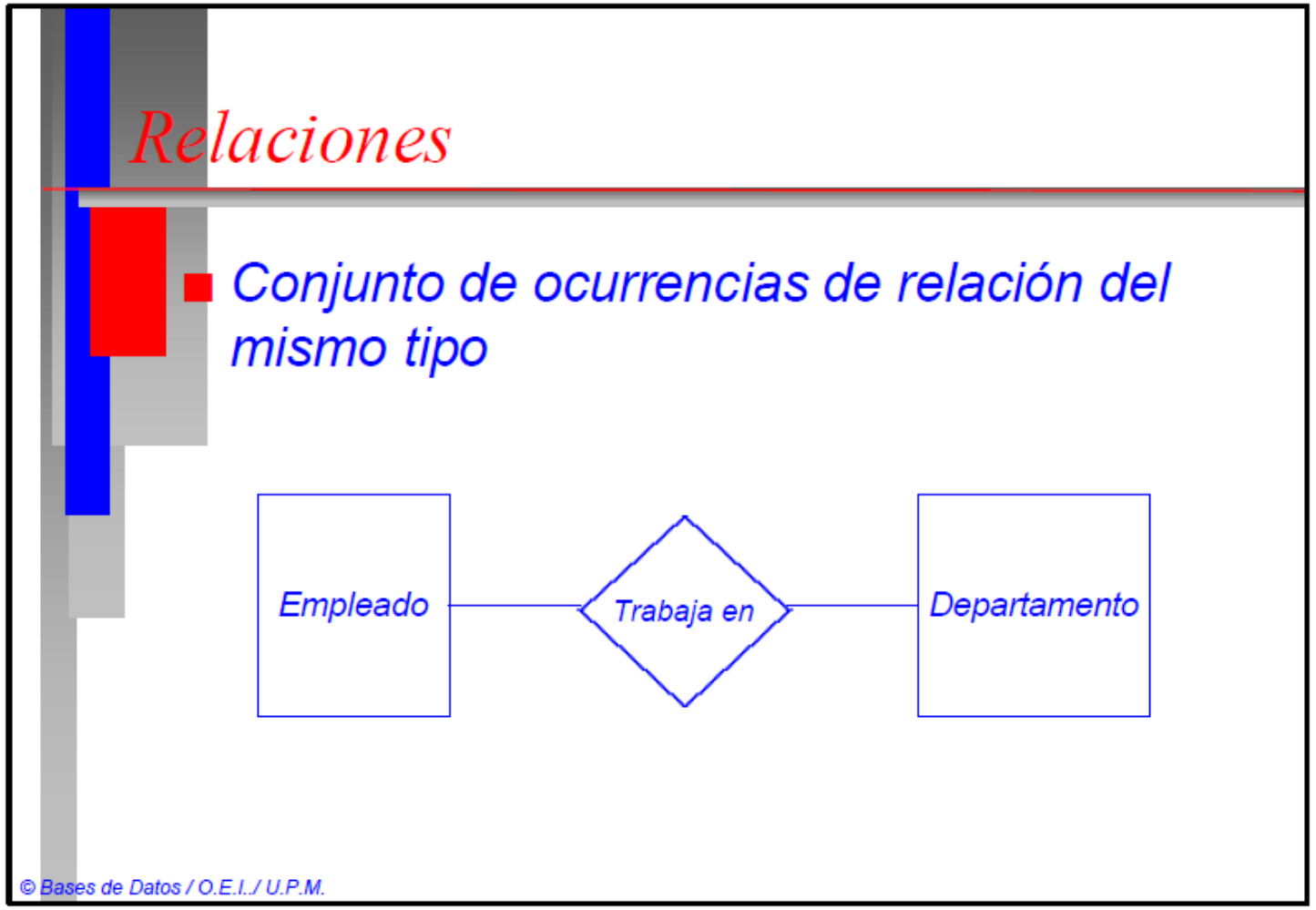


# El modelo Entidad Relación

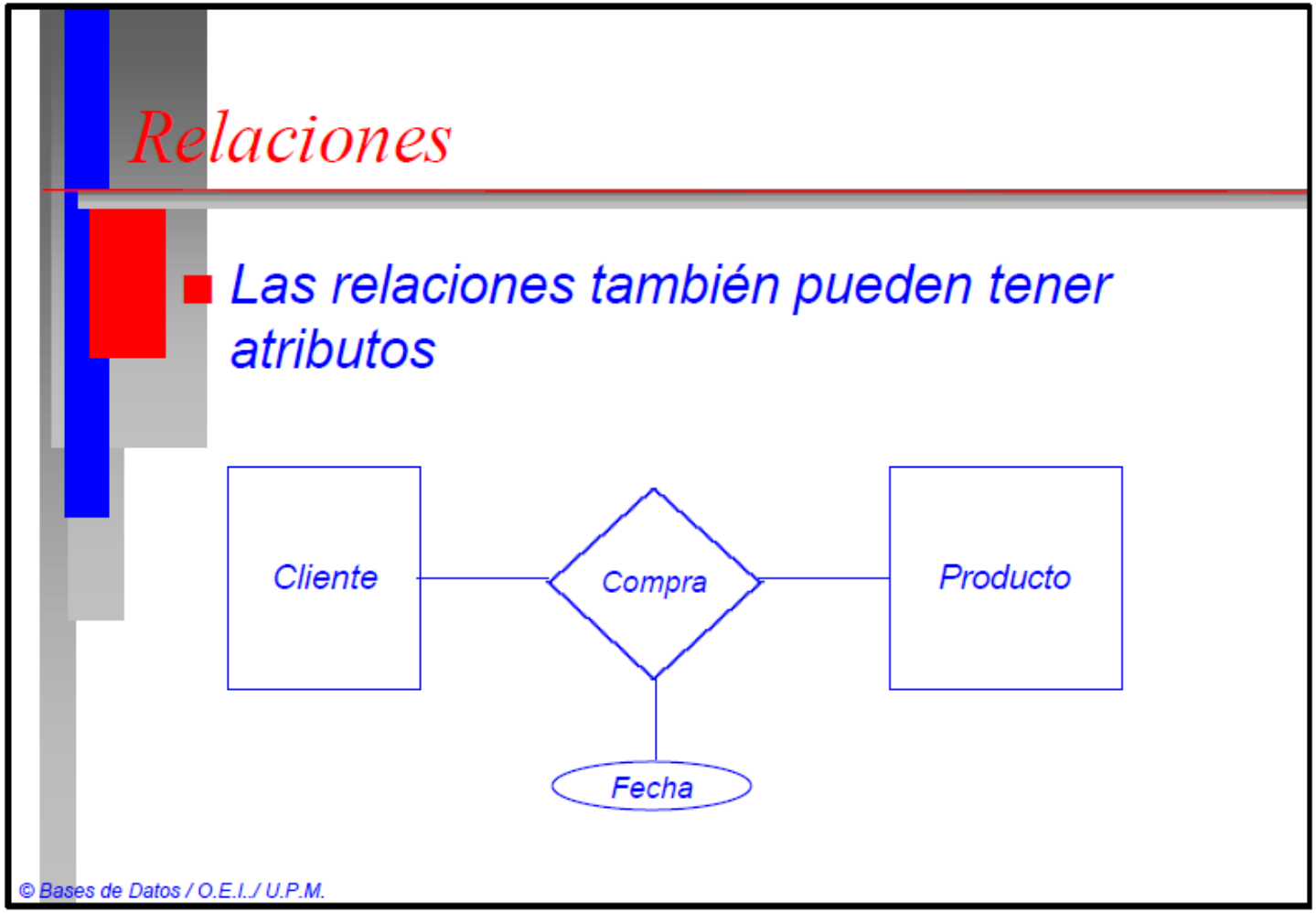


# El modelo Entidad Relación

# El modelo Entidad Relación



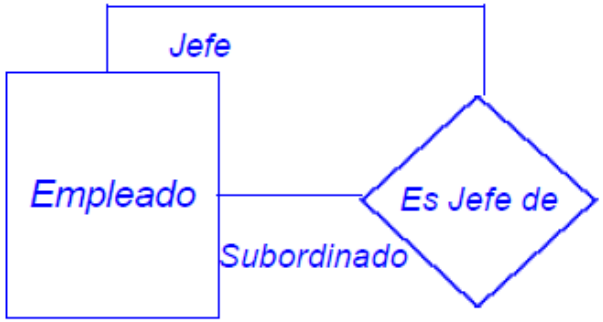
# El modelo Entidad Relación



# El modelo Entidad Relación

## Relaciones

- Es importante el "rol" o "papel" de cada ocurrencia

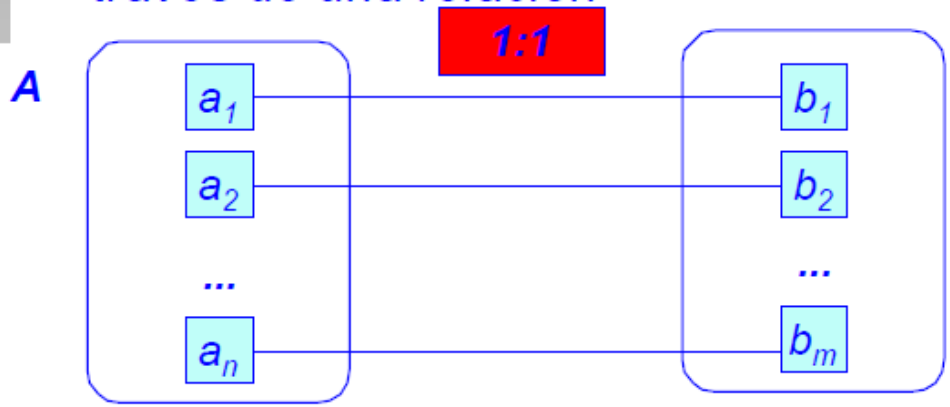



- Se denomina **grado** de una relación al número de entidades que relaciona

# El modelo Entidad Relación

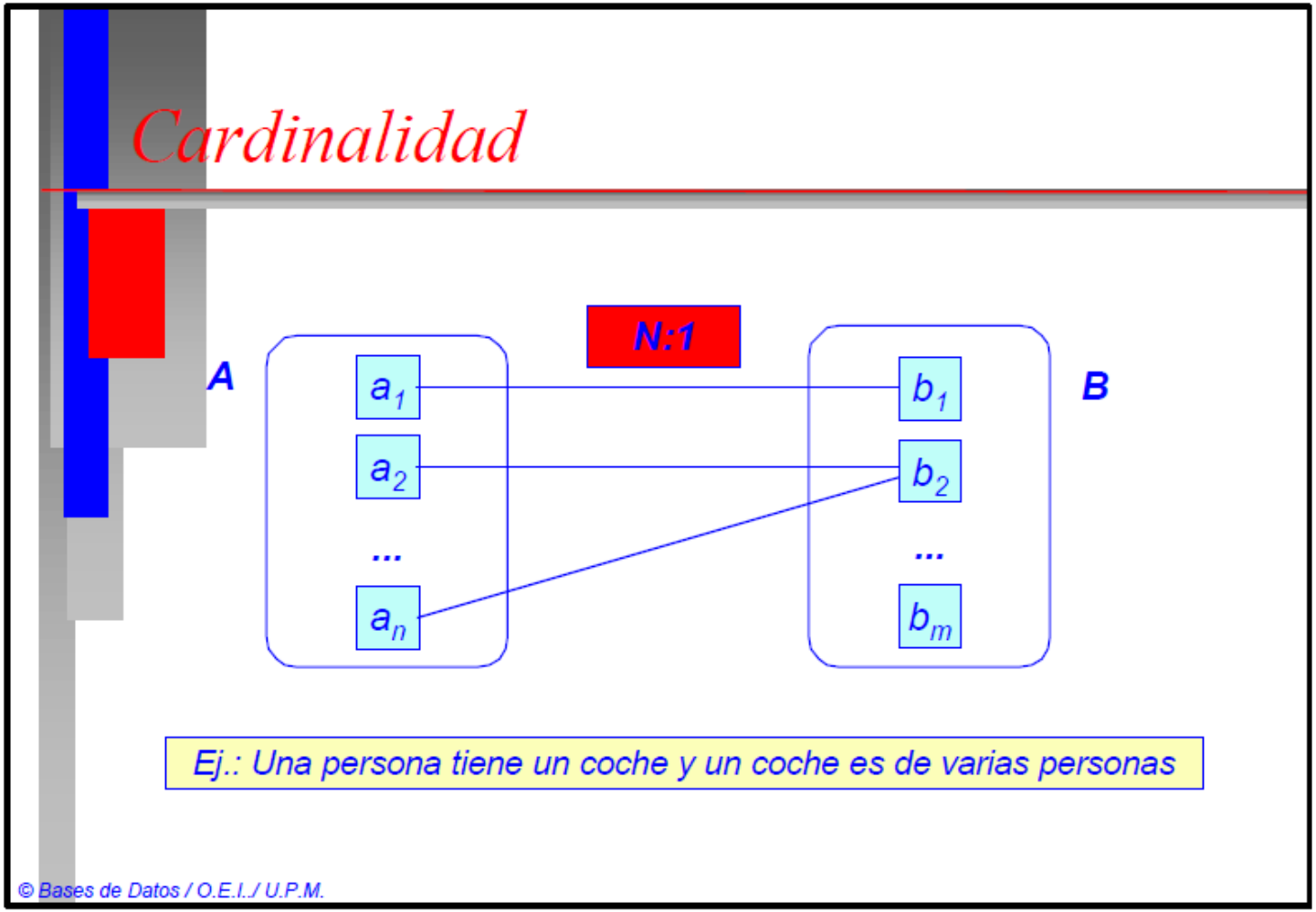
## Cardinalidad Máxima

- *Número de ocurrencias de entidad que se pueden asociar como máximo a otra a través de una relación*

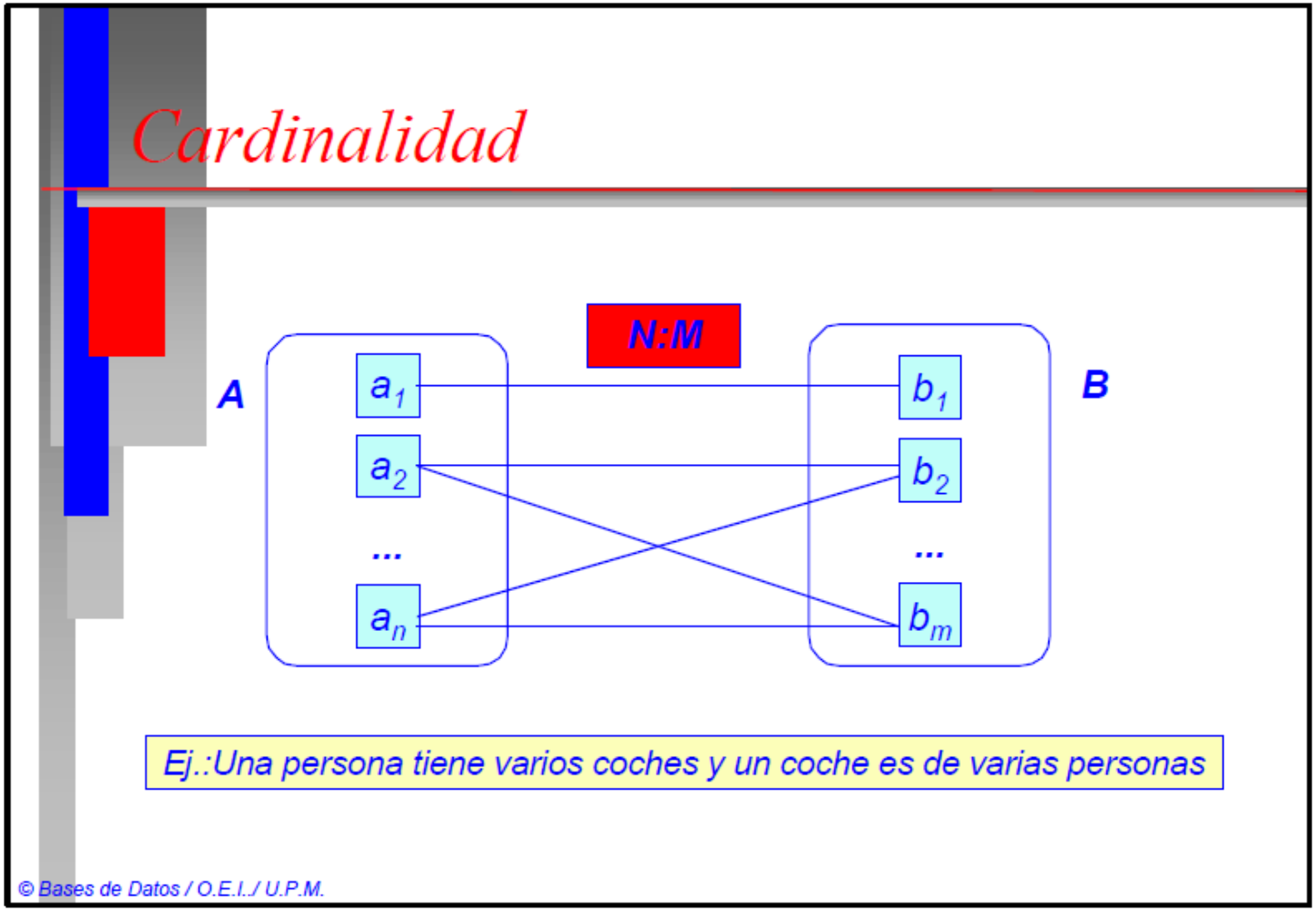



*Ej.:Una persona tiene un coche y un coche es de una sola persona*

# El modelo Entidad Relación



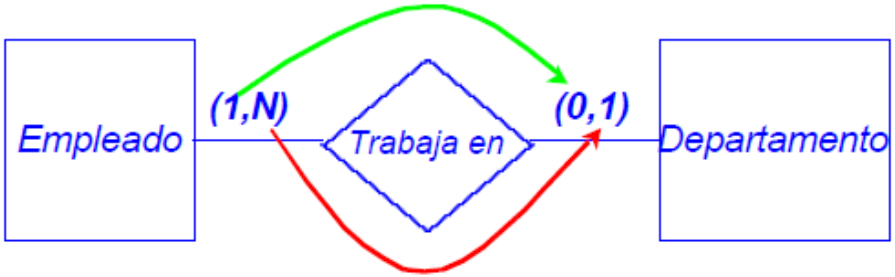
# El modelo Entidad Relación



# El modelo Entidad Relación

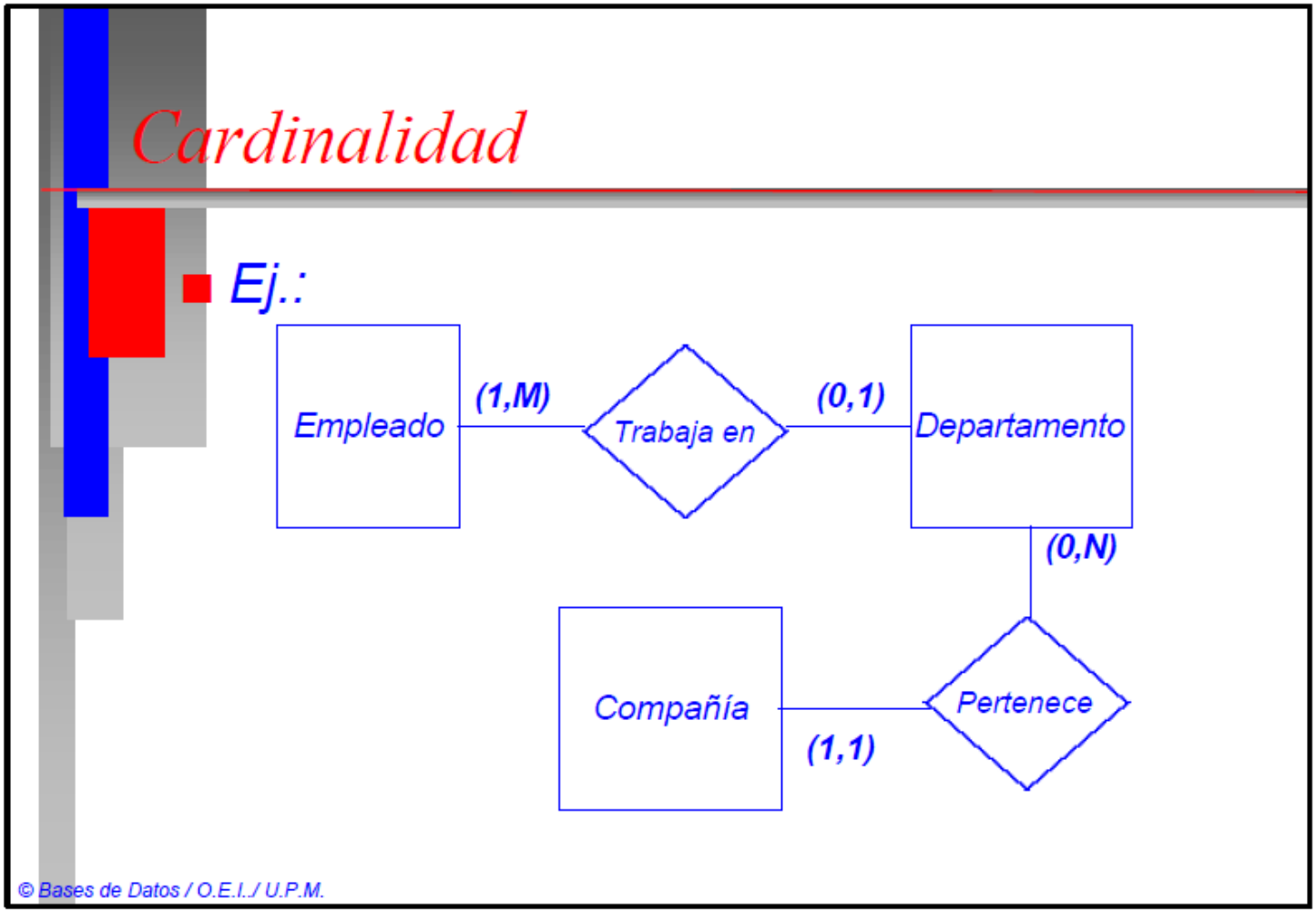
## Cardinalidad Mínima

- Número mínimo de ocurrencias de entidad que se deben asociar a otra a través de una relación
- Posibilidades: 0:0, 0:1, 1:0, 1:1

