palasi.com
IT and management

Bases de datos relacionales
Microsoft SQL Server
Data warehouse

Dr. Vicent-Ramon Palasí Lallana

Objetivos del curso

- Repasar la teoría de bases de datos que ustedes conocen.
- Aplicarla para crear bases de datos reales en un producto comercial como es SQL Server.

¿Por qué SQL Server?

- Es la base de datos con mayor relación calidad/precio del mercado. Oracle es aún mejor, pero su precio adicional sólo se justifica en una pequeña fracción de los casos.
- No demanda grandes recursos.
- Incluye programas de OLAP (o data warehousing) sin costo adicional.
- En mi modesta opinión, es el mejor producto de Microsoft (lo que no es decir mucho, pero realmente es un excelente producto).

Temario del curso

- 1. Introducción a las bases de datos relacionales.
- 2. Creación de la estructura de bases de datos.
- 3. Recuperación y actualización de datos.
- 4. Vistas.
- 5. Seguridad. Permisos de usuario.
- 6. OLAP (Data warehousing)

Temario del curso

- 1. Introducción a las bases de datos relacionales.
- 2. Creación de la estructura de bases de datos.
- 3. Recuperación y actualización de datos.
- 4. Vistas.
- 5. Seguridad. Permisos de usuario.
- 6. OLAP (Data warehousing)

1. Introducción a las bases de datos relacionales.

- 1.1. Breve repaso del modelo relacional.
- 1.2. Relación con la programación O-O.

Mail: palasi@palasi.com Web: www.palasi.com

palasi.com
IT and management

1. Introducción a las bases de datos relacionales.

- 1.1. Breve repaso del modelo relacional.
- 1.2. Relación con la programación O-O.

Persistencia de datos

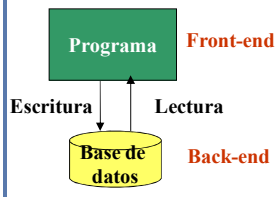
- Si un programa debe mantener datos entre ejecuciones del mismo, debe guardar los datos en un soporte permanente (normalmente, disco).
- A esto se le llama persistencia de datos.
- Sin persistencia de datos, cada vez que ejecutamos un programa debemos introducir todos los datos que necesita y esto no es práctico.

La persistencia de datos la obtenemos con una base de datos

- Una base de datos es un conjunto de archivos que comparten información de una forma integrada.
- Un sistema gestor de bases de datos (SGBD) o database management system (DBMS) es un programa software cuya principal función es guardar y recuperar los datos de una BD (a veces se le llama "motor de base de datos").
- Por ejemplo, en una base de datos Access, empleados.mdb es la base de datos y Microsoft Access es el SGBD.
- Una base de datos puede ser accedida desde un SGBD o desde otro programa.

Funcionamiento de una base de datos

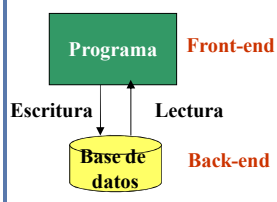
Una aplicación informática está compuesta de un programa (front-end) que se conecta con una base de datos (back-end).



- La base de datos almacena los datos en un soporte permanente.
- El programa guarda (escribe) o recupera (lee) datos de la BD.

Lo que veremos en este curso

En el anterior curso de programación nos habíamos centrado en el diseño del programa (front-end).



- En este curso nos centraremos en el diseño de la base de datos (back-end).

Diferentes tipos de bases de datos

- Según el modelo teórico que usen, las bases de datos se pueden dividir en:
  - Bases de datos jerárquicas. Si usan el modelo jerárquico.
  - Bases de datos en red. Si utilizan el modelo en red.
  - Bases de datos relacionales. Si usan el modelo relacional.
  - Bases de datos orientadas a objetos. Si usan el modelo orientado a objetos.

Mail: palasi@palasi.com Web: www.palasi.com

¿Qué implantación tienen en el mercado?

- Las bases de datos jerárquicas y en red hace mucho tiempo que quedaron obsoletas. Hoy en día es difícil encontrar alguna.
- Aunque hay algunas bases de datos orientadas a objetos, aún no han conseguido tener una porción sustancial del mercado.
- Las bases de datos relacionales ocupan casi la totalidad del mercado. Son el estándar actual y este curso se dedicará a ellas.
- Oracle, SQL Server, Sybase, PostgreSQL, MySQL, Interbase e incluso Access y Foxpro son todas bases de datos relacionales.

Bases de datos relacionales

- Se basan en el modelo relacional, un modelo teórico que definió E.F. Codd en los últimos años 60.
- Trabajaba en el centro de investigación de IBM y comenzó a estudiar las bases de datos de la época.
- Para su sorpresa, vio que los SGBDs eran "totalmente empíricos, sin ninguna teoría en absoluto". En sus propias palabras: "Comencé a leer documentación y me disgusté".
- Decidió desarrollar una teoría matemática para la gestión de las bases de datos.

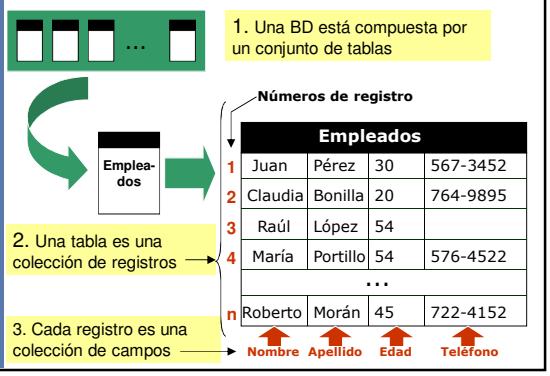
El modelo relacional

- El resultado apareció en 1970 en un artículo (hoy clásico) llamado "A relational Model of Data for Large Shared Data Banks".
- El artículo explicaba el modelo teórico de cómo debían ser las BDs, pero no decía cómo podía implementarse.
- Durante los años 70, varias empresas y universidades investigaron para conseguir implementarlo. Se desarrolló el lenguaje SQL.
- En 1976, Honeywell lanza el 1er. SGBD relacional comercial, llamado "Multics Relational Data Store".
- En 1979, Oracle presenta la primera versión de su SGBD y en 1983, IBM presenta DB2.

El modelo relacional

- La colección de datos usada por el software se llama base de datos.
- La base de datos se divide en tablas, cada una de las cuales representa un conjunto de entidades de la vida real (así, tabla de empleados, de clientes, de facturas, etc.)
- Cada tabla es un conjunto de registros, cada uno de los cuales representa una entidad en la vida real (así, el registro que representa a Juan Pérez).
- Cada registro tiene una serie de datos llamados campos o columnas.

El modelo relacional



El modelo relacional es un modelo plano

- Los datos no tienen ninguna estructura. Son sólo conjuntos de registros.
- Esto es contrario a la programación orientada a objetos, donde los datos tienen estructura (un objeto puede contener otros objetos, etc).
- Esto hace que la combinación de ambos no sea trivial.
- Trataremos este tema más adelante con más detalle.

Mail: palasi@palasi.com Web: www.palasi.com

Clave primaria en el modelo relacional

- Cada tabla debe tener un campo (o varios) llamado clave primaria, que identifica de forma única a cada uno de sus registros.
- Es decir, conociendo el valor del campo, conocemos automáticamente qué registro es.

En este caso, la clave primaria es el campo "DUI"

	Empleados				
1	566	Juan	Pérez	30	567-3452
2	099	Claudia	Bonilla	20	764-9895
3	567	Raúl	López	54	
4	927	María	Portillo	54	576-4522
		…			
n	456	Roberto	Morán	45	722-4152
	DUI	Nombre	Apellido	Edad	Teléfono

Claves candidatas

- Puede pasar que en una misma tabla, haya varios campos que puedan ser claves primarias.
- A estos campos se les llama "claves candidatas".

Las claves candidatas son DUI y ISSS, pues las dos pueden ser primarias

	Empleados				
1	566	Juan	Pérez	30	5645267
2	099	Claudia	Bonilla	20	7647598
3	567	Raúl	López	54	
4	927	María	Portillo	54	5487552
		…			
n	456	Roberto	Morán	45	7455452
	DUI	Nombre	Apellido	Edad	ISSS

Pero una tabla sólo puede tener UNA clave primaria

- De las claves candidatas, debemos elegir una que sea la clave primaria.
- En este caso, elegiremos el DUI.

	Empleados				
1	566	Juan	Pérez	30	5645267
2	099	Claudia	Bonilla	20	7647598
3	567	Raúl	López	54	
4	927	María	Portillo	54	5487552
		…			
n	456	Roberto	Morán	45	7455452
	DUI	Nombre	Apellido	Edad	ISSS

Clave foránea

- En el modelo relacional, dos tablas se relacionan mediante una clave foránea: un campo de una tabla que es clave primaria en la otra tabla.

Clave foránea

El campo "Jefe" contiene el DUI del jefe del departamento, que es clave primaria en otra tabla.

Departamentos		
001	Compras	566
014	Producción	927
Código	Nombre	Jefe

Empleados				
566	Juan	Pérez	30	567-3452
099	Claudia	Bonilla	20	764-9895
567	Raúl	López	54	
927	María	Portillo	54	576-4522
	…			
456	Roberto	Morán	45	722-4152
DUI	Nombre	Apellido	Edad	Teléfono

La clave foránea es la manera en que las diferentes tablas se relacionan

De esta manera, un registro de una tabla se relaciona con un registro de otra tabla.

Departamentos		
001	Compras	566
014	Producción	927
Código	Nombre	Jefe

Empleados				
566	Juan	Pérez	30	567-3452
099	Claudia	Bonilla	20	764-9895
567	Raúl	López	54	
927	María	Portillo	54	576-4522
	…			
456	Roberto	Morán	45	722-4152
DUI	Nombre	Apellido	Edad	Teléfono

Formas normales en el modelo relacional

- Para que las tablas estén bien definidas se deben cumplir una serie de condiciones teóricas, llamadas “formas normales”.

Primera forma normal

- Una tabla está en primera forma normal (1FN) si todos los campos de todos los registros contienen un valor como máximo.
- Dicho de otra manera, no hay ningún campo de ningún registro que tenga más de un valor.

Primera forma normal

- Ningún campo tiene más de un valor.
- No se dan situaciones como ésta

El campo teléfono tiene dos valores.

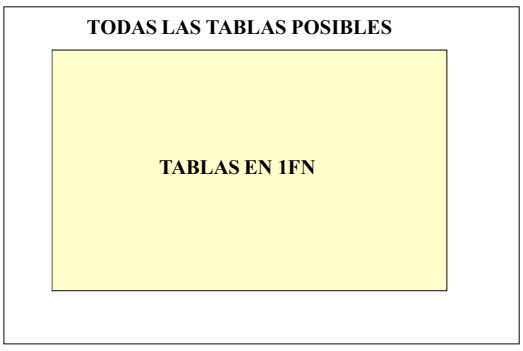
LA TABLA NO ESTA EN 1FN

	Empleados				
1	566	Juan	Pérez	30	567-3452, 345-678
2					
3	099	Claudia	Bonilla	20	764-9895
4	567	Raúl	López	54	
			...		
n	456	Roberto	Morán	45	722-4152
	DUI	Nombre	Apellido	Edad	Teléfono

Los SGBD relacionales en principio no aceptan más de un valor en un campo

- Sin embargo, se pueden producir trampas, como crear un campo de texto y añadir los diferentes valores separados por comas.
- Esto nos obliga a asegurarnos de que la primera forma normal se cumple.

Sólo unas cuantas tablas de las posibles están en 1FN



Normalizando

- El proceso por el que hacemos que una tabla cumpla con las formas normales, se llama “normalización”.
- ¿Qué hacemos con una tabla como la anterior que no cumple la 1FN?
- La normalizamos para que cumpla la 1FN.

Mail: palasi@palasi.com Web: www.palasi.com

Normalizando

- Si sabemos el número máximo de valores y este es pequeño, podemos crear tantos campos como ese número máximo.
- Así, si sabemos que, como máximo, hay dos teléfonos, podemos crear dos campos: "telef1" y "telef2".

	Empleados					
1	566	Juan	Pérez	30	567-3452	345-678
2	099	Claudia	Bonilla	20	764-9895	
3	567	Raúl	López	54		
4	927	María	Portillo	54	576-4522	
			...			
n	456	Roberto	Morán	45	722-4152	
	DUI	Nombre	Apellido	Edad	Teléf1	Teléf2

Esta no es la mejor solución

- Como vemos, se malgasta espacio.
- ¿Qué pasa si de una persona se quieren conservar más de tres teléfonos?

	Empleados					
1	566	Juan	Pérez	30	567-3452	345-678
2	099	Claudia	Bonilla	20	764-9895	
3	567	Raúl	López	54		
4	927	María	Portillo	54	576-4522	
			...			
n	456	Roberto	Morán	45	722-4152	
	DUI	Nombre	Apellido	Edad	Teléf1	Teléf2

Normalizando: una mejor solución

Teléfonos	
567-3452	566
345-678	
576-4522	927
Número	Propie-tario

Es mejor crear una nueva tabla con los campos que tenían varios valores y ligarla a la original por una clave foránea.

Empleados			
566	Juan	Pérez	30
099	Claudia	Bonilla	20
567	Raúl	López	54
927	María	Portillo	54
		...	
456	Roberto	Morán	45
DUI	Nombre	Apellido	Edad

Normalizando: una mejor solución

Teléfonos	
567-3452	566
345-678	
576-4522	927
Número	Propie-tario

No se malgasta espacio y pueden ponerse todos los teléfonos que se quiera.

Empleados			
566	Juan	Pérez	30
099	Claudia	Bonilla	20
567	Raúl	López	54
927	María	Portillo	54
		...	
456	Roberto	Morán	45
DUI	Nombre	Apellido	Edad

Segunda forma normal

- Se basa en la dependencia de campos.
- Decimos que un campo B depende de un campo A si
  - al determinar el valor del campo A
  - se determina de forma única el valor del campo B.

Dependencia de campos

- Así, el campo "Teléfono" depende del campo "DUI", ya que al determinar el DUI se determina unívocamente el teléfono.

	Empleados				
1	566	Juan	Pérez	30	567-3452
2	099	Claudia	Bonilla	20	764-9895
3	567	Raúl	López	54	
4	927	María	Portillo	54	576-4522
			...		
n	456	Roberto	Morán	45	722-4152
	DUI	Nombre	Apellido	Edad	Teléfono

Segunda forma normal

- Una tabla está en segunda forma normal (2FN) si:
- Está en 1FN.
- Todos los campos dependen de la clave primaria.

La tabla de empleados está en 2FN

- Está en primera forma normal y todos los campos dependen de la célula (clave primaria).
- Si sabemos el DUI, se determinan los demás campos.