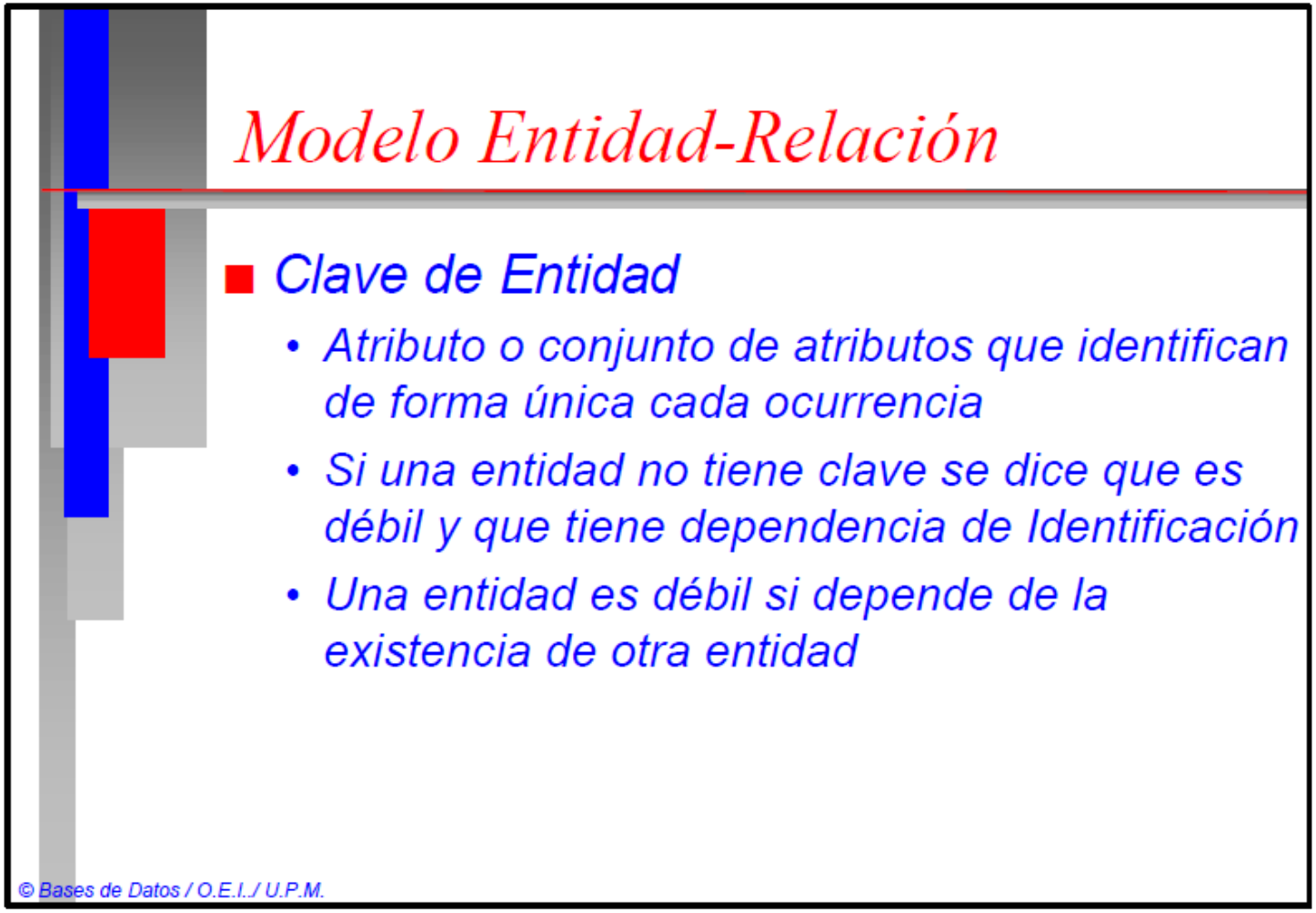



Nota: Hay que tener especial cuidado con las mínimas 1:1

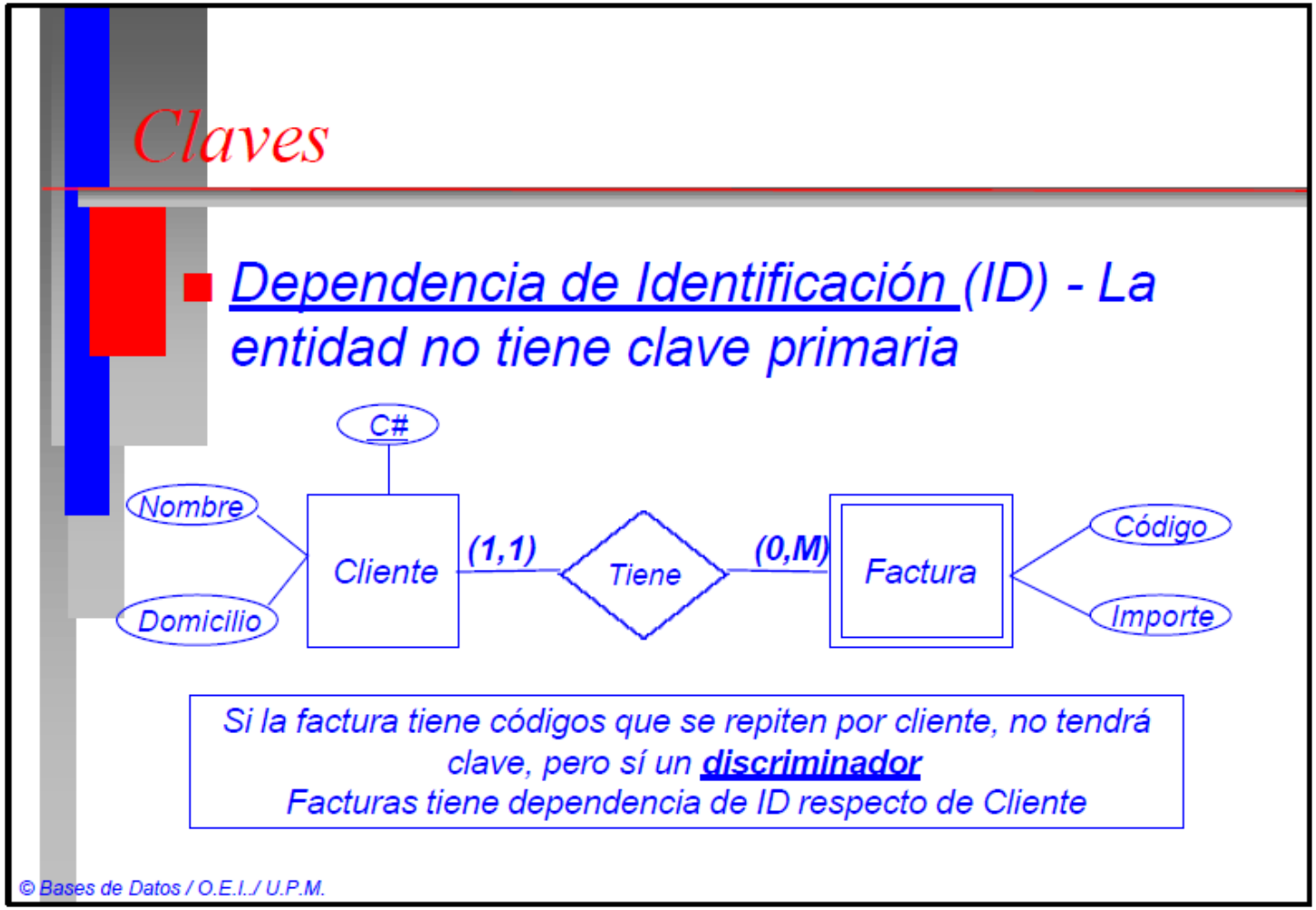
# El modelo Entidad Relación



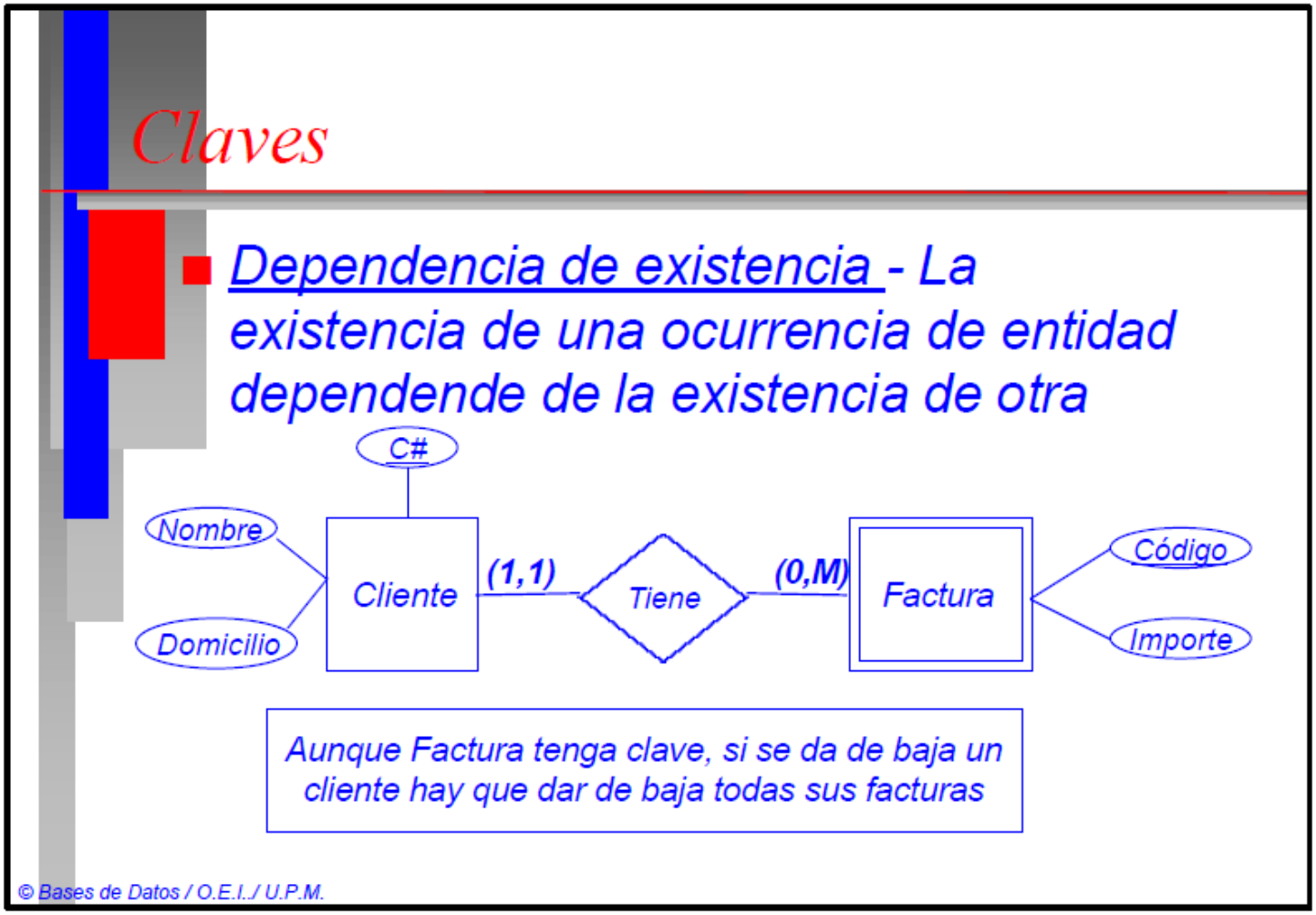
# El modelo Entidad Relación



# El modelo Entidad Relación

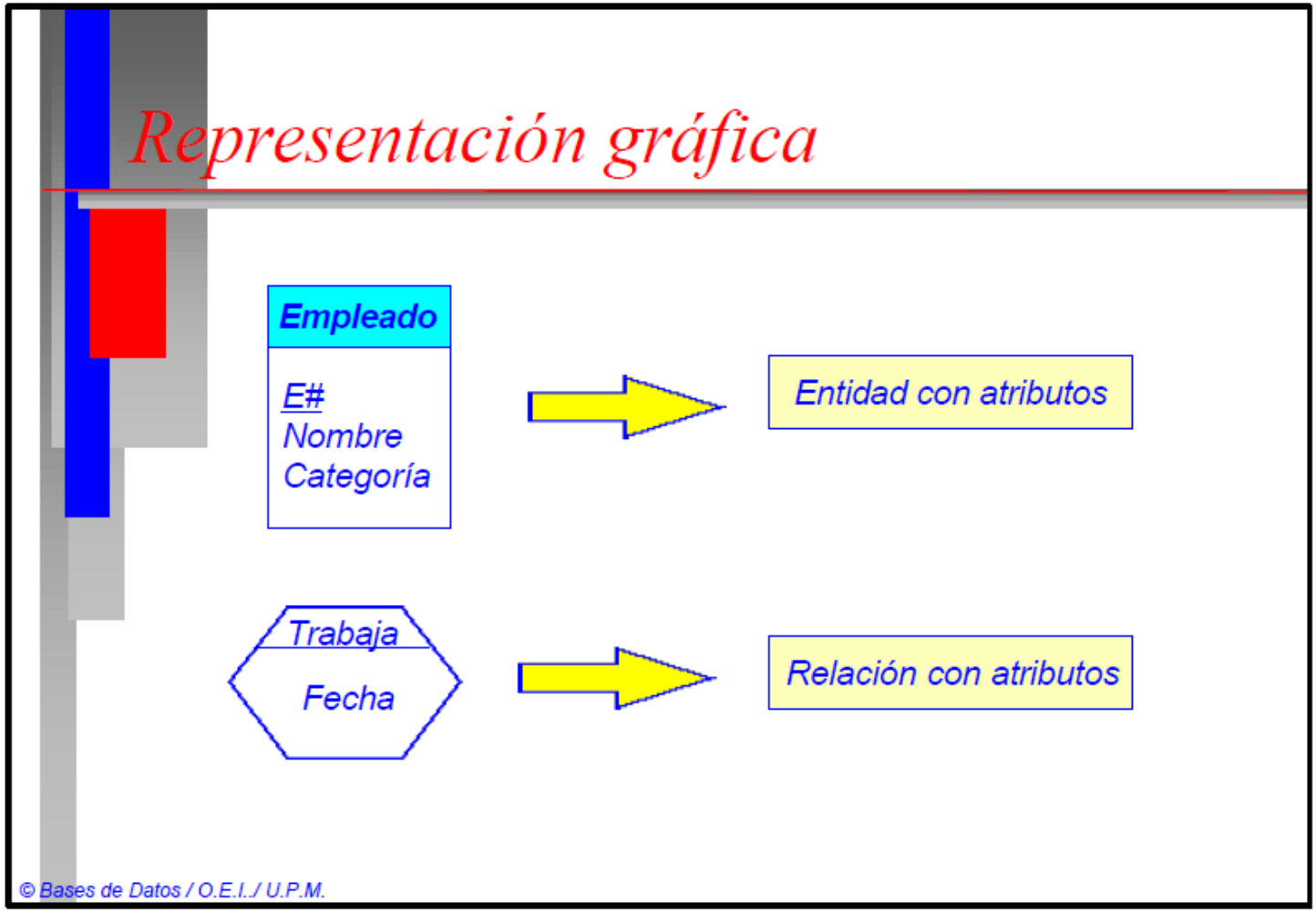


# El modelo Entidad Relación

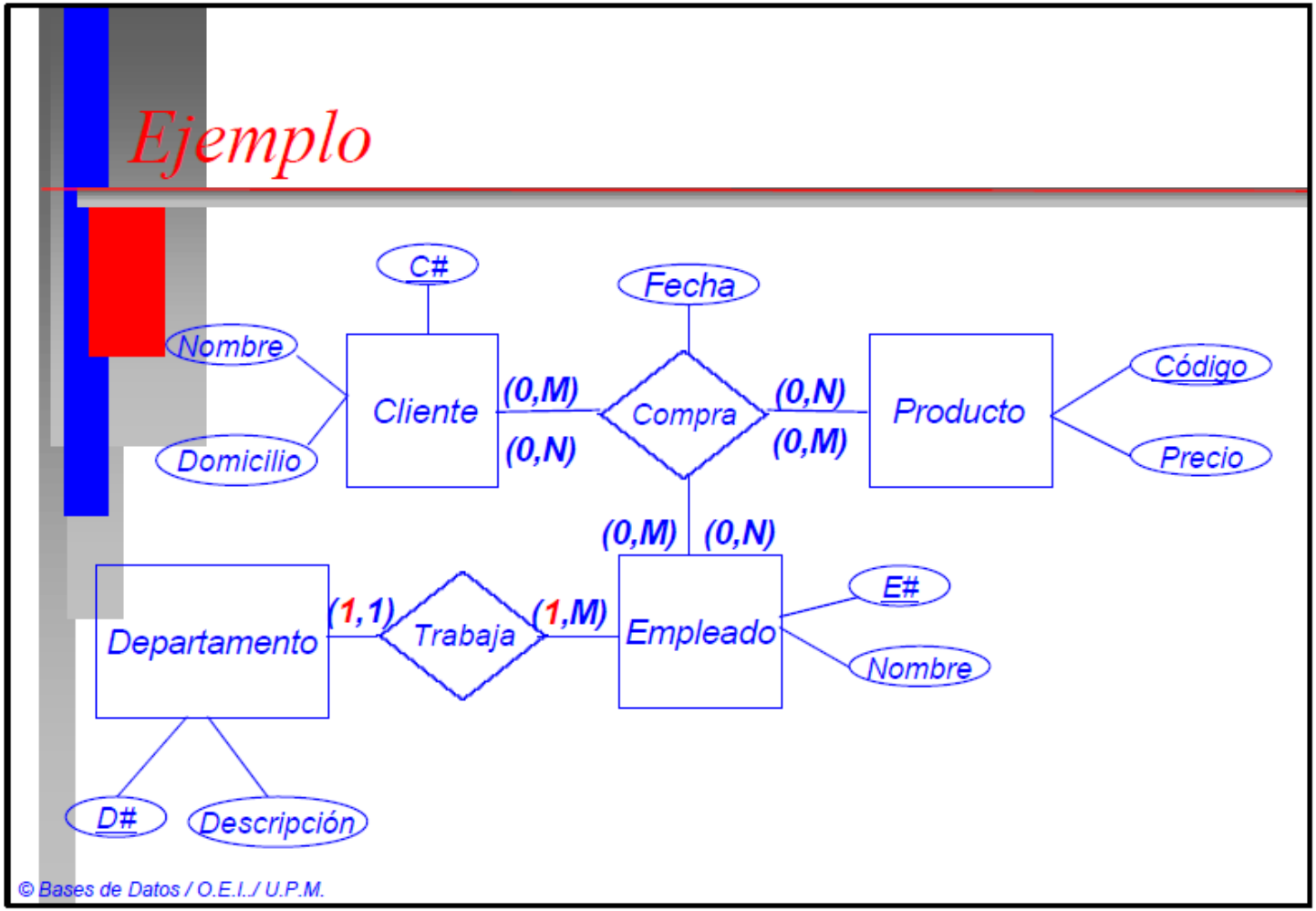


# El modelo Entidad Relación

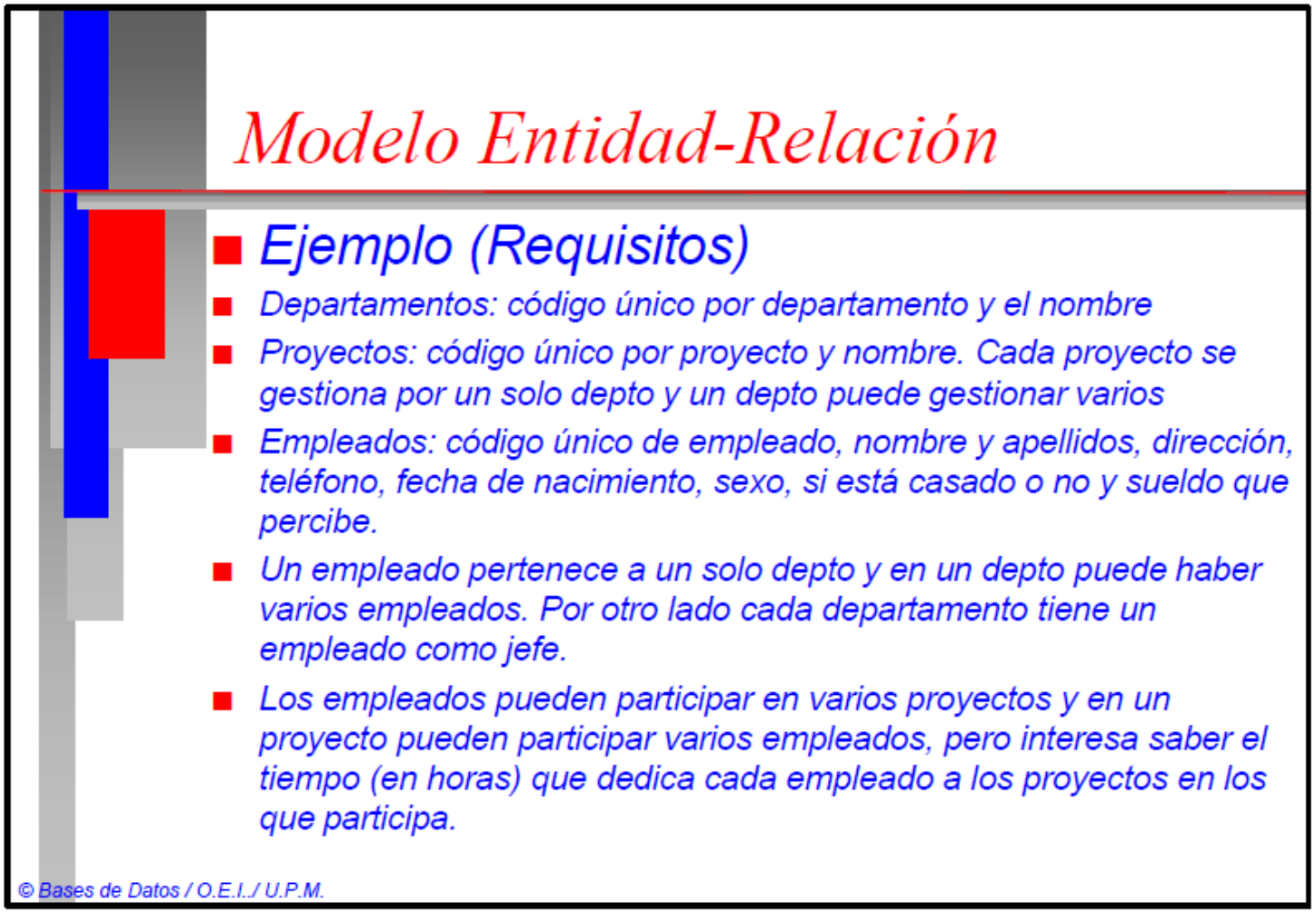
# El modelo Entidad Relación



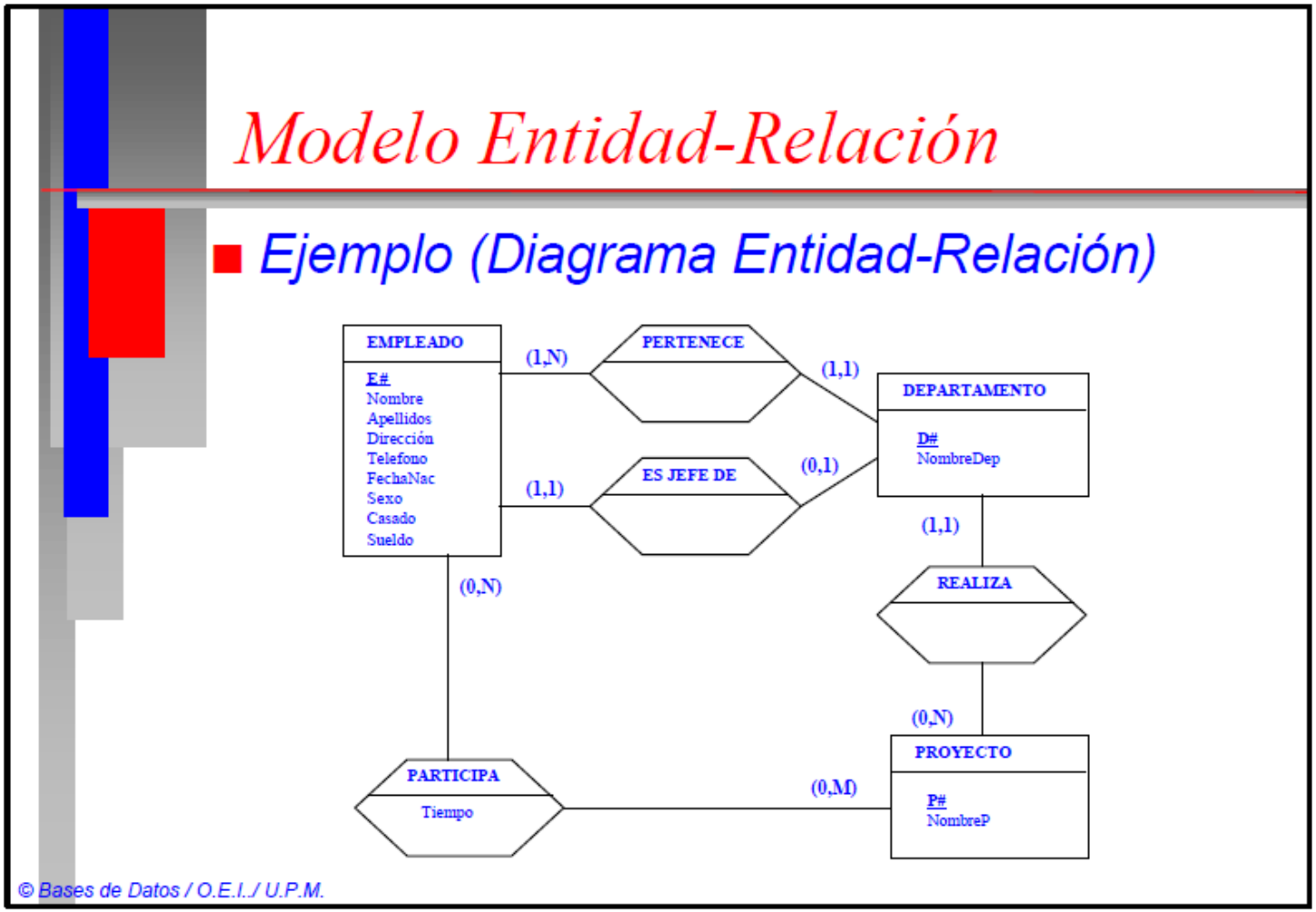
# El modelo Entidad Relación



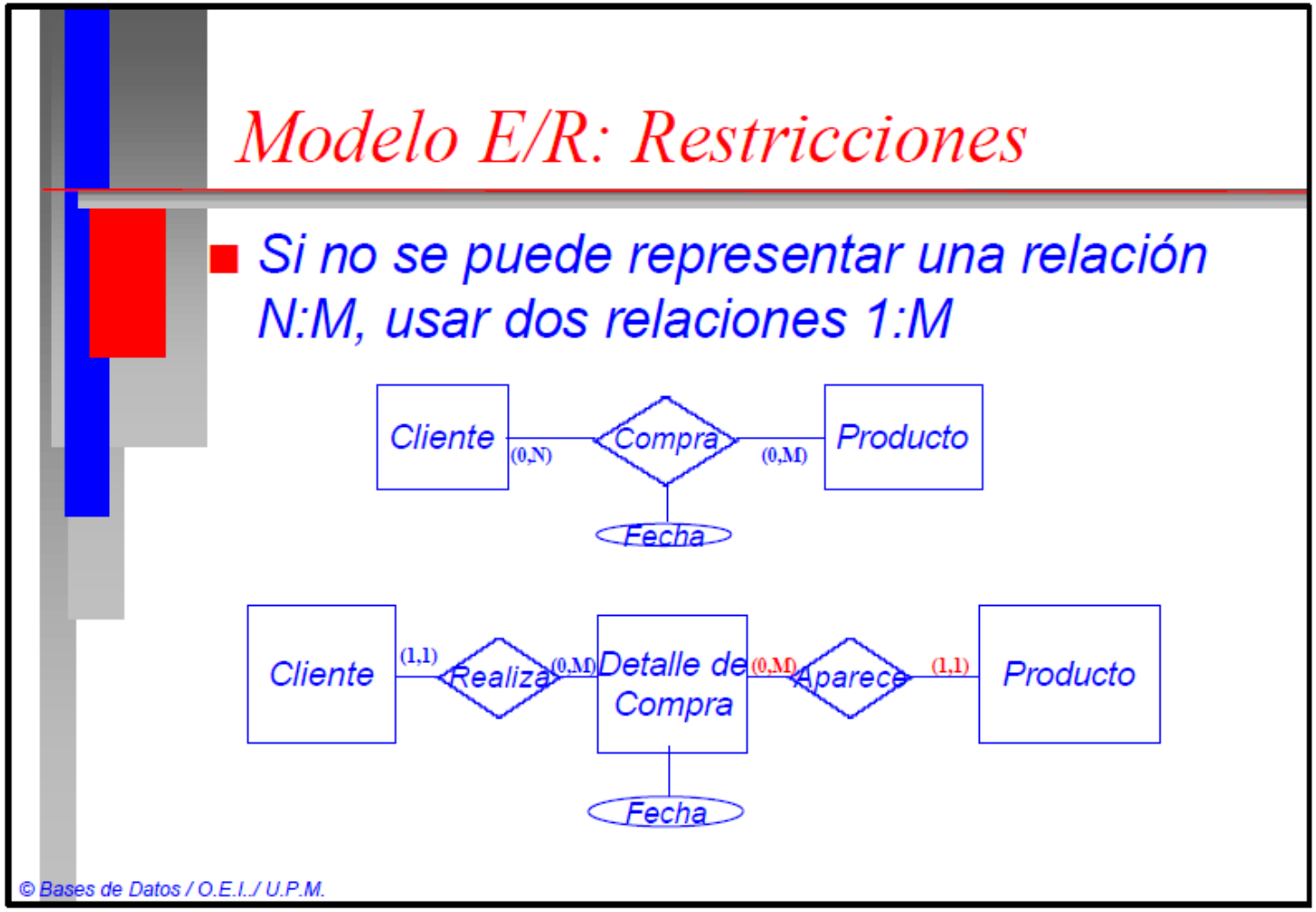
# El modelo Entidad Relación



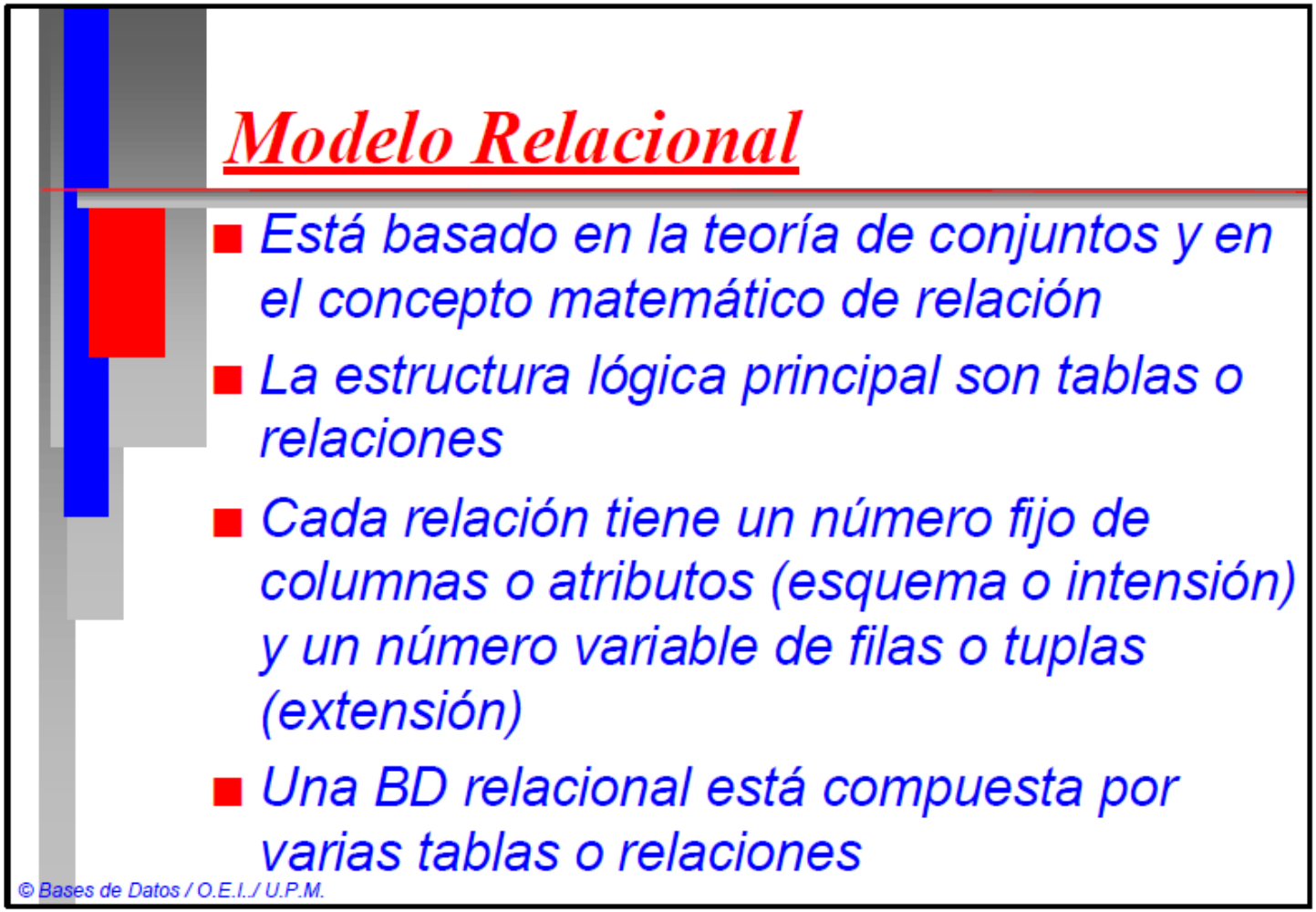
# El modelo Entidad Relación



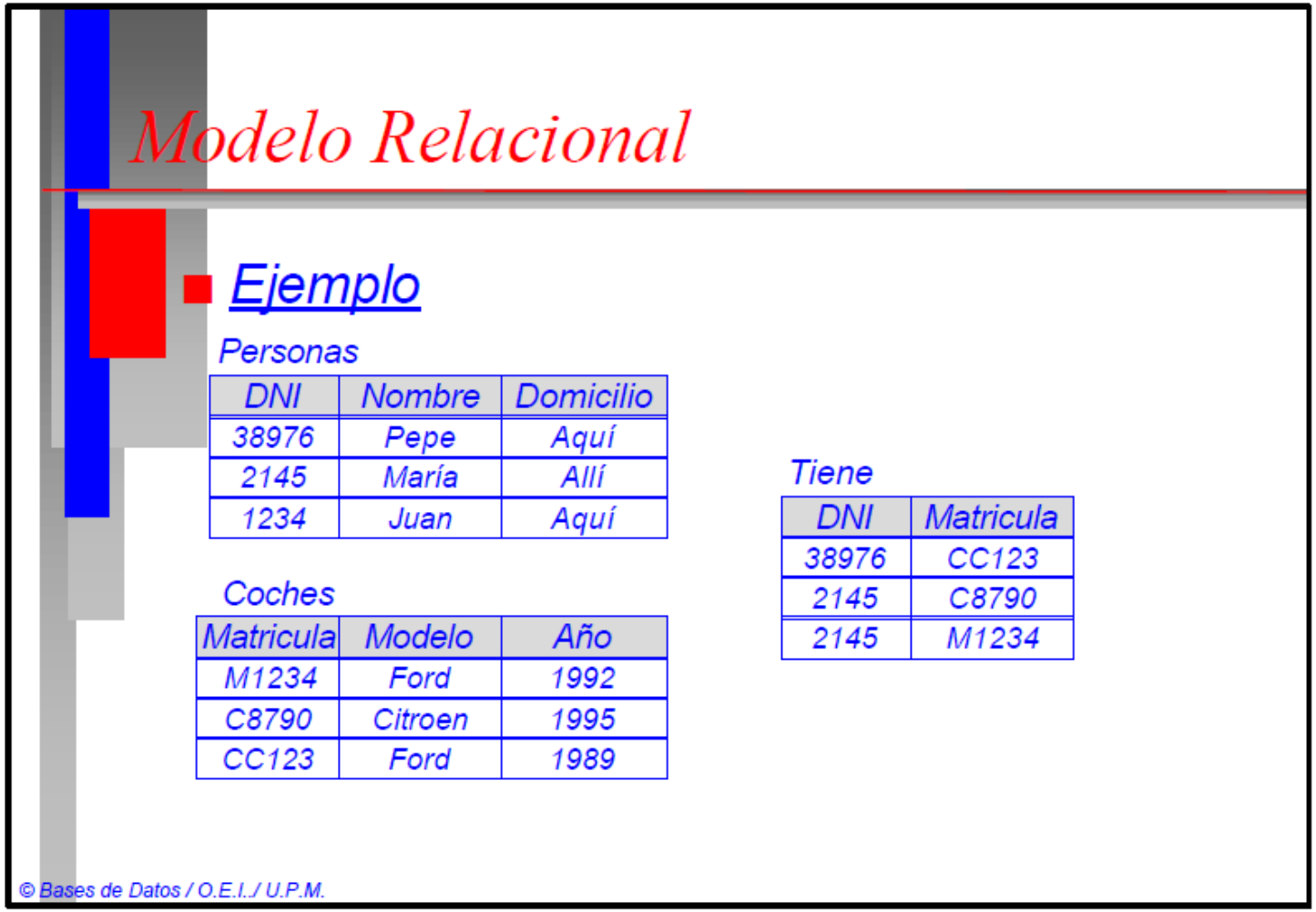
# El modelo Entidad Relación



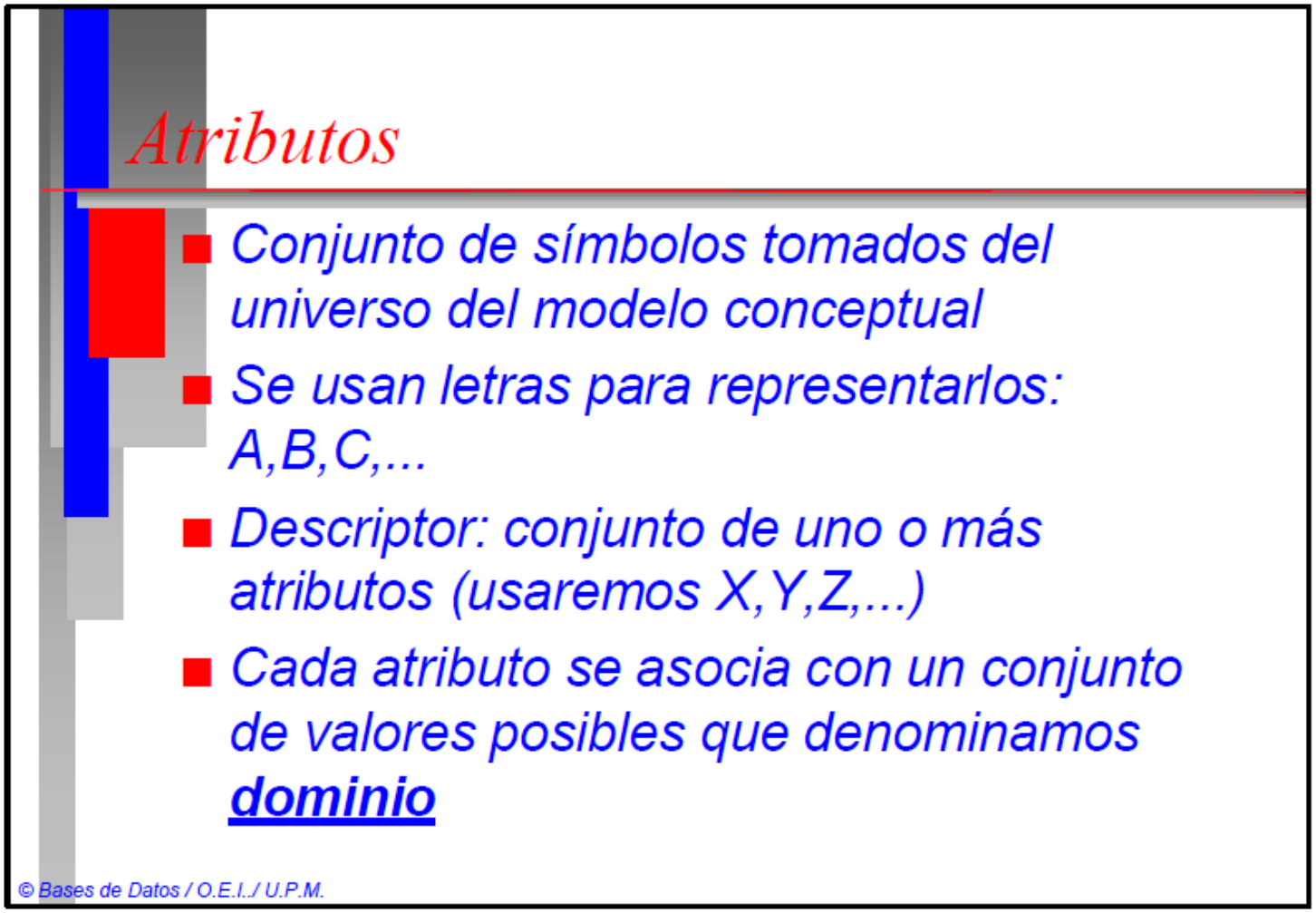
# El modelo Entidad Relación



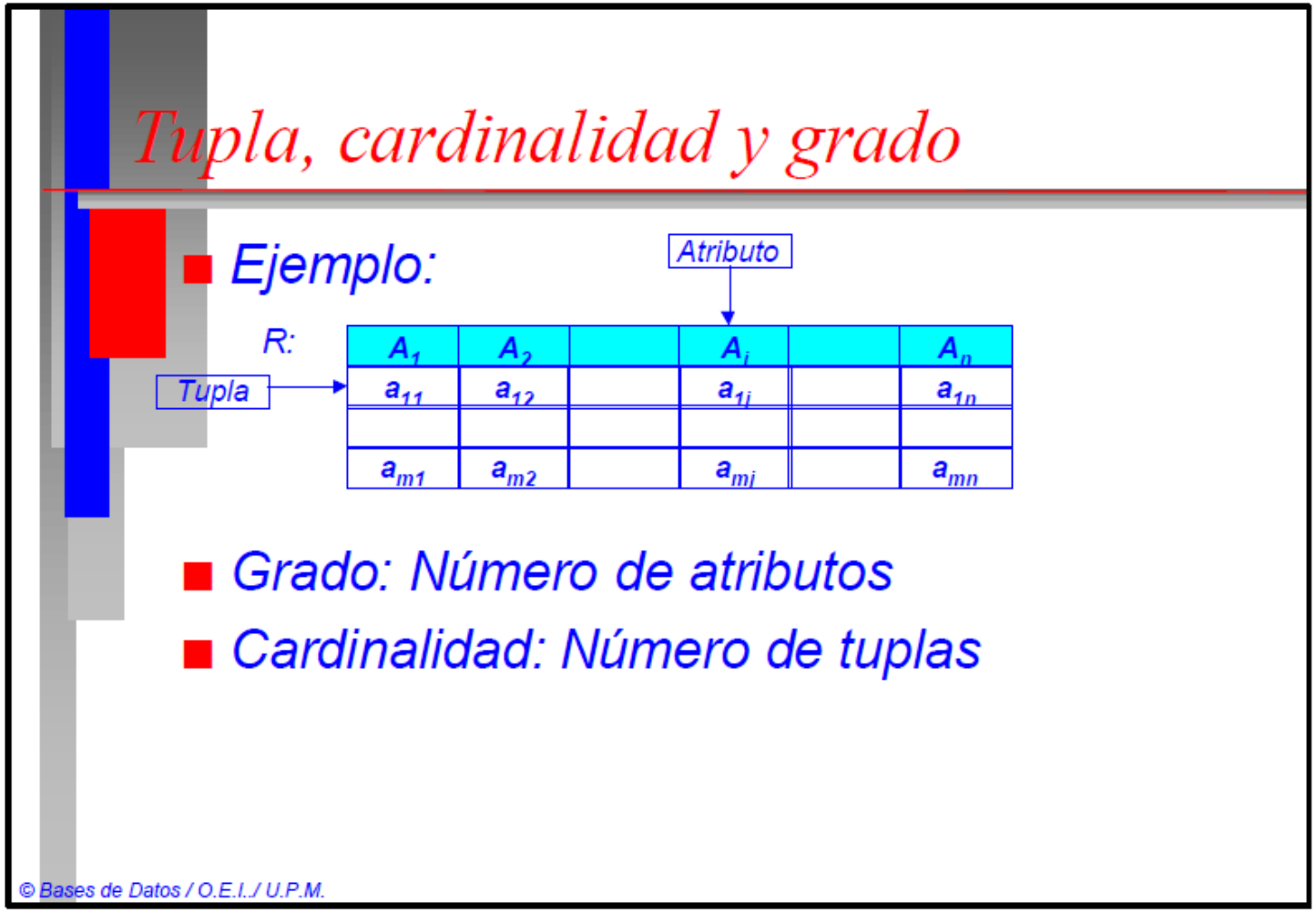
# El modelo Entidad Relación



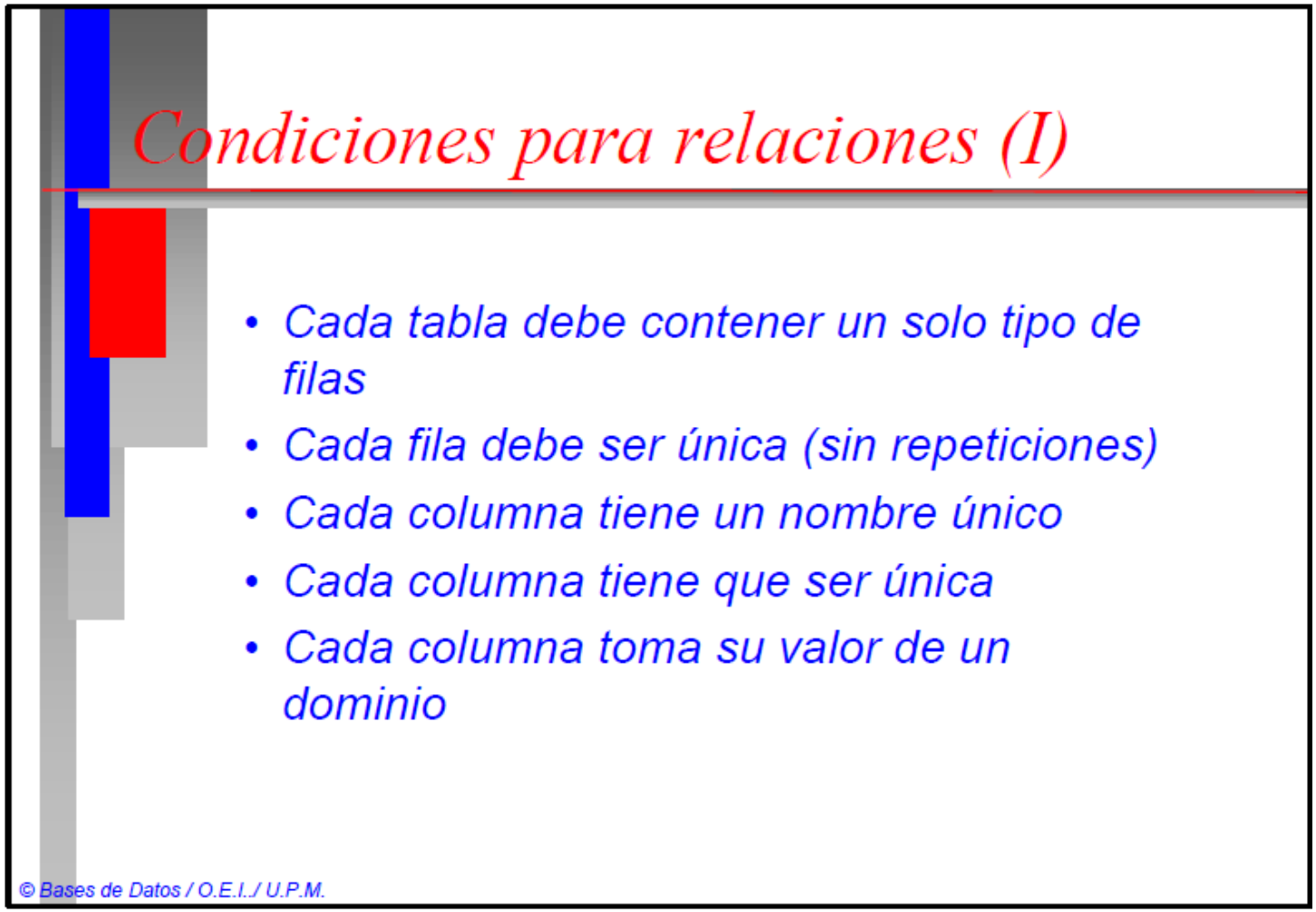
# El modelo Entidad Relación



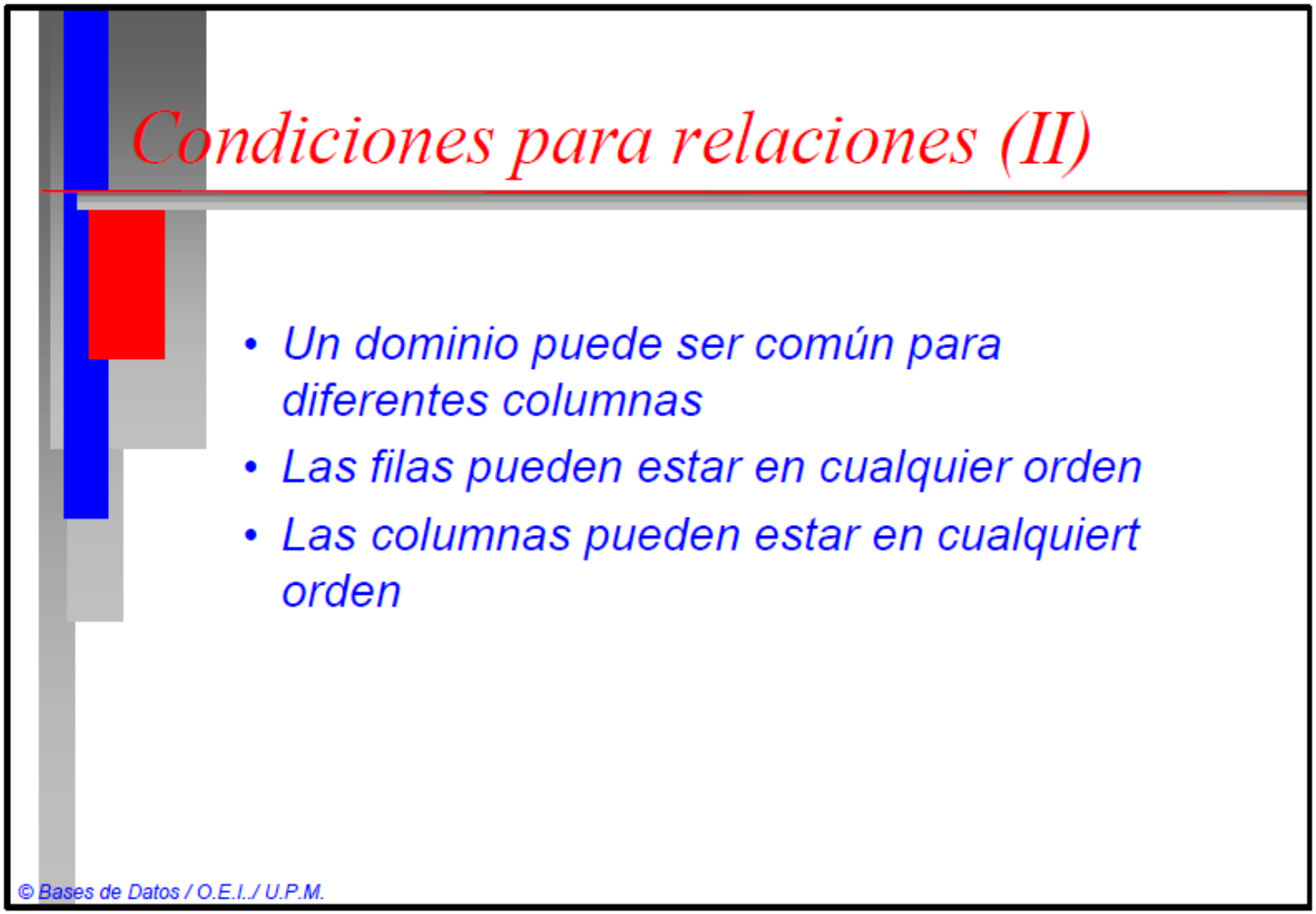
# El modelo Entidad Relación



# El modelo Entidad Relación



# El modelo Entidad Relación



# El modelo Entidad Relación

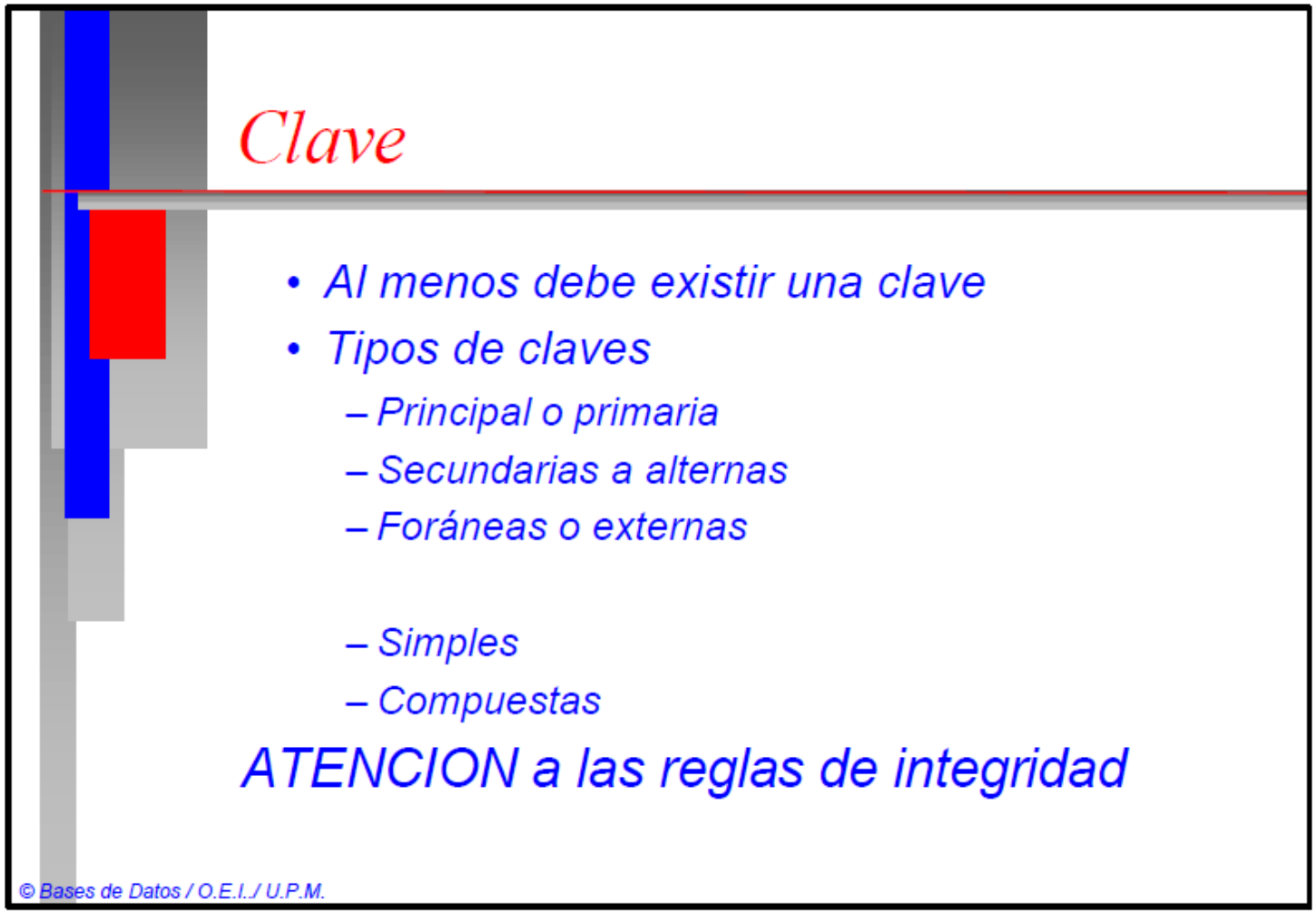
## Clave

- Cada relación tendrá una combinación de atributos que, tomados en conjunto, identifican de forma única cada tupla.

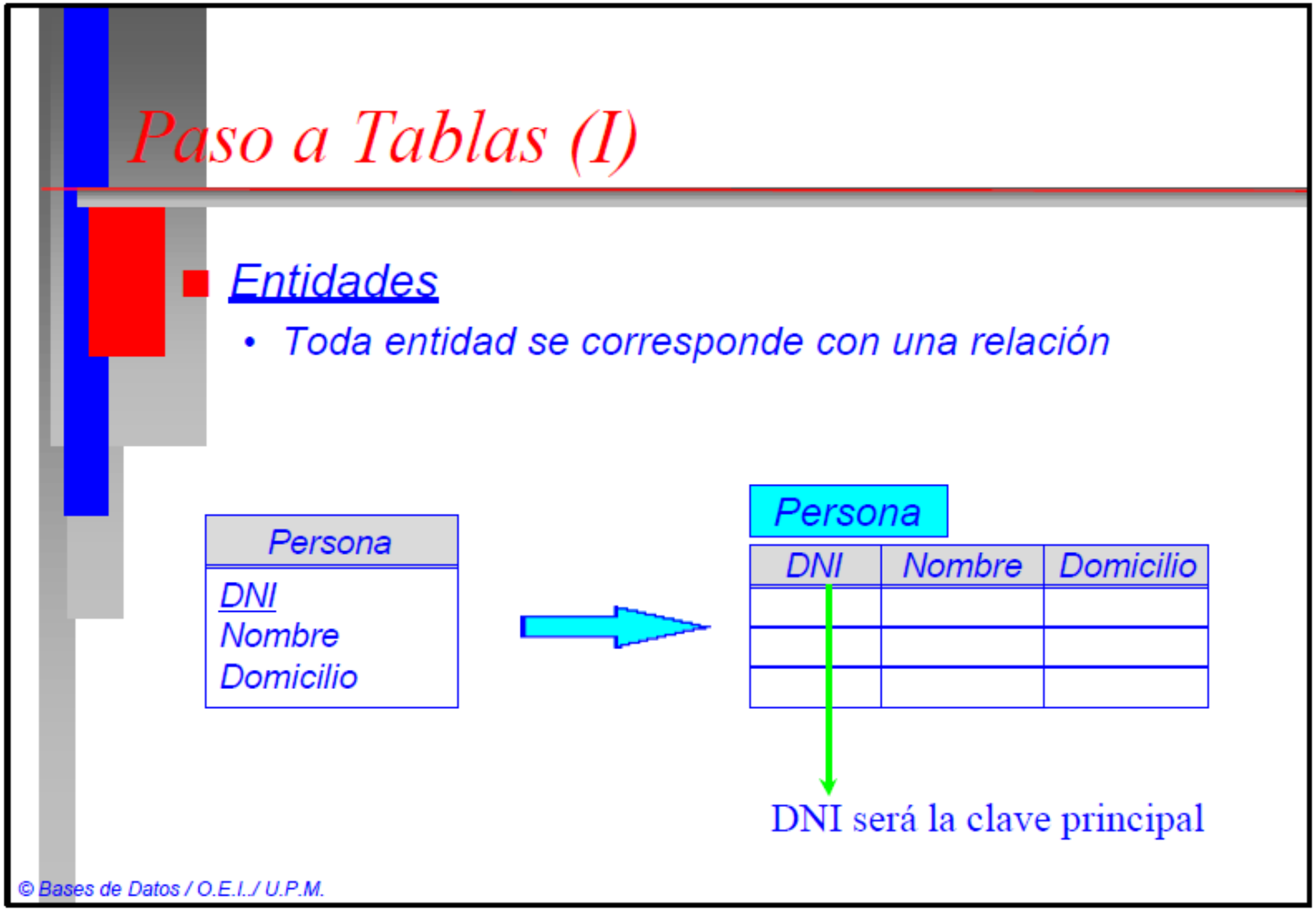
| DNI | Nombre | Domicilio | Teléfono |
|---|---|---|---|
| 321 | Pepe | Aquí | 987 |
| 134 | Pepe | Allí | 789 |
| 123 | Juan | Allí | 789 |

- Si tiene más de una, se elige la "principal" y las demás serán "alternas"

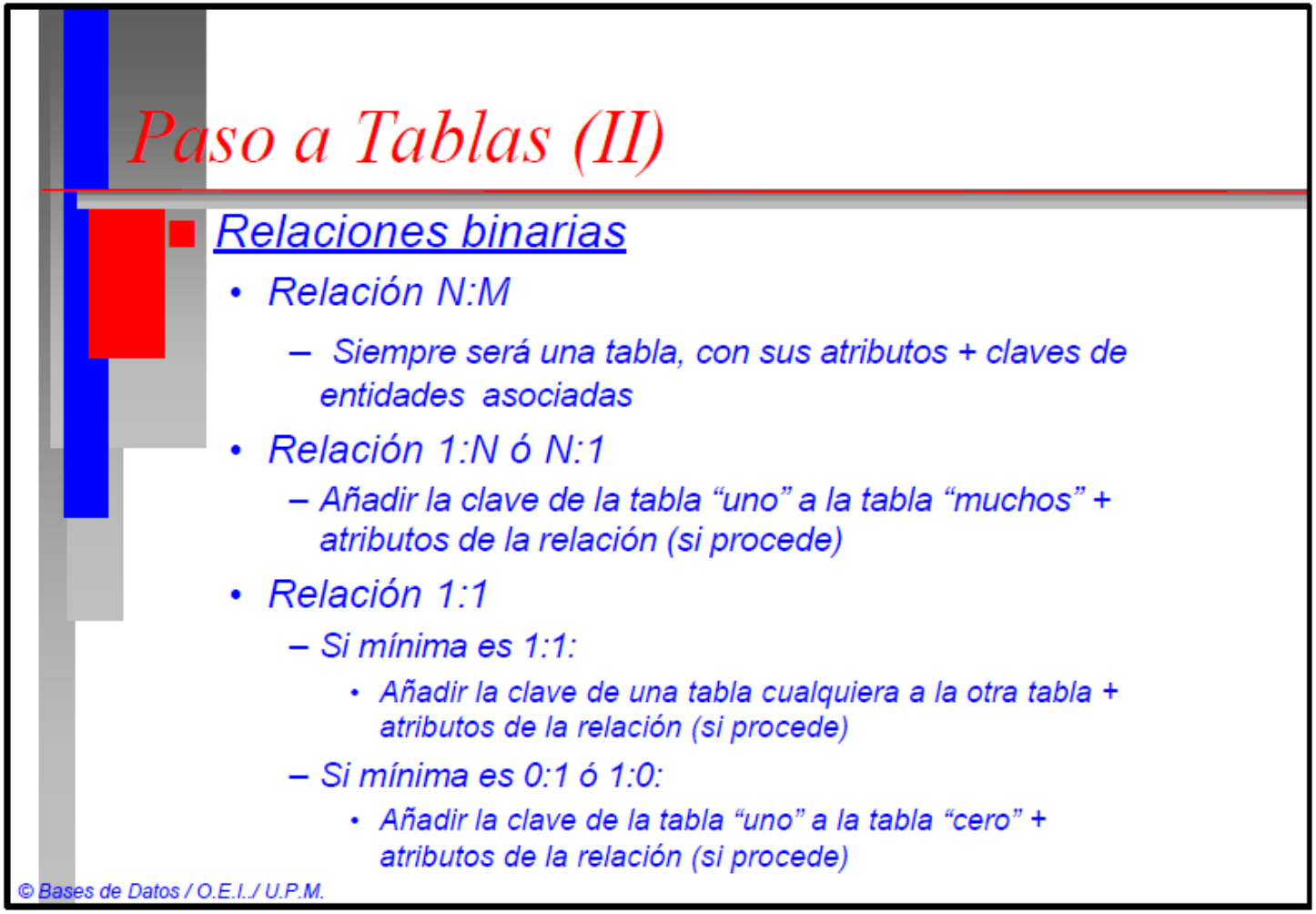
# El modelo Entidad Relación



# El modelo Entidad Relación



# El modelo Entidad Relación

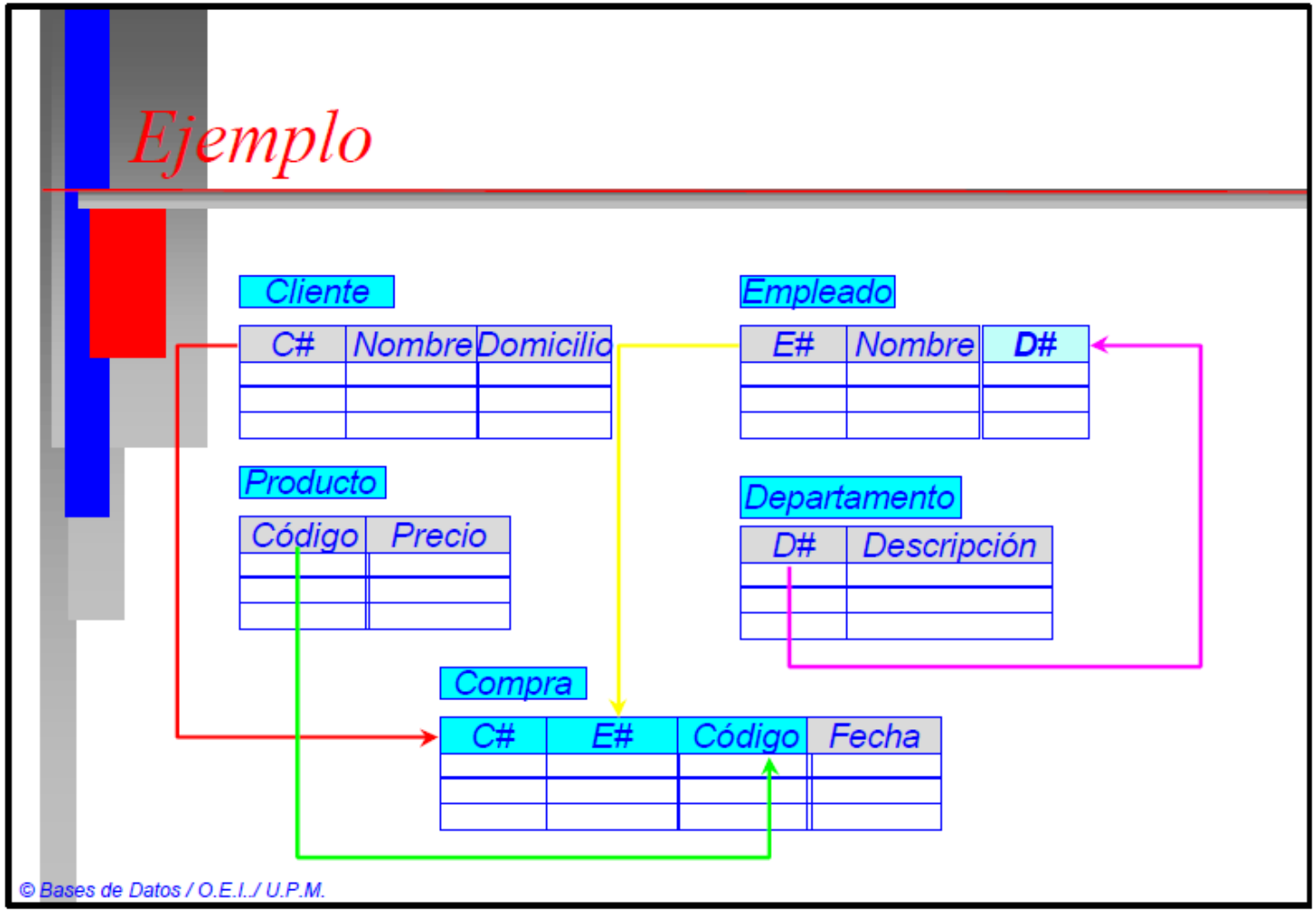


# El modelo Entidad Relación

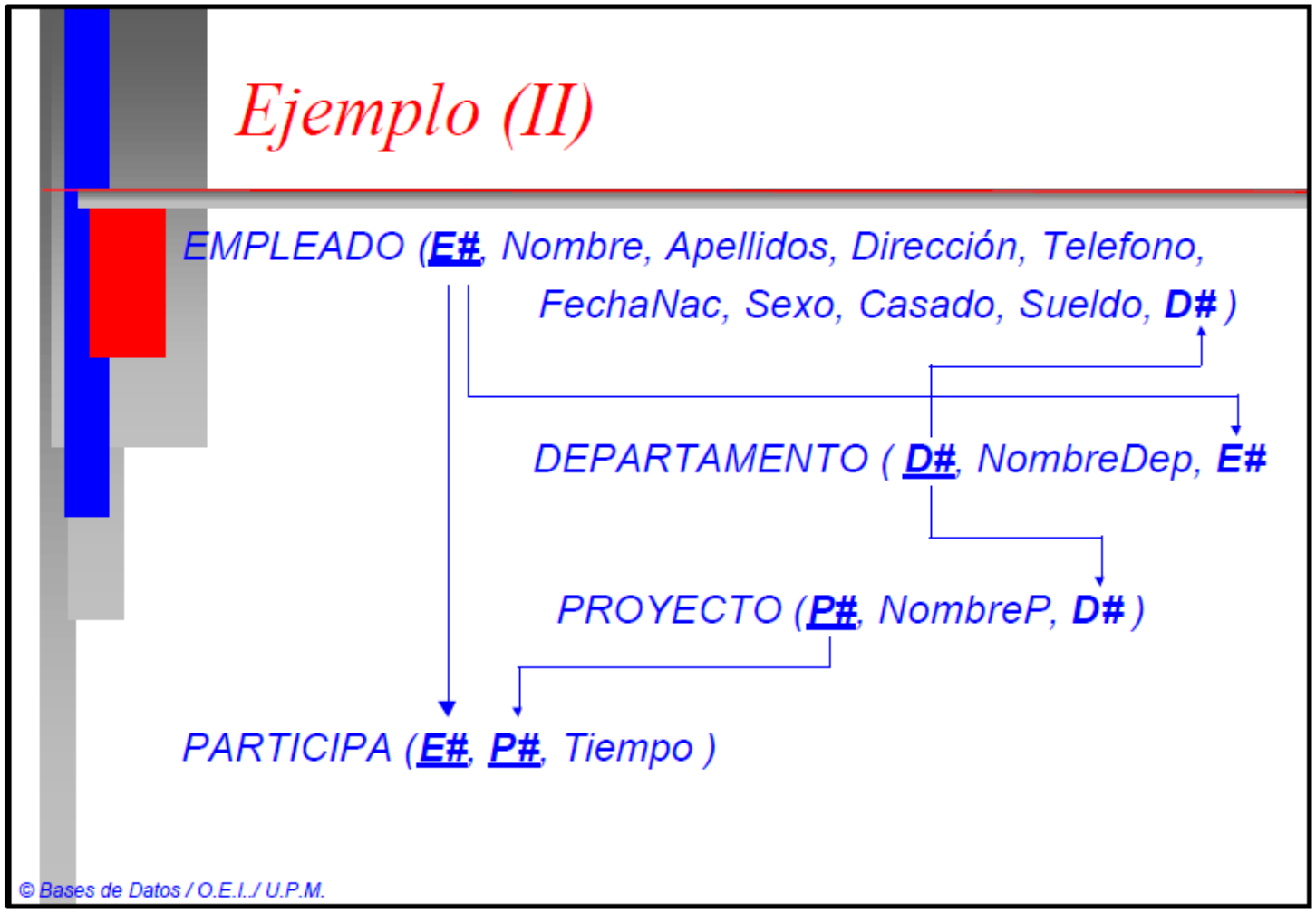
## Paso a Tablas (III)

- Relaciones ternarias y n-arias
  - Estudiar las relaciones de dos en dos y aplicar las reglas de relaciones binarias
  - Atención: se puede mejorar el diseño estudiando redundancias

# El modelo Entidad Relación



# El modelo Entidad Relación



# BASES DE DATOS

## Structured Query Language (SQL)

***Dr. Eugenio Santos Menéndez***

*Departamento de O.E.I.*
*Escuela Universitaria de Informática*
*Universidad Politécnica de Madrid*

# ÍNDICE TEMÁTICO

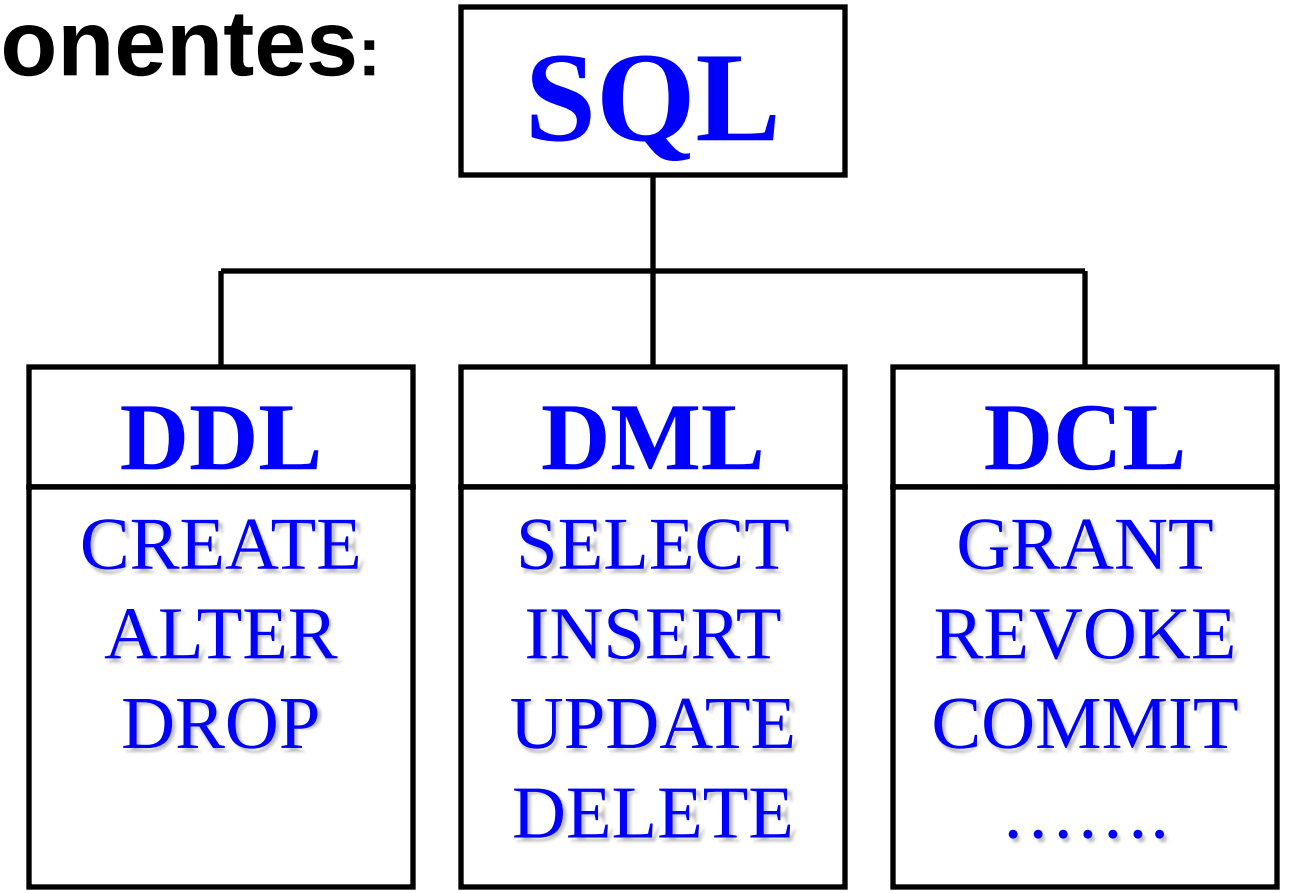
# 1. Introducción

# Introducción

- **Structured Query Language.**
- **Funcionalidad:**
  - Definir los diferentes objetos de un SGBDR.
  - Manipular los datos almacenados en la BDR.
  - Especificar restricciones de seguridad.
- **Lenguaje de Alto Nivel para dialogar con el SGBDR.**

# Introducción

- **Componentes:**



# Introducción

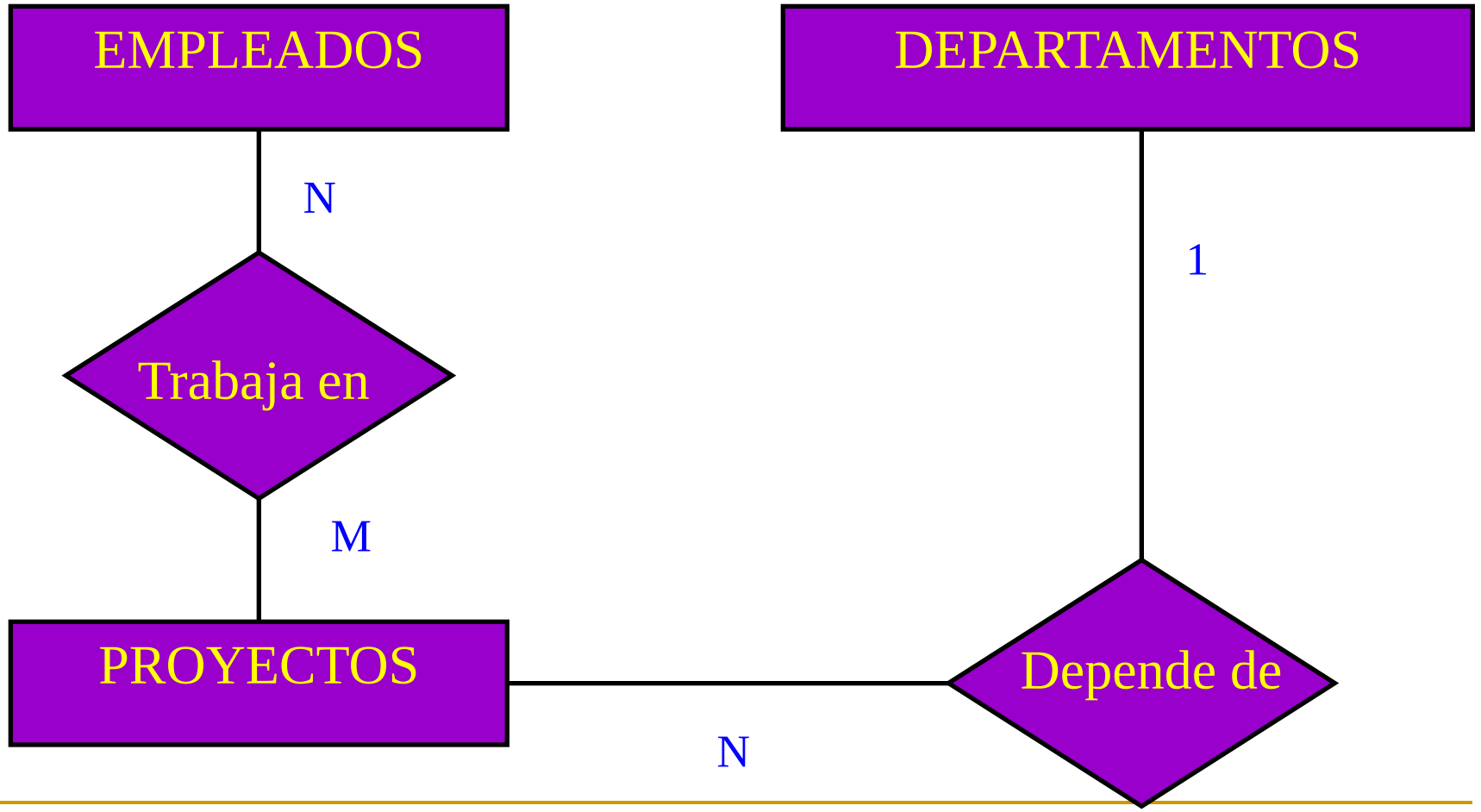
- **Características:**
  - Utilizado por todo tipo de usuarios:
    - Administradores de BDR.
    - Programadores.
    - Usuarios Finales.
  - Lenguaje no procedimental:
    - Se especifica **QUÉ** se quiere obtener, sin decir **CÓMO**.
  - Permite especificar cualquier consulta.

# Introducción

- **Operativa:**
  - Interacción en línea (*on-line*) con el SGBDR.
  - Acceso mediante identificador y palabra clave (*password*).
  - Las sentencias SQL se almacenan en Buffer.
  - El resultado de una consulta se presenta en forma de tabla.
  - Tiene ayuda interactiva (HELP).

# Introducción

- **Diagrama Entidad/Relación de BD Ejemplo:**



# Introducción

- **Transformando el DE/R a Modelo Relacional:**
  - Las tablas obtenidas son las siguientes:
    - T_EMP (Num_Emp, Apellidos, Nombre, Telefono, F_Alta, Sexo, F_Nacimiento, Proyecto)
    - T_DPTO (Num_Dpto, DNombre, Director)
    - T_PROY (Cod_Proy, PNombre, Responsable, Presupuesto, Dpto)

# 2. Lenguaje de Definición de Datos (LDD).

# Lenguaje de Definición de Datos

- **Creación de una Base de Datos:**

  CREATE DATABASE  nombre_base_datos;

- **Selección de una Base de Datos:**

  USE  nombre_base_datos;

- **Cierre de una Base de Datos:**

  CLOSE DATABASE;

- **Borrado de la Base de Datos:**

  **DROP DATABASE nombre_base_datos;**

# Lenguaje de Definición de Datos

- **Ejemplo de creación de la BDR “nominas”:**

  **CREATE DATABASE nominas;**

- **Si quisiéramos comenzar la sesión:**

  **DATABASE nominas; ó USE nominas;**

  - **En algunos gestores puede ser:**
    - **OPEN DATABASE nominas;**
    - **CONNECT DATABASE nominas;**

- **Si quisiéramos acabar la sesión:**

  **CLOSE DATABASE;**

- **Si quisiéramos iniciar una nueva sesión:**

  **DATABASE nominas;**

# Lenguaje de Definición de Datos

- **Tipos básicos de datos:**
  - Datos Alfanuméricos o Cadenas de Caracteres:
    - CHAR(longitud)
    - VARCHAR(longitud)
  - Datos Numéricos:
    - INTEGER
    - DECIMAL ó DECIMAL(precisión, decimal)
      - En algunos SGBDR se usa NUMBER o NUMERIC.
    - SMALLINT
    - REAL
    - FLOAT

  donde:

  **longitud = número máximo de caracteres del campo**
  **precisión = número de dígitos del número**
  **decimal = número de dígitos decimales del nº decimal**

# Lenguaje de Definición de Datos

- **Tipos básicos de datos (y II):**
  - Datos tipo fecha y tiempo:
    - DATE
      - Se puede elegir entre varios formatos.
    - TIME
      - También tiene diferentes formatos.
    - TIMESTAMP
      - Su valor es: fecha + tiempo + nanosegundos.
  - Secuencias:
    - SERIAL ó SERIAL(n)
      - El valor de cada registro se genera en secuencia ascendente.
  - Monetarios:
    - MONEY ó MONEY(m, n)
      - El formato de salida añade el símbolo de moneda: $, £, etc.

# Lenguaje de Definición de Datos

- **Valores nulos (NULL):**
  - NULL representa la ausencia de valor.
  - No es equivalente a cero o blanco.
  - Nada es igual a un valor nulo, ni siquiera otro valor nulo.
  - Se puede obligar a que un atributo no pueda tomar valor nulo al crear la tabla.

# Lenguaje de Definición de Datos

- **Creación de tablas (formato básico):**

```
CREATE TABLE nombre_tabla
(   <definición_atributo_1> [UNIQUE] [NOT NULL],
    <definición_atributo_2> [UNIQUE] [NOT NULL],
    .....
    <definición_atributo_n> [UNIQUE] [NOT NULL]      );
```

**donde:**

- **definición_atributo** = nombre_atributo tipo_dato (tamaño)
- **UNIQUE**: no se permiten valores duplicados en la columna
- **NOT NULL**: no se permiten valores nulos en la columna

# Lenguaje de Definición de Datos

- **Creación de tablas de la BD “nominas” (I):**

```
CREATE TABLE T_Emp
 (Num_Emp          CHAR(6) UNIQUE NOT NULL,
  Apellidos        CHAR(30) NOT NULL,
  Nombre           CHAR(12) NOT NULL,
  Telefono         CHAR(7),
  F_Alta           DATE,
  Sexo             CHAR(1),
  F_Nacimiento     DATE,
  Proyecto         CHAR(3) NOT NULL);
```

# Lenguaje de Definición de Datos

- **Creación de tablas de la BD “nominas” (y II):**

```
CREATE TABLE T_Dpto
  (Num_Dpto          CHAR(3) UNIQUE NOT NULL,
   DNombre           CHAR(20) NOT NULL,
   Director          CHAR(6) NOT NULL);

CREATE TABLE T_Proy
   (Cod_Proy         CHAR(3) UNIQUE NOT NULL,
    PNombre          CHAR(15) NOT NULL,
    Responsable      CHAR(6) NOT NULL,
    Presupuesto      INTEGER,
    Dpto             CHAR(3) NOT NULL);
```

# Lenguaje de Definición de Datos

- **Modificación de tablas:**
  - Añadir un nuevo atributo:

    **ALTER TABLE <nombre_tabla>**

    **ADD <definición_atributo>;**
  - Modificar un atributo ya existente:

    **ALTER TABLE <nombre_tabla>**

    **MODIFY <definición_atributo>;**

- **Eliminación de tablas:**

  **DROP TABLE <nombre_tabla>;**

# Lenguaje de Definición de Datos

- **Añadir a la tabla de empleados el atributo salario.**

```
ALTER TABLE T_Emp
ADD Salario INTEGER;
```

- **Modificar el tamaño del atributo telefono para que admita los 9 dígitos.**

```
ALTER TABLE T_Emp
ALTER COLUMN Telefono CHAR(9);
```

- **Borrar la tabla de empleados:**

```
DROP TABLE T_Emp;
```

# Lenguaje de Definición de Datos

- **Definición de Vista:**
  - Estructura tabular no física (tabla virtual), que permite consultar y/o modificar datos de la tabla real.
- **Creación de vistas:**

  **CREATE VIEW <nombre_vista> [(<lista_atributos>)]**
  **AS ( <cláusula SELECT> );**
  - Las filas de la vista serán aquellas que resulten de ejecutar la consulta sobre la que está definida.
- **Eliminación de vistas:**

  **DROP VIEW <nombre_vista>;**

# Lenguaje de Definición de Datos

- **Características:**
  - Se utilizan como si fuesen tablas reales.
  - No contienen datos propios.
  - No tienen asociada estructura física.
- **Ventajas de uso:**
  - Menor complejidad en consultas:
    - Permiten obtener igual información de forma más simple.
  - Aumento confidencialidad:
    - Permiten acceder sólo a ciertos datos de las tablas reales.

# Lenguaje de Definición de Datos

- **Esquema Externo (Vista) 'listín telefónico'.**

```
CREATE VIEW listin
AS (SELECT Apellidos, Nombre, Telefono
        FROM T_Emp);
```

- **Esquema Externo (Vista) 'relación de empleados asignados a cada proyecto'.**

```
CREATE VIEW proyectos (Proyecto, Apellidos, Nombre)
AS (SELECT PNombre, Apellidos, Nombre
        FROM T_Proy, T_Emp
        WHERE T_Proy.Cod_Proy = T_Emp.Proyecto);
```

# Lenguaje de Definición de Datos

- **Creación de índices:**

  **CREATE [UNIQUE] INDEX <nombre_índice>**

  **ON <nombre_tabla> (<lista_atributos>);**

  - Es el sistema el encargado de utilizar los índices, para optimizar el acceso a los datos.
  - El usuario sólo puede crear o eliminar índices, pero no indicar su utilización.