Empleados				
566	Juan	Pérez	30	567-3452
099	Claudia	Bonilla	20	764-9895
567	Raúl	López	54	
927	María	Portillo	54	576-4522
		...		
456	Roberto	Morán	45	722-4152
DUI	Nombre	Apellido	Edad	Teléfono

Una tabla que no está en segunda forma normal

- Un campo no depende de la clave primaria.
- En este caso el campo de productos contiene los productos que fabrica la empresa.

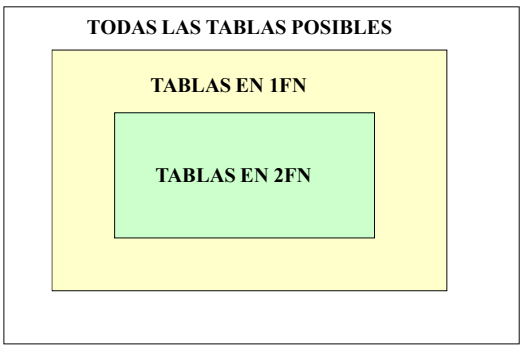
Empleados					
566	Juan	Pérez	30	567-3452	Arroz
099	Claudia	Bonilla	20	764-9895	Patatas
567	Raúl	López	54		
927	María	Portillo	54	576-4522	Legumbres
		...			
456	Roberto	Morán	45	722-4152	Plátano
DUI	Nombre	Apellido	Edad	Teléfono	Producto

Los productos no dependen de la célula del empleado

- El campo producto no depende de la clave primaria.
- En este caso, la tabla no está en 2FN.

Empleados					
566	Juan	Pérez	30	567-3452	Arroz
099	Claudia	Bonilla	20	764-9895	Patatas
567	Raúl	López	54		
927	María	Portillo	54	576-4522	Legumbres
		...			
456	Roberto	Morán	45	722-4152	Plátano
DUI	Nombre	Apellido	Edad	Teléfono	Producto

Las tablas en 2FN están en 1FN



Normalizando

- Normalizamos una tabla que no está en 2FN, poniendo los campos que no dependen de la clave primaria en una tabla diferente.
- Así, la tabla se convierte a la 2FN.

Mail: palasi@palasi.com Web: www.palasi.com

Los productos irán en una tabla a parte

Productos
Arroz
Plátanos

Nombre

Ahora la tabla "Empleados" está en forma normal.

Empleados				
566	Juan	Pérez	30	567-3452
099	Claudia	Bonilla	20	764-9895
567	Raúl	López	54	
927	María	Portillo	54	576-4522
	…			
456	Roberto	Morán	45	722-4152

DUI Nombre Apellido Edad Teléfono

Tercera forma normal

- Una tabla está en segunda forma normal (3FN) si:
- Está en 2FN.
- Los campos dependen **solamente** de las claves candidatas.

(Dicho de otra manera, ningún campo depende de otro que no sea clave candidata)

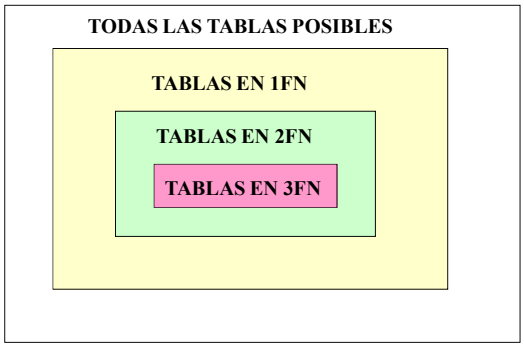
Una tabla que no está en 3FN

- El campo "País" depende del campo "Ciudad", que no es clave candidata.

Áreas			
566	Compras	San Salvador	El Salvador
099	Producción	Santa Ana	El Salvador
	…		
	Ventas	Tegucigalpa	Honduras

Código Nombre Ciudad País

Las tablas en 3FN están en 2FN y las tablas 2FN están en 1FN



Normalizando una tercera forma normal

- Los campos que dependen unos de otros (sin ser claves candidatas) se separan en tablas diferentes, que se ligan a la tabla original con una clave foránea.

Una solución podría ser ésta

Áreas		
001	Compras	San Salvador
014	Producción	Santa Ana
056	Ventas	Tegucigalpa

Código Nombre Ciudad

Ciudades	
San Salvador	El Salvador
Santa Ana	El Salvador
Tegucigalpa	Honduras

Ciudad País

Mail: palasi@palasi.com Web: www.palasi.com

Resumen de la 2FN y de la 3FN

- Las formas normales se resumen en la siguiente condición:

"Todos los campos de un registro deben depender de las claves candidatas y sólo de ellas."

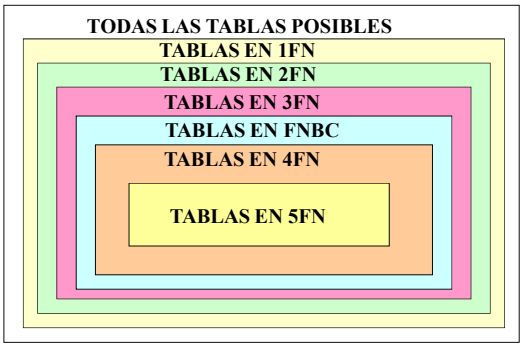
(la frase podría decir "y de todos los campos de ellas", pero veremos que sólo consideraremos claves de un solo campo)

- Esta es una condición para que las tablas estén bien definidas.

Otras formas normales

- Su violación es más difícil en la práctica.
- Forma normal de Boyce-Codd (FNBC).
- Cuarta forma normal (4FN).
- Quinta forma normal (5FN).
- Son de importancia más teórica que práctica.
- No la veremos aquí.

Relación entre las formas normales



Un último punto importante: La clave primaria

- La clave primaria puede ser un campo o varios. Puede ser entrada por el usuario, asignada automáticamente por el programa o por la BD.
- Sin embargo, por razones en las que no nos detendremos, se recomienda.
  - Que la clave primaria sea un solo campo
  - Que sea asignada por la BD como un campo autoincrementado (autonumérico).
- Hay algunas excepciones que veremos más adelante (como las llamadas "tablas de enlace").
- Esto nos evita muchos problemas y es lo que supondremos de ahora en adelante.

Para los curiosos. Las razones de lo que acabamos de decir.

- ¿Por qué sólo un campo?
- Si la clave primaria es más de un campo, esto complica las consultas, las claves foráneas y las relaciones de integridad referencial.

Para los curiosos. Las razones de lo que acabamos de decir.

- ¿Por qué sólo un campo?
- Si la clave primaria es más de un campo, esto complica las consultas, las claves foráneas y las relaciones de integridad referencial.

Para los curiosos. Las razones de lo que acabamos de decir.

- **¿Por qué no asignada por el usuario?**
- Si dejamos que la clave primaria la entre el usuario, Es difícil para el usuario asegurar la unicidad de la clave primaria. Esto implica programar índices y controles de nuestra parte, lo que complica el asunto.
- Además, si dejamos que la clave primaria la entre el usuario, cada vez que la cambie deberemos hacer una actualización en cascada por las claves foráneas, lo que es torpe e ineficiente. Ejemplo de las partidas.
- En cambio, un autonumérico no tiene un significado para el usuario. Por lo tanto, no lo cambiará. Es algo de estructura de la base de datos

Para los curiosos. Las razones de lo que acabamos de decir.

- ¿Por qué no asignada por el programa?
- Si la clave primaria es asignada por el programa, nos tendremos que encargar de programar la concurrencia para que sea única. En cambio, si lo dejamos a la BD, ésta se encarga de la concurrencia (mucho más sencillo).

Por ahora, conocemos lo básico del modelo relacional

Conceptos: BD, tabla, registro, campo, clave primaria, clave foránea y formas normales. Más adelante veremos más, pero por ahora es suficiente.

Departamentos

Código	Nombre	Jefe
001	Compras	566
014	Producción	927

Empleados

DUI	Nombre	Apellido	Edad	Teléfono
566	Juan	Pérez	30	567-3452
099	Claudia	Bonilla	20	764-9895
567	Raúl	López	54	
927	María	Portillo	54	576-4522
	...			
456	Roberto	Morán	45	722-4152

Ejercicio

- 1. El comedor de Doña Anita.

1. Introducción a las bases de datos relacionales.

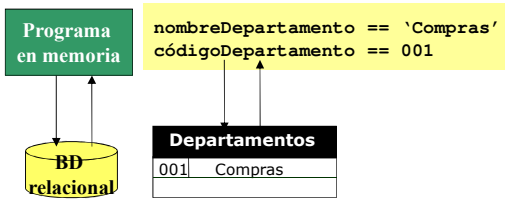
- 1.1. Breve repaso del modelo relacional.
- 1.2. Relación con la programación O-O.

Recordemos : persistencia de datos

- Propiedad por la cual un programa guarda sus datos en un soporte permanente para que se mantengan entre ejecuciones.
- Hay dos maneras principales de implementarla:
  - Persistencia de datos tradicional.
  - Persistencia de objetos.

Persistencia de datos tradicional

- El programa graba los datos directamente como registros usando las operaciones de una base de datos relacional.
- Es lo que todos tenemos en mente cuando pensamos "grabar datos en una base de datos".

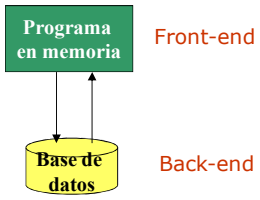


¿Qué problemas da la persistencia de datos tradicional?

- Si el programa que usa la BD no es orientado a objetos, no da ningún problema.
- Si el programa que usa la BD es orientado a objetos tenemos un problema, ya que los datos del programa son objetos.
- Las BDs relacionales no pueden guardar objetos de forma directa.
- ¿Por qué?

Tenemos dos modelos en una aplicación O-O

- Para el programa: Modelo orientado a objetos. El programa trata con objetos.
- Para la base de datos. Modelo relacional. La base de datos guarda registros.



Estos dos modelos son una pareja dispareja

- **Modelo orientado a objetos**: Modela conjuntamente los **datos y los procesos** de una aplicación.
- **Modelo relacional**: Modela el almacenamiento y recuperación de **datos** de una aplicación.

Diferencias entre ambos modelos

- El modelo orientado a objetos considera los procesos mientras el relacional no.
- Pero incluso en la forma de tratar los datos ambos modelos difieren.
- Por ejemplo, uno tiene estructura y el otro no.

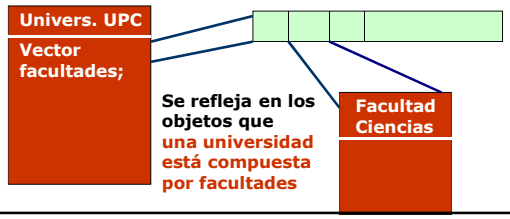
La realidad tiene una estructura

- Una universidad está compuesta de Facultades, cada una de las cuales está compuesta de Cursos y así sucesivamente.

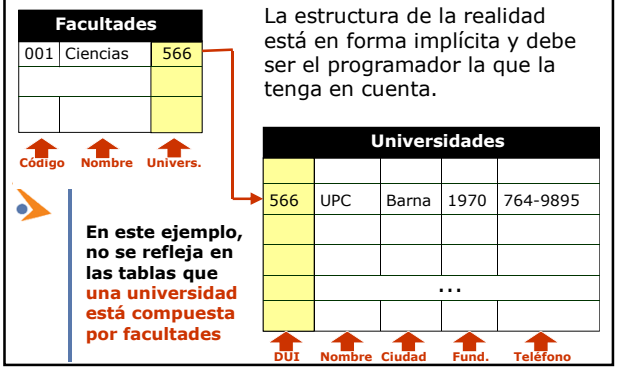
Mail: palasi@palasi.com Web: www.palasi.com

El modelo O-O refleja estructura de la realidad explícitamente

- Una universidad está compuesta de Facultades, cada una de las cuales está compuesta de Cursos y así sucesivamente.



El modelo relacional es relativamente plano



Otras diferencias entre modelo O-O y relacional

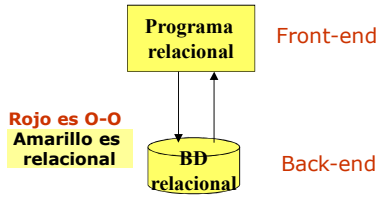
- Uno trata con objetos y otro con registros.
- Uno tiene herencia y otro no.
- Uno representa las relaciones como atributos, el otro como claves foráneas.

Cómo consecuencia

- No se pueden guardar objetos en una base de datos en forma directa.
- Entonces, ¿qué hacemos?
- Tenemos dos posibilidades:
  - Prescindir de la programación orientada a objetos.
  - Encontrar alguna forma de guardar objetos en una base de datos.

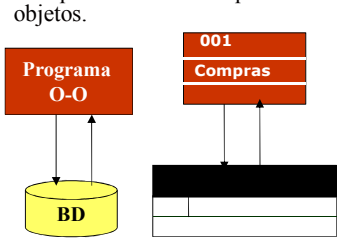
Prescindir de la programación O-O

- Esta es la forma tradicional
- El diseño de datos se hace de forma relacional.



Alguna forma de guardar objetos: Persistencia de objetos

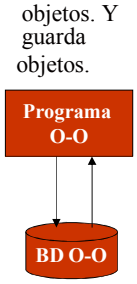
- Persistencia de objetos significa guardar los objetos en una base de datos.
- No tenemos suficiente con recuperar los datos de cualquier manera sino que éstos deben venir como objetos.



Rojo es O-O
Amarillo es relacional

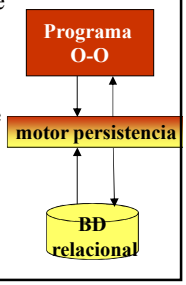
La persistencia de objetos puede implementarse de dos maneras

- O bien, la base de datos es orientada a objetos. Y guarda objetos.



- O bien, tenemos un motor de persistencia que traduce del modelo O-O al relacional y viceversa. Al guardar traduce los objetos a registros. Al recuperar, traduce los registros a objetos.

Esta es más común



Todo esto se produce en un contexto de programación en n-capas

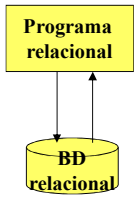
- Los objetos que guardamos son los de la capa de modelo, que modela la realidad.

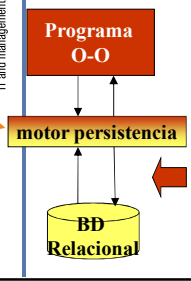
trabaja con Objetos de la capa de modelo

Objetos de la capa de modelo trabaja con Objetos de la capa de modelo

Resumiendo hay dos tipos de programar

- 1. Programar de forma relacional. El programa trabaja con registros y agrupaciones de registros (recordsets, resultsets, rowsets).