- **Eliminación de índices:**

  **DROP INDEX <nombre_índice>**

  **[ON <nombre_tabla>];**

# Lenguaje de Definición de Datos

- **Mejorar acceso a la tabla empleados para las consultas sobre nombre y apellidos.**

```
CREATE INDEX nombre_apellidos
ON T_Emp (Apellidos, Nombre);
```

- **Ídem para nombres de departamento.**

```
CREATE INDEX departamentos
ON T_Dpto (DNombre);
```

- **Ídem para nombres de proyecto.**

```
CREATE INDEX proyectos
ON T_Proy (PNombre);
```

# Lenguaje de Definición de Datos

- **Definición de claves primarias:**
  - En la creación de la tabla:

```
CREATE TABLE nombre_tabla
( nombre_columna   tipo_columna   atributos,
 ..... ,
 nombre_columna   tipo_columna   atributos )
PRIMARY KEY (lista_de_columnas);
```

  - Modificando la estructura de la tabla:

```
ALTER TABLE nombre_tabla
ADD PRIMARY KEY (lista_de_columnas);
```

**La columna o columnas que compongan la clave primaria deberán tener asignado el atributo NOT NULL**

# Lenguaje de Definición de Datos

- **Añadir a la tabla departamentos la definición de su clave primaria.**

```
ALTER TABLE T_Dpto
ADD PRIMARY KEY (Num_Dpto);
```

- **Ejemplo completo de creación:**

```
CREATE TABLE T_Dpto
 (Num_Dpto     CHAR(3) UNIQUE NOT NULL,
  DNombre              CHAR(20) NOT NULL,
  Director             CHAR(6) NOT NULL,
PRIMARY KEY (Num_Dpto));
```

# Lenguaje de Definición de Datos

- **Añadir a las tablas de empleados y proyectos la definición de sus claves primarias.**

```
ALTER TABLE T_Emp
ADD PRIMARY KEY (Num_Emp);

ALTER TABLE T_Proy
 ADD PRIMARY KEY (Cod_Proy);
```

# Lenguaje de Definición de Datos

- **Definición de claves Referenciales:**
  - En la creación de la tabla (formato completo):

```
CREATE TABLE nombre_tabla
( nombre_columna   tipo_columna   atributos,
  ..... ,
 nombre_columna   tipo_columna   atributos )
PRIMARY KEY (lista_de_columnas)
FOREIGN KEY  nombre_clave_referencial
                    (lista_de_columnas)
REFERENCES nombre_de_tabla
   ON UPDATE [RESTRICT | SET NULL | CASCADE]
   ON DELETE [RESTRICT | SET NULL | CASCADE]
FOREIGN KEY  .....  ;
```

# Lenguaje de Definición de Datos

- **Definición de claves Referenciales:**
  - Modificando la estructura de la tabla:

```
ALTER TABLE nombre_tabla
FOREIGN KEY  nombre_clave_referencial
                              (lista_de_columnas)
REFERENCES nombre_de_tabla
    ON UPDATE [RESTRICT | SET NULL | CASCADE]
    ON DELETE [RESTRICT | SET NULL | CASCADE];
```

# Lenguaje de Definición de Datos

- **Añadir a la tabla proyectos la definición de su clave ajena ‘dpto’.**

```
ALTER TABLE T_Proy
ADD FOREIGN KEY (Dpto)
REFERENCES T_Dpto
   ON UPDATE CASCADE
   ON DELETE (RESTRICT no funciona);
```

# Lenguaje de Definición de Datos

- **Ejemplo completo de creación:**

```
CREATE TABLE T_Proy
  (Cod_Proy          CHAR(3) UNIQUE NOT NULL,
   PNombre           CHAR(15) NOT NULL,
   Responsable       CHAR(6) NOT NULL,
   Presupuesto INTEGER,
   Dpto          CHAR(3) NOT NULL,
PRIMARY KEY (Cod_Proy),
FOREIGN KEY (Dpto)
REFERENCES T_Dpto
      ON UPDATE CASCADE
      ON DELETE CASCADE);
```

# Lenguaje de Definición de Datos

- **Añadir a la tabla de empleados la definición de su clave ajena ‘proyecto’.**

```
ALTER TABLE T_Emp
ADD FOREIGN KEY (proyecto)
REFERENCES T_Proy
   ON UPDATE CASCADE
   ON DELETE CASCADE;
```

# Lenguaje de Definición de Datos

- **Borrado de claves:**
  - Claves primarias:

    **ALTER TABLE nombre_tabla**

    **DROP PRIMARY KEY;**

  - Claves referenciales:

    **ALTER TABLE nombre_tabla**

    **DROP FOREIGN KEY nombre_clave_referencial;**

# Lenguaje de Definición de Datos

- **EJEMPLO BASE DE DATOS DE EMPLEADOS:**

```
--/****** Object:  Database [EMPRESA] ******/
CREATE DATABASE [EMPRESA]
GO
USE EMPRESA
GO

--/****** Object:  Table [dbo].[DEPARTAMENTO] ******/
CREATE TABLE [dbo].[DEPARTAMENTO](
    [ID Departamento] [int] PRIMARY KEY IDENTITY(1,1) NOT NULL,
    [Nombre Departamento] [nvarchar](50) NOT NULL,
    [id DNI jefe] [nchar](10) UNIQUE NULL )
 –UNIQUE: para poder después hacer una relación 1:1
GO
```

# Lenguaje de Definición de Datos

```
--/****** Object:  Table [dbo].[EMPLEADO] ******/
CREATE TABLE [dbo].[EMPLEADO](
        [DNI] [nchar](10)PRIMARY KEY  NOT NULL,
        [NOMBRE] [nvarchar](20) NOT NULL,
        [APELLIDO1] [nvarchar](20) NOT NULL,
        [APELLIDO2] [nvarchar](20) NULL,
        [CALLE] [nchar](10) NOT NULL,
        [PORTAL] [nchar](5) NOT NULL,
        [PISO] [nchar](3) NULL,
        [PUERTA] [nchar](3) NULL,
        [CIUDAD] [nchar](10) NOT NULL,
        [CODIGO POSTAL] [int] NOT NULL,
        [PAIS] [nchar](10) NULL,
        [SUELDO ANUAL] [money] NULL,
        [Fecha de nacimiento] [date] NOT NULL,
        [Sexo] [bit] NOT NULL,
        [id departamento] [int] NOT NULL )
GO
```

# Lenguaje de Definición de Datos

```
--/****** Object:  Table [dbo].[PROYECTO] ******/
CREATE TABLE [dbo].[PROYECTO](
        [ID Proyecto] [int] PRIMARY KEY IDENTITY(1,1) NOT NULL,
        [Nombre Proyecto] [nvarchar](50) NOT NULL,
        [id departamento] [int] NOT NULL )

GO

--/****** Object:  Table [dbo].[PARTICIPA] ******/
CREATE TABLE [dbo].[PARTICIPA](
        [id empleado] [nchar](10)  NOT NULL,
        [id proyecto] [int]  NOT NULL,
        [Tiempo] [real] NOT NULL);

GO

ALTER TABLE [dbo].[PARTICIPA]
ADD PRIMARY KEY ([id empleado],[id proyecto])

GO
```

# Lenguaje de Definición de Datos

- /****** Object:  ForeignKey [FK_EMPLEADO_DEPARTAMENTO]    ******/
- ALTER TABLE [dbo].[EMPLEADO]  WITH CHECK ADD  CONSTRAINT [FK_EMPLEADO_DEPARTAMENTO] FOREIGN KEY([DNI])
- REFERENCES [dbo].[DEPARTAMENTO] ([id jefe])
- GO

- ALTER TABLE [dbo].[EMPLEADO] CHECK CONSTRAINT [FK_EMPLEADO_DEPARTAMENTO]
- GO

- /****** Object:  ForeignKey [FK_EMPLEADO_DEPARTAMENTO1]   ******/
- ALTER TABLE [dbo].[EMPLEADO]  WITH CHECK ADD  CONSTRAINT [FK_EMPLEADO_DEPARTAMENTO1] FOREIGN KEY([id departamento])
- REFERENCES [dbo].[DEPARTAMENTO] ([ID Departamento])
- GO

- ALTER TABLE [dbo].[EMPLEADO] CHECK CONSTRAINT [FK_EMPLEADO_DEPARTAMENTO1]
- GO

# Lenguaje de Definición de Datos

```
/****** Object:  ForeignKey [FK_PROYECTO_DEPARTAMENTO] ******/
ALTER TABLE [dbo].[PROYECTO]  WITH CHECK ADD  CONSTRAINT
[FK_PROYECTO_DEPARTAMENTO] FOREIGN KEY([id departamento])
REFERENCES [dbo].[DEPARTAMENTO] ([ID Departamento])
GO

ALTER TABLE [dbo].[PROYECTO] CHECK CONSTRAINT
[FK_PROYECTO_DEPARTAMENTO]
GO

--/****** Object:  ForeignKey [FK_PARTICIPA_EMPLEADO]
ALTER TABLE [dbo].[PARTICIPA]  WITH CHECK ADD  CONSTRAINT
[FK_PARTICIPA_EMPLEADO] FOREIGN KEY([id empleado])
REFERENCES [dbo].[EMPLEADO] ([DNI])
GO

ALTER TABLE [dbo].[PARTICIPA] CHECK CONSTRAINT [FK_PARTICIPA_EMPLEADO]
GO
```

# Lenguaje de Definición de Datos

- /****** Object:  ForeignKey [FK_PARTICIPA_PROYECTO] */

- ALTER TABLE [dbo].[PARTICIPA]  WITH CHECK ADD  CONSTRAINT [FK_PARTICIPA_PROYECTO] FOREIGN KEY([id proyecto])
- REFERENCES [dbo].[PROYECTO] ([ID Proyecto])
- GO

- ALTER TABLE [dbo].[PARTICIPA] CHECK CONSTRAINT [FK_PARTICIPA_PROYECTO]
- GO

# 3. Lenguaje de Manipulación de Datos (DML).

## 3.1. Operaciones de Actualización.

# Lenguaje de Manipulación de Datos

## 3.1. Operaciones de Actualización

- **Inserción de filas:**
  - Inserción de una fila:

    **INSERT**
    **INTO <nombre_tabla> [(<lista_de _atributos>)]**
    **VALUES (<valor_1>, <valor_2>,...,<valor_n>);**
  - Inserción de varias filas:

    **INSERT**
    **INTO <nombre_tabla> [(<lista_de_atributos>)]**
    **( <cláusula SELECT> );**

    **La cláusula "SELECT" especifica una consulta cuyo resultado (filas) se insertará en la tabla especificada.**

# Lenguaje de Manipulación de Datos

## 3.1. Operaciones de Actualización

- **En el día de hoy se incorpora a la empresa el empleado Nicolás García Ramos nacido el 27/09/1968, se le asigna al proyecto ‘P56’.**

```
INSERT
INTO T_Emp
VALUES (‘E-1237’, ‘García Ramos’, ‘Nicolás’,
                  ’913565656’, SYSDATE, ’V’,
                  ’27/09/1968’, ‘P56’);
```

  - Al no especificar la lista de atributos los valores deben escribirse en el mismo orden en que se definieron en el CREATE TABLE.

# Lenguaje de Manipulación de Datos

## 3.1. Operaciones de Actualización

- **Se inicia el proyecto ‘P59’, ‘Portal Educativo’, con un presupuesto de 3.000.000 de pesetas. Su responsable será ‘E-0376’ y el departamento el ‘D12’.**

```
INSERT INTO T_Proy
    (Cod_Proy, PNombre, Presupuesto, Responsable, Dpto)
VALUES (‘P59’, ‘Portal Educativo’, 3000000, ’E-0376’, ‘D12’);
```

# Lenguaje de Manipulación de Datos

## 3.1. Operaciones de Actualización

- **Se ha creado una nueva tabla T_Formacion con igual estructura que T_Emp. Se desea inicializarla con todos aquellos empleados cuya fecha de alta en la empresa sea posterior al 01/01/1999'.**

```
INSERT
INTO T_Formacion
(SELECT * FROM T_Emp WHERE F_Alta = '01/01/1999);
```

# Lenguaje de Manipulación de Datos

## 3.1. Operaciones de Actualización

- **Modificación de filas:**

```
UPDATE <nombre_tabla>
SET <atributo_1> = <valor_1>,
      <atributo_2> = <valor_2>,
       ..........
     <atributo_n> = <valor_n>
[WHERE <condición>];
```

  - La modificación afectará a todas las filas que cumplan la condición, si se especifica ésta. Si no se especifica condición, la modificación afectará a todas las filas de la tabla.
  - El valor que se asigne al atributo debe ser una constante, o el resultado de una subconsulta (que irá entre paréntesis).

# Lenguaje de Manipulación de Datos

## 3.1. Operaciones de Actualización

- **Una vez finalizado el proyecto ‘P02’ se reasignan los empleados de dicho proyecto al proyecto ‘P45’.**

```
UPDATE T_Emp
SET Proyecto = ‘P45’
WHERE Proyecto = ‘P02’;
```

# Lenguaje de Manipulación de Datos

## 3.1. Operaciones de Actualización

- **El proyecto ‘P23’ es aumentado en 2000000 de presupuesto y reasignado al departamento ‘D25’.**

```
UPDATE T_Proy
SET Dpto = ‘D25’,
      Presupuesto = (SELECT Presupuesto + 2000000
                     FROM T_Proy
                     WHERE Cod_Proy = ‘P23’)
WHERE Cod_Proy = ‘P23’;
```

# Lenguaje de Manipulación de Datos

## 3.1. Operaciones de Actualización

- **Eliminación de filas:**

```
DELETE
FROM <nombre_tabla>
[WHERE <condición>];
```

  - No se pueden eliminar partes de una fila.
  - Si no aparece la cláusula "WHERE" se eliminarán todas las filas de la tabla, no eliminándose la definición de ésta en el esquema.

# Lenguaje de Manipulación de Datos

## 3.1. Operaciones de Actualización

- **Eliminar el Proyecto ‘P02’ una vez finalizado.**

```
DELETE
FROM T_Proy
WHERE Cod_Proy = ‘P02’ ;
```

# 3. Lenguaje de Manipulación de Datos (DML).

## 3.2. Operaciones de Consulta o Recuperación.

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Sintaxis de la sentencia:**

```
SELECT [UNIQUE/DISTINCT] <expresión>
FROM  <lista_de_tablas>
[WHERE <condición>]
[GROUP BY <lista_de_atributos>
[HAVING <condición_de_grupo> ]]
[ORDER BY <lista_de_atributos> [ASC/DESC] ];
```

  - SELECT: especifica la información que se desea obtener.
  - FROM: indica las tablas o vistas en las que se encuentran los atributos implicados en la consulta.
  - WHERE: especifica la condición de búsqueda.
  - GROUP BY: permite agrupar el resultado.
  - HAVING: especifica una condición de grupo.
  - ORDER BY: permite ordenar el resultado.

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Operadores:**
  - Los operadores que se pueden utilizar para expresar condiciones de fila (cláusula WHERE) o de grupo (cláusula HAVING) son:
    - De comparación: <, <=, >, >=, <>, =
    - Lógicos: AND, OR, NOT
    - BETWEEN ... AND ...
    - LIKE
    - IN
    - IS NULL
    - Cuantificadores: ANY, SOME, ALL
    - Existencial: EXISTS

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Reglas de Evaluación de Operadores:**
  - Orden de Evaluación:
    1. Operadores de Relación, BETWEEN, IN, LIKE, IS NULL.
    2. NOT.
    3. AND.
    4. OR.
  - Se pueden utilizar paréntesis para establecer el orden de evaluación deseado por el usuario.

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Condiciones de selección (I).**

1. Recuperación simple:
   - Obtener todos los datos de todos los proyectos:

```
SELECT Cod_Proy, PNombre, Responsable, Presupuesto, Dpto
FROM T_Proy;
```

   ó su equivalente:

```
SELECT *
FROM T_Proy;
```

   *: equivale a todos los atributos de una tabla.
   - Obtener los códigos y nombre de proyectos actualmente en desarrollo.

```
SELECT DISTINCT Cod_Proy, PNombre
FROM T_Proy;
```

     - DISTINCT elimina los valores repetidos.

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Condiciones de selección (II).**

  2. Recuperación calificada:

  - Obtener los números de los empleados que son varones.

    ```
    SELECT Num_Emp
    FROM T_Emp
    WHERE Sexo = 'V';
    ```

  - Obtener los apellidos y nombre de los empleados varones que tengan un salario inferior a 200000.

    ```
    SELECT Apellidos, Nombre
    FROM T_Emp
    WHERE Sexo = 'V' AND Salario < 200000;
    ```

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Condiciones de selección (III).**

3. Recuperación con más de una tabla:
   - Obtener para cada proyecto, su código y los empleados asignados.

```
SELECT  Cod_Proy, Apellidos, Nombre
FROM T_Proy, T_Emp
WHERE Cod_Proy = Proyecto;
```

4. Recuperación con el operador 'BETWEEN ... AND ...' (Establece una comparación dentro de un intervalo cerrado).
   - Obtener el nombre de los proyectos cuyo presupuesto esté comprendido entre 5.000.000 y 15.000.000 Ptas.

```
SELECT PNombre
FROM  T_Proy
WHERE Presupuesto BETWEEN 5000000 AND 15000000;
```

También se puede utilizar **NOT BETWEEN.**

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Condiciones de selección (IV).**

  5. Recuperación con el operador 'LIKE‘:

     Establece una comparación entre cadenas de caracteres, empleando los siguientes comodines:

     - '%' : sustituye a una cadena de caracteres cualquiera.
     - '_' : sustituye a un único carácter cualquiera.

     - Obtener los nombres de los empleados que comiencen por 'C‘.

       ```
       SELECT Nombre
       FROM T_Emp
       WHERE Nombre LIKE ‘C%’;
       ```

       También se puede utilizar NOT LIKE.

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Condiciones de selección (V).**

6. Recuperación con el operador "IN“:

**Comprueba la pertenencia de un valor a un conjunto dado.**

- Obtener los nombres de los proyectos desarrollados en los departamentos ‘D01’ o ‘D02’.

```
SELECT PNombre
FROM T_Proy
WHERE Dpto IN ('D01',D02');
```

- Obtener los empleados asignados al proyecto ‘Portal Formacion‘.

```
SELECT Apellidos, Nombre
FROM T_Emp
WHERE Proyecto IN (SELECT Cod_Proy
                   FROM T_Proy
                   WHERE PNombre = ' Portal Formacion ');
```

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Condiciones de selección (VI).**

6. Recuperación con el operador "IN" (Cont.)

**ó su equivalente:**

```
SELECT Apellidos, Nombre
FROM T_Emp, T_Proy
WHERE Proyecto = Cod_Proy AND PNombre = 'Portal Formacion';
```

**ó su otra equivalente:**

```
SELECT Apellidos, Nombre
FROM T_Emp
WHERE Proyecto IN (SELECT Cod_proy FROM T_Proy
                                    WHERE Pnombre = 'Portal
    formacion' );
```

También se puede utilizar **NOT IN.**

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Condiciones de selección (VII).**

  7. Recuperación con el operador 'IS NULL‘:

  Comprueba si un valor determinado es nulo (NULL).

  - Obtener los empleados que no tienen telefono.

```
SELECT Apellidos, Nombre
FROM T_Emp
WHERE Telefono IS NULL;
```

También se puede utilizar **IS NOT NULL.**

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Condiciones de selección (VIII).**

  8. Consulta con cuantificadores (ALL: todos, ANY: alguno).

     Van acompañados de un operador de comparación: > ALL, >= ALL, < ALL, ...; > ANY, >= ANY, < ANY, ...

     - Obtener los empleados no asignados al proyecto 'Piloto‘.

```
SELECT Apellidos, Nombre
FROM T_Emp
WHERE Proyecto <> ALL (SELECT Cod_Proy
                       FROM T_Proy
                       WHERE Pnombre = 'Piloto');
```

       - La segunda sentencia SELECT se denomina subconsulta.

## Condiciones de selección (IX).

8. Consulta con cuantificadores. (Cont.)

- Obtener los empleados no asignados al mismo proyecto que el empleado Ángel Pérez Pérez.

```
SELECT Apellidos, Nombre
FROM T_Emp
WHERE Proyecto <> ANY (SELECT Proyecto
                       FROM T_Emp
                       WHERE Nombre = 'Ángel'
                          AND Apellidos = 'Pérez Pérez');
```

  - Esta consulta no tiene solución sin usar subconsultas.

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Condiciones de selección (y X).**

9. Recuperación con el operador 'EXISTS‘: Indica la existencia o no de un conjunto.
   - Obtener los empleados asignados al proyecto ‘Internet 2000’.

```
SELECT Apellidos, Nombre
FROM T_Emp
WHERE EXISTS (SELECT *
                    FROM T_Proy
                    WHERE Proyecto = Cod_Proy
                       AND PNombre = ‘Internet 2000');
```

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Otros operadores para Consultas (I).**
  1. Consultas con UNION, DIFERENCIA e INTERSECCIÓN:
     - Unión de conjuntos: operador UNION.
     - Diferencia de conjuntos: operador MINUS.
     - Intersección de conjuntos: operador INTERSECT.
  - Obtener los empleados asignados al proyecto ‘P09’ o que tengan salario inferior a 150.000 Pts.

```
SELECT Apellidos, Nombre
FROM T_Emp
WHERE Proyecto = ‘P09'
UNION
SELECT Apellidos, Nombre
FROM T_Emp
WHERE Salario < 150000;
```

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Otros operadores para Consultas (II).**
  - Obtener los empleados con salario inferior a 150.000 Pts. y que no estén asignados al proyecto ‘P01’.

```
SELECT Apellidos, Nombre
FROM T_Emp
WHERE Salario < 150000
MINUS ó EXCEPT
SELECT Apellidos, Nombre
FROM T_Emp
WHERE Proyecto = ‘P01';
```

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Otros operadores para Consultas (III).**
  - Obtener los departamentos de los que dependen los proyectos ‘P15’ y ‘P25’ .

```
SELECT Dpto
FROM T_Proy
WHERE Cod_Proy = ‘P15'
INTERSECT
SELECT Dpto
FROM T_Proy
WHERE Cod_Proy = ‘P25';
```

- **Deseamos saber de todas las fechas donde hay una operación de venta.**

- Tabla ***Store_Information***

| store_name | Sales | Date |
|---|---|---|
| Los Angeles1 | 500 € | 05-Jan-1999 |
| San Diego | 250 € | 07-Jan-1999 |
| Los Angeles | 300 € | 08-Jan-1999 |
| Boston | 700 € | 08-Jan-1999 |

- Tabla ***Internet_Sales***

| Date | Sales |
|---|---|
| 07-Jan-1999 | 250 € |
| 10-Jan-1999 | 535 € |
| 11-Jan-1999 | 320 € |
| 12-Jan-1999 | 750 € |

- **SELECT Date FROM Store_Information**
  **UNION**
  **SELECT Date FROM Internet_Sales**

- *Resultado:*
- **05-Jan-1999**
- **07-Jan-1999**
- **08-Jan-1999**
- **10-Jan-1999**
- **11-Jan-1999**
- **12-Jan-1999**

- **Deseamos encontrar las fechas en donde se realizó una operación de venta en un negocio como así también las fechas donde hay una venta a través de Internet.**

- Tabla ***Store_Information***

| store_name | Sales | Date |
|---|---|---|
| Los Angeles1 | 500 € | 05-Jan-1999 |
| San Diego | 250 € | 07-Jan-1999 |
| Los Angeles | 300 € | 08-Jan-1999 |
| Boston | 700 € | 08-Jan-1999 |

- Tabla ***Internet_Sales***

| Date | Sales |
|---|---|
| 07-Jan-1999 | 250 € |
| 10-Jan-1999 | 535 € |
| 11-Jan-1999 | 320 € |
| 12-Jan-1999 | 750 € |

- **SELECT Date FROM Store_Information**
  **UNION ALL**
  **SELECT Date FROM Internet_Sales**

- *Resultado:*
- **05-Jan-1999**
- **07-Jan-1999**
- **08-Jan-1999**
- **08-Jan-1999**
- **07-Jan-1999**
- **10-Jan-1999**
- **11-Jan-1999**
- **12-Jan-1999**

- **Deseamos encontrar todas las fechas donde hay ventas tanto en el negocio como en Internet**

- Tabla ***Store_Information***

| store_name | Sales | Date |
|---|---|---|
| Los Angeles1 | 500 € | 05-Jan-1999 |
| San Diego | 250 € | 07-Jan-1999 |
| Los Angeles | 300 € | 08-Jan-1999 |
| Boston | 700 € | 08-Jan-1999 |

- Tabla ***Internet_Sales***

| Date | Sales |
|---|---|
| 07-Jan-1999 | 250 € |
| 10-Jan-1999 | 535 € |
| 11-Jan-1999 | 320 € |
| 12-Jan-1999 | 750 € |

- **SELECT Date FROM Store_Information**
  **INTERSECT**
  **SELECT Date FROM Internet_Sales**

- *Resultado:*

| **Date** |
|---|
| 07-Jan-1999 |

- **Deseamos encontrar todas las fechas donde hay ventas en el negocio, pero no aquellas realizadas por Internet**

- Tabla ***Store_Information***

| store_name | Sales | Date |
|---|---|---|
| Los Angeles1 | 500 € | 05-Jan-1999 |
| San Diego | 250 € | 07-Jan-1999 |
| Los Angeles | 300 € | 08-Jan-1999 |
| Boston | 700 € | 08-Jan-1999 |

- Tabla ***Internet_Sales***

| Date | Sales |
|---|---|
| 07-Jan-1999 | 250 € |
| 10-Jan-1999 | 535 € |
| 11-Jan-1999 | 320 € |
| 12-Jan-1999 | 750 € |

- **SELECT Date FROM Store_Information**
  **EXCEPT**
  **SELECT Date FROM Internet_Sales**

- *Resultado:*

- **Date**
- 05-Jan-1999
  08-Jan-1999

- *Por favor note que el comando* **EXCEPT ó *MINUS*** *sólo arrojará valores distintos.*

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Otros operadores para Consultas (y IV).**

  2. Expresiones en la cláusula SELECT.

  No sólo se pueden seleccionar atributos, sino expresiones en las que aparezcan atributos y/o constantes y operadores aritméticos.

  - Obtener el salario anula de los empleados de la empresa.

    **SELECT Apellidos, Nombre, 'Salario Anual:', (Salario * 14)**

    **FROM T_Emp;**

    - **'Salario Anual:' es un literal o constante, que aparecerá en todas las tuplas resultado de la consulta.**

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Funciones agregadas:**
  - COUNT (*): contador de tuplas (totalizador)
  - COUNT (DISTINCT): contador de tuplas (parcial), no tiene en cuenta valores nulos ni duplicados
  - AVG: media aritmética de un atributo o una expresión numérica
  - SUM: suma de atributos o expresiones numéricas
  - MAX: valor máximo de un atributo o expresión numérica
  - MIN: valor mínimo de un atributo o expresión numérica

    - Devuelven un valor único, numérico.
    - No se pueden combinar, con columnas que devuelvan más de un valor, a menos que la consulta contenga una cláusula GROUP BY.

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Obtener el número total de proyectos realizados.**

```
SELECT COUNT(*)
FROM T_Proy;
```

- **Obtener el número total de proyectos del Departamento 'D02'.**

```
SELECT COUNT (DISTINCT Cod_Proy)
FROM T_Proy
WHERE Dpto = 'D02';
```

- **Obtener el presupuesto medio y total de los proyectos del Departamento 'D21'.**

```
SELECT AVG(Presupuesto), SUM(Presupuesto)
FROM T_Proy
WHERE Dpto = 'D21';
```

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Cláusula GROUP BY:**

  GROUP BY <lista_de_atributos>

  - Agrupa el resultado, devolviendo una única fila por grupo.
  - El agrupamiento no se realiza ordenado.
  - Los atributos que aparezcan en GROUP BY, deben aparecer en la cláusula SELECT.
    - Obtener por cada proyecto, el código de éste y el número total de empleados asignados.

      ```
      SELECT Proyecto, COUNT(*)
      FROM T_Emp
      GROUP BY Proyecto;
      ```

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Cláusula HAVING:**

  HAVING <condición_de_grupo>

  - Siempre va acompañada de la cláusula GROUP BY.
  - Especifica una condición de grupo.
    - Obtener para los conductores que hayan utilizado la misma máquina más de una vez entre el 12/09/94 y el 18/09/94, el código de conductor, el código de máquina y el tiempo total empleado

```
SELECT codigo_conductor, codigo_maquina, SUM (tiempo)
FROM trabajos
WHERE fecha BETWEEN 12/09/94 AND 18/09/94
GROUP BY codigo_conductor, codigo_maquina
HAVING COUNT(*) > 1;
```

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Cláusula ORDER BY:**

  ORDER BY <lista_de_atributos> [ASC | DESC]

  - El resultado de la consulta se ordena en base a los atributos que se indiquen en la lista.
  - Los atributos de ordenación deben aparecer en SELECT.
    - Obtener los nombres de los proyectos ordenados ascendentemente por departamento y presupuesto.

      ```
      SELECT Pnombre, Dpto, Presupuesto
      FROM T_Proy
      ORDER BY Dpto, Presupuesto;
      ```

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Uso de Alias de columnas (I):**
  - SQL visualiza el resultado de una consulta, dejando como nombre en la cabecera el mismo nombre con que están definidos los atributos en la tabla.
  - Para cambiar esos nombres de cabecera se pueden usar Alías de Atributo en la Sentencia SELECT:

```
SELECT Cod_Proy CODIGO, PNombre PROYECTO
FROM T_Proy;
```

    - Si el Alías lleva espacios en blanco o el carácter '_' deberá ir entrecomillado.

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Uso de Alias de columnas (y II):**
  - También, se usan para nombrar una columna obtenida por medio de alguna expresión:
    - **Obtener el presupuesto medio de los proyectos del Departamento 'D21'.**

```
SELECT AVG(Presupuesto) 'PRESUPUESTO MEDIO'
FROM T_Proy
WHERE Dpto = 'D21';
```

# Lenguaje de Manipulación de Datos

## 3.2. Operaciones de Consulta.

- **Uso de Alias de tablas (utilidad):**
  - Abreviar un nombre de tabla que se usa a menudo.
  - Hacer más clara una instrucción complicada de SQL.
  - Distinguir entre dos ocurrencias del mismo nombre de tabla en cualquier instrucción SELECT.
    - Obtener los empleados que tengan el salario más alto.

```
SELECT E.Apellidos + ', ' + E.Nombre CONDUCTOR
FROM T_Emp E, T_Emp F
WHERE E.Salario > = F.Salario;
```

# 4. Lenguaje de Control de Datos (DCL).

# Lenguaje de Control de Datos

- **Generalidades:**
  - Control de Accesos a los datos (Seguridad).
  - Control de Integridad.
- **Control de Accesos:**
  - Niveles de acceso soportados por los SGBDR:
    - Base de Datos.
    - Tabla.
    - Atributo.
    - Tupla.
  - Se establecen privilegios de acceso por nivel.

# Lenguaje de Control de Datos

- **Concesión de Privilegios: Permite dar a los usuarios el acceso completo o restringido a la base de datos:**

  GRANT <privilegio_de_acceso>

  [ON <lista_de_tablas>]

  TO <lista_de_usuarios> | PUBLIC

  [WITH GRANT OPTION];

  donde:

  - <privilegio_de_acceso>: CONNECT, RESOURCE, DBA, ALL PRIVILEGES, ALTER, INDEX, SELECT, UPDATE, INSERT, DELETE.
  - PUBLIC: se conceden los privilegios a todos los usuarios.
  - WITH GRANT OPTION: se concede el privilegio de poder otorgar privilegios a otros usuarios.

# Lenguaje de Control de Datos

- **Nivel de Base de Datos (I):**
  - El SGBDR chequea los privilegios del usuario al iniciar la sesión.
  - Los posibles privilegios o permisos son:
    - CONNECT: Conectarse a la BDR.
    - RESOURCE: Crear objetos.
    - DBA:
      - Ejecución de comandos restrictivos.
      - Acceso a cualquier objeto.
      - Privilegio RESOURCE implícito.

# Lenguaje de Control de Datos

- **Nivel de Base de Datos (y II):**
  - Las password están encriptadas.
  - Cada usuario puede cambiar su password:

    ```
    GRANT CONNECT TO Nombre_Usuario
    IDENTIFIED BY Nueva_Password;
    ```

  - Un DBA puede modificar los privilegios de cualquier usuario.

# Lenguaje de Control de Datos

- **Nivel de Tabla (I):**
  - Las tablas son propiedad del usuario que las creó.
  - Los posibles privilegios o permisos son:
    - ALTER: Permite modificar la definición de la tabla.
    - INDEX: Permite crear índices sobre la tabla.
    - DELETE: Autoriza el borrado de tuplas.
    - INSERT: Autoriza la inserción de nuevas tuplas.
    - SELECT: Permite la realización de consultas.
    - UPDATE: Permite la actualización de tuplas.
    - ALL PRIVILEGES: Concede todos los privilegios.

# Lenguaje de Control de Datos

- **Nivel de Tabla (y II):**
  - Si un privilegio se quiere dar a todos los usuarios de la BDR se utiliza PUBLIC.
    - Ejemplo de acceso en consulta a todos los usuarios sobre la tabla T_Proy:

```
GRANT SELECT
ON T_Proy
TO PUBLIC;
```

    - Ejemplo de acceso total a ‘Usuario1‘ sobre T_Emp:

```
GRANT ALL PRIVILEGES
ON T_Emp
TO Usuario1;
```

# Lenguaje de Control de Datos

- **Sinónimos (I):**
  - El uso de Sinónimos proporciona un mayor nivel de seguridad sobre las tablas de la BDR.
  - Un Sinónimo es un nombre alternativo para una tabla; y proporciona independencia respecto al nombre real.

    **CREATE [PUBLIC] SINONYM <Nombre_Sinonimo>**

    **FROM <Nombre_Tabla>;**

    - **Si se utiliza PUBLIC el sinónimo es de uso público, es decir, para todos los usuarios.**

# Lenguaje de Control de Datos

- **Sinónimos (y II):**
  - Crear el sinónimo ‘Proyectos’ sobre la tabla T_Proy.

    ```
    CREATE SINONYM Proyectos
    FROM T_Proy;
    ```

  - Ejemplo de acceso en consulta a todos los usuarios sobre la tabla T_Proy usando el sinónimo anterior:

    ```
    GRANT SELECT
    ON Proyectos
    TO PUBLIC;
    ```

    - **Con el uso del sinónimo se oculta el nombre real de la tabla.**

# Lenguaje de Control de Datos

- **Niveles Atributo y Tupla (I):**
  - Se implantan a través de la definición de vistas.
  - Nivel de Atributo:
    - Se crea una vista sin condiciones:

```
CREATE VIEW listin
AS (SELECT Apellidos, Nombre, Telefono
      FROM T_Emp);
```

    - Se establecen los permisos sobre la vista:

```
GRANT SELECT
ON listin
TO UsuarioX;
```

      - **UsuarioX no tiene acceso al resto de atributos de T_Emp.**

# Lenguaje de Control de Datos

- **Niveles Atributo y Tupla (y II):**
  - Nivel de Tupla:
    - Se crea una vista con sólo las tuplas permitidas:

```
CREATE VIEW Proyectos_Menores
AS (SELECT PNombre, Presupuesto
        FROM T_Proy WHERE Presupuesto < 5000000);
```

    - Se establecen los permisos sobre la vista:

```
GRANT SELECT
ON Proyectos_Menores
TO UsuarioY;
```

      - **UsuarioY no tiene acceso a proyectos de presupuesto superior a 5.000.000 de pesetas.**

# Lenguaje de Control de Datos

- **Revocación de privilegios**
  - Se utiliza para anular privilegios ya concedidos a los usuarios:

```
REVOKE <privilegio_de_acceso>
[ON <lista_de_tablas>]
TO  <lista_de_usuarios> | PUBLIC;
```

    - Quitar a ‘Usuario1’ el privilegio ALTER sobre T_Emp:

```
REVOKE ALTER
ON T_Emp
TO Usuario1;
```

# SQL

SQL is a standard language for accessing and manipulating databases.

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL is an ANSI (American National Standards Institute) standard

http://www.w3schools.com/sql/default.asp

# What Can SQL do?

SQL can execute queries against a database
SQL can retrieve data from a database
SQL can insert records in a database
SQL can update records in a database
SQL can delete records from a database
SQL can create new databases
SQL can create new tables in a database
SQL can create stored procedures in a database
SQL can create views in a database
SQL can set permissions on tables, procedures, and views

# SQL is a Standard - BUT....

- Although SQL is an ANSI (American National Standards Institute) standard, there are many different versions of the SQL language.
- However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.
- **Note:** Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard!

# Using SQL in Your Web Site

To build a web site that shows some data from a database, you will need the following:

- An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
- A server-side scripting language, like PHP or ASP
- SQL
- HTML / CSS

# RDBMS

- RDBMS stands for Relational Database Management System.
- RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- The data in RDBMS is stored in database objects called tables.
- A table is a collections of related data entries and it consists of columns and rows.

# Database Tables

A database most often contains one or more tables.

Each table is identified by a name (e.g. "Customers" or "Orders").

Tables contain records (rows) with data.

# Below is an example of a table called "Persons":

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

The table above contains three records (one for each person) and five columns (P_Id, LastName, FirstName, Address, and City).

# SQL Statements

- Most of the actions you need to perform on a database are done with SQL statements.
- The following SQL statement will select all the records in the "Persons" table:

```
SELECT * FROM Persons
```

Keep in Mind That...