- 2. Programar de forma O-O con datos de objetos. Es decir, el programa trata con objetos.

Independientemente de esto, hay dos tipos de diseñar los datos de 1 aplicación

- El diseño relacional es el que veremos en este curso. A partir de diagramas entidad-relación se deducen las tablas, los campos, se diseña la BD tradicional, etc.
- El diseño O-O es el que se verá en un curso de análisis y diseño O-O. Se diseñan los objetos de una capa de modelo con análisis y diseño O-O.

Dicho de forma sencilla

- El diseño relacional. Se empieza diseñando tablas (si después queremos que el programa sea O-O habrá que ver cómo traducir estas tablas en objetos).
- El diseño O-O. Se empieza diseñando objetos (si después queremos guardar en una BD relacional habrá que ver cómo se traducen los objetos en tablas).

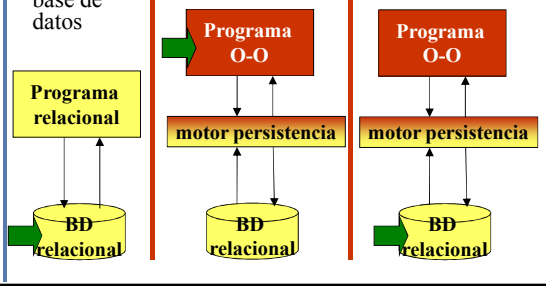
Tenemos dos formas de programar y dos formas de diseñar los datos

- Si los combinamos, salen tres posibilidades.
- El diseño es relacional y el programa es relacional.
- El diseño es O-O y el programa es O-O.
- El diseño es relacional y el programa es O-O.

Mail: palasi@palasi.com Web: www.palasi.com

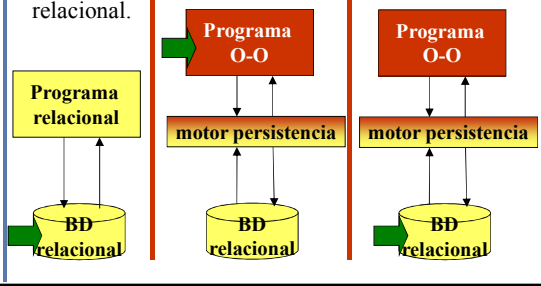
El diseño se hace donde indica la flecha verde

- Diseñamos relacional: tablas, base de datos
- Diseñamos O-O y el motor traduce a relacional
- Diseñamos relacional y el motor traduce a O-O.



Tres formas de programar

- Diseño relacional, programa relacional.
- Diseño O-O y programa O-O.
- Diseño relacional y programa O-O.



Ventajas e inconvenientes de estas tres formas

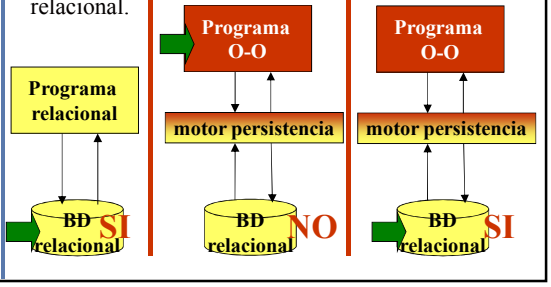
	Dis. y prog relacional.	Dis. y prog. O-O.	Dis. Rel. Prog O-O
Flexibilidad	Baja	Alta	Media
Eficiencia teórica	Alta (en la práctica, puede ser menor)	En teoría, algo menor que la anterior	Un poco más que la anterior
Aprov. pot. BD	Alta	Media	Alta
Adecuado para	Programas pequeños que no ameritan la sobrecarga de 1 motor	Programas que crean una BD nueva y necesitan flexibilidad en el mantenimiento	Que usan una BD preexistente o que desean para aprovechar más la potencia de la BD que con un diseño O-O.

Este curso trata sobre el diseño relacional de los datos

	Dis. y prog relacional.	Dis. y prog. O-O.	Dis. Rel. Prog O-O
Programa	Relacional	O-O	O-O
Diseño de datos	Relacional	O-O	Relacional
Curso en el que se tratará	Este curso	El curso de análisis y diseño O-O y, de forma más básica, en el curso de programación en Java.	Este curso

Se tratará el diseño relacional de los datos

- Diseño relacional, programa relacional.
- Diseño O-O y programa O-O.
- Diseño relacional y programa O-O.



Veremos pues el diseño relacional de los datos

- Esto puede aplicarse tanto a un programa tradicional relacional como a un programa orientado a objetos pero con diseño de datos relacional.
- Se puede hacer programación orientada a objetos con este diseño relacional.

Mail: palasi@palasi.com Web: www.palasi.com

Temario del curso

- 1. Introducción a las bases de datos relacionales.
- 2. Creación de la estructura de bases de datos.
- 3. Recuperación y actualización de datos.
- 4. Vistas.
- 5. Seguridad. Permisos de usuario.
- 6. OLAP (Data warehousing)

2. Creación de la estructura de bases de datos.

- 2.1. Método de modelización sencillo para el diseño de bases de datos.
- 2.2. Método de modelización complejo para el diseño de bases de datos.
- 2.3. Introducción a SQL Server.
- 2.4. Instalación de SQL Server.
- 2.5. Principales herramientas de SQL Server.
- 2.6. Creación de bases de datos y tablas en SQL Server.
- 2.7. Índices.

2. Creación de la estructura de bases de datos.

- 2.1. Método de modelización sencillo para el diseño de bases de datos.
- 2.2. Método de modelización complejo para el diseño de bases de datos.
- 2.3. Introducción a SQL Server.
- 2.4. Instalación de SQL Server.
- 2.5. Principales herramientas de SQL Server.
- 2.6. Creación de bases de datos y tablas en SQL Server.
- 2.7. Índices.

Método sencillo de modelización de datos

- Para diseñar la estructura de datos hay métodos bien establecidos.
- Nosotros vamos a ver dos métodos. Uno lo llamaremos "el método sencillo" y el otro "el método complejo".
- El primero es adecuado para diseño de pequeñas bases de datos. El segundo para bases más completas.
- Comenzaremos con el "método sencillo", que se deriva directamente de las formas normales y, concretamente, del resumen que habíamos hecho "Todo dato de una tabla debe depender de las claves candidatas y sólo de ellas"

Método sencillo de modelización de datos

- 1. Se escribe en una lista todos los datos del programa. Se les da un nombre y un tipo.
- 2. Si ya se conocen algunas tablas se dividen los campos entre ellas. Si no, se supone que es una sola tabla con todos los campos.
- 3. Se identifica la clave primaria de cada una de las tablas.
- 4. Todo dato de una tabla debe depender de las claves candidatas y sólo de ellas. Si esto se cumple, ya hemos acabado.
- 5. En caso de que esto no se cumpla los datos que no lo cumplan deben ir en otras tablas. Debemos crear nuevas tablas, ligarlas a la primera con una clave foránea y volver al punto 3.

Ejemplo de método de modelización de datos

- Programa de inscripción de asistentes a un congreso. Hay tres tipos de tarifas: estudiante, invitado y normal cuyos precios respectivos son 50, 100 y 200.
- Si el país de origen del asistente es centroamericano, el asistente llegará por autobús. En caso contrario, vendrá por avión y habrá que ir a recogerlo en el aeropuerto.
- De cada asistente nos interesa su nombre, apellidos, número de DUI, nacionalidad, qué tipo de tarifa, qué cantidad debe pagar, teléfono, dirección en el país, hotel en el que se aloja y si debemos recogerlo en el aeropuerto o no.

Mail: palasi@palasi.com Web: www.palasi.com

Método sencillo de modelización de datos

- 1. Se escribe en una lista todos los datos del programa. Se les da un nombre y un tipo.
- 2. Si ya se conocen algunas tablas se dividen los campos entre ellas. Si no, se supone que es una sola tabla con todos los campos.
- 3. Se identifica la clave primaria de cada una de las tablas.
- 4. Todo dato de una tabla debe depender de las claves candidatas y sólo de ellas. Si esto se cumple, ya hemos acabado.
- 5. En caso de que esto no se cumpla los datos que no lo cumplan deben ir en otras tablas. Debemos crear nuevas tablas, ligarlas a la primera con una clave foránea y volver al punto 3.

1. Lista de todos los datos

- Nombre. (nombre). Texto(50)
- Apellidos (apellidos) Texto(50)
- DUI (DUI) Numérico(20).
- Nacionalidad (pais) Texto(50).
- Tipo de tarifa (tipotarifa) Texto(10).
- Cantidad a pagar (importe) Numérico (3).
- Teléfono (telefono). Texto(7).
- Dirección en el país (direccion). Texto (150).
- Hotel (hotel). Texto (50).
- ¿Recoger en el aeropuerto? (avion) Lógico (1).

Método sencillo de modelización de datos

- 1. Se escribe en una lista todos los datos del programa. Se les da un nombre y un tipo.
- 2. Si ya se conocen algunas tablas se dividen los campos entre ellas. Si no, se supone que es una sola tabla con todos los campos.
- 3. Se identifica la clave primaria de cada una de las tablas.
- 4. Todo dato de una tabla debe depender de las claves candidatas y sólo de ellas. Si esto se cumple, ya hemos acabado.
- 5. En caso de que esto no se cumpla los datos que no lo cumplan deben ir en otras tablas. Debemos crear nuevas tablas, ligarlas a la primera con una clave foránea y volver al punto 3.

2. No hay información sobre tablas

- Por ello, supondremos por ahora que sólo existe una tabla con todos los campos.
- El nombre que parece más lógico para ella es el nombre de “asistentes”.

Método sencillo de modelización de datos

- 1. Se escribe en una lista todos los datos del programa. Se les da un nombre y un tipo.
- 2. Si ya se conocen algunas tablas se dividen los campos entre ellas. Si no, se supone que es una sola tabla con todos los campos.
- 3. Se identifica la clave primaria de cada una de las tablas.
- 4. Todo dato de una tabla debe depender de las claves candidatas y sólo de ellas. Si esto se cumple, ya hemos acabado.
- 5. En caso de que esto no se cumpla los datos que no lo cumplan deben ir en otras tablas. Debemos crear nuevas tablas, ligarlas a la primera con una clave foránea y volver al punto 3.

3. Identificar claves primarias

- Nombre. T(50)
- Apellidos T(50)
- DUI N(20).
- Pais T(50).
- Tipotarifa T(10).
- Importe N(3).
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).
- Avion L(1).

- Debe identificar unívocamente al asistente.
- Nombre y apellidos identifican pero no unívocamente.
- No hay ningún campo que sea clave candidata (es decir, que pueda ser clave primaria).

3. Identificar claves primarias

- Como no hay ningún campo que pueda ser clave primaria y necesitamos una clave primaria, debemos introducir un nuevo campo.
- Lo haremos de tipo autonumérico pues ya hemos visto que es lo mejor para las claves primarias.
- Le llamaremos "CodAsist" (en este caso, lo podemos aprovechar como código de inscripción).

3. Identificar claves primarias

- CodAsist. Autonum(6). CLAVE
- Nombre. T(50)
- Apellidos T(50)
- DUI N(20).
- Pais T(50).
- Tipotarifa T(10).
- Importe N(3).
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).
- Avion L(1).

Método sencillo de modelización de datos

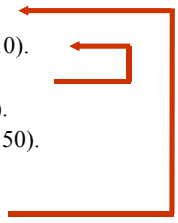
1. Se escribe en una lista todos los datos del programa. Se les da un nombre y un tipo.
2. Si ya se conocen algunas tablas se dividen los campos entre ellas. Si no, se supone que es una sola tabla con todos los campos.
3. Se identifica la clave primaria de cada una de las tablas.
4. Todo dato de una tabla debe depender de las claves candidatas y sólo de ellas. Si esto se cumple, ya hemos acabado.
5. En caso de que esto no se cumpla los datos que no lo cumplan deben ir en otras tablas. Debemos crear nuevas tablas, ligarlas a la primera con una clave foránea y volver al punto 3.

4. Los datos deben depender de las claves candidatas y sólo de ellas

- CodAsist. A(6). CLAVE. Identif. asistente
- Nombre. T(50). Depende de clave, del asistente.
- Apellidos T(50). Depende de clave.
- DUI N(20). Depende de clave.
- Pais T(50). Depende de clave.
- Tipotarifa T(10). Depende de clave.
- Importe N(3). Depende de clave y de tipotarifa.
- Telefono T(7). Depende de clave.
- Direccion T(150). Depende de clave.
- Hotel. T(50). Depende de clave.
- Avion L(1). Depende de clave y de pais.

4. Los datos deben depender de las claves candidatas y sólo de ellas

- CodAsist. A(6). **CLAVE (primaria pero también la única candidata).**
- Nombre. T(50).
- Apellidos T(50).
- DUI N(20).
- Pais T(50).
- Tipotarifa T(10).
- Importe N(3).
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).
- Avion L(1).



4.Hay campos que dependen de otros que no son la clave

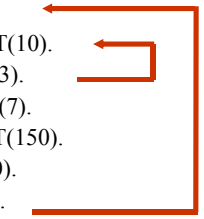
- Tal como dice el método, separamos estos campos en una nueva tabla y la ligamos con una clave foránea.

Método sencillo de modelización de datos

- 1. Se escribe en una lista todos los datos del programa. Se les da un nombre y un tipo.
- 2. Si ya se conocen algunas tablas se dividen los campos entre ellas. Si no, se supone que es una sola tabla con todos los campos.
- 3. Se identifica la clave primaria de cada una de las tablas.
- 4. Todo dato de una tabla debe depender de las claves candidatas y sólo de ellas. Si esto se cumple, ya hemos acabado.
- 5. En caso de que esto no se cumpla los datos que no lo cumplan deben ir en otras tablas. Debemos crear nuevas tablas, ligarlas a la primera con una clave foránea y volver al punto 3.

5. Separando los datos en tablas

- CodAsist. A(6). CLAVE PRIMARIA y CANDIDA.
- Nombre. T(50).
- Apellidos T(50).
- DUI N(20).
- Pais T(50).
- Tipotarifa T(10).
- Importe N(3).
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).
- Avion L(1).



5. Separando los datos en tablas

Asistentes

- CodAsist. A(6). Nombre. T(50).
- Apellidos T(50).
- DUI N(20).
- <clave foránea tabla pais>
- <clave foránea tabla tipo de tarifa>
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).