SQL is not case sensitive

# Semicolon after SQL Statements?

- Some database systems require a semicolon at the end of each SQL statement.
- Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.
- We are using MS Access and SQL Server 2000 and we do not have to put a semicolon after each SQL statement, but some database programs force you to use it.

# SQL DML and DDL

- SQL can be divided into two parts:
  - The Data Manipulation Language (DML)
  - The Data Definition Language (DDL).

# The query and update commands form the DML part of SQL:

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database

# The DDL part of SQL permits database tables to be created or deleted

It also define indexes (keys), specify links between tables, and impose constraints between tables.

The most important DDL statements in SQL are:

- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

# The SQL SELECT Statement

The SELECT statement is used to select data from a database.

The result is stored in a result table, called the result-set.

SQL SELECT Syntax:

- *SELECT column_name(s)*
  *FROM table_name*
- and
- *SELECT * FROM table_name*

- **Note:** SQL is not case sensitive. SELECT is the same as select.

# An SQL SELECT Example

The "Persons" table:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Now we want to select the content of the columns named "LastName" and "FirstName" from the table above.

We use the following SELECT statement:

*SELECT LastName,FirstName FROM Persons*

**Note: If Field name or Table name has blank character, use [ ]**

*SELECT [Last Name] , [First Name] FROM Persons*

# The result-set will look like this:

| LastName | FirstName |
|---|---|
| Hansen | Ola |
| Svendson | Tove |
| Pettersen | Kari |

# SELECT * Example

Now we want to select all the columns from the "Persons" table.

We use the following SELECT statement:

*SELECT * FROM Persons*

**Tip:** The asterisk (*) is a quick way of selecting all columns!

# The result-set will look like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

# Navigation in a Result-set

Most database software systems allow navigation in the result-set with programming functions, like: Move-To-First-Record, Get-Record-Content, Move-To-Next-Record, etc.

Programming functions like these are not a part of this tutorial. To learn about accessing data with function calls, please visit our ADO tutorial or our PHP tutorial.

# The SQL SELECT DISTINCT Statement

In a table, some of the columns may contain duplicate values. This is not a problem, however, sometimes you will want to list only the different (distinct) values in a table.

The DISTINCT keyword can be used to return only distinct (different) values.

SQL SELECT DISTINCT Syntax

*SELECT DISTINCT column_name(s)*
*FROM table_name*

# SELECT DISTINCT Example

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Now we want to select only the distinct values from the column named "City" from the table above.

We use the following SELECT statement:

```
SELECT DISTINCT City FROM Persons
```

| City |
|---|
| Sandnes |
| Stavanger |

# The WHERE Clause

The WHERE clause is used to extract only those records that fulfill a specified criterion.

SQL WHERE Syntax

- *SELECT column_name(s)*
  *FROM table_name*
  *WHERE column_name operator value*

# WHERE Clause Example

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Now we want to select only the persons living in the city "Sandnes" from the table above.

We use the following SELECT statement:

*SELECT * FROM Persons*
*WHERE City='Sandnes'*

# The result-set will look like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

# Quotes Around Text Fields

SQL uses single quotes around text values (most database systems will also accept double quotes).

Although, numeric values should not be enclosed in quotes.

For text values:

This is correct:

```
SELECT * FROM Persons WHERE FirstName='Tove'
```

This is wrong:

```
SELECT * FROM Persons WHERE FirstName=Tove
```

# For numeric values:

This is correct:

```
SELECT * FROM Persons WHERE Year=1965
```

This is wrong:

```
SELECT * FROM Persons WHERE Year='1965'
```

# Operators Allowed in the WHERE Clause

| Operator | Description |
|---|---|
| = | Equal |
| < > | Not equal |
| > | Greater than |
| < | Less than |
| > = | Greater than or equal |
| < = | Less than or equal |
| BETWEEN | Between an inclusive range |
| LIKE | Search for a pattern |
| IN | If you know the exact value you want to return for at least one of the columns |

**Note:** In some versions of SQL the <> operator may be written as !=

# The AND & OR Operators

The "Persons" table:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Now we want to select only the persons with the first name equal to "Tove" AND the last name equal to "Svendson":

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE FirstName='Tove'
AND LastName='Svendson'
```

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

# OR Operator Example

Now we want to select only the persons with the first name equal to "Tove" OR the first name equal to "Ola":

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE FirstName='Tove'
OR FirstName='Ola'
```

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

# Combining AND & OR

You can also combine AND and OR (use parenthesis to form complex expressions).

Now we want to select only the persons with the last name equal to "Svendson" AND the first name equal to "Tove" OR to "Ola":

We use the following SELECT statement:

```
SELECT * FROM Persons WHERE
LastName='Svendson'
AND (FirstName='Tove' OR FirstName='Ola')
```

The result-set will look like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

# The ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set by a specified column.

The ORDER BY keyword sort the records in ascending order by default.

If you want to sort the records in a descending order, you can use the DESC keyword.

SQL ORDER BY Syntax:

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC
```

# ORDER BY Example

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Tom | Vingvn 23 | Stavanger |

Now we want to select all the persons from the table above, however, we want to sort the persons by their last name.

We use the following SELECT statement:

```
SELECT * FROM Persons
ORDER BY LastName
```

The result-set will look like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 4 | Nilsen | Tom | Vingvn 23 | Stavanger |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

# ORDER BY DESC Example

Now we want to select all the persons from the table above, however, we want to sort the persons descending by their last name.

We use the following SELECT statement:

```
SELECT * FROM Persons
ORDER BY LastName DESC
```

The result-set will look like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Tom | Vingvn 23 | Stavanger |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |

# The INSERT INTO Statement

The INSERT INTO statement is used to insert a new row in a table.

SQL INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two forms.

# The INSERT INTO Statement

The first form doesn't specify the column names where the data will be inserted, only their values:

```
INSERT INTO table_name
VALUES (value1, value2, value3,...)
```

The second form specifies both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

# SQL INSERT INTO Example

We have the following "Persons" table:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Now we want to insert a new row in the "Persons" table.
We use the following SQL statement:

```
INSERT INTO Persons
VALUES (4,'Nilsen', 'Johan', 'Bakken 2', 'Stavanger')
```

The "Persons" table will now look like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Johan | Bakken 2 | Stavanger |

# Insert Data Only in Specified Columns

It is also possible to only add data in specific columns.

The following SQL statement will add a new row, but only add data in the "P_Id", "LastName" and the "FirstName" columns:

```
INSERT INTO Persons (P_Id, LastName, FirstName)
VALUES (5, 'Tjessem', 'Jakob')
```

The "Persons" table will now look like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Johan | Bakken 2 | Stavanger |
| 5 | Tjessem | Jakob | | |

# The UPDATE Statement

The UPDATE statement is used to update existing records in a table.

SQL UPDATE Syntax:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

**Note:** Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

# SQL UPDATE Example

The "Persons" table:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Johan | Bakken 2 | Stavanger |
| 5 | Tjessem | Jakob | | |

Now we want to update the person "Tjessem, Jakob" in the "Persons" table.
We use the following SQL statement:

```
UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'
WHERE LastName='Tjessem' AND FirstName='Jakob'
```

# SQL UPDATE Example

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Johan | Bakken 2 | Stavanger |
| 5 | Tjessem | Jakob | Nissestien 67 | Sandnes |

# SQL UPDATE Warning

Be careful when updating records. If we had omitted the WHERE clause in the example above, like this:

```
UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'
```

The "Persons" table would have looked like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Nissestien 67 | Sandnes |
| 2 | Svendson | Tove | Nissestien 67 | Sandnes |
| 3 | Pettersen | Kari | Nissestien 67 | Sandnes |
| 4 | Nilsen | Johan | Nissestien 67 | Sandnes |
| 5 | Tjessem | Jakob | Nissestien 67 | Sandnes |

# The DELETE Statemen

The DELETE statement is used to delete rows in a table.

SQL DELETE Syntax:

```
DELETE FROM table_name
WHERE some_column=some_value
```

**Note:** Notice the WHERE clause in the DELETE syntax. The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

# SQL DELETE Example

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Johan | Bakken 2 | Stavanger |
| 5 | Tjessem | Jakob | Nissestien 67 | Sandnes |

Now we want to delete the person "Tjessem, Jakob" in the "Persons" table.
We use the following SQL statement:

```
DELETE FROM Persons
WHERE LastName='Tjessem' AND FirstName='Jakob'
```

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Johan | Bakken 2 | Stavanger |

# Delete All Rows

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name
```

*or*

```
DELETE * FROM table_name
```

**Note:** Be very careful when deleting records. You cannot undo this statement!

# Test your SQL Skills

We will use the **Customers** table in the Northwind database:

| CompanyName | ContactName | Address | City |
|---|---|---|---|
| Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin |
| Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå |
| Centro comercial Moctezuma | Francisco Chang | Sierras de Granada 9993 | México D.F. |
| Ernst Handel | Roland Mendel | Kirchgasse 6 | Graz |
| FISSA Fabrica Inter. Salchichas S.A. | Diego Roel | C/ Moralzarzal, 86 | Madrid |
| Galería del gastrónomo | Eduardo Saavedra | Rambla de Cataluña, 23 | Barcelona |
| Island Trading | Helen Bennett | Garden House Crowther Way | Cowes |
| Königlich Essen | Philip Cramer | Maubelstr. 90 | Brandenburg |
| Laughing Bacchus Wine Cellars | Yoshi Tannamuri | 1900 Oak St. | Vancouver |
| Magazzini Alimentari Riuniti | Giovanni Rovelli | Via Ludovico il Moro 22 | Bergamo |
| North/South | Simon Crowther | South House 300 Queensbridge | London |
| Paris spécialités | Marie Bertrand | 265, boulevard Charonne | Paris |

# Test your SQL Skills

To see how SQL works, you can copy the SQL statements below and paste them into the textarea, or you can make your own SQL statements.

```
SELECT * FROM customers
```

```
SELECT CompanyName, ContactName FROM customers
```

```
SELECT * FROM customers WHERE companyname LIKE 'a%'
```

```
SELECT CompanyName, ContactName
FROM customers
WHERE CompanyName > 'a'
```

*When using SQL on text data, "alfred" is greater than "a" (like in a dictionary).*

```
SELECT CompanyName, ContactName
FROM customers
WHERE CompanyName > 'g'
AND ContactName > 'g'
```

```
SELECT * FROM customers
```

# The TOP Clause

The TOP clause is used to specify the number of records to return.

The TOP clause can be very useful on large tables with thousands of records. Returning a large number of records can impact on performance.

**Note:** Not all database systems support the TOP clause.

SQL Server Syntax:

```
SELECT TOP number|percent column_name(s)
FROM table_name
```

# The LIKE Operator

- The LIKE operator is used to search for a specified pattern in a column.

- SQL LIKE Syntax

- *SELECT column_name(s)*
  *FROM table_name*
  *WHERE column_name LIKE pattern*

# LIKE Operator Example

The "Persons" table:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Now we want to select the persons living in a city that starts with "s" from the table above.
We use the following SELECT statement:

*SELECT * FROM Persons*
*WHERE City LIKE 's%'*

The "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

The result-set will look like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Next, we want to select the persons living in a city that ends with an "s" from the "Persons" table.
We use the following SELECT statement:

*SELECT * FROM Persons*
*WHERE City LIKE '%s'*

The result-set will look like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

Next, we want to select the persons living in a city that contains the pattern "tav" from the "Persons" table.

We use the following SELECT statement:

*SELECT * FROM Persons*
*WHERE City LIKE '%tav%'*

The result-set will look like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

It is also possible to select the persons living in a city that NOT contains the pattern "tav" from the "Persons" table, by using the NOT keyword.

```
SELECT * FROM Persons
WHERE City NOT LIKE '%tav%'
```

The result-set will look like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

# SQL Wildcards

- SQL wildcards can be used when searching for data in a database.
- SQL wildcards can substitute for one or more characters when searching for data in a database.
- SQL wildcards must be used with the SQL LIKE operator.

**With SQL, the following wildcards can be used:**

| Wildcard | Description |
|---|---|
| % | A substitute for zero or more characters |
| _ | A substitute for exactly one character |
| [charlist] | Any single character in charlist |
| [^charlist]<br>or<br>[!charlist] | Any single character not in charlist |

# Using the % Wildcard

Now we want to select the persons living in a city that starts with "sa" from the "Persons" table.

We use the following SELECT statement:

*SELECT * FROM Persons*
*WHERE City LIKE 'sa%'*

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

Next, we want to select the persons living in a city that contains the pattern "nes" from the "Persons" table.

```
SELECT * FROM Persons
WHERE City LIKE '%nes%'
```

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

# Using the _ Wildcard

Now we want to select the persons with a first name that starts with any character, followed by "la" from the "Persons" table.

```
SELECT * FROM Persons
WHERE FirstName LIKE '_la'
```

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |

Next, we want to select the persons with a last name that starts with "S", followed by any character, followed by "end", followed by any character, followed by "on" from the "Persons" table.

```
SELECT * FROM Persons
WHERE LastName LIKE 'S_end_on'
```

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

# Using the [charlist] Wildcard

Now we want to select the persons with a last name that starts with "b" or "s" or "p" from the "Persons" table

*SELECT * FROM Persons*
*WHERE LastName LIKE '[bsp]%'*

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Next, we want to select the persons with a last name that do not start with "b" or "s" or "p" from the "Persons" table.

*SELECT * FROM Persons*
*WHERE LastName LIKE '[!bsp]%'*

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |

# SQL Joins

- SQL joins are used to query data from two or more tables, based on a relationship between certain columns in these tables.
- The JOIN keyword is used in an SQL statement to query data from two or more tables, based on a relationship between certain columns in these tables.
- Tables in a database are often related to each other with keys.
- A primary key is a column (or a combination of columns) with a unique value for each row. Each primary key value must be unique within the table. The purpose is to bind data together, across tables, without repeating all of the data in every table

# SQL JOIN

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

| O_Id | OrderNo | P_Id |
|---|---|---|
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 1 |
| 4 | 24562 | 1 |
| 5 | 34764 | 15 |

Note that the "O_Id" column is the primary key in the "Orders" table and that the "P_Id" column refers to the persons in the "Persons" table without using their names.
Notice that the relationship between the two tables above is the "P_Id" column.

# SQL JOIN

- **Different SQL JOINs**

- Before we continue with examples, we will list the types of JOIN you can use, and the differences between them.

- **JOIN**: Return rows when there is at least one match in both tables
- **LEFT JOIN**: Return all rows from the left table, even if there are no matches in the right table
- **RIGHT JOIN**: Return all rows from the right table, even if there are no matches in the left table
- **FULL JOIN**: Return rows when there is a match in one of the tables
-

# SQL INNER JOIN Keyword

- The INNER JOIN keyword return rows when there is at least one match in both tables.
- **SQL INNER JOIN Syntax**
- SELECT column_name(s)
  FROM table_name1
  INNER JOIN table_name2
  ON table_name1.column_name=table_name2.column_name
- **PS:** INNER JOIN is the same as JOIN**.**

# SQL INNER JOIN Example

The "Persons" table:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

The "Orders" table:

| O_Id | OrderNo | P_Id |
|---|---|---|
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 1 |
| 4 | 24562 | 1 |
| 5 | 34764 | 15 |

# SQL INNER JOIN Example

- Now we want to list all the persons with any orders.
- SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
  FROM Persons
  INNER JOIN Orders
  ON Persons.P_Id=Orders.P_Id
  ORDER BY Persons.LastName

| LastName | FirstName | OrderNo |
|---|---|---|
| Hansen | Ola | 22456 |
| Hansen | Ola | 24562 |
| Pettersen | Kari | 77895 |
| Pettersen | Kari | 44678 |

The INNER JOIN keyword return rows when there is at least one match in both tables. If there are rows in "Persons" that do not have matches in "Orders", those rows will NOT be listed.

# SQL LEFT JOIN Keyword

- The LEFT JOIN keyword returns all rows from the left table (table_name1), even if there are no matches in the right table (table_name2).
- SELECT column_name(s)
  FROM table_name1
  LEFT JOIN table_name2
  ON table_name1.column_name=table_name2.column_name

- **PS:** In some databases LEFT JOIN is called LEFT OUTER JOIN.

# SQL LEFT JOIN Example

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

| O_Id | OrderNo | P_Id |
|---|---|---|
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 1 |
| 4 | 24562 | 1 |
| 5 | 34764 | 15 |

Now we want to list all the persons and their orders - if any, from the tables above.

# SQL LEFT JOIN Example

- SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
  FROM Persons
  LEFT JOIN Orders
  ON Persons.P_Id=Orders.P_Id
  ORDER BY Persons.LastName

| LastName | FirstName | OrderNo |
|---|---|---|
| Hansen | Ola | 22456 |
| Hansen | Ola | 24562 |
| Pettersen | Kari | 77895 |
| Pettersen | Kari | 44678 |
| Svendson | Tove | |

The LEFT JOIN keyword returns all the rows from the left table (Persons), even if there are no matches in the right table (Orders).

# SQL RIGHT JOIN Keyword

- The RIGHT JOIN keyword returns all the rows from the right table (table_name2), even if there are no matches in the left table (table_name1).

- SELECT column_name(s)
  FROM table_name1
  RIGHT JOIN table_name2
  ON table_name1.column_name=table_name2.column_name

- **PS:** In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

# SQL RIGHT JOIN Example

The "Persons" table:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

The "Orders" table:

| O_Id | OrderNo | P_Id |
|---|---|---|
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 1 |
| 4 | 24562 | 1 |
| 5 | 34764 | 15 |

# SQL RIGHT JOIN Example

- Now we want to list all the orders with containing persons - if any, from the tables above.
- SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
  FROM Persons
  RIGHT JOIN Orders
  ON Persons.P_Id=Orders.P_Id
  ORDER BY Persons.LastName

| LastName | FirstName | OrderNo |
|---|---|---|
| Hansen | Ola | 22456 |
| Hansen | Ola | 24562 |
| Pettersen | Kari | 77895 |
| Pettersen | Kari | 44678 |
| | | 34764 |

The RIGHT JOIN keyword returns all the rows from the right table (Orders), even if there are no matches in the left table (Persons).

# SQL FULL JOIN Keyword

- The FULL JOIN keyword return rows when there is a match in one of the tables.
- SELECT column_name(s)
  FROM table_name1
  FULL JOIN table_name2
  ON table_name1.column_name=table_name2.column_name
- Now we want to list all the persons and their orders, and all the orders with their persons.
- SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
  FROM Persons
  FULL JOIN Orders
  ON Persons.P_Id=Orders.P_Id
  ORDER BY Persons.LastName

| LastName | FirstName | OrderNo |
|---|---|---|
| Hansen | Ola | 22456 |
| Hansen | Ola | 24562 |
| Pettersen | Kari | 77895 |
| Pettersen | Kari | 44678 |
| Svendson | Tove | |
| | | 34764 |

The FULL JOIN keyword returns all the rows from the left table (Persons), and all the rows from the right table (Orders). If there are rows in "Persons" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Persons", those rows will be listed as well.

# LENGUAJE SQL. EJERCICIOS

Seleccionar todo de la tabla asignatura

- *SELECT * FROM Asignatura;*

Seleccionar apellidos y nombre de la tabla alumno

- *SELECT apellidos, nombre FROM Alumno;*

Seleccionar apellidos y nombre de la tabla alumno, pero ordenar por apellido

- *SELECT apellidos, nombre FROM Alumno ORDER BY apellidos;*

# LENGUAJE SQL. EJERCICIOS

Seleccionar el número de alumnos de la tabla Alumno

- *SELECT Count(*) AS Numero FROM Alumno;*

Seleccionar el mayor número de matrícula de todos los alumnos

- *SELECT Max(numMatricula) FROM Alumno;*

Seleccionar el número medio de créditos de las asignaturas de la tabla Asignatura

- *SELECT Avg(Creditos) FROM Asignatura;*

# LENGUAJE SQL. EJERCICIOS

Seleccionar el número mínimo de créditos de las asignatura

- *SELECT Min(Creditos) AS Minimo FROM Asignatura;*

Seleccionar los distintos valores de créditos que tienen las asignaturas

- *SELECT Distinct(Creditos) FROM Asignatura;*

Seleccionar los distintos valores de créditos que tienen las asignaturas

- *SELECT Distinct(Creditos) FROM Asignatura ORDER BY Creditos;*

# LENGUAJE SQL. EJERCICIOS

Seleccionar todo de las asignaturas con más de 6 créditos (ordenar por créditos)

- *SELECT * FROM Asignatura WHERE creditos > 6 ORDER BY Creditos;*
-

# LENGUAJE SQL. EJERCICIOS

- 

Seleccionar los apellidos y nombre de aquellos alumnos cuyo número de matrícula esté comprendido entre6200 y 6500

- *SELECT Alumno.Apellidos, Alumno.nombre FROM Alumno*
- *WHERE numMatricula BETWEEN 6200 AND 6500*
- 

# LENGUAJE SQL. EJERCICIOS

Seleccionar los apellidos y el nombre de aquellos alumnos cuyos números de matrícula sean 6602, 6603, 6605 ó 6607

- *SELECT Alumno.Apellidos, Alumno.nombre FROM Alumno*
- *WHERE numMatricula IN (6602, 6603, 6605 ,6607)*

# LENGUAJE SQL. EJERCICIOS

Seleccionar el nombre de cada asignatura junto con el nombre de su Unidad Docente

- *SELECT asignatura.nombre, unidadDocente.nombre FROM Asignatura, UnidadDocente WHERE Asignatura.UnidadDocente = UnidadDocente.id;*
-

# LENGUAJE SQL. EJERCICIOS

Seleccionar el nombre de cada asignatura junto con el nombre de su Unidad Docente, ordenar por el nombre de la unidad docente

- *SELECT asignatura.nombre, unidadDocente.nombre FROM Asignatura, UnidadDocente WHERE Asignatura.UnidadDocente = UnidadDocente.id ORDER BY UnidadDocente.Nombre;*
-

# LENGUAJE SQL. EJERCICIOS

Seleccionar el nombre las unidades docentes que tengan asignaturas de más de 10 créditos

- *SELECT Distinct(unidadDocente.nombre) FROM Asignatura, UnidadDocente WHERE Asignatura.UnidadDocente = UnidadDocente.id AND Asignatura.creditos > 10;*

# LENGUAJE SQL. EJERCICIOS

Seleccionar los apellidos de cada alumno y las asignaturas en las que está matriculado

- *SELECT Apellidos, Alumno.nombre, Asignatura.nombre FROM Alumno, Asignatura, AlumnoAsignatura WHERE Alumno.numMatricula= AlumnoAsignatura.numMatricula AND AlumnoAsignatura.CodAsignatura = Asignatura.PlanCodigo*
-

# LENGUAJE SQL. EJERCICIOS

Seleccionar el número de matrícula, los apellidos y el nombre de los alumnos matriculados en la asignatura de informática

- *SELECT Alumno.NumMatricula, Apellidos, Alumno.nombre FROM Alumno, Asignatura, AlumnoAsignatura WHERE Alumno.numMatricula= AlumnoAsignatura.numMatricula AND AlumnoAsignatura.CodAsignatura = Asignatura.PlanCodigo AND Asignatura.nombre = 'Informática'*

# LENGUAJE SQL. EJERCICIOS

Seleccionar los apellidos de cada alumno y las asignaturas en las que está matriculado, pero solamente para aquellos alumnos cuyo apellido empiece por C

- *SELECT Apellidos, Alumno.nombre, asignatura.nombre FROM Alumno, Asignatura, AlumnoAsignatura WHERE alumno.numMatricula= alumnoAsignatura.numMatricula AND alumnoAsignatura.CodAsignatura = asignatura.PlanCodigo AND Alumno.Apellidos LIKE 'C*'*
-

# LENGUAJE SQL. EJERCICIOS

Seleccionar los apellidos y el nombre de cada alumno junto con el número de asignaturas en las que está matriculado

- *SELECT Alumno.Apellidos, Alumno.nombre, Count(*) AS Numero*
- *FROM Alumno, Asignatura, AlumnoAsignatura*
- *WHERE Alumno.numMatricula=alumnoAsignatura. numMatricula AND AlumnoAsignatura.CodAsignatura=asignatura. PlanCodigo*
- *GROUP BY Alumno.Apellidos, Alumno.nombre;*
-

# LENGUAJE SQL. EJERCICIOS

Seleccionar los apellidos y el nombre de cada alumno junto con el número de asignaturas y el número total de créditos en los que está matriculado

- *SELECT Alumno.Apellidos, Alumno.nombre, Count(*), Sum(Asignatura.creditos)*
- *FROM Alumno, Asignatura, AlumnoAsignatura*
- *WHERE Alumno.numMatricula=alumnoAsignatura.numMatricula AND AlumnoAsignatura.CodAsignatura = asignatura.PlanCodigo*
- *GROUP BY Alumno.Apellidos, Alumno.nombre;*
-

# LENGUAJE SQL. EJERCICIOS

Seleccionar todas las asignaturas que imparta la misma Unidad Docente que imparte Topografía I

- *SELECT Asignatura.nombre FROM Asignatura WHERE UnidadDocente = (SELECT UnidadDocente from Asignatura WHERE nombre = 'Topografía I')*
- *(o también…)*
- *SELECT Asignatura.nombre FROM Asignatura WHERE UnidadDocente IN (SELECT UnidadDocente from Asignatura WHERE nombre = 'Topografía I')*

# LENGUAJE SQL. EJERCICIOS

Los apellidos y el nombre de todos los alumnos que comparten alguna asignatura con el alumno cuyo número de matrícula es 6605

- *SELECT DISTINCT (Alumno.Apellidos) AS Expr1, Alumno.nombre*
- *FROM Alumno, Asignatura, AlumnoAsignatura*
- *WHERE Alumno.numMatricula = alumnoAsignatura.numMatricula AND AlumnoAsignatura.CodAsignatura = asignatura.PlanCodigo*
- *AND Asignatura.nombre IN (SELECT Asignatura.nombre*
- *FROM Alumno, Asignatura, AlumnoAsignatura*
- *WHERE Alumno.numMatricula=alumnoAsignatura.numMatricula AND AlumnoAsignatura.CodAsignatura = asignatura.PlanCodigo*
- *AND Alumno.NumMatricula = 6605);*

# LENGUAJE SQL. EJERCICIOS

Seleccionar aquellas unidades docentes que imparten más de 5 asignaturas

- *SELECT unidadDocente.nombre, UnidadDocente.Id FROM Asignatura, UnidadDocente WHERE Asignatura.UnidadDocente = UnidadDocente.id GROUP BY UnidadDocente.nombre,UnidadDocente.Id*
- *HAVING COUNT(asignatura.nombre) >5;*

# LENGUAJE SQL. EJERCICIOS

Seleccionar aquellos alumnos que están matriculados en más de 30 créditos

- *SELECT Alumno.Apellidos, Alumno.nombre, Count(*) AS N, Sum(Asignatura.creditos)*
- *FROM Alumno, Asignatura, AlumnoAsignatura*
- *WHERE Alumno.numMatricula=alumnoAsignatura.numMatricula AND AlumnoAsignatura.CodAsignatura = asignatura.PlanCodigo*
- *GROUP BY Alumno.Apellidos, Alumno.nombre*
- *HAVING SUM(Asignatura.Creditos) > 30*

# LENGUAJE SQL. EJERCICIOS

Seleccionar aquellos alumnos que están matriculados en más de 6 asignaturas

- *SELECT Alumno.Apellidos, Alumno.nombre, Count(*) AS N, Sum(Asignatura.creditos)*
- *FROM Alumno, Asignatura, AlumnoAsignatura*
- *WHERE Alumno.numMatricula=alumnoAsignatura.numMatricula AND AlumnoAsignatura.CodAsignatura = asignatura.PlanCodigo*
- *GROUP BY Alumno.Apellidos, Alumno.nombre*
- *HAVING COUNT(asignatura.nombre) > 6*

# About SQL Server

- Microsoft SQL Server is a Relational Database Management System (RDBMS) designed to run on platforms ranging from laptops to large multiprocessor servers. SQL Server is commonly used as the backend system for websites and corporate CRMs and can support thousands of concurrent users.

SQL Server comes with a number of tools to help you with your database administration and programming tasks.

- SQL Server is much more robust and scalable than a desktop database management system such as Microsoft Access.

- Anyone who has ever tried using Access as a backend to a website will probably be familiar with the errors that were generated when too many users tried to access the database!

- Although SQL Server can also be run as a desktop database system, it is most commonly used as a server database system.

- Server based database systems are designed to run on a central server, so that multiple users can access the same data simultaneously.

- The users normally access the database through an application.

- For example, a website could store all its content in a database. Whenever a visitor views an article, they are retrieving data from the database. As you know, websites aren't normally limited to just one user. So, at any given moment, a website could be serving up hundreds, or even thousands of articles to its website visitors. At the same time, other users could be updating their personal profile in the members' area, or subscribing to a newsletter, or anything else that website users do.

- Generally, it's the application that provides the functionality to these visitors. It is the database that stores the data and makes it available. Having said that, SQL Server does include some useful features that can assist the application in providing its functionality.

# SQL Server Edition in this Tutorial

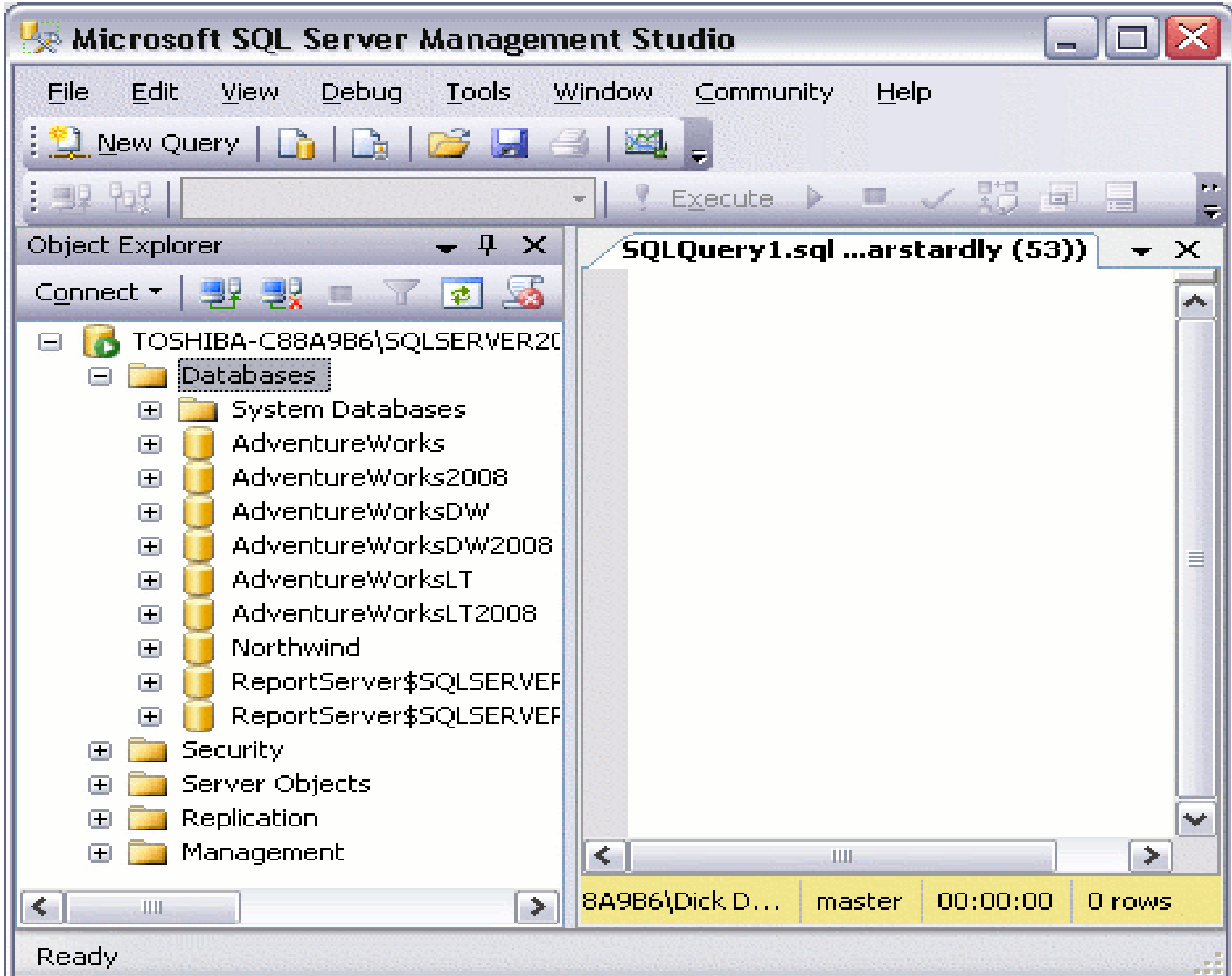
- The examples in this tutorial were made using the (free) Express edition of SQL Server 2008.

- SQL Server database engine - create, store, update and retrieve your data
- SQL Server Management Studio Basic - visual database management tool for creating, editing and managing databases
- Full-text Search - powerful, high-speed engine for searching text-intensive data
- Reporting Services - integrated report creation and design environment to create reports

# SQL Server Management Studio (**SSMS**) is the main administration console for SQL Server.

- SSMS enables you to create database objects (such as databases, tables, views etc), view the data within your database, you can configure user accounts, transfer data between databases, and more.

- Here's what SQL Server Management Studio looks like when you first open it up:



- The left pane contains the **Object Explorer**. The Object Explorer provides navigation to databases, server objects (such as triggers), log files, and more.

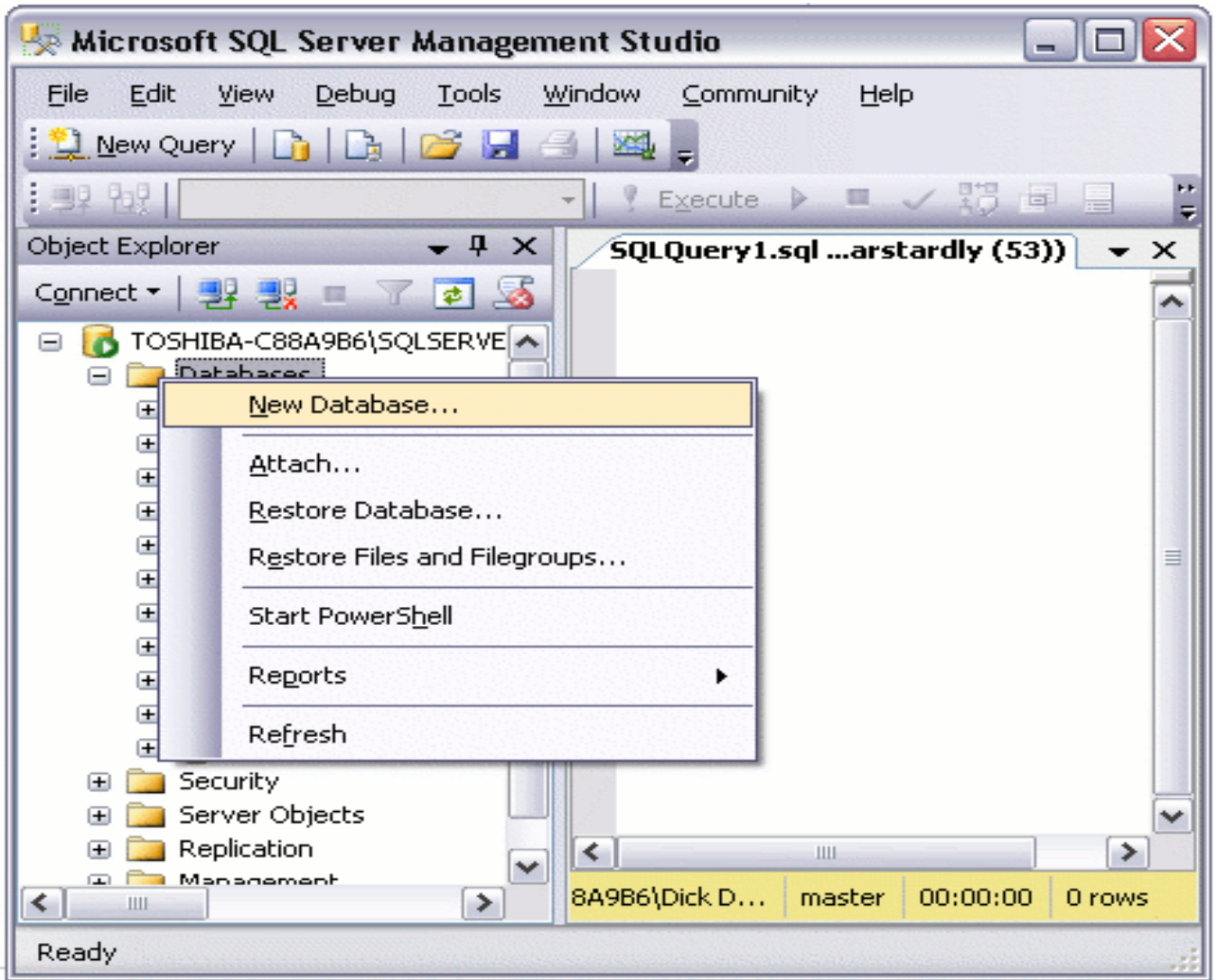
- The right pane allows you **to write queries** against the database and view the results. In this screenshot I have opened a blank query by clicking the "New Query" button. You can also bring up other windows, such as the Properties window.