TiposTarifa

- TipoTarifa T(10)
- Importe N(3)

Paises

- Pais T(50)
- Avion L(1)

Método sencillo de modelización de datos

- 1. Se escribe en una lista todos los datos del programa. Se les da un nombre y un tipo.
- 2. Si ya se conocen algunas tablas se dividen los campos entre ellas. Si no, se supone que es una sola tabla con todos los campos.
- 3. Se identifica la clave primaria de cada una de las tablas.
- 4. Todo dato de una tabla debe depender de las claves candidatas y sólo de ellas. Si esto se cumple, ya hemos acabado.
- 5. En caso de que esto no se cumpla los datos que no lo cumplan deben ir en otras tablas. Debemos crear nuevas tablas, ligarlas a la primera con una clave foránea y volver al punto 3.

3. Identificar claves primarias

- TipoTarifa y Pais podrían ser claves primarias pues identifican unívocamente la tabla.
- Pero, como hemos visto, no es buena práctica que las claves primarias puedan ser introducidas por el usuario.

TiposTarifa

- TipoTarifa T(10)
- Importe N(3)

Paises

- Pais T(50)
- Avion L(1)

Recordemos: ¿Por qué no claves primarias introducidas por el usuario?

- Si dejamos que la clave primaria la entre el usuario, Es difícil para el usuario asegurar la unicidad de la clave primaria. Esto implica programar índices y controles de nuestra parte, lo que complica el asunto.
- Además, si dejamos que la clave primaria la entre el usuario, cada vez que la cambie deberemos hacer una actualización en cascada por las claves foráneas, lo que es torpe e ineficiente. Ejemplo de las partidas.
- En cambio, un autonumérico no tiene un significado para el usuario. Por lo tanto, no lo cambiará. Es algo de estructura de la base de datos

3. Identificar claves primarias

- Colocaremos dos campos nuevos que no sean introducidos por el usuario para ser clave primaria.
- Es conveniente que sean campos autonuméricos

TiposTarifa
- CodTarifa A(3)
- TipoTarifa T(10)
- Importe N(3)

Paises
- CodPais A(3)
- Pais T(50)
- Avion L(1)

Claves

Determinando cuáles son las claves foráneas

Asistentes
- CodAsist. A(6).
- Nombre. T(50).
- Apellidos T(50).
- DUI N(20).
- CodPais N(3)
- CodTarifa N(3)
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).

TiposTarifa
- CodTarifa A(3)
- TipoTarifa T(10)
- Importe N(3)

Paises
- CodPais A(3)
- Pais T(50)
- Avion L(1)

Método sencillo de modelización de datos

1. Se escribe en una lista todos los datos del programa. Se les da un nombre y un tipo.
2. Si ya se conocen algunas tablas se dividen los campos entre ellas. Si no, se supone que es una sola tabla con todos los campos.
3. Se identifica la clave primaria de cada una de las tablas.
4. Todo dato de una tabla debe depender de las claves candidatas y sólo de ellas. Si esto se cumple, ya hemos acabado.
5. En caso de que esto no se cumpla los datos que no lo cumplan deben ir en otras tablas. Debemos crear nuevas tablas, ligarlas a la primera con una clave foránea y volver al punto 3.

4. Los datos deben depender de las claves candidatas

Asistentes
- CodAsist. A(6).
- Nombre. T(50).
- Apellidos T(50).
- DUI N(20).
- CodPais N(3)
- CodTarifa N(3)
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).

TiposTarifa
- CodTarifa A(3)
- TipoTarifa T(10)
- Importe N(3)

Paises
- CodPais A(3)
- Pais T(50)
- Avion L(1)

CodTarifa y TipoTarifa
CodPais y Pais
son candidatas.

Ya hemos acabado

Asistentes
- CodAsist. A(6).
- Nombre. T(50).
- Apellidos T(50).
- DUI N(20).
- CodPais N(3)
- CodTarifa N(3)
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).

TiposTarifa
- CodTarifa A(3)
- TipoTarifa T(10)
- Importe N(3)

Paises
- CodPais A(3)
- Pais T(50)
- Avion L(1)

Una observación

- Este ejercicio se ha realizado sobre el modelo relacional sin entrar en detalles específicos de una base de datos.
- Según sea la base de datos, se deberá adaptar algunos aspectos menores del resultado
- Por ejemplo, en SQL Server, el campo autonumérico se llama "identity". En Oracle, sería una secuencia.

Mail: palasi@palasi.com Web: www.palasi.com

Ejercicio de modelización de datos

- 2. Ejercicio de modelización sencillo

2. Creación de la estructura de bases de datos.

- 2.1. Método de modelización sencillo para el diseño de bases de datos.
- 2.2. Método de modelización complejo para el diseño de bases de datos.
- 2.3. Introducción a SQL Server.
- 2.4. Instalación de SQL Server.
- 2.5. Principales herramientas de SQL Server.
- 2.6. Creación de bases de datos y tablas en SQL Server.
- 2.7. Índices.

El método de modelización anterior es a veces demasiado engorroso

- Sobre todo, cuando hay demasiados datos. Escribirlos todos y aplicar el método mecánico puede ser una pérdida de tiempo, cuando hay datos que ya se saben que se reúnen en una tabla.
- Presentamos un método más real para problemas grandes de modelización.

Método de modelización "complejo"

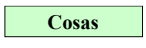
- 1. Realizar un diagrama entidad-relación de Chen.
- 2. Añadir los atributos al diagrama.
- 3. Traducir este diagrama a tablas.
- 4. Aplicar el método de modelización "sencillo" que acabamos de ver para asegurarnos de que todos los campos dependen de la clave primaria y sólo de ella.

Comenzamos viendo qué es un diagrama entidad-relación de Chen

- Modelo semántico que indica la relación entre los datos del programa.
- Es bastante antiguo (años 1975-76).
- Es útil cuando deseamos realizar un diseño de una base de datos relacional.

Un diagrama entidad-relación consta de 2 elementos

- Entidad. Representa un conjunto de "cosas" de la vida real. Se representa por un rectángulo.



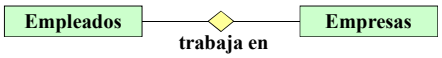
- Relación. Representa una relación entre dos entidades. Se representa con una línea con un rombo que une las entidades.

Mail: palasi@palasi.com Web: www.palasi.com

Ejemplo

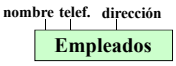
- Entidades. Todos los alumnos, todos los empleados, todos los departamentos, etc.

- Relación. Representa una relación entre dos entidades. Se representa con una línea con un rombo que une las entidades.

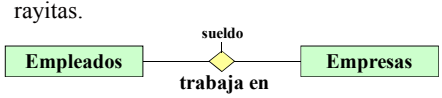


Tanto las entidades como las relaciones pueden tener atributos

- Que son propiedades de las entidades o relaciones.



- Se pueden escribir por aparte o señalarse con rayitas.



¿Esto no nos está sonando mucho?

- Fijémonos que esto es muy parecido al modelo de dominio. Los dos modelan el mundo real.
- Las entidades serían las clases conceptuales.
- Las relaciones serían las asociaciones entre clases conceptuales.
- Los atributos de las entidades son los atributos de las clases conceptuales.

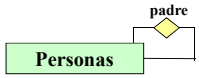
Sin embargo, hay diferencias,

Además de la notación:

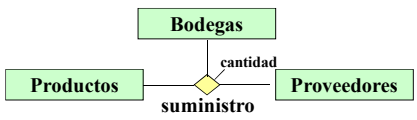
- En el diagrama entidad-relación sólo hay los datos que acabarán en el sistema.
- En entidad-relación, las entidades son algo estático que tiene atributos, pero no tiene comportamientos (métodos).
- Esto refleja la diferencia entre el modelo O-O que modela datos y procesos y el relacional, que sólo modela datos.

Pueden darse estos casos

- Una relación de una entidad con ella misma. Una persona es padre de otra persona.

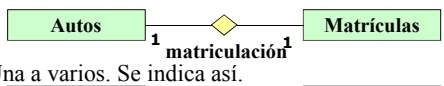


- Una relación entre más de dos entidades. Un proveedor suministra un producto a una bodega. Fíjense que esta relación tiene un atributo.

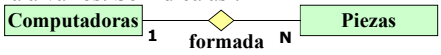


Las relaciones tienen multiplicidad

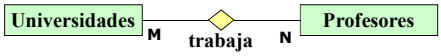
- Que indican cuantos elementos de una entidad se relacionan con uno de otra entidad. Hay tres tipos de multiplicidad:
- Una a una. Se indica así.



- Una a varios. Se indica así.



- Varios a varios. Se indica así.



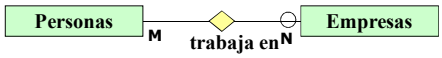
Cada extremo puede tener un circulito

- Que indican que la multiplicidad puede ser cero.

- Un empleado puede trabajar en varias empresas o en ninguna (este "ninguna" es el circulito).
- Los circulitos no son demasiado importantes y muchas veces se omiten.

Para hallar la multiplicidad

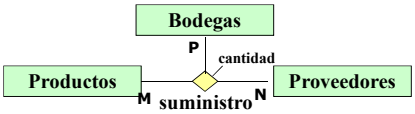
- Se pregunta, ¿cuántos elementos de una entidad se relacionan con otra entidad?



- ¿Cuántos empleados se relacionan con una empresa? Varios (M).
- ¿Cuántas empresas se relacionan con un empleado? Varias o ninguna (M o el circulito).

Para hallar la multiplicidad en relaciones con más de dos entidades

- ¿Cuántos proveedores suministran un producto en una bodega? N.
- ¿Cuántos productos son suministrados en una bodega por un proveedor? M
- ¿En cuántas bodegas un proveedor suministra un producto? P.
- Como ven, siempre es ¿cuántos X es 1 de los otros?



Método de modelización "complejo"

- 1. Realizar un diagrama entidad-relación de Chen.
- 2. Añadir los atributos al diagrama.
- 3. Traducir este diagrama a tablas.
- 4. Aplica… "sencillo" … asegurarnos de que todos los campos dependen de la clave primaria y sólo de ella.

Hemos visto el paso 1 y el 2.

Método de modelización "complejo"

- 1. Realizar un diagrama entidad-relación de Chen.
- 2. Añadir los atributos al diagrama.
- 3. Traducir este diagrama a tablas.
- 4. Aplicar el método de modelización "sencillo" que acabamos de ver para asegurarnos de que todos los campos dependen de la clave primaria y sólo de ella.

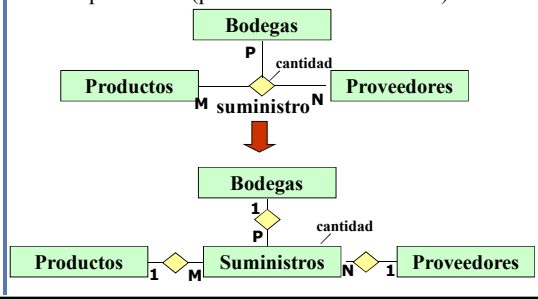
Para traducir el diagrama E-R a tablas se siguen los siguientes pasos

- 3.1. Se eliminan las relaciones que son entre más de dos entidades.
- 3.2. Se eliminan las relaciones 1 a 1.
- 3.3. Se eliminan las relaciones M a N.
- 3.4. Se obtienen las tablas con sus campos.
- 3.5. Se obtienen las claves primarias.
- 3.6. Se obtienen las claves foráneas.

Mail: palasi@palasi.com Web: www.palasi.com

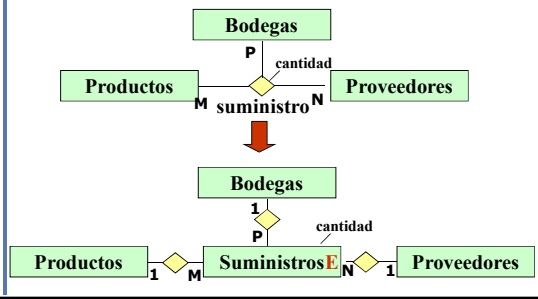
3.1 Se eliminan las relaciones que son entre más de dos entidades

- La relación se convierte en una entidad y se crean relaciones binarias. Se deben volver a pensar las multiplicidades (pueden salir 1 a 1 o 1 a N).



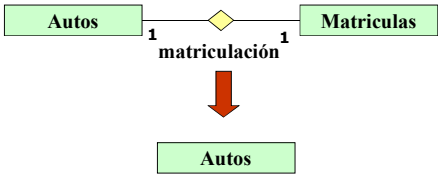
Para marcar las nuevas entidades que aparecen y distinguirlas de existentes

- Se pueden marcar con una "E", que significa "entidades de enlace"



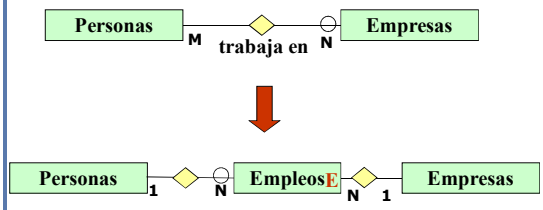
3.2. Eliminar las relaciones 1 a 1

- Si la relación es 1 a 1, se unen las dos entidades (los atributos de la nueva entidad son los que tenían las dos entidades existentes).



3.3. Eliminar las relaciones M a N

- Si la relación es M a N, se convierte la relación en una entidad. Aparecen dos relaciones 1 a N (el N se encuentra en la nueva entidad). La nueva entidad se marca con una "E", que significa "entidad de enlace".

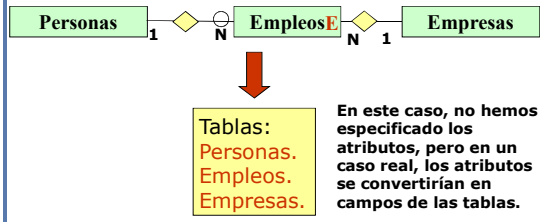


Con estos primeros tres pasos

- Las únicas relaciones que quedan son de 1 a N.
- Todas las otras las hemos eliminado.

3.4. Se obtienen las tablas con sus campos

- Tal como tenemos el diagrama, las tablas son las entidades que han quedado. Los campos de una tabla son los atributos de la entidad correspondiente. Se pone tipo y longitud de campos.



3.5. Se obtienen las claves primarias

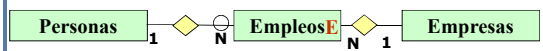
- Cada tabla que no sea de enlace tendrá una clave primaria, que será un autonumérico (como hemos dicho).
- Las de enlace también se les puede poner un autonumérico como clave primaria (es mejor para el mantenimiento), pero no es necesario.

Tablas:
Personas. Empleos. Empresas.

Tablas:
Personas. Campos: CodPersona A(6) (clave).
Empleos. Campos: Ninguno por ahora.
Empresas. Campos: CodEmpresa A(6) (clave).

3.6. Se obtienen las claves foráneas

- Cada relación que haya quedado quiere decir que habrá una clave foránea en el extremo N.



Tablas:
Personas. Campos: CodPersona A(6) .
Empleos. Campos: **CodPersona N(6). CodEmpresa N(6)**
Empresas. Campos: CodEmpresa A(6).

Como vemos, la tabla de enlace queda con una clave primaria compuesta.

Y ya está. Ya hemos convertido el diagrama E-R en tablas.

- Tendremos una lista de tablas con sus campos (tipos, nombre y longitud).

Método de modelización "complejo"

- 1. Realizar un diagrama entidad-relación de Chen.
- 2. Añadir los atributos al diagrama.
- 3. Traducir este diagrama a tablas.
- 4. Aplicar el método de modelización "sencillo" que acabamos de ver para asegurarnos de que todos los campos dependen de la clave primaria y sólo de ella.

Aplicamos el método de modelización sencillo

- Así vemos si todo ha quedado correcto.
- Acabaremos cuando se cumpla el resumen de las formas normales: "todos los campos de una tabla deben depender de las claves candidatas y sólo de ellas".

Ejercicio

- 3. Agencias de viaje.
- Si hay tiempo,
- 4. Información policial.

Mail: palasi@palasi.com Web: www.palasi.com

2. Creación de la estructura de bases de datos.

- 2.1. Método de modelización sencillo para el diseño de bases de datos.
- 2.2. Método de modelización complejo para el diseño de bases de datos.
- 2.3. Introducción a SQL Server.
- 2.4. Instalación de SQL Server.
- 2.5. Principales herramientas de SQL Server.
- 2.6. Creación de bases de datos y tablas en SQL Server.
- 2.7. Índices.

SQL Server

- Sistema Gestor de Base de datos (SGBD) relacional con arquitectura cliente-servidor desarrollado por Microsoft.
- Algunos límites:
  - El máximo tamaño de una BD es de 1,048,516 terabytes. Con miles de terabytes no se detectan problemas de desempeño.
  - El número de transacciones por minuto es de 709,220.
  - El número máximo de usuarios concurrentes es ilimitado, pero puede servir con eficiencia 26000 en un único servidor (SAP benchmark).
- SQL Server funciona de forma adecuada incluso en los entornos más exigentes, siempre que el entorno sean el adecuado.

Historia de SQL Server

- En 1988, Microsoft y Sybase Corporation desarrollaron SQL Server para el SO OS/2 de IBM.
- En los primeros años noventa, Microsoft rompió con IBM e implementó SQL Server para su nuevo sistema operativo Windows NT, ligándolo a él.
- En 1993, Windows NT 3.1 y SQL Server 4.2.
- En 1994, Microsoft y Sybase rompieron. Continuaron desarrollando el producto, que se dividió en dos: SQL Server (Microsoft) y la BD de Sybase.
- En 1998, apareció SQL Server 7.0. En 2000, SQL Server 2000.
- En 2005 habrá SQL Server 2005 (Yukon). Ya hay betas disponibles.

Relación con otros productos

- Dado el origen común del SGBD Sybase y SQL Server, las versiones antiguas de ambos productos se parecen mucho, aunque las versiones más modernas difieren.
- Oracle y SQL Server se parecen poco, excepto que los conceptos son los mismos, pues se basan en el modelo relacional.
- Access es una base de datos pequeña para hasta 20 usuarios y unos 50000 registros. Sus funciones están limitadas.

Comparación con Oracle

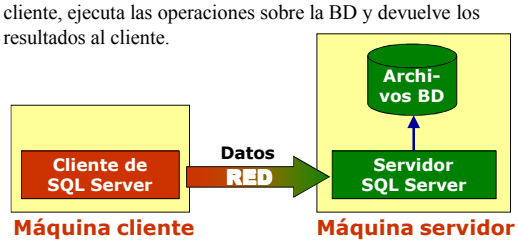
- Ventajas de SQL Server:
  - Mucho más barato. Relación calidad/precio más alta.
  - Más fácil de instalar y administrar.
  - No demanda tantos recursos.
- Ventajas de Oracle.
  - Más estable y confiable.
  - Está disponible en todas las plataformas y no sólo en Windows.
  - El lenguaje (PL/SQL) es más potente que el de SQL Server (Transact-SQL).
  - Permite un ajuste más fino de las propiedades.

SQL Server tiene una arquitectura cliente/servidor

Una sesión de SQL Server consta de dos procesos

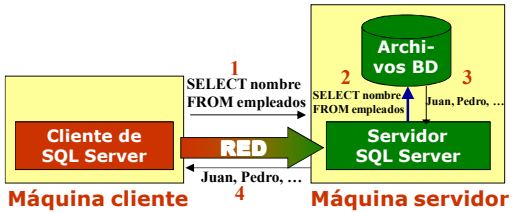
•El proceso cliente, que requiere operaciones sobre la base de datos.

•El proceso servidor que acepta estos requerimientos del cliente, ejecuta las operaciones sobre la BD y devuelve los resultados al cliente.



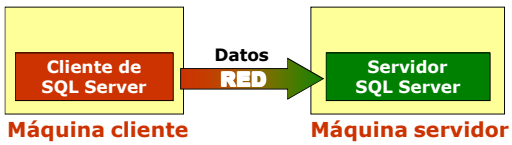
Ejemplo.

Pide el nombre de todos los empleados de una empresa.



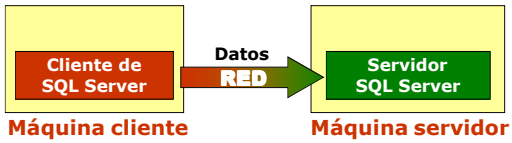
A partir de ahora se dibujará así

- No especificaremos por separado el almacenamiento de los archivos de BD.
- Aunque siempre el servidor accederá a estos archivos, dibujarlos complicaría el gráfico.
- Supondremos siempre que el servidor debe acceder a los archivos, aunque no los dibujemos.



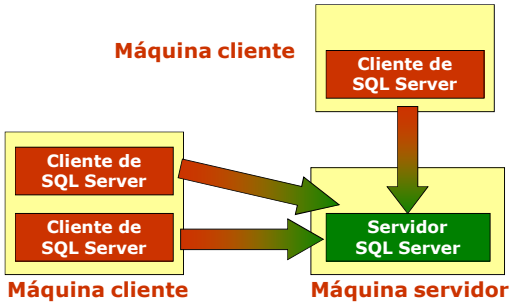
Cliente y servidor pueden estar en máquinas diferentes

- O bien en la misma máquina.
- En todo caso, se comunican mediante diversos protocolos de red (lenguajes para la comunicación de red).
  - Named pipes.
  - TCP/IP.
  - Multiprotocolo.
  - Otros.



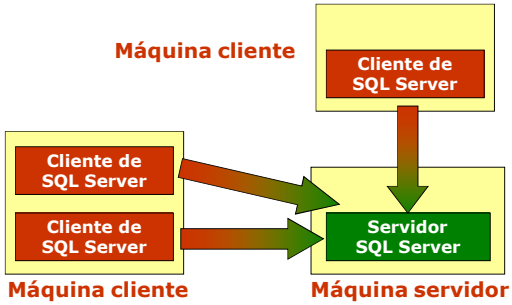
El servidor es sólo uno.

- Es un proceso que accede a los archivos de la base de datos .



Los clientes pueden ser muchos

- Un único servidor puede manejar múltiples requerimientos de los clientes al mismo tiempo.



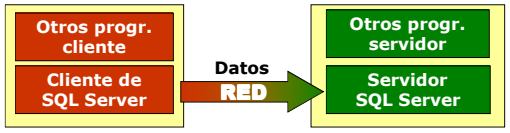
Los clientes pueden ser muchos y variados

- Clientes pueden ser las herramientas de SQL Server para consultar o modificar datos.
- Puede ser un programa que accede a la base de datos para buscar sus datos.
- Puede ser un servidor de Web que busca los datos para generar una página Web con ellos.
- Etcétera.

Mail: palasi@palasi.com Web: www.palasi.com

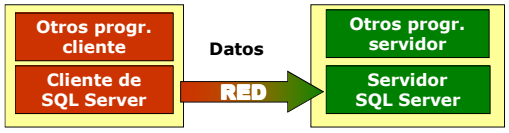
A parte del servidor y el cliente, SQL Server tiene otros programas

- Son utilidades que nos ayudan a realizar trabajos útiles con la base de datos.
- Según si se encuentran en la máquina cliente o en la máquina servidor, podemos llamarlas "otros programas del cliente" y "otros programas del servidor".



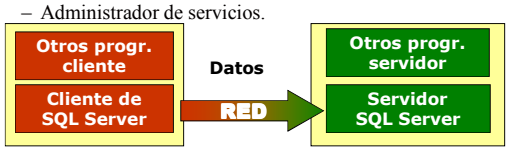
SQL Server tiene dos tipos de programas

- Herramientas cliente. Incluye el acceso a la BD desde el cliente y otros programas que hay en la máquina cliente.
- Herramientas servidor. Incluye el servidor de BD y otros programas que hay en la máquina servidor.



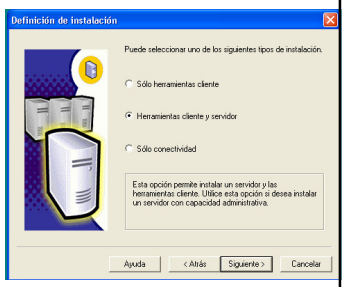
SQL Server tiene dos tipos de programas

- Herramientas cliente (algunas).
  - Librerías de acceso a la BD desde el cliente (conectividad del cliente)
  - Administrador corporativo
  - Analizador de consultas.
- Herramientas servidor (algunas).
  - Servidor de la BD.
  - Administrador de servicios.



SQL Server tiene tres tipos de instalaciones

- Sólo conectividad.
- Sólo herramientas cliente.
- Herramientas cliente y servidor.



SQL Server tiene tres tipos de instalaciones

- Sólo conectividad. Es la instalación mínima para las máquinas clientes. Sólo instala las librerías necesarias para que los programas se conecten al servidor. No instala ninguna utilidad de BD.
- Sólo herramientas cliente. Es la instalación típica para las máquinas clientes. Instala todas las herramientas del cliente.
- Herramientas cliente y servidor. Es la instalación típica para la máquina servidor. Instala todas las herramientas del cliente y del servidor.

¿Cómo instalaremos SQL Server?

- En las máquinas clientes instalaremos con la opción "herramientas clientes" y en la máquina servidor instalaremos las "herramientas cliente y servidor".

Mail: palasi@palasi.com Web: www.palasi.com

2. Creación de la estructura de bases de datos.

Más detalle. Instalando SQL Server

- Veremos la versión de evaluación.
- Consta de los siguientes pasos.

- 1. Descargar la versión de evaluación.
- 2. Instalar la base de datos.
- 3. Instalar Analysis Services.
- 4. Descargar el Service Pack 3
- 5. Instalar el Service Pack 3.

1. Descargar la versión de evaluación

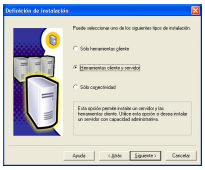
- La versión de evaluación se baja de http://www.microsoft.com/sql/evaluation/trial/ haciendo clic en el enlace "Register and download SQL Server trial software".
- Se debe tener una cuenta en Hotmail, de la que nos pedirá la contraseña.
- Se selecciona el idioma y se hace clic en el botón de "Descargar".

2. Instalar la base de datos (1)

- Como resultado de la descarga, aparecerá un archivo autodescomprimible "ESN_SQLEVAL.EXE". Ejecutándolo, se descomprimirá y generará un directorio con su contenido.
- En ese directorio, ejecutaremos "autorun.exe".
- En él, elegimos "Componentes de SQL Server 2000" y, después, "Instalar Servidor de bases de datos".

2. Instalar la base de datos (2)

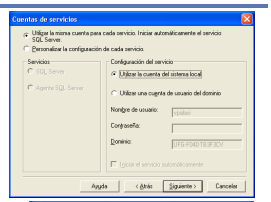
- Todas las opciones son las predeterminadas (Hay que hacer "Siguiente") excepto las siguientes pantallas.



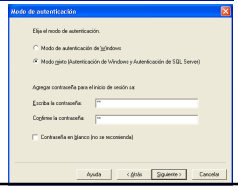
- Como hemos visto, en las máquinas clientes, instalaremos con la opción "sólo herramientas clientes" y en la máquina servidor instalaremos las "herramientas cliente y servidor".

2. Instalar la base de datos (3)

- En esta pantalla, se elige "Utilizar la cuenta del sistema local"



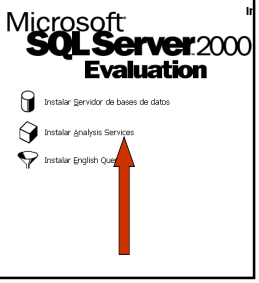
- En esta se elige "Modo mixto" y se escribe la contraseña del administrador.



Mail: palasi@palasi.com Web: www.palasi.com

3. Instalar Analysis Services

- Ejecutaremos "autorun.exe" y, en él, elegimos "Componentes de SQL Server 2000" y, después, "Instalar Analysis Services".
- Se conservan todas las opciones predeterminadas.



4. Descargar el Service Pack 3

- Se descarga de http://www.microsoft.com/sql/downloads/2000/sp3.asp.
- Se elige lenguaje y se descargan los 3 archivos
  - esn_sql2kasp3.exe
  - esn_sql2kdesksp3.exe
  - esn_sql2ksp3.exe

5. Instalar Service Pack 3

- Cada uno de estos archivos se ejecuta y se descomprimirá en un directorio.
- En ese directorio, ejecutamos "setup.exe" con todas las opciones predeterminadas.

2. Creación de la estructura de bases de datos.

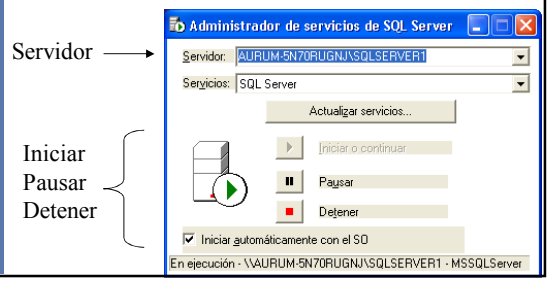
- 2.1. Método de modelización sencillo para el diseño de bases de datos.
- 2.2. Método de modelización complejo para el diseño de bases de datos.
- 2.3. Introducción a SQL Server.
- 2.4. Instalación de SQL Server.
- 2.5. Principales herramientas de SQL Server.
- 2.6. Creación de bases de datos y tablas en SQL Server.
- 2.7. Índices.