# System Databases

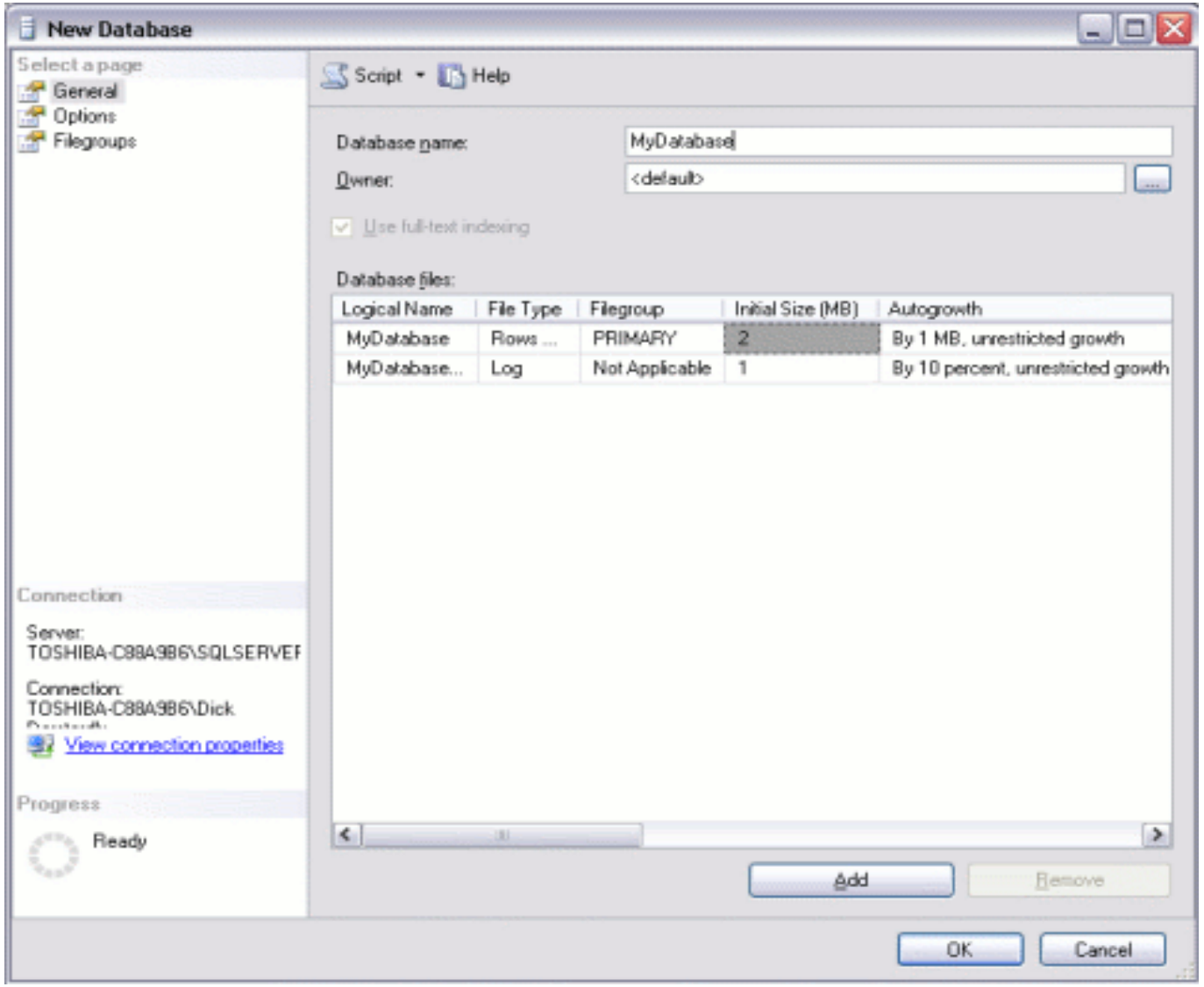
| Database | Type | Description |
|---|---|---|
| master | System database | Stores system level information such as user accounts, configuration sett on all other databases. |
| model | System database | This database is used as a template for all other databases that are create |
| msdb | System database | Used by the SQL Server Agent for configuring alerts and scheduled jobs |
| tempdb | System database | Holds all temporary tables, temporary stored procedures, and any other t storage requirements generated by SQL Server. |

# Creating a New Database

- Right click on the "Databases" icon and select "New Database...":



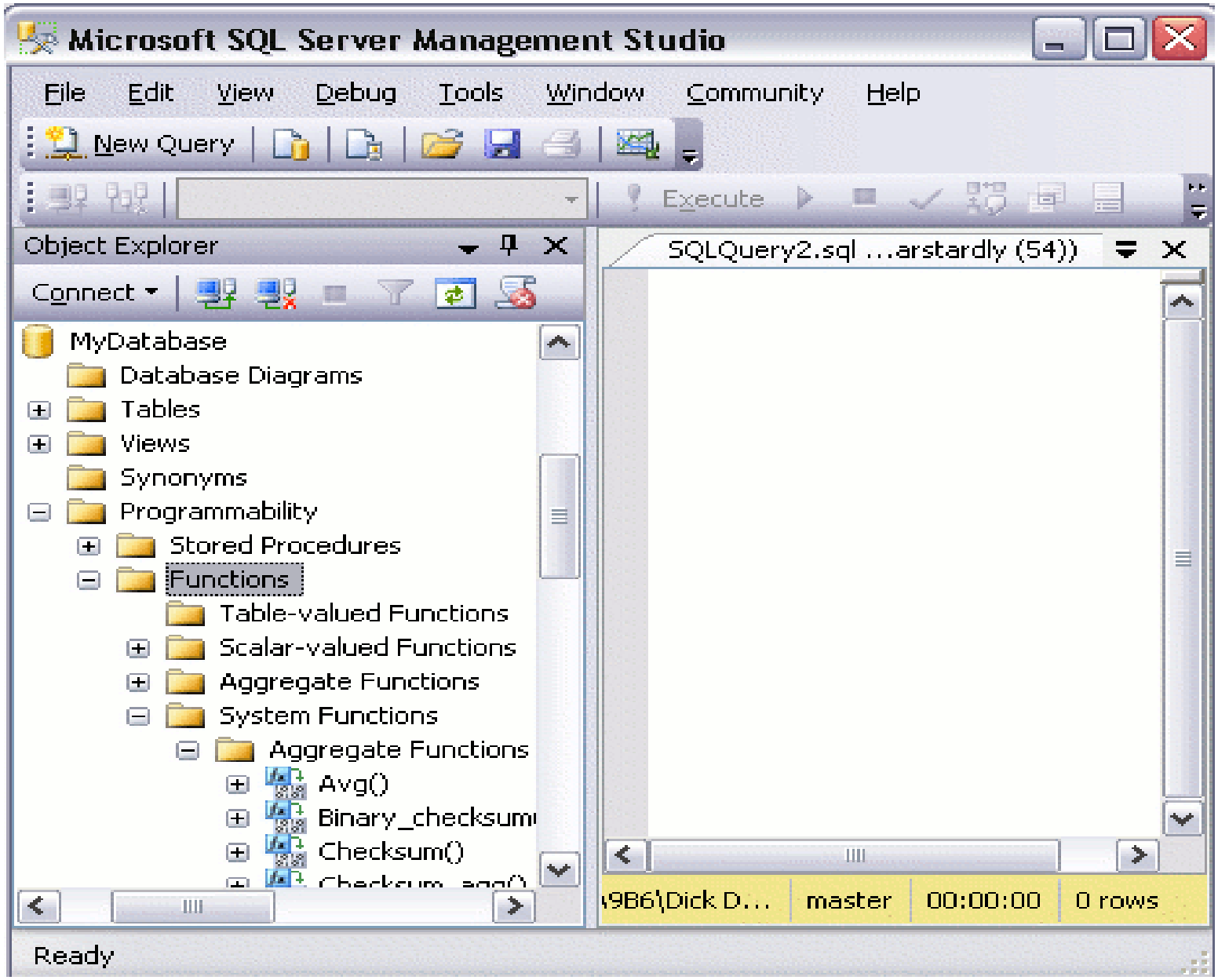
- Name your database and click "OK":



# Your New Database

- You will now notice your new database appears under the "Databases" section of SQL Server Management Studio.

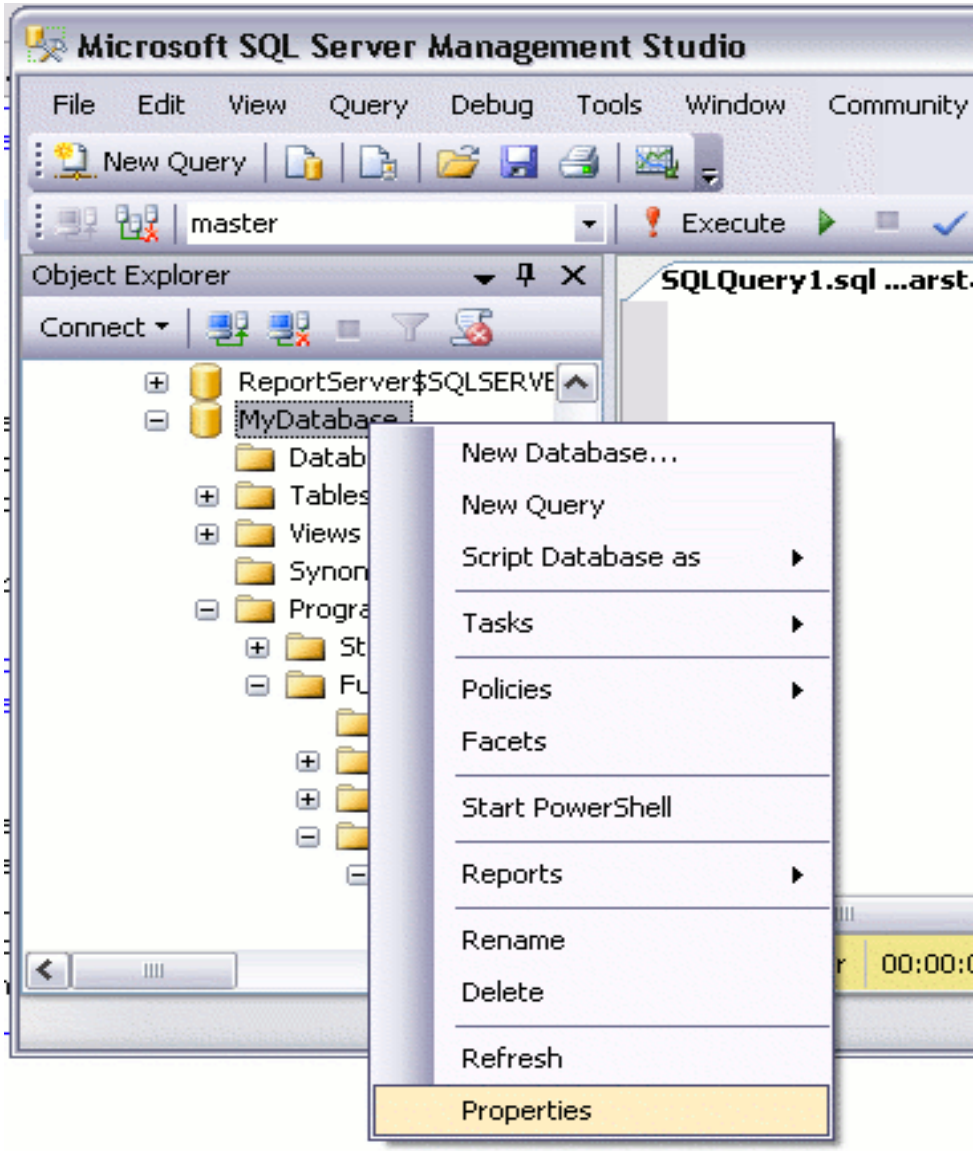
- Your new database is based on the "Model" database. The Model database is a system database which is used as a template whenever a new database is created. If you use the left pane to navigate to your database and expand the tree, you will notice that your database already contains a number of objects. For example, it already contains system functions, system views, system stored procedures, and (hidden) system tables. These are system objects which provide information about the database.

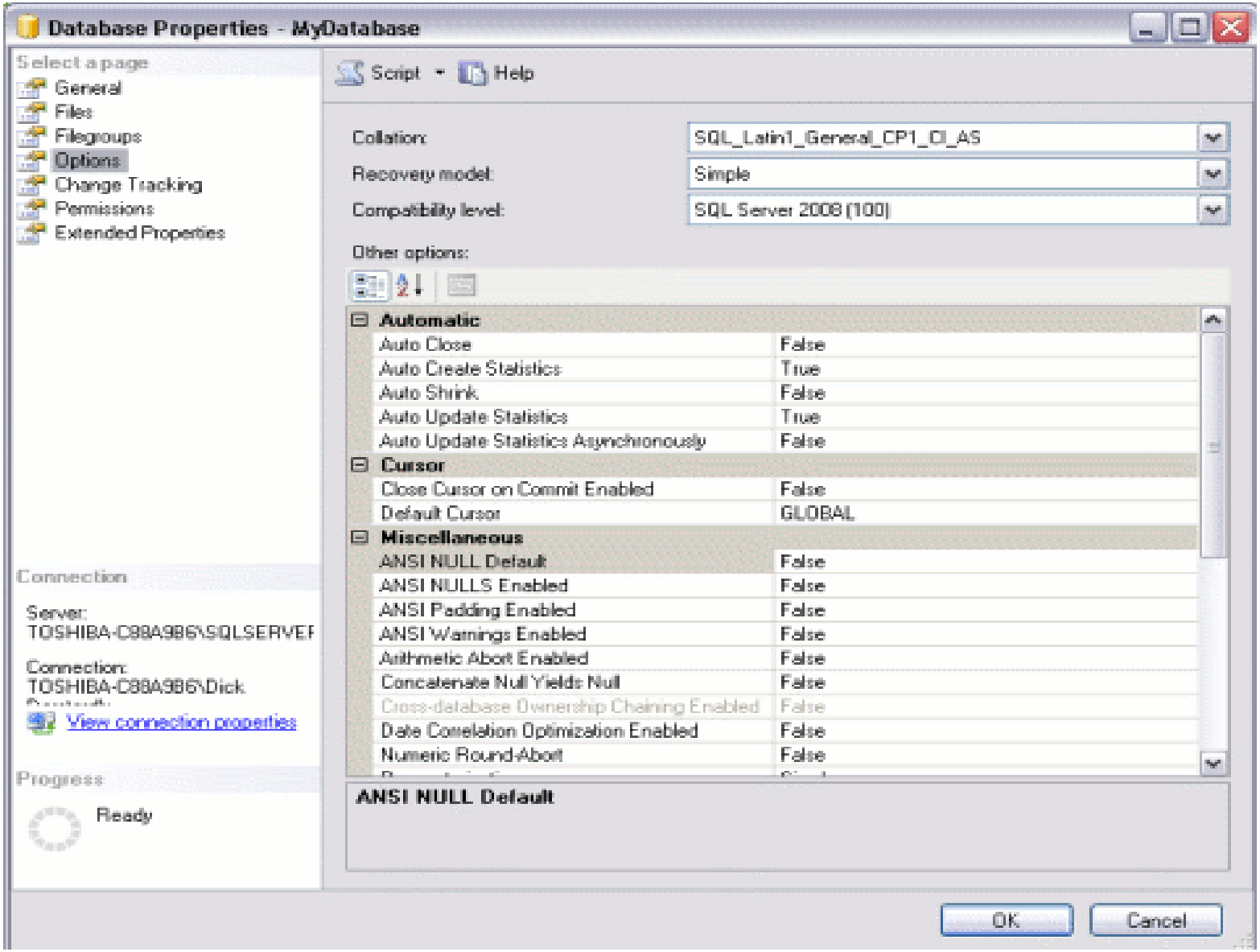

Microsoft SQL Server Management Studio
File
Edit
View
Debug
Tools
Window
Community
Help
New Query
Execute
Object Explorer
Connect
SQLQuery2.sql ...arstardly (54))
MyDatabase
Database Diagrams
Tables
Views
Synonyms
Programmability
Stored Procedures
Functions
Table-valued Functions
Scalar-valued Functions
Aggregate Functions
System Functions
Aggregate Functions
Avg()
Checksum()
master
00:00:00
0 rows
Ready

# Other Options

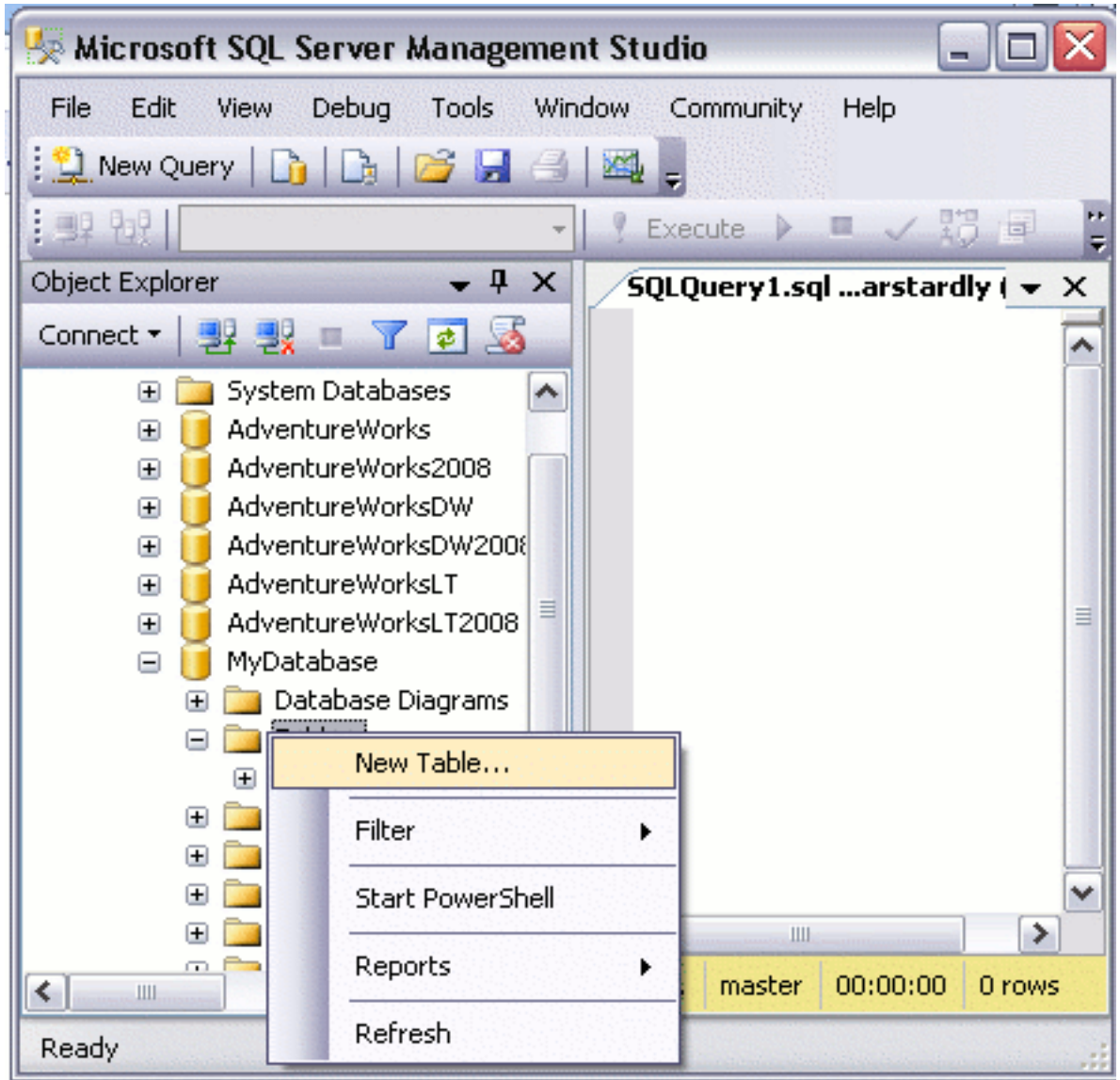
- We have just created a database using the default options. When we created the database, a "Data File" and a "Transaction Log" were created. They were created in the default location for our server.
- If we'd wanted to, we could have specified a different location for these files. We also could have changed specifications on whether to allow the file to grow automatically (as it stores more and more data), and if so, how that growth should be managed. We could have done that at step 2. But all is not lost. We can still do it now that we've created the database. We can do it via the Properties dialog box.

- To view or change the database properties, simply right click on the database and select "Properties":

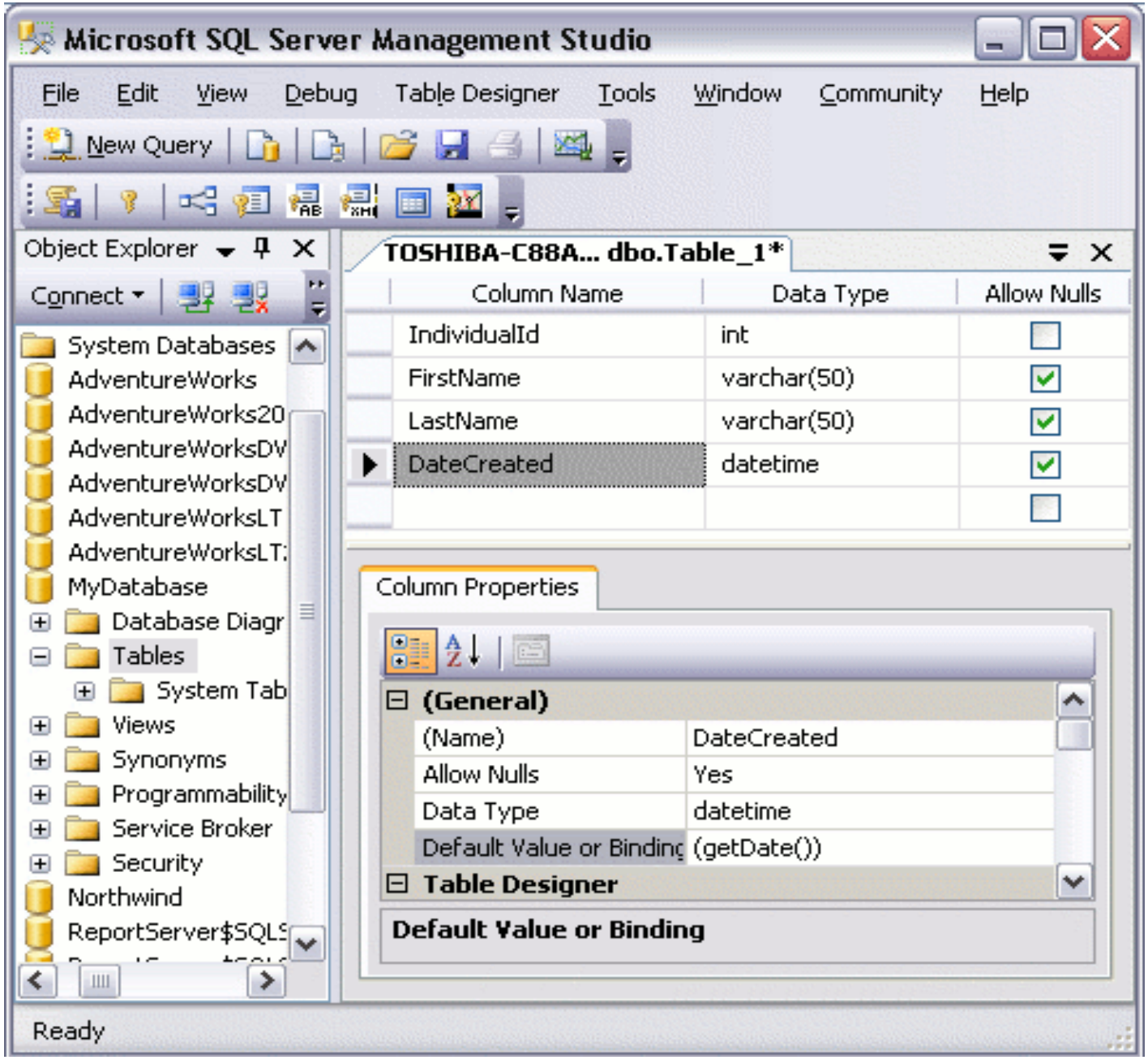



Database Properties - MyDatabase
Select a page
General
Files
Filegroups
Options
Change Tracking
Permissions
Extended Properties
Script
Help
Collation:
SQL_Latin1_General_CP1_CI_AS
Recovery model:
Simple
Compatibility level:
SQL Server 2008 (100)
Other options:
Automatic
Auto Close
False
Auto Create Statistics
True
Auto Shrink
False
Auto Update Statistics
True
Auto Update Statistics Asynchronously
False
Cursor
Close Cursor on Commit Enabled
False
Default Cursor
GLOBAL
Miscellaneous
ANSI NULL Default
False
ANSI NULLS Enabled
False
ANSI Padding Enabled
False
ANSI Warnings Enabled
False
Arithmetic Abort Enabled
False
Concatenate Null Yields Null
False
Cross-database Ownership Chaining Enabled
False
Date Correlation Optimization Enabled
False
Numeric Round-Abort
False
ANSI NULL Default
Connection
Server:
TOSHIBA-C88A9B6\SQLSERVEF
Connection:
TOSHIBA-C88A9B6\Dick
View connection properties
Progress
Ready
OK
Cancel

- This lesson demonstrates how to create a table in a SQL Server database using SQL Server Management Studio (SSMS).
- Ensuring you have the right database expanded, right click on the "Tables" icon and select "New Table...":

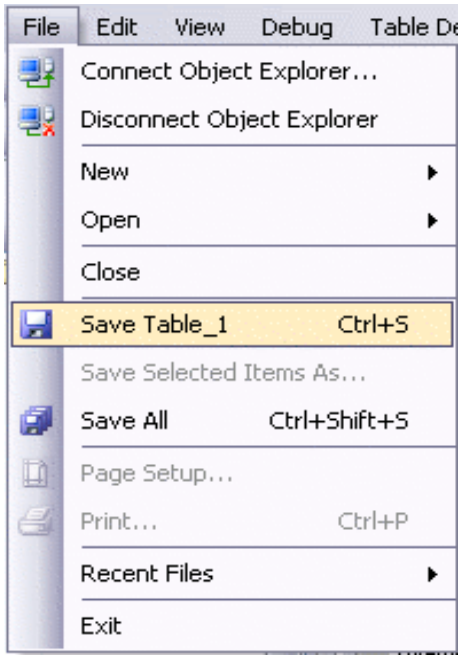


- While you have this screen open, do the following:
  - Using the values in the screenshot, complete the details in the "Column Name" column, the "Data Type" column, "Length" column, and "Allow Nulls" column.
  - Make the Individual Id column an "identity column", by setting "Is Identity" to "Yes" (this option is under the "Identity Specification" section in the bottom pane). This column is going to be an auto-number column - it will contain an incrementing number for each record that is created. **AUTONUMÉRICO**
  - Set the "Default Value" of the DateCreated column to *(getdate())*. (This will automatically insert the current date into that field for each new record).

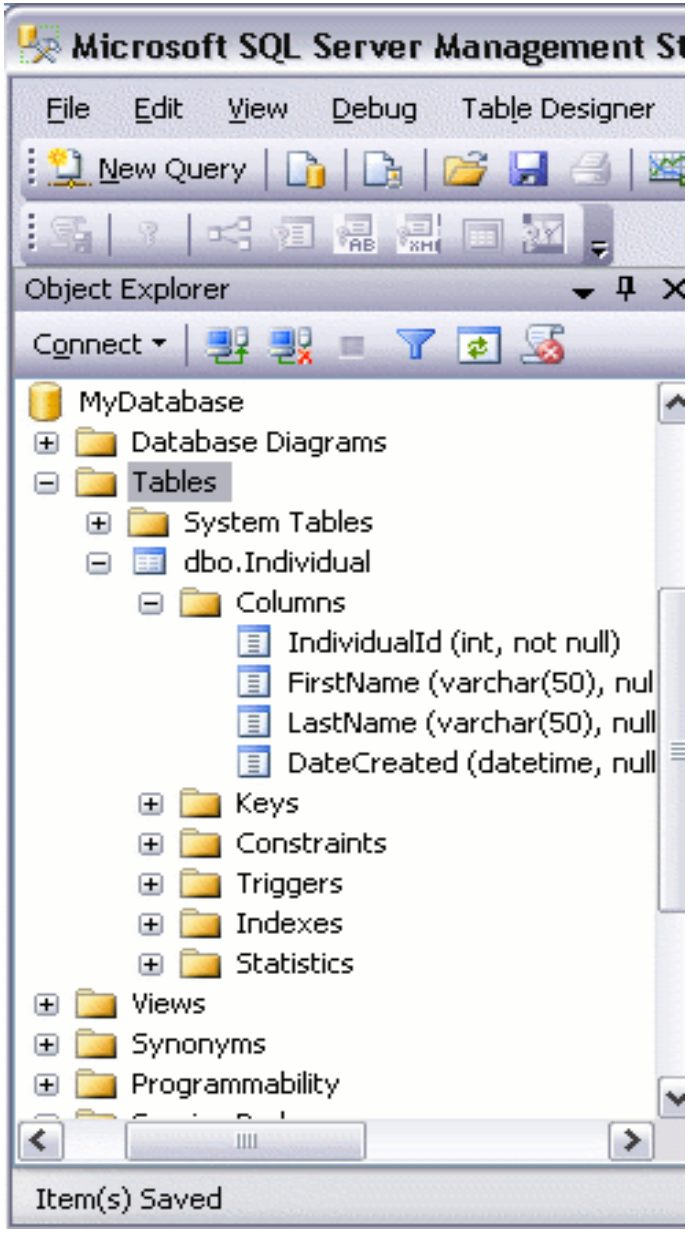

Microsoft SQL Server Management Studio
File Edit View Debug Table Designer Tools Window Community Help
New Query
Object Explorer
Connect
System Databases
AdventureWorks
AdventureWorks20
AdventureWorksDV
AdventureWorksDV
AdventureWorksLT
AdventureWorksLT
MyDatabase
Database Diagr
Tables
System Tab
Views
Synonyms
Programmability
Service Broker
Security
Northwind
ReportServer$SQL
TOSHIBA-C88A... dbo.Table_1*
Column Name
Data Type
Allow Nulls
IndividualId
int
FirstName
varchar(50)
LastName
varchar(50)
DateCreated
datetime
Column Properties
(General)
(Name)
DateCreated
Allow Nulls
Yes
Data Type
datetime
Default Value or Bindinç
(getDate())
Table Designer
Default Value or Binding
Ready

- What we are doing at this stage, is creating the column names, specifying the type of data that can be entered into them, and setting default values.
- **Restricting the data type** for each column is very important and helps maintain data integrity. For example, it can prevent us from accidentally entering an email address into a field for storing the current date.

Save the table by selecting *File* > *Save Table_1*:

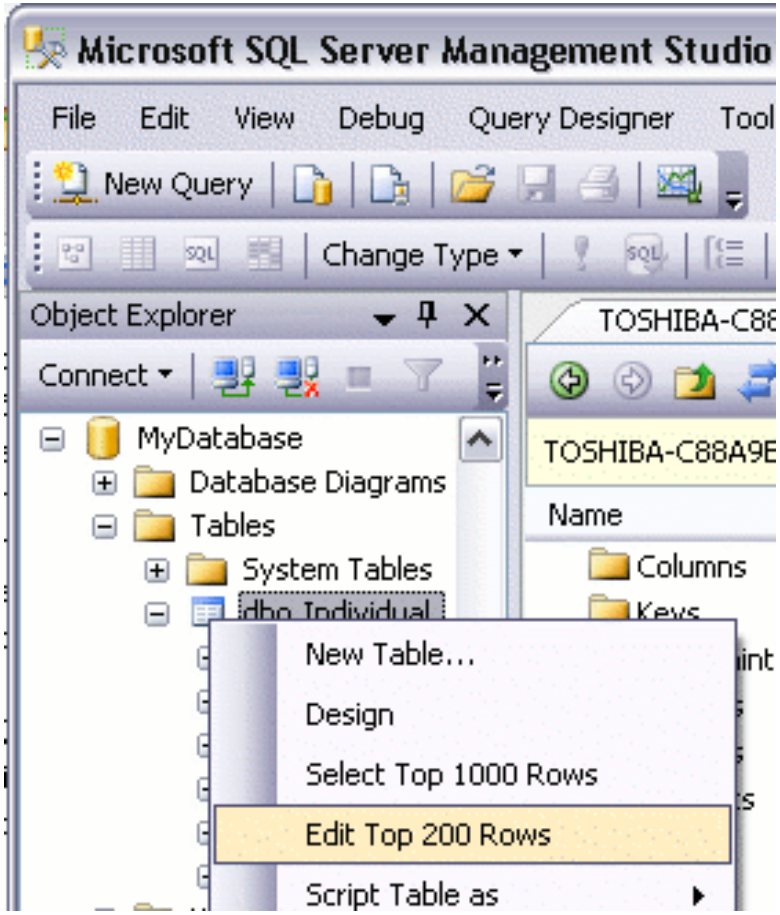



When prompted, name your table:


Microsoft SQL Server Management St
File
Edit
View
Debug
Table Designer
New Query
Object Explorer
Connect
MyDatabase
Database Diagrams
Tables
System Tables
dbo.Individual
Columns
IndividualId (int, not null)
FirstName (varchar(50), nul
LastName (varchar(50), null
DateCreated (datetime, null
Keys
Constraints
Triggers
Indexes
Statistics
Views
Synonyms
Programmability
Item(s) Saved

# Editing Table Rows

- We can use the "Edit Top 200 Rows" option to add data to our table.
- To use this option, right click on the table you wish to open, and select "Edit Top 200 Rows":



- You can now start entering the data directly into your table.
- Note that you don't need to enter data into the IndividualId and DateCreated columns. This is because the they will be populated automatically (remember, we set IndividualId to "Is Identity" and DateCreated to "GetDate()"))

| | IndividualId | FirstName | LastName | DateCreated |
|---|---|---|---|---|
| | 1 | Homer | Simpson | 2009-04-27 11:3... |
| ! | NULL | Barney | Rubble | NULL |
| ! | NULL | Ozzy | Osbourne | NULL |
| ✎ | NULL | Fred ! | Flinsto | NULL |
| * | NULL | NULL | NULL | NULL |

## Disadvantages of Entering Data Directly to your Table

- The above method is fine if you only have a small amount of data to enter or update. If you have a lot of data to enter, this could become very tedious. Also, if you have multiple environments (for example, a development environment, staging environment, and production environment), with duplicate databases configured in each environment, you will need to re-enter the same data into each environment.

- When you're first learning SQL Server, this may not be a major concern. However, in an environment such as described, entering data directly into the table becomes quite inefficient.

# A Better Method - SQL Scripts

- In most cases, you will probably find it more efficient to write a *SQL script*. Using a script enables you to re-enter the data as many times as you like. If you need to rebuild your database for any reason, you can simply run your script to enter the data. If you have multiple environments, once again you can run your script against each environment.

- Once you get used to writing and running scripts, you will probably find it quicker than entering data directly into the table.

# Database Driven Websites

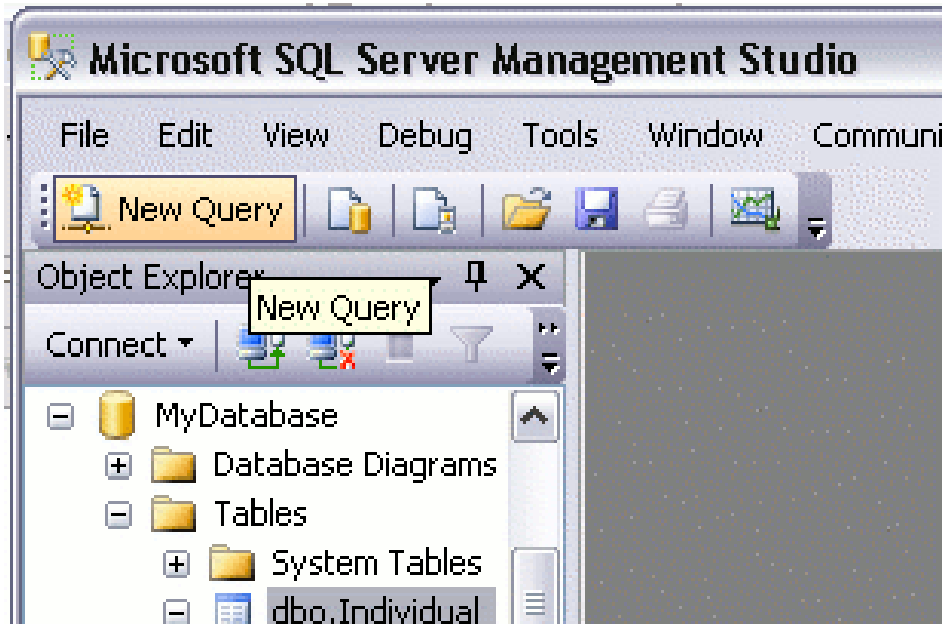
When you create a database driven website, you actually write SQL scripts to insert data into the database.