Una herramienta del servidor que veremos

- Administrador de servicios. Se encarga de controlar los servidores de bases de datos que tenemos funcionando.
- Se accede a ella desde **Programas | Microsoft SQL Server | Administrador de servicios**.
- La veremos de forma simplificada.

Administrador de servicios

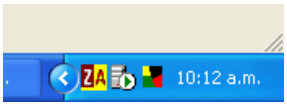
- Desde aquí, podemos detener, pausar o iniciar el servidor de SQL Server.



Mail: palasi@palasi.com Web: www.palasi.com

Otras forma de acceder al Administrador de servicios

- En la bandeja del sistema hay un icono que representa un servidor.



- Haciendo clic derecho podemos iniciar, pausar o detener el servidor. Haciendo doble clic se abre el administrador de servicios.

Algunas herramientas del cliente que veremos

- **Administrador corporativo.** Sirve para administrar los diferentes servidores de nuestra instalación junto con sus bases de datos y usuarios.
- **Analizador de consultas.** Sirve para ejecutar consultas y actualizaciones SQL directamente dentro de una base de datos. Cuando queramos ejecutar una sentencia SQL lo haremos siempre desde el analizador de consultas.

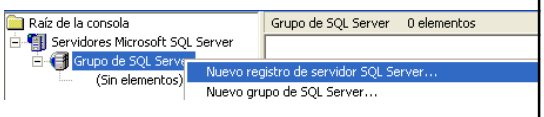
Cómo acceder a ellas

- **Administrador corporativo.** Hay que acceder a **Programas | Microsoft SQL Server | Administrador corporativo**.
- **Analizador de consultas.** Dos formas:
- **Programas | Microsoft SQL Server | Analizador de consultas.**
- Desde el administrador corporativo, ejecutar la opción **Herramientas | Analizador de consultas SQL**

Configurando el administrador corporativo

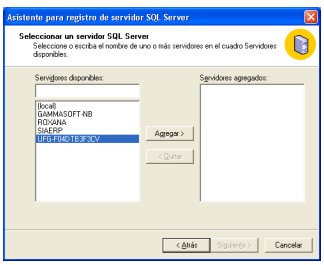
- Lo primero que debemos hacer es registrar nuestro servidor.
- Registrar un servidor significa indicarle al administrador corporativo que queremos usarlo.
- Esto sólo se debe hacer una vez por servidor. Como normalmente, sólo tenemos un servidor, quiere decir que sólo se hace una vez.

Se expande el árbol de la izquierda



- Hasta que aparezca "Grupo de SQL Server". Se hace clic derecho en él y se selecciona la opción "Nuevo registro de servidor SQL Server".

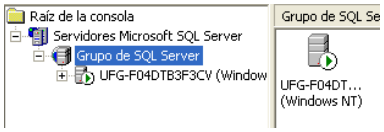
En la ventana que aparece hacemos "Siguiente"



El nombre se ve en el administrador de servicios

- Aparece una ventana con los servidores disponibles. Elegimos el nombre del que queremos registrar y hacemos clic en "Agregar".

Después, se hace "Siguiente" en todo y al fin cerrar



- Se ve el servidor que se ha registrado.

Hay operaciones que pueden hacerse con las dos herramientas

- Por ejemplo, las operaciones de crear la estructura de la base de datos se pueden realizar:
- De forma gráfica con el administrador corporativo.
- En forma de comandos SQL (texto) con el analizador de consultas.

SQL (Structured Query Language)

- Lenguaje de programación para la manipulación de bases de datos.
- Desarrollado por IBM, se ha convertido en el estándar internacional para las BDs internacionales.
- Se divide en dos subconjuntos de comandos:
  - DDL (Data Definition Language). Comandos de definición de la estructura de bases de datos
  - DML (Data Manipulation Language). Comandos para la recuperación y actualización de datos.

Historia del SQL

- 1974. Definición, por parte de IBM, de un lenguaje llamado SEQUEL para la manipulación de bases de datos relacionales.
- Se implementó un prototipo y las experimentaciones del prototipo llevaron a una modificación del lenguaje.
- Esta nueva versión, llamada SQL, apareció en 1977. Pronto, todos los SGBDs que aparecieron en los ochenta lo adoptaron.
- En 1986, ISO lo convierte en un estándar internacional. Esta versión se llama SQL/86.
- Varias revisiones producen el SQL/89, SQL/92 (el estándar actual).
- Recientemente, ha aparecido SQL/99, pero no ha sido implementado aún por los SGBDs.

El problema del SQL

- Es que hay tantas versiones de SQL como SGBDs.
- A pesar de que hay SQL estándar, los motores de bases de datos no lo siguen e implementan su propia versión de SQL.
- Esto reduce la portabilidad de programas de un SGBD a otro (una solución: implementar una capa de indirección como la capa de datos de una arquitectura en tres capas).
- La versión de SQL que usa SQL Server, se llama Transact-SQL.

2. Creación de la estructura de bases de datos.

Vamos a ver cómo se crea la estructura de la base de datos

- Hasta ahora habíamos visto como se diseñan las tablas con los métodos de modelización que habíamos estudiado.
- Ahora, vamos a ver como se crean esas tablas en SQL Server.

Lo primero: Creación de una base de datos

- La forma más sencilla en SQL es:

```
CREATE DATABASE nombreBD
```

- Este comando se puede complicar mucho pero por ahora lo mantendremos simple.
- Este comando no es SQL estándar: es una extensión de SQL Server. Es DDL.

Sigan paso a paso

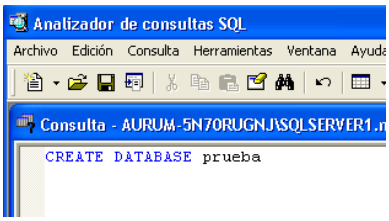
- Abran el administrador de consultas y creen una base de datos llamada “prueba”.
- Lo veremos paso a paso.

Abran el analizador de consultas

- Les saldrá una ventana como ésta.

- Por ahora, no la estudiaremos. Seleccionen “Autenticación de Windows”. Hagan “Aceptar”.

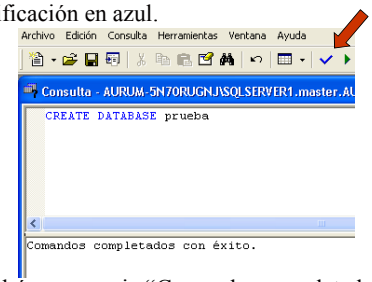
Saldrá una ventana de texto



- Escribimos en ella la instrucción SQL

Comprobamos que la instrucción SQL está bien escrita

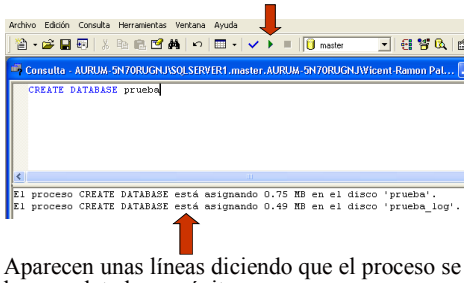
- Haciendo clic en el icono con la marca de verificación en azul.



- Saldrá un mensaje:“Comandos completados con éxito”.

Ejecutamos la instrucción

- Se hace clic en el icono con una flecha verde.



- Aparecen unas líneas diciendo que el proceso se ha completado con éxito.

Es bueno después de ejecutar una instrucción

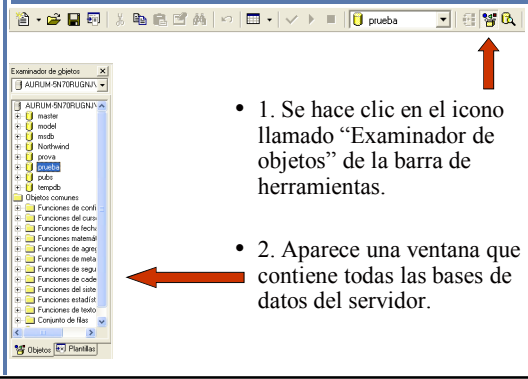
- Borrarla del texto para no volverla a ejecutar de nuevo con accidente. Se puede borrar a mano o hacer clic en el icono del "borrador".

- Si queremos utilizar ese comando SQL varias veces, lo podemos guardar con **Archivo|Guardar como** y después lo recuperaremos con **Abrir**.

¿Cómo vemos la base de datos que se acaba de crear?

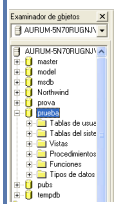
- Esto es un caso particular de un problema más general.
- Las sentencias SQL que escribimos en el analizador de consultas pueden referirse a todos los objetos que hay en mi servidor.
- Estos objetos pueden ser muchos y muy variados (BDs, tablas, índices, permisos, etc).
- Es necesario tener una forma para ver cuáles son todos los objetos de la BDs.

Viendo todos los objetos del servidor (si aún no se ven)

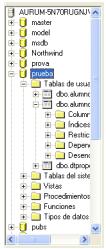


- 1. Se hace clic en el icono llamado "Examinador de objetos" de la barra de herramientas.
- 2. Aparece una ventana que contiene todas las bases de datos del servidor.

Si expandimos una BD, podemos ver todos los objetos de esa BD

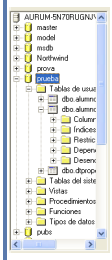


- Esto se ve a la izquierda.
- Si expandimos una de las tablas (a la derecha) podemos ver toda la información sobre la tabla: campos, índices, etc.



- De esta manera, podemos buscar el objeto sobre el cual queremos escribir la sentencia SQL y ver sus propiedades.

Nota: si realizamos alguna operación de actualización de la BD

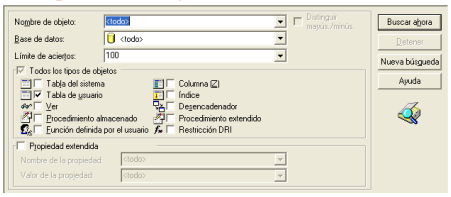


- El examinador de objetos se vera desactualizado.
- Para actualizarlo, basta con seleccionar la ventana del examinador de objetos, hacer clic derecho y seleccionar "Actualizar".
- A veces hace falta reiniciar el analizador de consultas. Esto es un error.

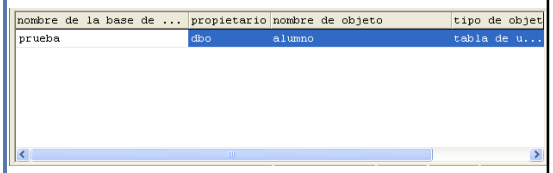
Mail: palasi@palasi.com Web: www.palasi.com

Una manera alternativa

- Es hacer clic en el icono "Buscar objeto". Aparecerá una ventana donde podremos buscar el objeto por nombre y tipo. Hacemos clic en "Buscar ahora". Esto no funciona para las BDs pero sí para otros objetos.



Buscando un objeto

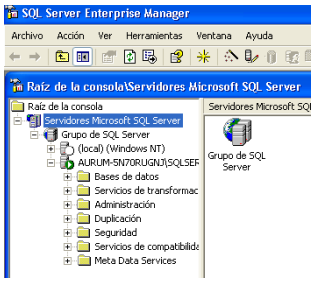


- En la parte inferior de la ventana de búsqueda, aparecerán los objetos que cumplen esos criterios (los resultados de la búsqueda).

Otra forma de ver la BD que se acaba de crear

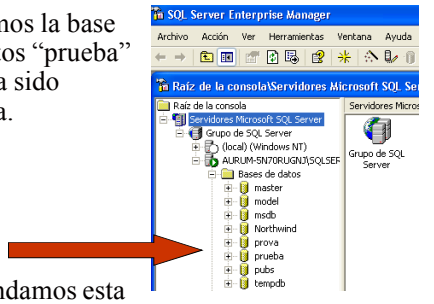
- En el administrador corporativo hacemos clic en el icono de actualizar para actualizar los datos.

- Expandimos el árbol de la izquierda hasta que vemos nuestro servidor de BD.
- Lo expandimos. Debajo hay una carpeta llamada "Bases de datos".



Expandamos la carpeta "Bases de datos"

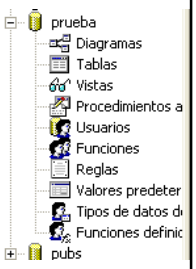
- Veremos la base de datos "prueba" que ha sido creada.



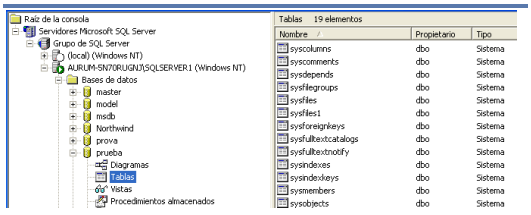
- Expandamos esta base de datos.

Expandamos la base de datos "prueba"

- Veremos todos los objetos que forman una base de datos:
  - Diagramas.
  - Tablas.
  - Vistas.
  - Procedimientos almacenados.
  - Usuarios.
  - Funciones.
  - Reglas.
  - Valores predeterminados.
  - Tipos de datos definidos por el usuario.
  - Funciones definidas por el usuario.



Hagamos clic sobre "Tablas"



- Aparecen a la derecha una lista de tablas de la base de datos "prueba".

Un momento, por favor

- ¿Por qué la base de datos "prueba" ya tiene tablas?
- ¿No la acabamos de crear?
- ¿No debería ser vacía?

Creación de BDs en SQL Server

- Cuando se crea una base de datos, se incluyen todos los objetos (tablas, usuarios, etc.) que hay en una BD llamada ***model***.
- La BD ***model*** es una especie de plantilla, a partir de la cual, se crean las BDs nuevas.
- En la BD ***model*** normalmente incluye todos los objetos que necesita el servidor para poder administrar la BD. También podemos añadir objetos propios, si queremos que aparezcan en todas las BD.

Podemos ver la base "model" en el administrador corporativo

- Vemos que tiene las mismas tablas que habían aparecido en nuestra base nueva "prueba".

Resumiendo: Lo que sabemos hasta ahora

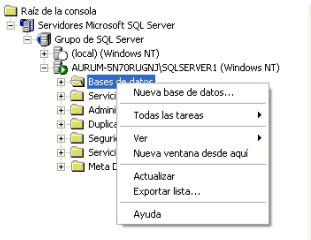
- La forma sencilla en SQL de crear una base de datos es:

```
CREATE DATABASE nombreBD
```

- Se crea una tabla nueva con todos los objetos que hay en la BD ***model***.
- La BD ***model*** contiene de forma predeterminada los objetos del sistema (no tocar), pero podemos añadir otros objetos si queremos que aparezcan en todas las BDs nuevas.

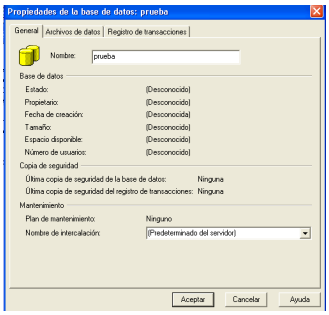
Una forma más sencilla

- En el administrador corporativo hacemos clic derecho sobre la carpeta "Bases de datos" y seleccionamos la opción "Nueva base de datos".



Una forma más sencilla

- En la ventana que aparece podemos escribir el nombre de la BD y hacer clic en **Aceptar**.



Mail: palasi@palasi.com Web: www.palasi.com

Creación de tablas con SQL

```
CREATE TABLE nombreTabla (
nombrecampo1 tipocampo1,
...
nombrecampon tipocampon)
```

- Comando que crea una tabla. El comando simplificado tal como se presenta aquí es SQL estándar, pero en realidad tiene opciones que no lo son. Es DDL.

Creación de tablas

```
CREATE TABLE nombreTabla (
nombrecampo1 tipocampo1,
...
nombrecampon tipocampon)
```

- Los nombres de los campos deben seguir las reglas de SQL Server para los identificadores.

Reglas de SQL para los identificadores

- Los identificadores deben empezar por una letra, _ (subrayado), @ o bien # . Los dos últimos símbolos tienen significado especial, así que no los usaremos por ahora.
- Los identificadores pueden seguir con letras, números o los caracteres _, # o $.
- Aunque los identificadores pueden tener espacios en blanco, esto no se recomienda y aquí no lo consideraremos.

Creación de tablas

```
CREATE TABLE nombreTabla (
nombrecampo1 tipocampo1,
...
nombrecampon tipocampon)
```

- ¿Cuáles son los tipos que podemos elegir para los campos?

Tipos de datos numéricos y binarios en SQL Server

`bit`	0 o 1.
`tinyint`	Números enteros de 0 a 255
`smallint`	Números enteros de –32768 a 32767
`int`	Enteros de –2,147,483,648 a 2,147,483,647
`bigint`	Enteros de -9223372036854775808 a 9223372036854775807
`decimal(p,e)` `numeric(p,e)`	Números reales de punto fijo con p dígitos en total y e dígitos decimales (a p se le llama precisión y a e escala).
`real` `float(n)`	Reales de punto flotante. Se utilizan para cálculos científicos, pero no para aplicaciones empresariales.

Tipos de datos de texto e imagen en SQL Server

`char(n)`	Cadenas de exactamente n caracteres. Entre comillas simples. Si se entran cadenas más cortas, se rellenarán con espacios. Cadenas más largas se truncarán.
`varchar(n)`	Una cadena de n caracteres máximo. No se rellena con espacios. Entre comillas simples. Ahorra espacio en disco pero es más lento.
`text`	Texto mayor de 8000 caracteres.
`image`	Imágenes (pero SQL Server no sabe en qué formato:sólo son bits).

- Los tipos de datos `text` e `image` degradan el rendimiento. Una alternativa es tener estos datos como archivos a parte y sólo guardar la ruta en la BD.

Mail: palasi@palasi.com Web: www.palasi.com

Tipos de datos monetarios y temporales en SQL Server

`smallmoney`	Reales de –214748.3648 a 214748.3647 (4 dígitos decimales). Precedidos por $.
`money`	De –922,337,203,685,477.5808 a 922,337,203,685,477.5807 (4 decim.) Precedidos por $.
`smalldatetime`	Almacena datos de fecha y hora con precisión hasta el minuto. 'yyyy-dd-mm hh:mm'
`datetime`	Almacena datos de fecha y hora con precisión hasta 3,33 milisegundos. 'yyyy-dd-mm hh:mm:ss.mmm'
`timestamp`	Indica automáticamente el día y la hora en que se creó o actualizó el registro.

Otros tipos de datos en SQL Server

`binary(n)` `varbinary(n)`	n bytes de información binaria en hexadecimal (prefijo Ox).
Definidos por el usuario	No los veremos por ahora.
Campos calculados	No los veremos por ahora.
`nchar, ntext, nvarchar`	Igual que sus correspondientes sin n. Almacenan textos con formato Unicode, para compatibilidad con otros lenguajes.
`uniqueidentifier`	No lo veremos.
`cursor`	No lo veremos.
`sql_variant`	No lo veremos.
`table`	No lo veremos.

Almacenamiento de tipos de datos numéricos en SQL Server

`bit`	1 byte
`tinyint`	1 byte
`smallint`	2 bytes
`int`	4 bytes
`bigint`	8 bytes
`decimal(p,e)` `numeric(p,e)`	Si p va de 1 – 9, 5 bytes Si p va de 10-19, 9 bytes Si p va de 20-28, 13 bytes. Si p va de 29-38, 17 bytes
`real`	4 bytes.
`float(n)`	Si n va de 1-24, 4 bytes. Si n va de 25-53, 7 bytes.

Almacenamiento de tipos de datos en SQL Server

`char(n)`	n bytes exactamente.
`varchar(n)`	Como máximo n bytes.
`text`	El tamaño del texto en caracteres.
`image`	El tamaño de la imagen.
`smallmoney`	4 bytes.
`money`	8 bytes.
`smalldatetime`	4 bytes.
`datetime`	8 bytes.
`timestamp`	8 bytes.

Almacenamiento de tipos de datos en SQL Server

`binary(n)` `varbinary(n)`	n bytes
Definidos por el usuario	No los veremos por ahora.
Campos calculados	No los veremos por ahora.
`nchar, ntext, nvarchar`	El doble que sus correspondientes sin n.
`uniqueidentifier`	No lo veremos.
`cursor`	No lo veremos.
`sql_variant`	No lo veremos.
`table`	No lo veremos.

Valores constantes de los diferentes tipos de datos (1)

- Cadenas de caracteres (`char`, `varchar`, `text`). Entre comillas simples `'San Salvador'`. Las comillas dentro de la cadena se indican como dos comillas ('Don''t speak')
- Cadenas de caracteres Unicode (`nchar`, `nvarchar`, `ntext`). Similar, pero antes de la primera comilla hay una N mayúscula. `N'Π se lee pi'`.
- Constantes enteras (`bit`, `tinyint`, `smallint`, `int`, `bigint`). Números sin el punto decimal. `1450`
- Constantes reales (`decimal`, `numeric`, `real`, `float`). Números con el punto decimal. `23.34`

Mail: palasi@palasi.com Web: www.palasi.com

Valores constantes de los diferentes tipos de datos (2)

- Constantes de dinero (`smallmoney, money`). Números. Pueden tener un punto decimal opcional y un símbolo de dólar opcional `123.54 $125`.
- Constantes de fecha y hora (`datetime, smalldatetime, timestamp`). Entre comillas simples, conteniendo una expresión de fecha y hora, que pueden
  - Formatos con separación (por ejemplo, `'15/12/1998'` o `'15-12-1998'`)
  - Formatos sin separación (por ejemplo, `'19981215'`). (La hora se añade en formato **hh:mm:ss.** Así `'19981215 13:34'` o `'15-12-1998 13:34'`)

Extendiendo el "create table"

```
CREATE TABLE nombreTabla (
nombrecampo1 tipocampo1,
...
nombrecampon tipocampon)
```

- Después del tipo de campo, podemos añadir las siguientes palabras clave (en ese orden):
  - **`DEFAULT`** *valor*. Establece el valor predeterminado que llenará ese campo si no se inserta otro valor.
  - **`IDENTITY`**. Indica que es un campo autonumérico.
  - **`NOT NULL.`** No permitiremos valores nulos en este campo.
  - **`NULL.`** Permitiremos nulos. No se suele usar porque es el comportamiento por defecto.

¿Qué es un valor nulo?

- Cuando no ponemos valor en el campo de un registro, se dice que tiene un valor nulo o NULL.
- Este es una especie de valor vacío o "desconocido".

Algunas restricciones

- Las palabras clave deben estar en este orden:
  - **`DEFAULT`** *valor*.
  - **`IDENTITY.`**
  - **`NOT NULL.`**
  - **`NULL.`**
- Un campo IDENTITY no puede tener valores nulos ni valores predeterminados.

Ejemplo: tabla "alumno"

```
USE prueba

CREATE TABLE alumno (
 idAlumno smallint IDENTITY NOT NULL,
 nombre char(50) NOT NULL,
 nota tinyint,
 precioMatricula smallmoney,
 fechaMatricula smalldatetime
)
```

- Un momento, ¿Qué es ese comando USE?

El comando USE

- Cuando entramos en el Analizador de consultas, todos los comandos SQL que escribimos se aplican sobre la base de datos "master".
- Si queremos que se apliquen a otra BD, utilizamos el comando (no es SQL estándar).

```
USE nombreBD
```

`USE` indica la base de datos en la que queremos trabajar. Todos los comandos que escribamos, se ejecutarán sobre esa BD hasta que utilicemos **`USE`** de nuevo.

Mail: palasi@palasi.com Web: www.palasi.com

Creen esta tabla "alumno"

```
USE prueba

CREATE TABLE alumno (
 idAlumno smallint IDENTITY NOT NULL,
 nombre char(50) NOT NULL,
 nota tinyint,
 precioMatricula smallmoney,
 fechaMatricula smalldatetime
)
```

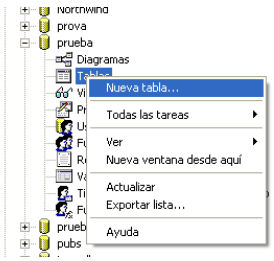
- Comprueben que se ha creado en el analizador de consultas.

Ejercicio

- Escriban las sentencias SQL que les permitan crear todas las tablas que han obtenido diseñando el ejemplo de "Agencias de viajes".
- Escriban en un archivo de texto pero no las ejecuten.

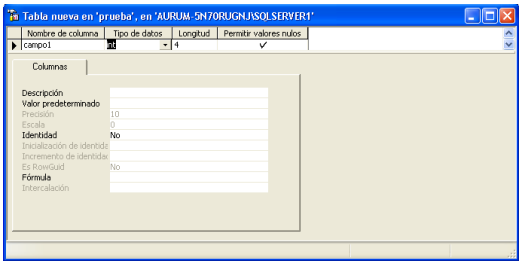
Crear tablas gráficamente

- En el administrador corporativo, expandimos la BD en que queremos crear la tabla.
- Hacemos clic derecho sobre el nodo **Tablas**.
- Seleccionamos la opción **Nueva tabla...**



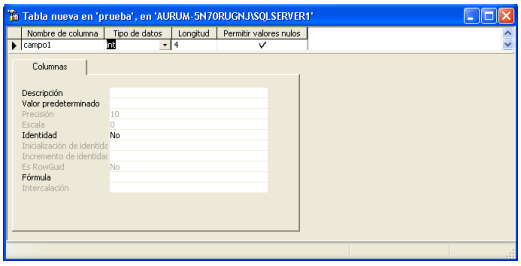
Aparece el cuadro de diálogo

- Allá podemos especificar los diferentes campos de la tabla ("columnas" en nomenclatura de SQL Server) con sus diferentes propiedades.



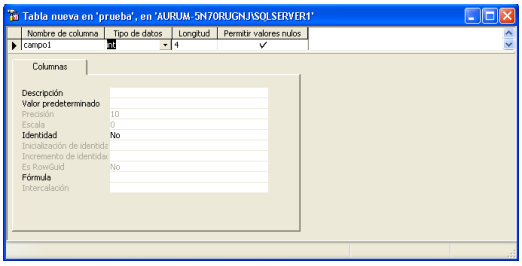
Aparece el cuadro de diálogo

- En la parte superior se puede especificar el nombre de campo, el tipo, la longitud y si permite valores nulos o no.



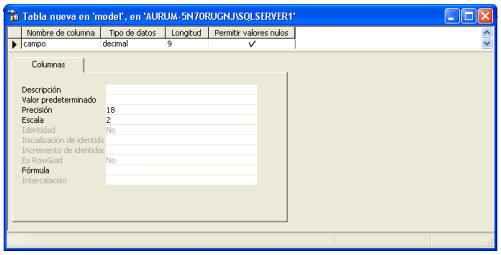
Aparece el cuadro de diálogo

- En la parte inferior, se puede especificar si el campo es Identity ("Identidad") y su valor predeterminado.



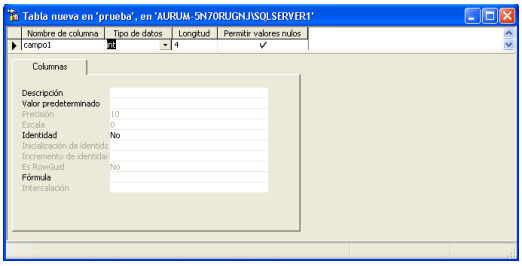
Si el tipo es `numeric` o `decimal`

- En la parte inferior, se puede especificar el número total de dígitos (precisión) y el número de dígitos decimales (escala). En este caso, no se debe especificar longitud.



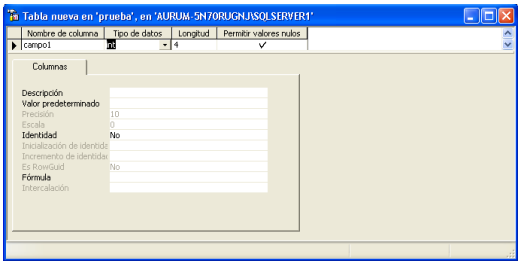
Como ven, es la misma información que con el CREATE TABLE

- Sólo que ahora es gráfica.



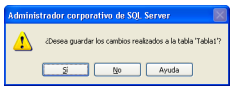
Cuando hayamos definido todos los campos

- Hacemos clic en el botón en forma de X de la esquina superior derecha de la ventana.

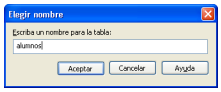


Aparece un mensaje de confirmación

- Diciendo si queremos grabar. Contestamos "Sí".



- Nos pedirá un nombre de la tabla. Escribiremos el nombre.



- Haremos clic en "Aceptar" y habremos acabado de crear la tabla.

Ejercicio. Creen estas tablas en SQL Server. Asistente con el analizador.

Asistentes

- CodAsist. A(6).
- Nombre. T(50).
- Apellidos T(50).
- DUI N(20).
- CodPais N(3)
- CodTarifa N(3)
- Telefono T(7).
- Direccion T(150).
- Hotel. T(50).

TiposTarifa

- CodTarifa A(3)
- TipoTarifa T(10)
- Importe N(3)

Paises

- CodPais A(3)
- Pais T(50)
- Avion L(1)

Las otras con el administrador corporativo

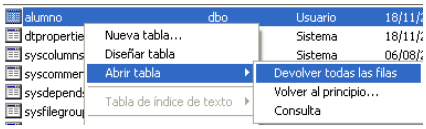
Ejercicio

- Creen todas las tablas del ejemplo de agencia de viajes en su SQL Server.
- Creen la mitad de forma gráfica y la mitad con comandos SQL.