You also use SQL scripts to read data, update data, and delete data.

These SQL scripts can be placed directly into your website code (PHP, ColdFusion etc), or they can be stored in the database as *Stored Procedures* or *Views*.

# Create a New Query

- Before we generate our SQL script, we need somewhere to enter it into. This part is easy. Just click the "New Query" button:




- A blank, white sheet should now appear on the right pane

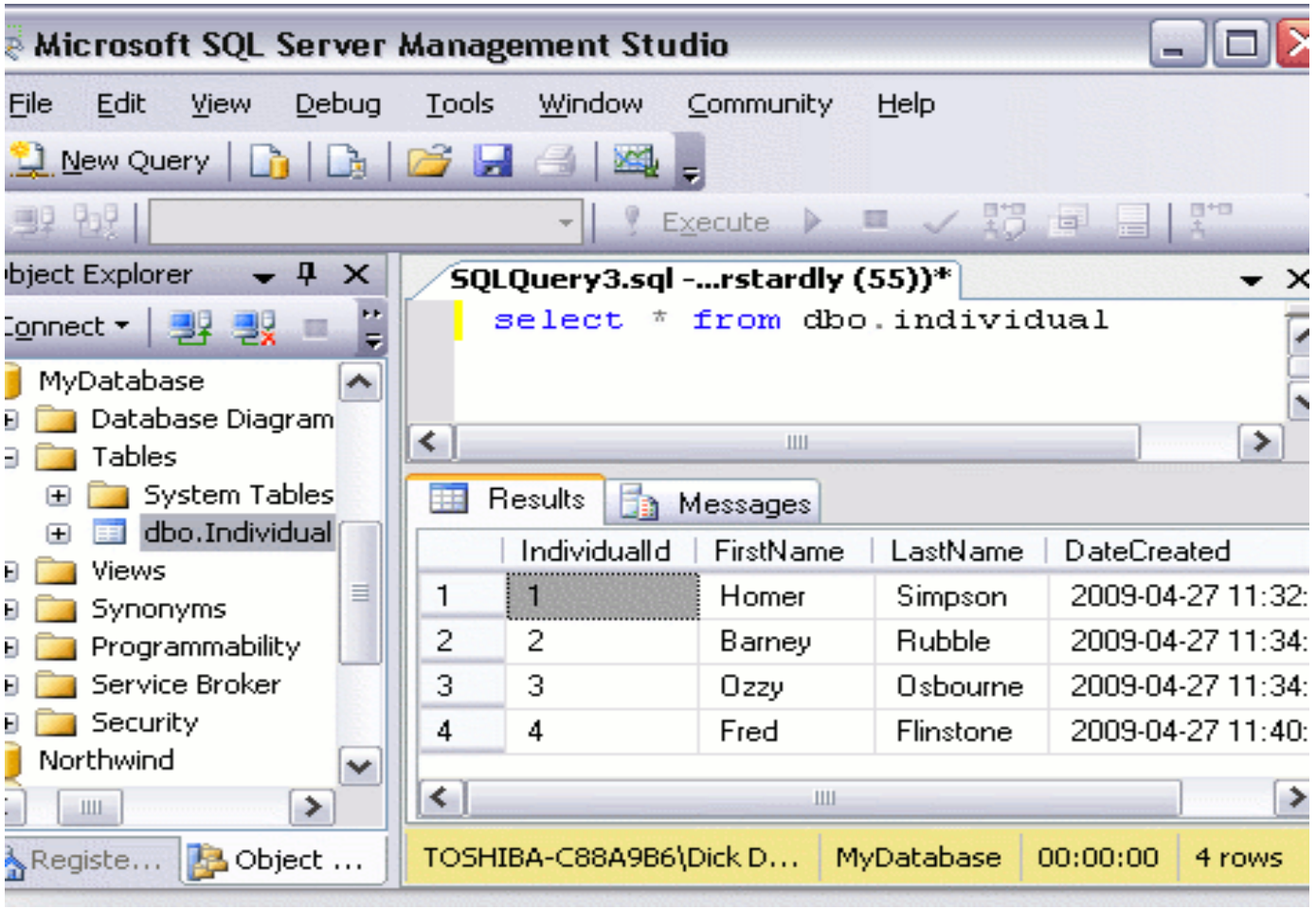
# Write/Run Your SQL Script

- You are now ready to write SQL queries against your database. You can use this interface to create database objects (such as databases, tables, views etc), insert data into a database table, select data, update data, delete data.

- To run an SQL query:

- 1. Type your query into the workspace on the right pane
- 2.- Click "Execute" (you can also press F5)

- The following screenshot shows an example of using a SQL 'select' statement to select data from a database:




- As you can see, the results of the query are displayed in the bottom pane.

# Database Administration Tasks

- Most of the database administration tasks that can be performed in **SSMS** via the graphical user interface can be performed programmatically via SQL scripts.
- This tutorial concentrates on using the graphical user interface, mainly because it's usually a lot easier for new users to get their head around.
- Once you become more familiar with SQL Server, you may find yourself using SQL scripts to perform many of the tasks that you started out doing via the graphical user interface.

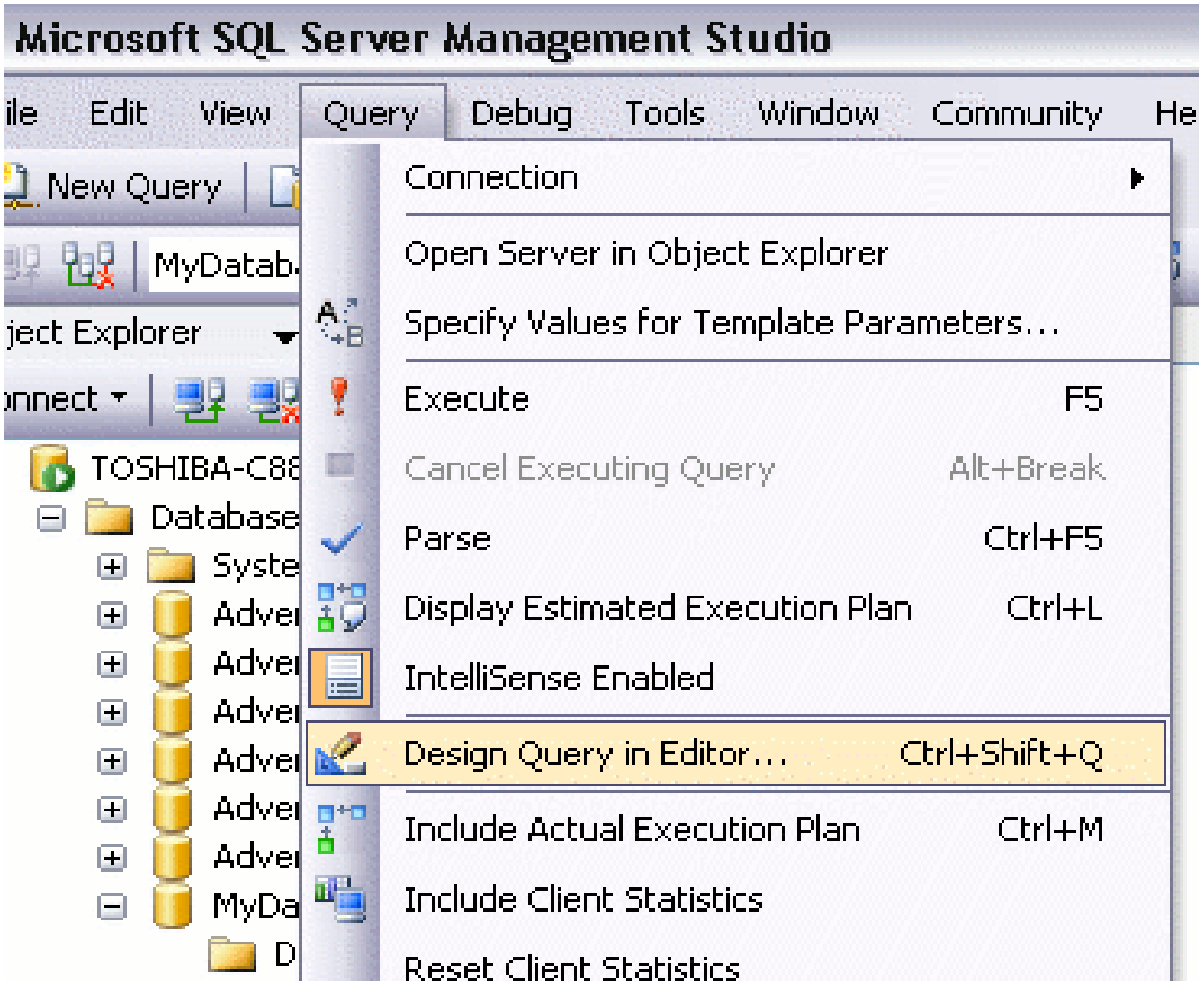
# SQL Server - Query Designer

- The graphical query designer is a graphical user interface that allows you to build queries to run against your SQL Server database. This can be particularly useful when building complex queries that involves many tables, views etc.

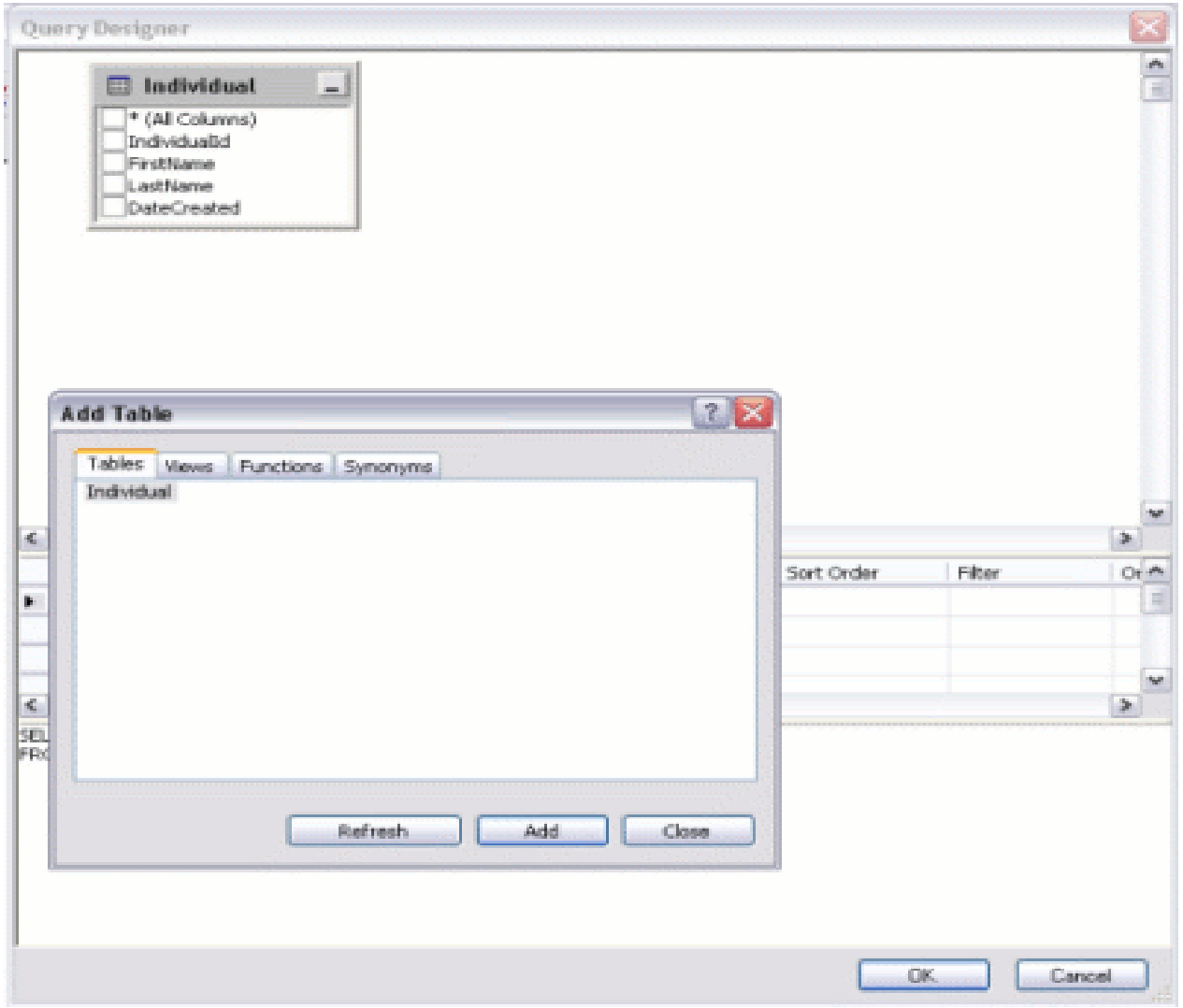
- The query designer can also be beneficial for those who are learning how to write SQL. This is because you don't need to remember the SQL syntax in order to write queries against your database - the query designer generates the SQL for you.

# Building Your Queries

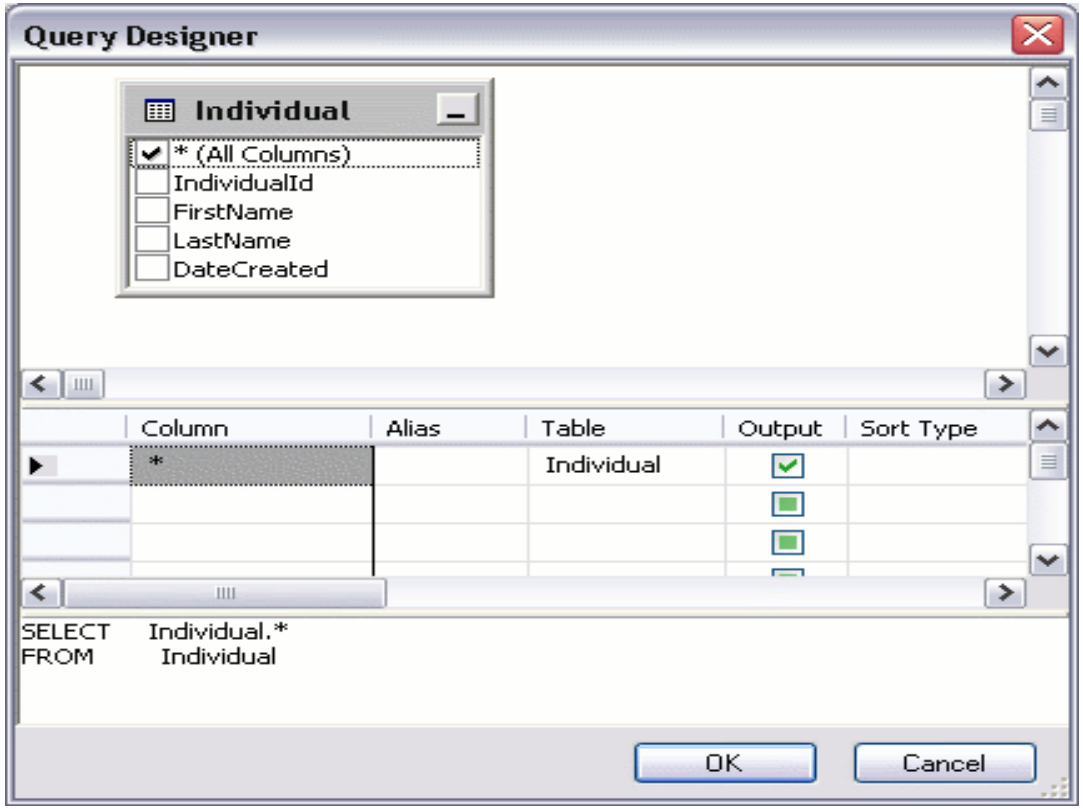
- Select *Query* > *Design Query in Editor...*:



- Add the tables you want to run the query against. In this case, we only have one table to choose from.



- Select the column/s you want to display in your query:




Click "OK“

Once you've clicked OK, you will find the query has been added to your workspace.

You can then run it as you would any other query.

- In SQL Server, a view is a pre-written query that is stored on the database.
- A view consists of a SELECT statement, and when you run the view, you see the results of it like you would when opening a table.
- Some people like to think of a view as a virtual table.
- This is because a view can pull together data from multiple tables, as well as aggregate data, and present it as though it is a single table.

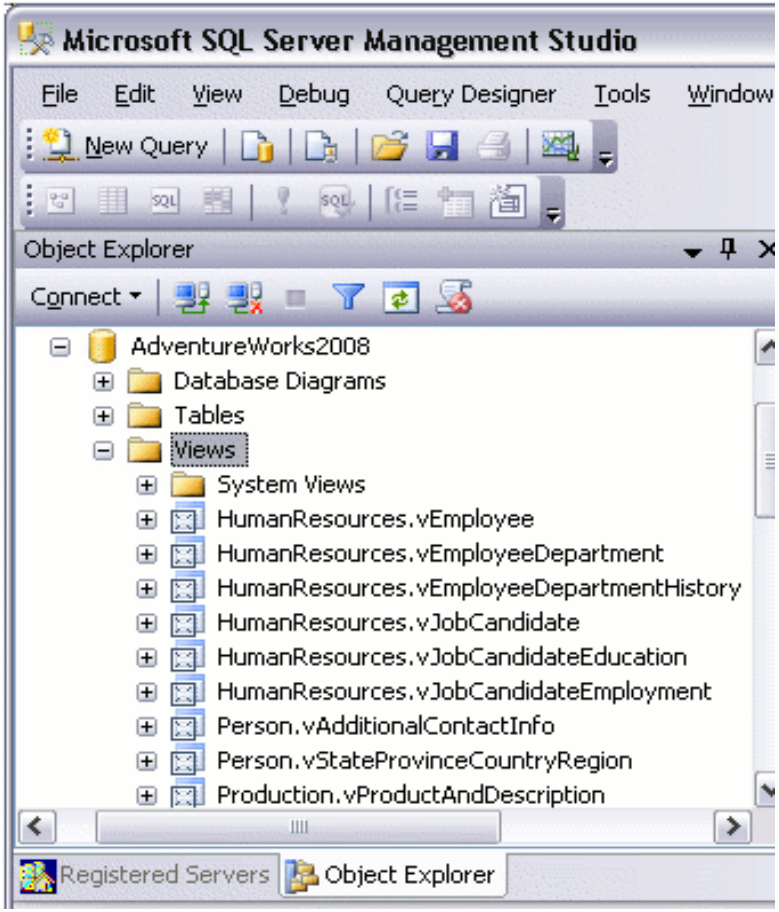
# Benefits of Views

- A view can be useful when there are multiple users with different levels of access, who all need to see portions of the data in the database (but not necessarily all of the data).

- Views can do the following:
  - Restrict access to specific rows in a table
  - Restrict access to specific columns in a table
  - Join columns from multiple tables and present them as though they are part of a single table
  - Present aggregate information (such as the results of the COUNT function)

# Accessing Views

- Any view that you create ends up being located under the "Views" folder of your database.
- The following screenshot shows a number of views that are located within the "AdventureWorks2008" database:



# Creating a View

- You create a view by using the CREATE VIEW statement, followed by the SELECT statement.

- CREATE VIEW  ViewName AS
- SELECT ...

- Example:

- CREATE VIEW "Alphabetical list of products" AS
- SELECT Products.*, Categories.CategoryName
- FROM Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID
- WHERE (((Products.Discontinued)=0))

# Modifying a View

- You can modify an existing view by using ALTER instead or CREATE.
- Example:

- ALTER VIEW "Alphabetical list of products" AS
- SELECT Products.*, Categories.CategoryName
- FROM Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID
- WHERE (((Products.Discontinued)=0))

- You can also right click on the view and select "Design".

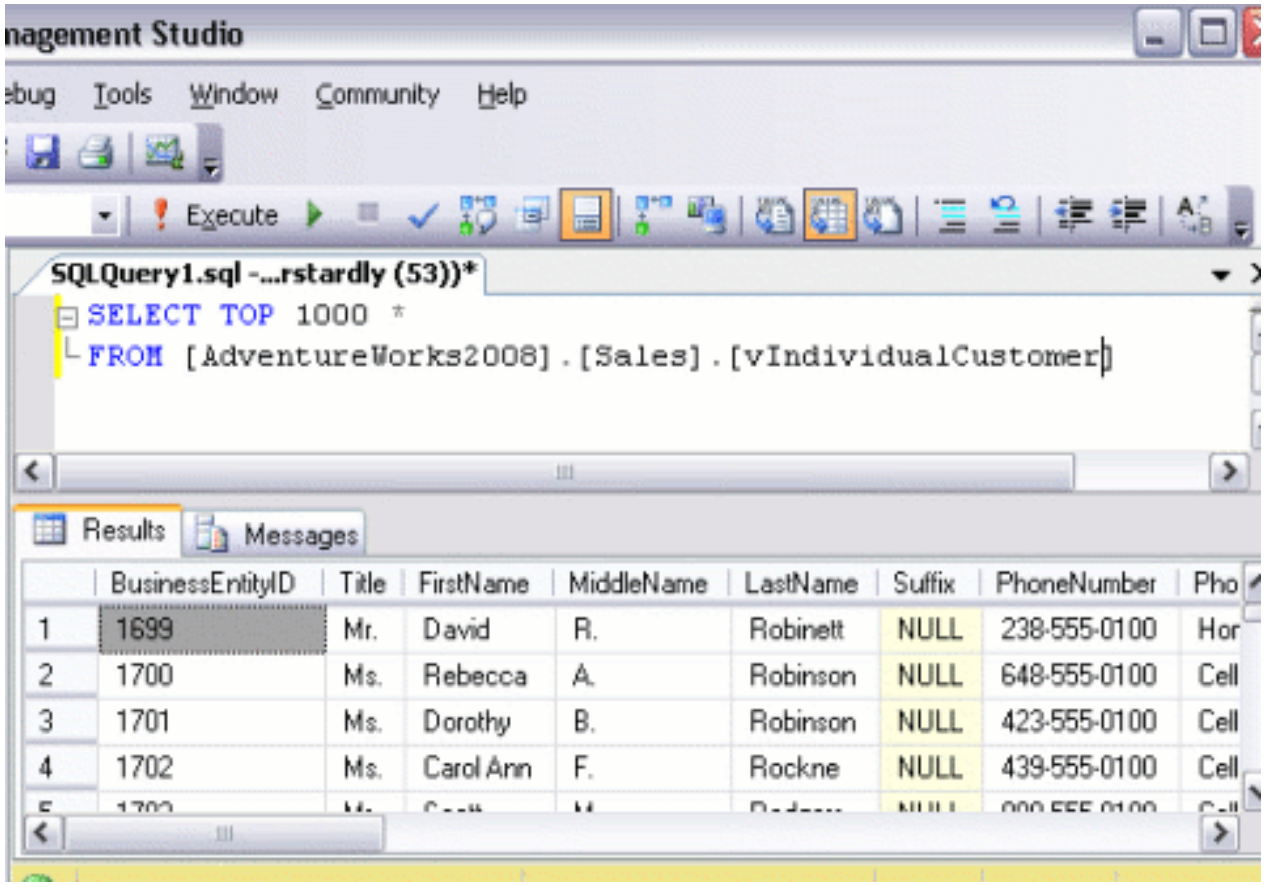
# Running a View

- You run a view by using a SELECT statement.

- SELECT TOP 1000 *
- FROM [AdventureWorks2008].[Sales].[vIndividualCustomer]

- You can also right-click on the view and select "Select Top 1000 Rows".

- Running the above view results in this:




- As you can see, it looks just like you've selected rows from a table. The difference is that, each column could potentially be coming from a different table.

# SQL Server - Stored Procedures

- Stored procedures are a powerful part of SQL Server.
- They can assist programmers and administrators greatly in working with the database configuration and its data.
- A stored procedure is a precompiled group of Transact-SQL statements, and is saved to the database (under the "Stored Procedures" node).
- Programmers and administrators can execute stored procedures either from the SQL Server Management Studio or from within an application as required.

- Transact-SQL, which is based on SQL (**Structured Query Language**), is the programming language used to interface between applications and their databases. T
- ransact-SQL is a relatively easy language to learn and I highly recommend becoming familiar with it.

# Benefits of Stored Procedures

- **Modular programming:**

- You can write a stored procedure once, then call it from multiple places in your …

- **Performance:**
- **Faster execution**: Stored procedures are parsed and optimized as soon as they are created and the stored procedure is stored in memory. This means that it will execute a lot faster than sending many lines of SQL code from your application to the SQL Server. Doing that requires SQL Server to compile and optimize your SQL code every time it runs.
- **Reduced network traffic**: If you send many lines of SQL code over the network to your SQL Server, this will impact on network performance. This is especially true if you have hundreds of lines of SQL code and/or you have lots of activity on your application. Running the code on the SQL Server (as a stored procedure) eliminates the need to send this code over the network. The only network traffic will be the parameters supplied and the results of any query.

- **Security:**

- Users can execute a stored procedure without needing to execute any of the statements directly.
- Therefore, a stored procedure can provide advanced database functionality for users who wouldn't normally have access to these tasks, but this functionality is made available in a tightly controlled way.

# Creating a Stored Procedure

- You create stored procedures in the SQL Server Management Studio using the *CREATE PROCEDURE* statement, followed by the code that makes up the stored procedure.
  - CREATE PROCEDURE StoredProcedureName AS ...
- The following code creates a stored procedure called "MyStoredProcedure":
  - CREATE PROCEDURE MyStoredProcedure AS
  - SET ROWCOUNT 10
  - SELECT Products.ProductName AS TenMostExpensiveProducts, Products.UnitPrice
  - FROM Products
  - ORDER BY Products.UnitPrice DESC
- Once you run this code in the SQL Server Management Studio, the stored procedure is created and appears under the "Stored Procedures" node.

# Modifying a Stored Procedure

- If you need to modify an existing stored procedure, you simply replace the CREATE with ALTER.

- ALTER PROCEDURE MyStoredProcedure AS...

# Running a Stored Procedure

- You can run a stored procedure by using *EXECUTE* or *EXEC*.
- For example, to run the above stored procedure, type the following:

- EXEC MyStoredProcedure

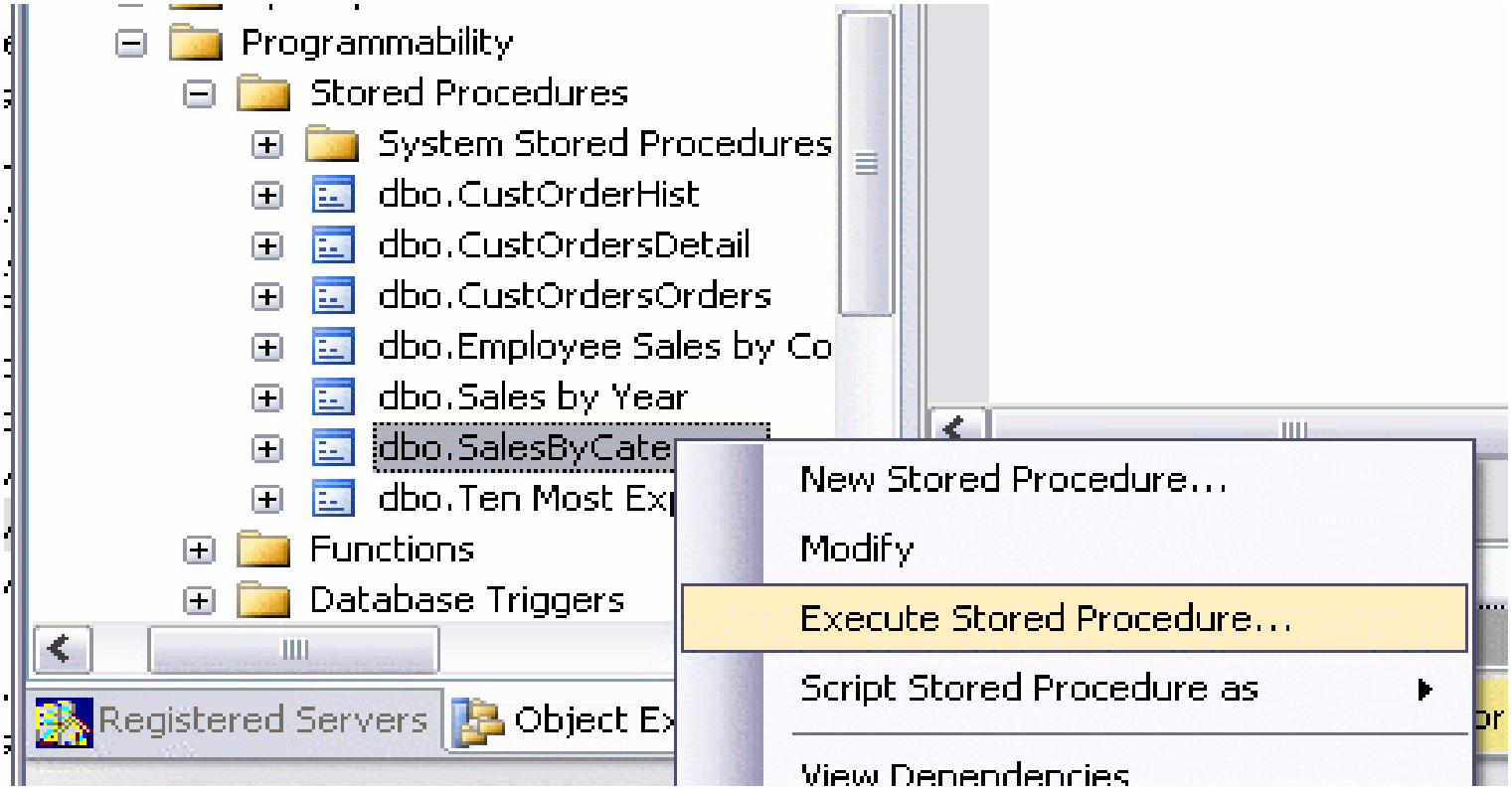
- If the stored procedure has spaces in its name, enclose it between double quotes:

- EXEC "My Stored Procedure"

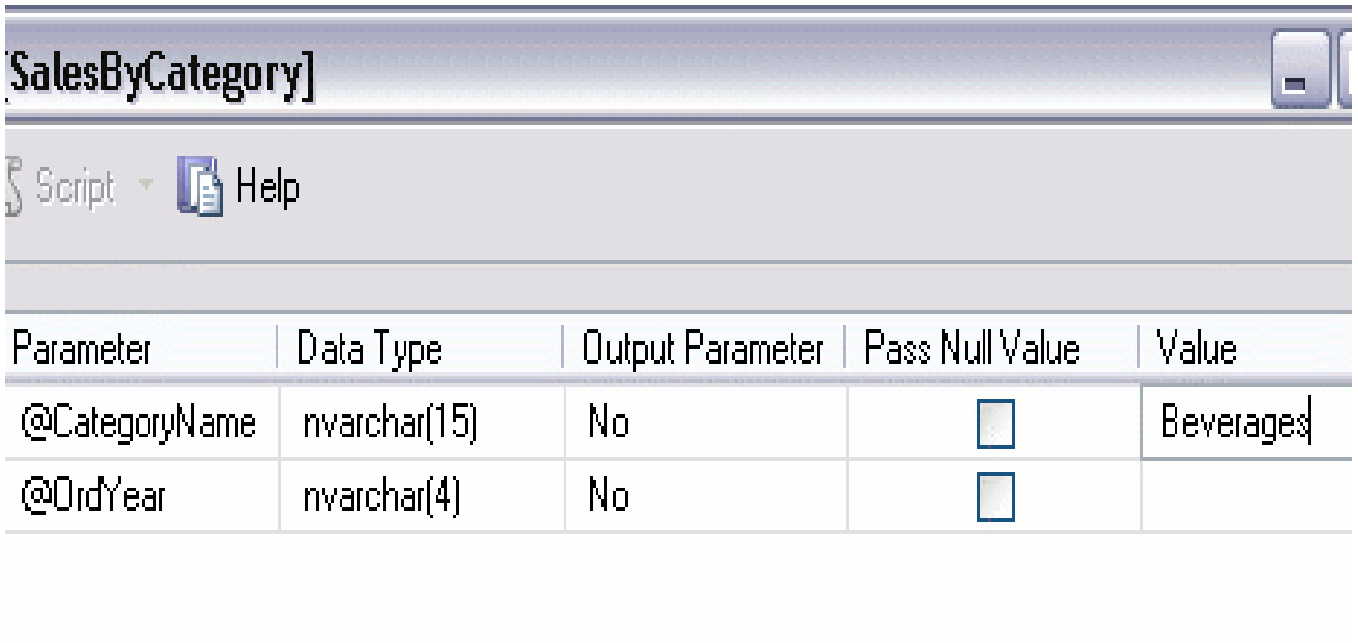
- If your stored procedure accepts any parameters, they are placed after the procedure name:
- EXEC MyStoredProcedure @ParameterName="MyParameter“
- So, here's an example:
- EXEC SalesByCategory @CategoryName ="Beverages"

# Using The GUI

- You can also use the graphical user interface to initiate the execution of a stored procedure. To initiate a stored procedure this way:
  - Navigate to the stored procedure
  - Right click on the stored procedure and select "Execute Stored Procedure...":



- A dialog will appear. Enter your chosen parameter values etc:




- Click "OK"

- SQL Server will now generate the SQL code and execute the stored procedure:

SQLQuery4.sql -...arstardly (56))

```
USE [Northwind]
GO

DECLARE @return_value int

EXEC    @return_value = [dbo].[SalesByCategory]
        @CategoryName = N'Beverages'

SELECT  'Return Value' = @return_value

GO
```

Results | Messages

| | ProductName | TotalPurchase |
|---|---|---|
| 1 | Chai | 6296.00 |
| 2 | Chang | 6299.00 |
| 3 | Chartreuse verte | 4261.00 |
| 4 | Côte de Blaye | 67324.00 |
| 5 | Guaraná Fantástica | 2318.00 |
| 6 | Ipoh Coffee | 7526.00 |

# Parameters

- A parameter is a value that your stored procedure uses in order to perform it's task.
- When you write a stored procedure, you can specify the parameters that need to be supplied from the user.
- For example, if you write a stored procedure to select the address details about an individual, your stored procedure needs to know which individual to select.
- In this case, the user can provide an IndividualId or UserId to tell the stored procedure which individual to look up.

# System Stored Procedures

- SQL Server includes a large number of system stored procedures to assist in database administration tasks.
- Many of the tasks you can perform via SQL Server Management Studio can be done via a system stored procedure.
- For example, some of the things you can do with system stored procedures include:

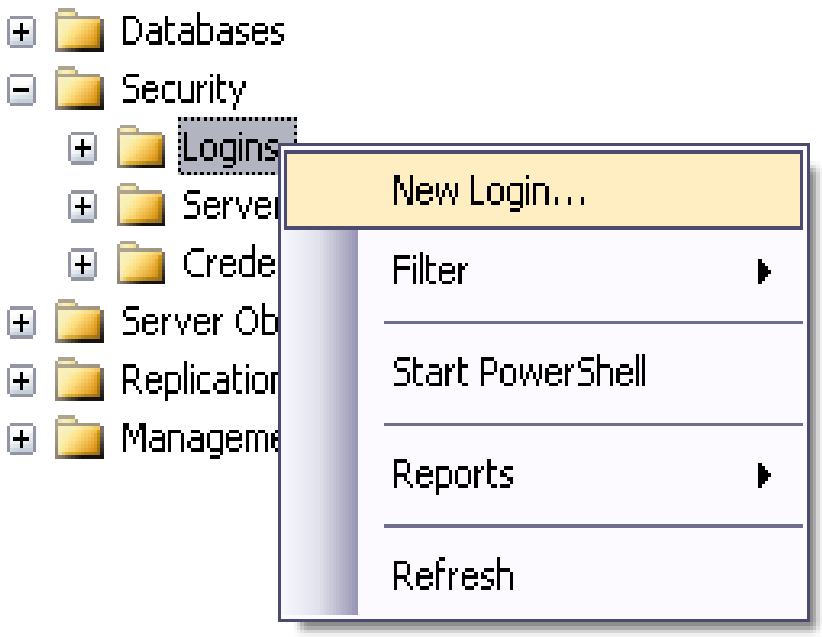
  - configure security accounts
  - set up linked servers
  - create a database maintenance plan
  - create full text search catalogs
  - configure replication
  - set up scheduled jobs
  - and much more.

# SQL Server - User Logins

- SQL Server allows for the creation of user logins.
- Each individual who needs access to SQL Server can be given their own user account.
- When the administrator configures these user logins, he/she can assign them to any number of roles and schemas, depending on the access that the individual is entitled to.

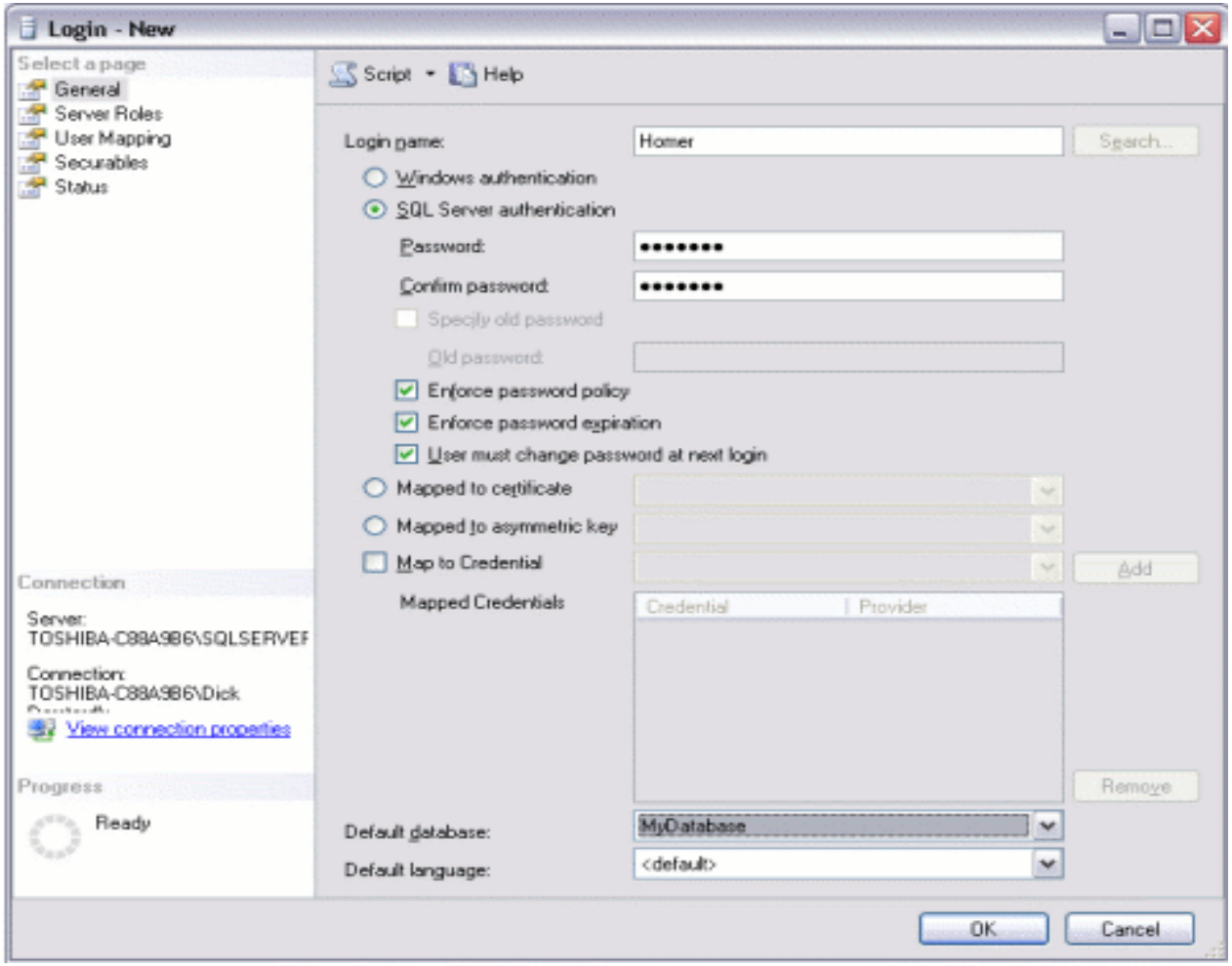
# To Create a New User Login

- Using SQL Server Management Studio, expand the "Security" option and right click on "Logins"
- Click on "New Login"

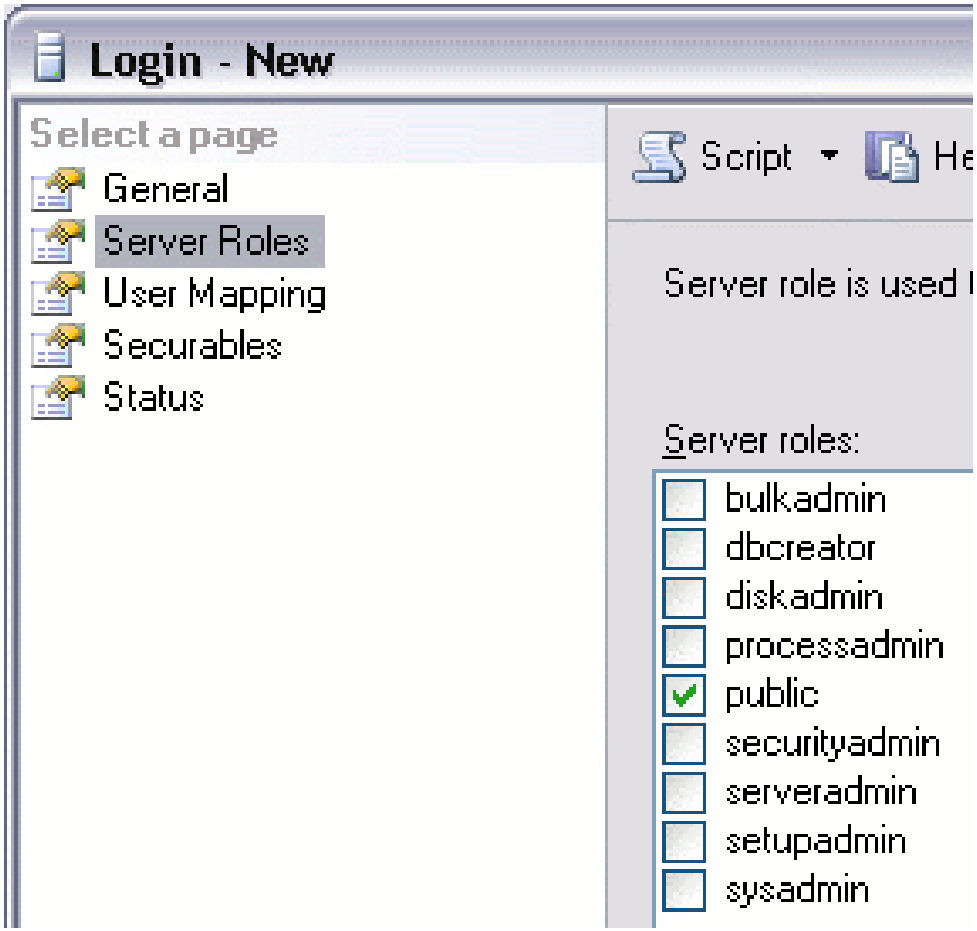


- Complete the login properties in the "General" tab by providing a name for the login, choosing the Authentication method (providing a password if you choose "SQL Server authentication"), and selecting the database to use as a default. If you don't choose a language, it will use the default for the current installation of SQL Server.

- If you get an error that reads "The MUST_CHANGE option is not supported by this version of Microsoft Windows", simply uncheck the "User must change password at next login" option. The error occurs because your operating system doesn't support this option.

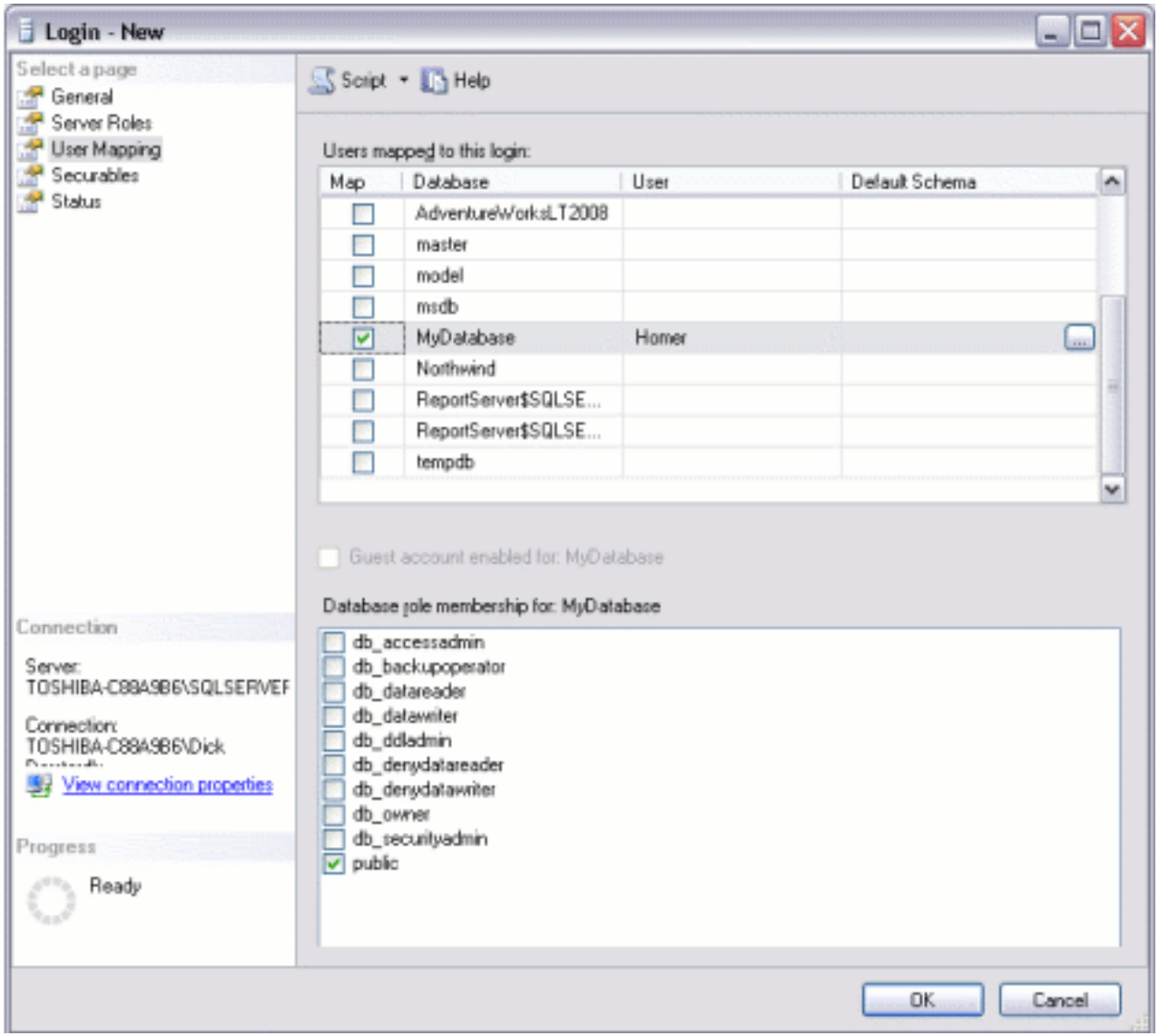

Login - New
Select a page
General
Server Roles
User Mapping
Securables
Status
Script
Help
Login name:
Homer
Search...
Windows authentication
SQL Server authentication
Password:
Confirm password:
Specify old password
Old password:
Enforce password policy
Enforce password expiration
User must change password at next login
Mapped to certificate
Mapped to asymmetric key
Map to Credential
Add
Mapped Credentials
Credential
Provider
Remove
Default database:
MyDatabase
Default language:
<default>
Connection
Server:
TOSHIBA-C88A9B6\SQLSERVEF
Connection:
TOSHIBA-C88A9B6\Dick
View connection properties
Progress
Ready
OK
Cancel

- Click the "Server Roles" tab if you need to apply any server-wide security privileges.



- Click the "User Mapping" tab to specify which databases this user account is allowed to access.
- By default, the login will be assigned to the "Public" role, which provides the login with basic access.
- If the login needs more access in one or more databases, it can be assigned to another role with greater privileges.

- Note that these roles are "Database Roles" and are different to the server roles in the previous tab.
- Server roles are for administering the SQL Server.
- Database roles are created within each database and specify what the login can do within that database.

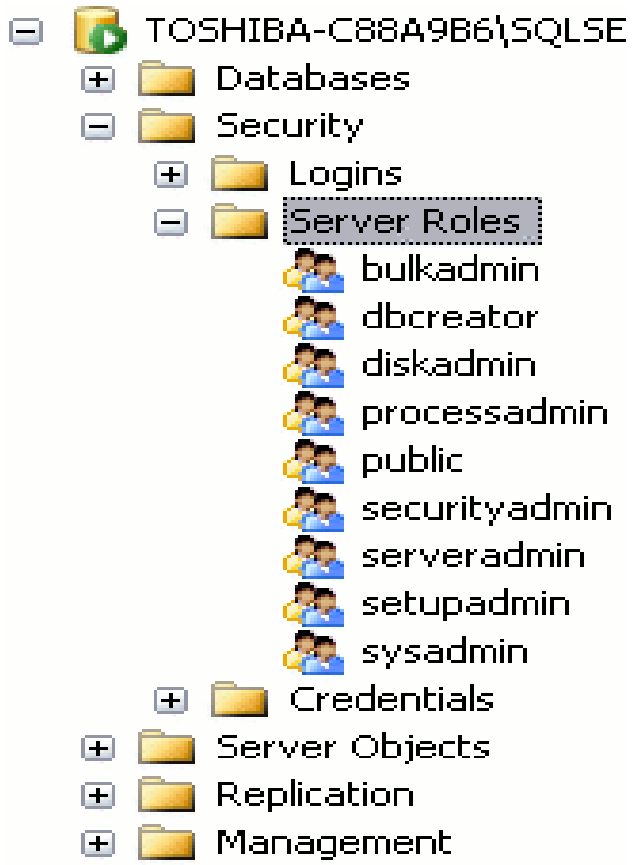

Login - New
Select a page
General
Server Roles
User Mapping
Securables
Status
Script
Help
Users mapped to this login:
Map
Database
User
Default Schema
AdventureWorksLT2008
master
model
msdb
MyDatabase
Homer
Northwind
ReportServer$SQLSE...
ReportServer$SQLSE...
tempdb
Guest account enabled for: MyDatabase
Database role membership for: MyDatabase
db_accessadmin
db_backupoperator
db_datareader
db_datawriter
db_ddladmin
db_denydatareader
db_denydatawriter
db_owner
db_securityadmin
public
Connection
Server:
TOSHIBA-C88A9B6\SQLSERVEF
Connection:
TOSHIBA-C88A9B6\Dick
View connection properties
Progress
Ready
OK
Cancel

# SQL Server - Server Roles

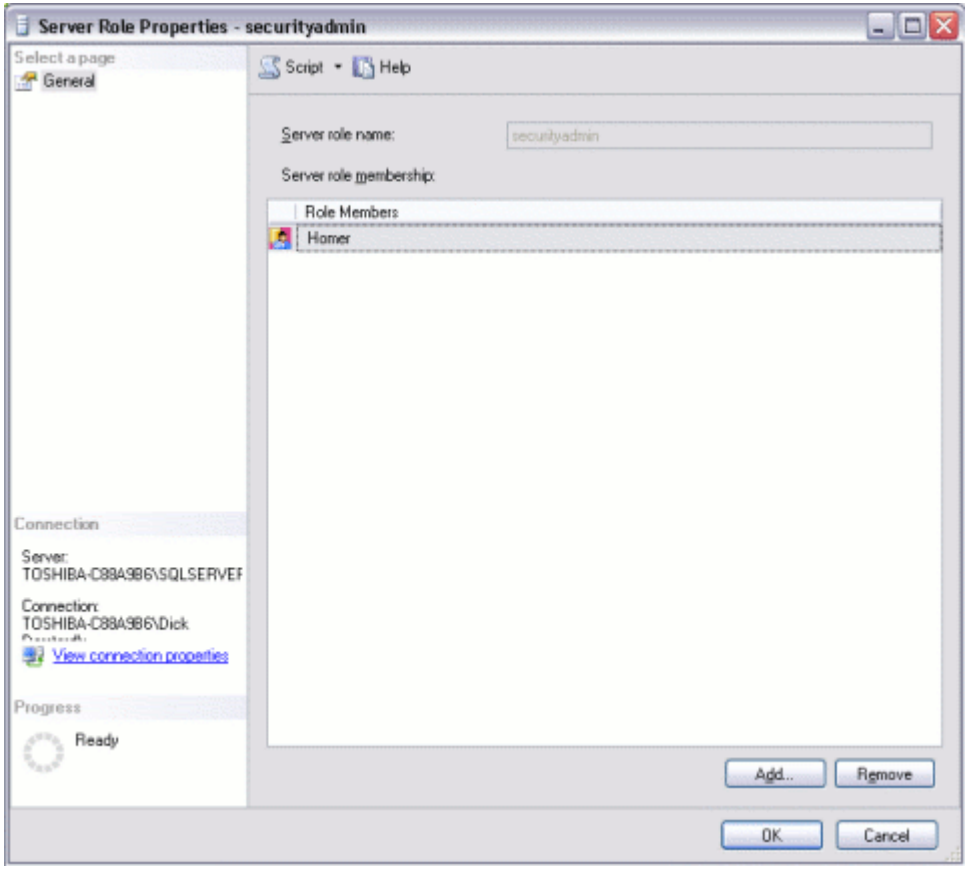
- When creating a new user login in SQL Server, you get the option of assigning the login one or more server roles.
- Server roles (not to be confused with database roles) are available for various database administration tasks.
- Not everyone should be assigned to a server role. In fact, only advanced users such as database administrators should be assigned a server role.

# Accessing the Server Roles

- To access the server roles in SQL Server Management Studio, expand the Security folder:



- You view the properties of a server role by right clicking on it. You can then add users to the server role by clicking Add.
- In the screenshot below, Homer has been added to the *securityadmin* role.



# Explanation of Server Roles

**sysadmin:**

- Can perform any task in SQL Server.

**serveradmin:**

Can set server-wide configuration options, can shut down the server.

**setupadmin:**

Can manage linked servers and startup procedures.

# Explanation of Server Roles

**securityadmin:**

- Can manage logins and database permissions, read logs, change passwords.

**processadmin:**

Can manage processes running in SQL Server.

- **dbcreator:**
  - Can create, alter, and drop databases.

# Explanation of Server Roles

**diskadmin:**

- Can manage disk files.

**bulkadmin:**

Can execute BULK INSERT statements.

- **public:**
  - Every SQL Server user account belongs to this server role. When a server principal has not been granted or denied specific permissions on a securable object, the user inherits the permissions granted to public on that object. Only assign public permissions on an object when you want the object to be available to all users.

# Explanation of Server Roles

- As you can see, some of these roles allow very specific tasks to be performed.

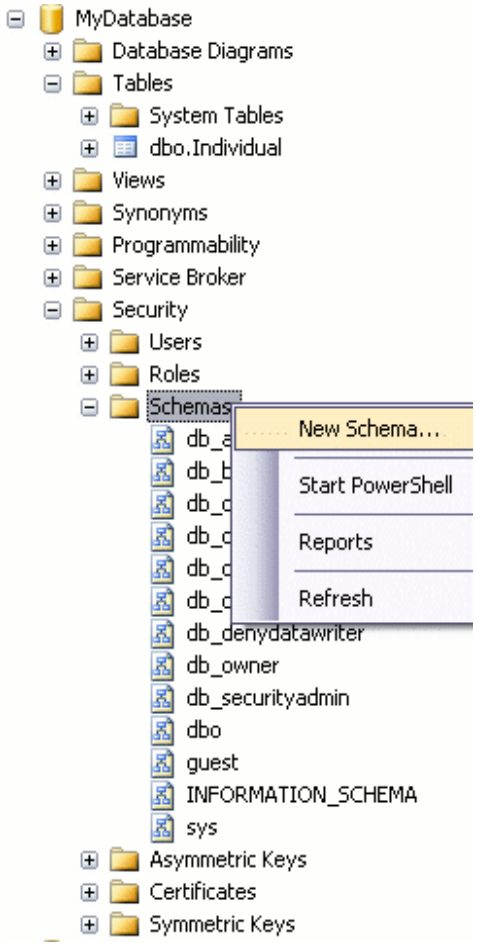
- If you don't have many technical users, it's likely that you'll only use one or two of these roles (including sysadmin).

# SQL Server - Database Schemas

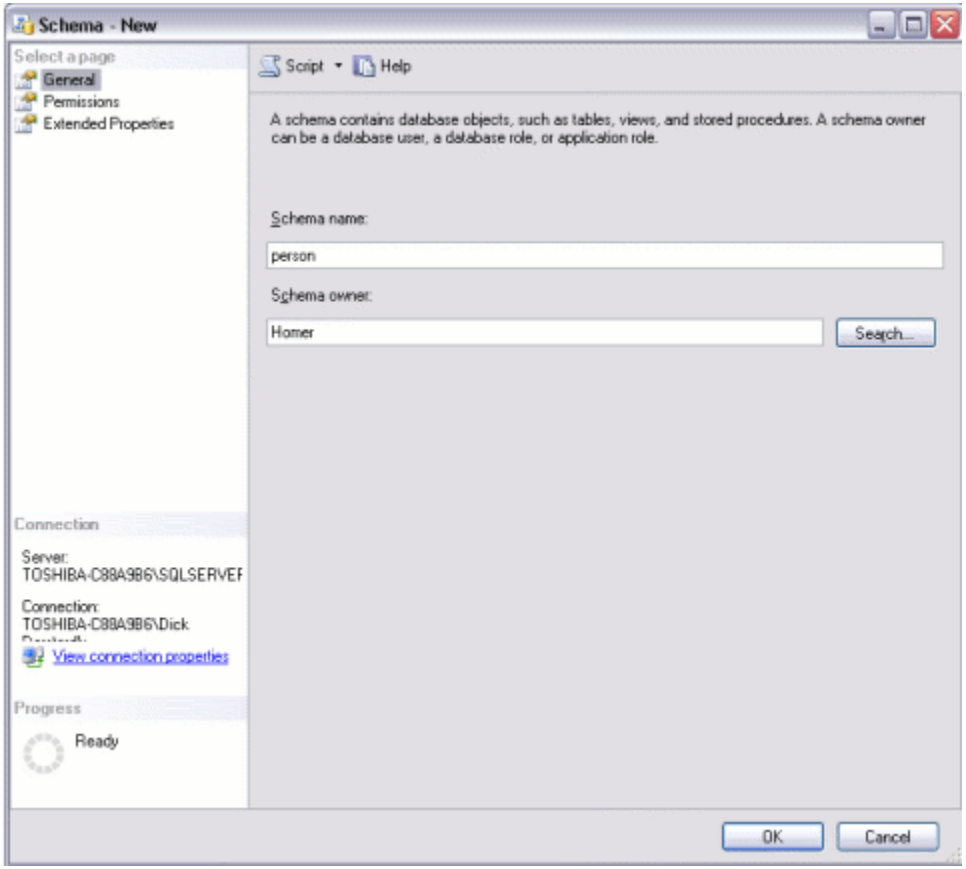
- A database schema is a way to logically group objects such as tables, views, stored procedures etc. Think of a schema as a container of objects.
- You can assign a user login permissions to a single schema so that the user can only access the objects they are authorized to access.
- Schemas can be created and altered in a database, and users can be granted access to a schema.
- A schema can be owned by any user, and schema ownership is transferable.

# Creating a Database Schema

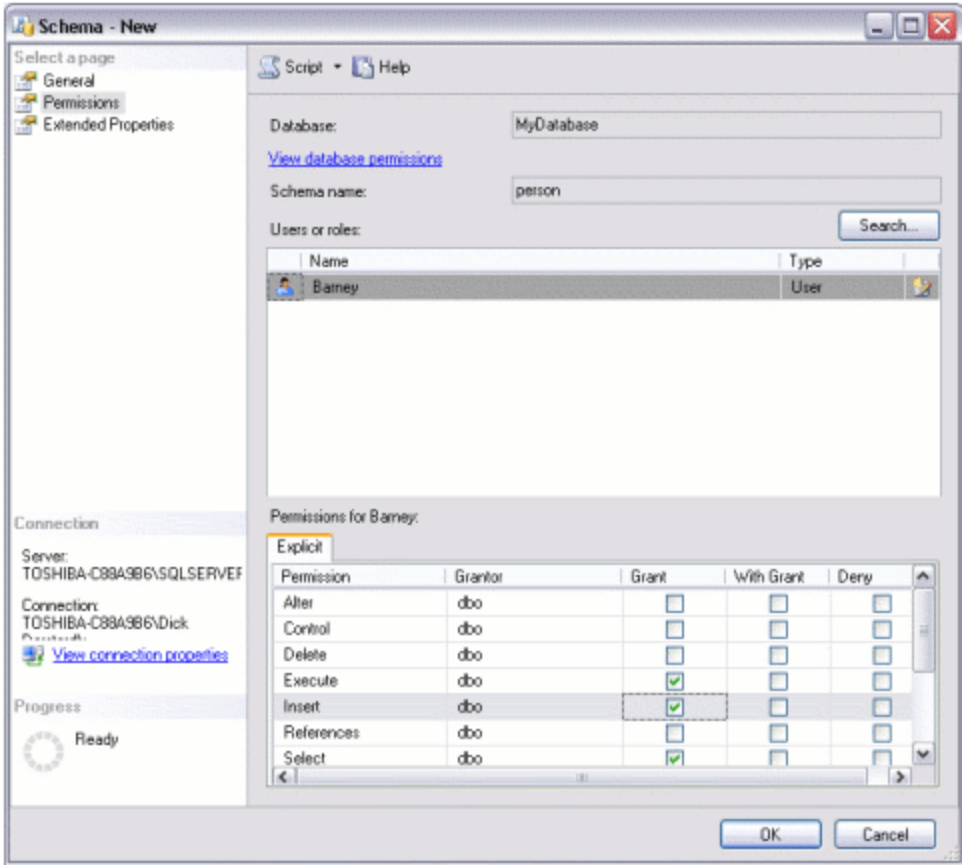
- Navigate to *Security* > *Schemas*
- Right click on *Schemas* and select *New Schema*.... Like this:



- Complete the details in the *General* tab for the new schema. In this example, the schema name is "person" and the schema owner is "Homer".



- Add users to the schema as required and set their permissions:

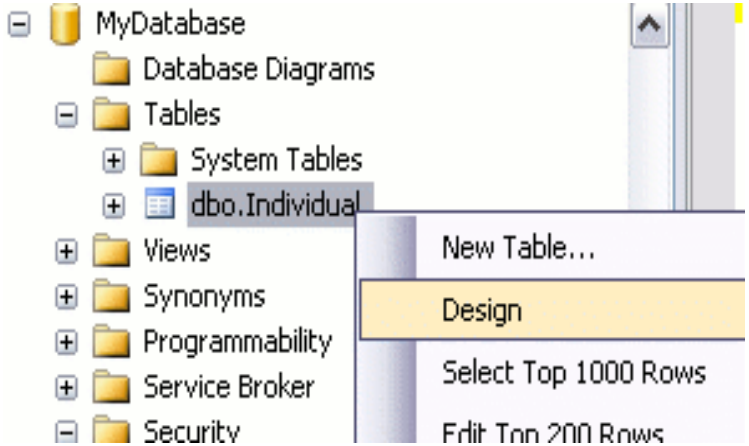



- Add any extended properties (via the Extended Properties tab)

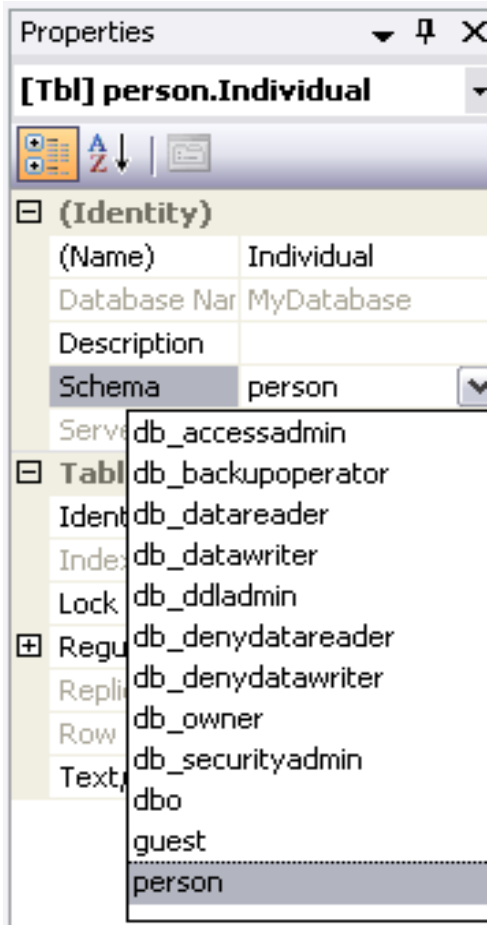
# Add a Table to the New Schema

- Now that we have a new schema, we can add objects such as tables, views, and stored procedures to it.
- For example, we could transfer the table that we created in the earlier lesson to the new schema.
- When we created that table (called "Individual"), it was created in the default database schema ("dbo").
- We know this because it appears in our object browser as "dbo.Individual".

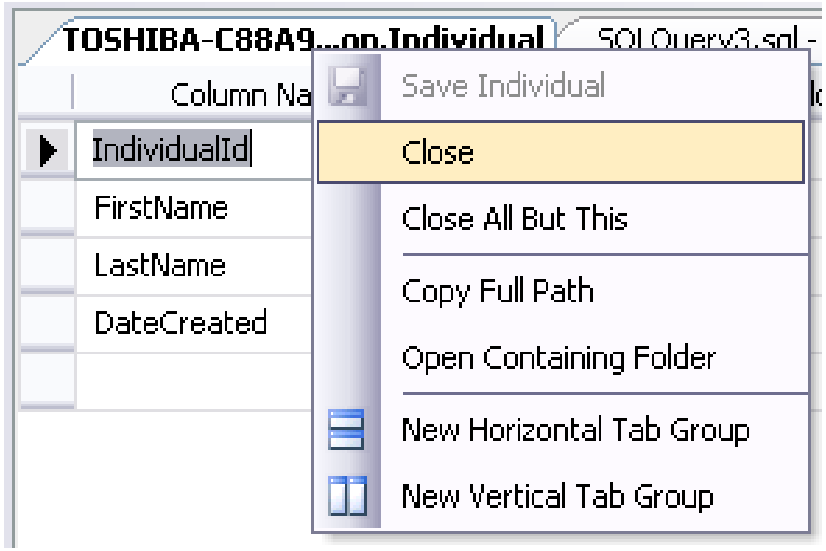
- To transfer the "Individual" table to the person "schema":
- In Object Explorer, right click on the table name and select "Design":



- From Design view, press F4 to display the Properties window.
- From the Properties window, change the schema to the desired schema:



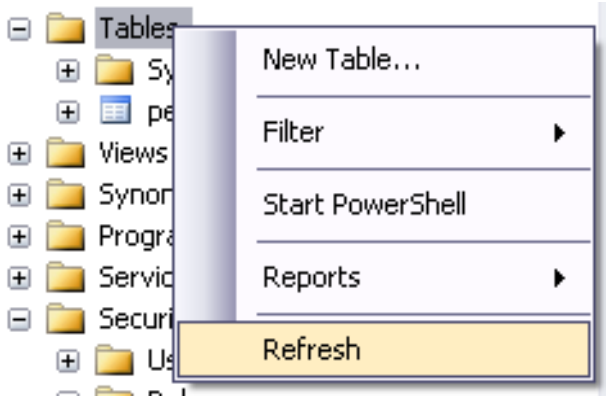
- Close Design View by right clicking the tab and selecting "Close":




- Your table has now been transferred to the "person" schema.

# Confirm your Change

- Refresh the Object Browser view:



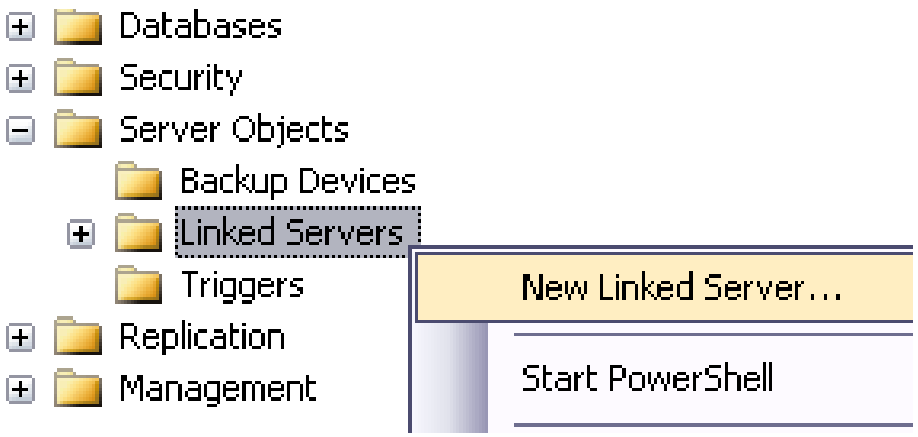

- You will now see that Object Browser displays the new schema for the table (person.Individual):

# SQL Server - Linked Servers

- The Linked Servers option allows you to connect to another instance of SQL Server running on a different machine, perhaps remotely in a different city/country.
- This can be useful if you need to perform distributed queries (query a remote database).
- Setting up a linked server is quite straight forward in **SSMS**, all you need is details of the remote server, and the database that you need to query.

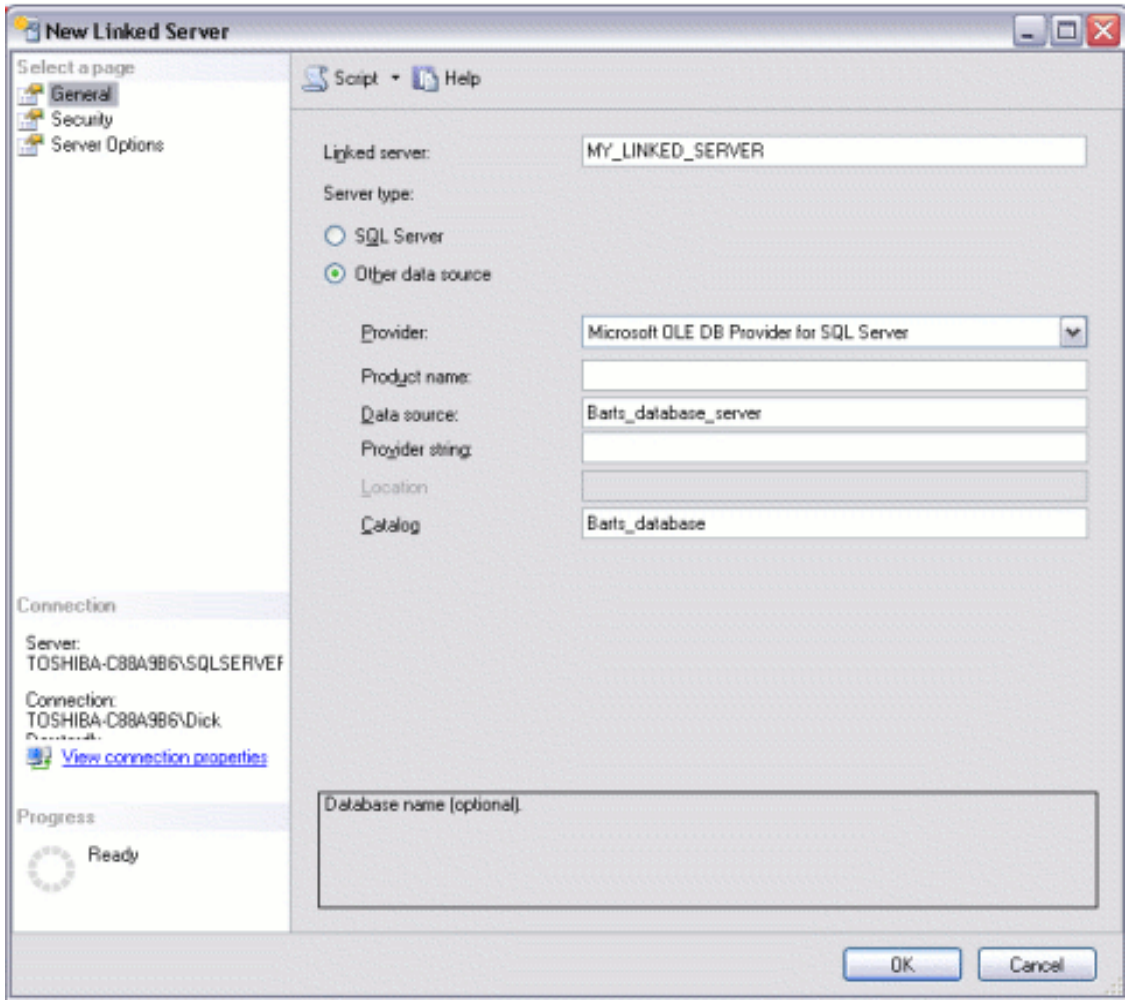
# Creating a Linked Server

- Navigate to *Server Objects* > *Linked Servers*
- Right click on *Linked Servers* and select *New Linked Server...*. Like this:




Complete the details for the linked server. In this example, *Data source* refers to the name of the SQL Server machine ("Barts_database_server"), *Catalogue* refers to the name of the database ("Barts_database").

- You can also configure options in the other two tabs depending on your requirements.



# Distributed Queries

- Once you have configured your linked server, you will be able to run queries etc against it.
- When you run a query against a linked server, it is referred to as a *distributed query*.
- When you execute a distributed query against a linked server, you must include a fully qualified, four-part table name for each data source to query.
- This four-part name should be in the form:
  - *linked_server_name.catalog.schema.object_name*.

- An example:

- **Barts_database_server.Barts_database.Person.Enemy**

- This example is based on the linked server example above.
- It assumes that the remote database has a schema called "Person" and a table called "Enemy".