Mail: palasi@palasi.com Web: www.palasi.com

Ahora que tenemos una tabla, queremos añadir registros

- ¿Cómo se añaden registros a una tabla?
- La forma más sencilla es la siguiente.
  - En el administrador corporativo, se hace clic derecho en la tabla.
  - Se selecciona "Abrir tabla" y "Devolver todas las filas".
  - Aparece una ventana donde se pueden añadir los registros



Ejercicio

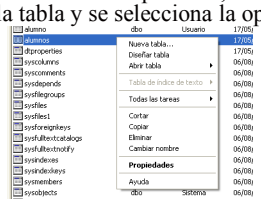
- Llenen las tablas de asistentes con algunos datos de prueba.
- Al menos cinco países, cinco asistentes y las tarifas "estudiante", "invitado" y "normal" con importes de 50, 100 y 200 dólares respectivamente.

Eliminar una tabla

- Desde el analizador de consultas (SQL estándar:DDL).

```
DROP TABLE nombreTabla
```

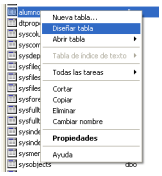
- Desde el administrador corporativo, clic derecho en el icono de la tabla y se selecciona la opción "Eliminar".



- No se pueden eliminar las tablas de sistema.

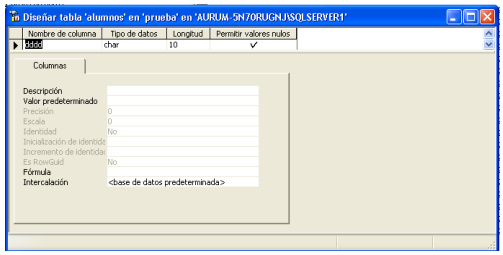
Modificar una tabla

- Se puede utilizar la instrucción SQL ALTER TABLE (vean su sintaxis en la ayuda).
- Es más sencillo desde el administrador corporativo, hacer clic derecho en el icono de la tabla y seleccionar la opción "Diseñar tabla".



Aparecerá una ventana idéntica a la que introducíamos los campos

- Cuando creábamos la tabla. Aquí podemos modificar los campos. No la estudiamos porque es idéntica.



La pareja dispareja

- Como ven muchas cosas se pueden hacer tanto desde el administrador corporativo como desde el analizador de consultas.
- El administrador corporativo es más visual y operaciones como modificar tablas son mucho más fáciles.
- El analizador de consultas permite guardar las instrucciones SQL y, por eso, podemos repetir varias veces las mismas operaciones.
- Además el SQL es más estándar que el entorno gráfico, que cambia completamente en las BD.
- Cada uno tiene sus ventajas e inconvenientes. Además hay acciones que sólo se pueden realizar en uno de ellos. Por lo tanto, es bueno saber ambos

Mail: palasi@palasi.com Web: www.palasi.com

Revisitando el CREATE DATABASE

- Habíamos visto que creábamos una base de datos con:

```
CREATE DATABASE nombreBD
```

- Sin embargo este comando tiene muchas más opciones. No las veremos todas, pero, a partir de ahora veremos las principales.

CREATE DATABASE

```
CREATE DATABASE nombreBD ON PRIMARY
(NAME=nombreLogico,
 FILENAME=rutaArchivo,
 SIZE =tamañoInicial,
 MAXSIZE=tamañoMaximo,
 FILEGROWTH=incremento
)
```

Todos estos elementos son opcionales, pero si se incluye **NAME** también debería incluirse **FILENAME**.

CREATE DATABASE

```
CREATE DATABASE nombreBD ON PRIMARY
(NAME=nombreLogico,
 FILENAME=rutaArchivo,
 SIZE =tamañoInicial,
 MAXSIZE=tamañoMaximo,
 FILEGROWTH=incremento
)
```

FILENAME. Especifica la ruta del archivo donde se guardará los datos de la BD (extensión MDF).

NAME. Especifica el alias por el cual se conocerá a este archivo dentro de SQL Server.

CREATE DATABASE

```
CREATE DATABASE nombreBD ON PRIMARY
(NAME=nombreLogico,
 FILENAME=rutaArchivo,
 SIZE =tamañoInicial,
 MAXSIZE=tamañoMaximo,
 FILEGROWTH=incremento
)
```

SIZE. Especifica el tamaño inicial de la base de datos. No se añaden decimales. Se añaden los sufijos KB, MB, GB o TB. Cuando no se especifica, se usa el tamaño de la BD **model**.

CREATE DATABASE

```
CREATE DATABASE nombreBD ON PRIMARY
(NAME=nombreLogico,
 FILENAME=rutaArchivo,
 SIZE =tamañoInicial,
 MAXSIZE=tamañoMaximo,
 FILEGROWTH=incremento
)
```

MAXSIZE. Especifica el tamaño máximo de la base de datos (también sufijos). Si no se especifica, el archivo aumenta hasta que el disco esté lleno (no se recomienda).

CREATE DATABASE

```
CREATE DATABASE nombreBD ON PRIMARY
(NAME=nombreLogico,
 FILENAME=rutaArchivo,
 SIZE =tamañoInicial,
 MAXSIZE=tamañoMaximo,
 FILEGROWTH=incremento
)
```

FILEGROWTH. Cantidad de espacio que se añade al archivo cuando éste se llena. Sólo enteros, seguidos de KB, MB, GB, TB o % (porcentaje del tamaño del archivo en el momento en que tiene lugar el incremento). Si no se especifica, es 10%. Si se especifica 0, el archivo no crece.

Ejemplo de CREATE DATABASE

```
CREATE DATABASE alumno ON PRIMARY
(NAME=datosAlumno,
 FILENAME=C:\MSSQL7\DATA\datosAlumno.mdf,
 SIZE=20MB,
 MAXSIZE=100MB,
 FILEGROWTH=10MB
)
```

Como vemos, el archivo que contiene los datos de la BD es "datosAlumno.mdf". Se le llama archivo principal y sus propiedades se especifican aquí.

¿El archivo principal es el único archivo de la BD?

- De hecho, no. Hay como mínimo otro archivo (de extensión ldf) que se llama el registro de transacciones.
- Sirve para registrar las transacciones (esto lo veremos después).
- También se puede especificar el tamaño y crecimiento de este archivo.

CREATE DATABASE

```
CREATE DATABASE nombreBD ON PRIMARY
(NAME=nombreLogico,
 FILENAME=rutaArchivo,
 SIZE =tamañoInicial,
 MAXSIZE=tamañoMaximo,
 FILEGROWTH=incremento
)
LOG ON
(NAME=nombreLogico,
 FILENAME=rutaArchivo,
 SIZE =tamañoInicial,
 MAXSIZE=tamañoMaximo,
 FILEGROWTH=incremento
)
```

Archivo principal

Registro de transacciones

El tamaño de la base de datos en disco

- Será la suma de los tamaños del archivo principal y el registro de transacciones.
- ¿Cómo calculamos el tamaño?
- Vamos a ver cómo calculamos el tamaño del archivo principal y el registro de transacciones.

Calculando tamaño inicial y máximo de cada tabla

- 1º, se calcula el tamaño de cada campo, según su tipo. Para ello, consultamos las tablas que acabamos de ver que relacionan cada tipo con el tamaño.
- Se suman los tamaños de todos los campos y esto nos dará el tamaño de cada registro.
- Multiplicamos el tamaño de cada registro por el número de registros que esperamos tener al principio. Esto nos da el tamaño inicial.
- Multiplicamos el tamaño de cada registro por el máximo número de registros que esperamos tener. Esto nos da el tamaño máximo.
- Es bueno dar un poco de margen.

Calculando los tamaños del archivo principal

- Para calcular SIZE, se suman los tamaños iniciales de cada tabla.
- Para calcular MAXSIZE, se suman los tamaños máximos de cada tabla.
- FILEGROWTH se suele especificar en torno al 10%.
- Es bueno dar un poco de margen.

Calculando los tamaños del registro de transacciones

- Si es una base de datos de sólo lectura, los tamaños serán de un 5% de los tamaños del archivo principal.
- Si es una base de datos de uso normal, entre 10 y 15% del tamaño del archivo principal.
- Si es una base de datos con muchas actualizaciones e inserciones, entre 25 y 30% del tamaño del archivo principal.

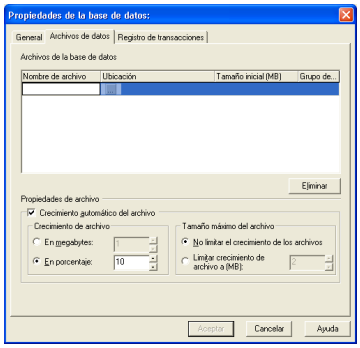
El tamaño de la base de datos en disco

- Se calculará sumando los tamaños del archivo principal y el registro de transacciones.

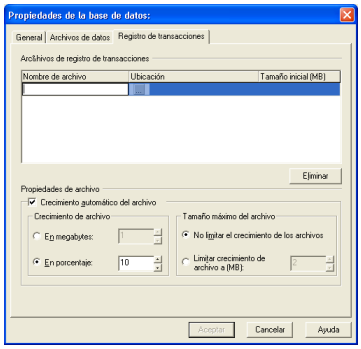
Todo esto puede especificarse gráficamente

- Por supuesto, desde el administrador corporativo, que es la herramienta gráfica de SQL Server.
- Hacemos clic derecho sobre la base de datos a modificar y se selecciona la opción "Nueva base de datos", que hemos visto,
- Aparece una ventana (que ya hemos visto) con dos pestañas que nos permiten especificar estos datos del archivo principal y el registro de transacciones.
- No las explicaremos en detalle, pues contienen los mismos datos que acabamos de explicar.

Pestaña que me permite especificar el archivo principal.



Pestaña que me permite especificar el registro de transacciones.



Para borrar una base de datos

- Desde el analizador de consultas.

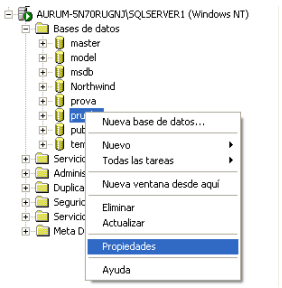
```
DROP DATABASE nombreBD
```

- Si queremos modificar las propiedades de una base de datos, podemos hacerlo desde el analizador de consultas con ALTER DATABASE (mirar la sintaxis en la ayuda).
- Sin embargo, es más fácil hacerlo desde el administrador corporativo.

Mail: palasi@palasi.com Web: www.palasi.com

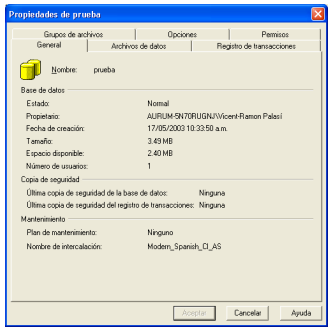
Modificando las propiedades de la BD

- Hacemos clic derecho en el icono de la BD y seleccionamos "Propiedades".



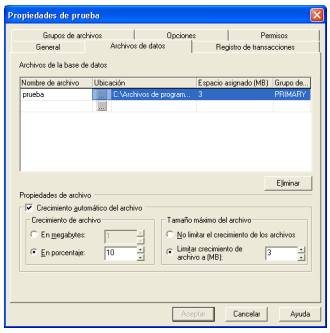
Aparecerá una ventana

- En las pestañas "Archivos de datos" y "Registro de transacciones" podemos modificar los datos.



No explicamos estas pestañas

- Son las mismas que cuando creábamos la base de datos. Por lo tanto, ya las conocemos.



Hacer copia de seguridad de una base de datos

```
BACKUP DATABASE nombreBD
 TO DISK = 'rutaArchivoRespaldo'
 WITH INIT
```

- Una copia de la base de datos se guardará en el archivo de respaldo cuya ruta que acabamos de especificar.
- En este apartado, ruta incluye el nombre del archivo y la extensión.

Restaurar copia de seguridad de una base de datos

```
RESTORE DATABASE nombreBD
 FROM DISK = 'rutaArchivoRespaldo'
```

- La base de datos se restaurará desde el archivo de respaldo a la misma ubicación que tenía .
- ¿Qué pasa si queremos restaurarlo en otra ubicación? (por ejemplo, para copiarlo a otro servidor de BD).

Restaurar copia de seguridad de una base de datos a otra ubicación (1)

- Son dos pasos. El primero: vemos que archivos existen dentro del archivo de respaldo con

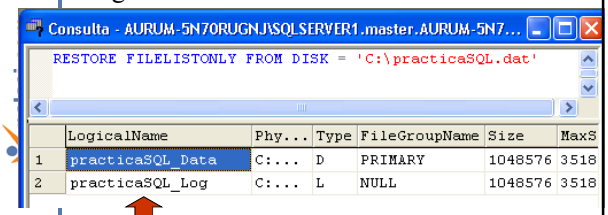
```
RESTORE FILELISTONLY FROM DISK
 = 'rutaArchivoRespaldo'
```

- Esto nos debe dar los "nombres lógicos" de los archivos que hay dentro del archivo de respaldo.

Mail: palasi@palasi.com Web: www.palasi.com

Restaurar copia de seguridad de una base de datos a otra ubicación (1)

- Por ejemplo, restaurando la base de datos siguiente



- Estos son los nombres lógicos.

Restaurar copia de seguridad de una base de datos a otra ubicación (2)

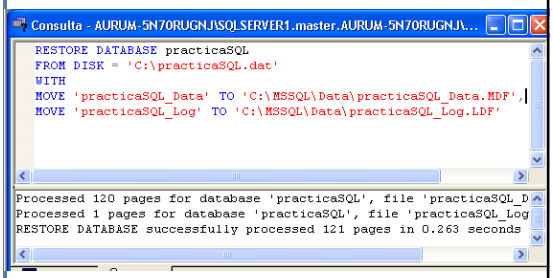
- El segundo paso es

```
RESTORE DATABASE nombreBD
 FROM DISK = 'rutaArchivoRespaldo'
WITH
MOVE 'nombrelogico1' TO 'ruta1',
MOVE 'nombrelogico2' TO 'ruta2',
...
MOVE 'nombrelogicon' TO 'rutan'
```

- Es decir, se especifica la ruta donde irá a parar cada uno de los archivos.

Restaurar copia de seguridad de una base de datos a otra ubicación (2)

- En nuestro caso:



Todo esto también se puede hacer desde el administrador corporativo

- Las opciones son **Herramientas|Copia de seguridad de la base de datos** y **Herramientas|Restaurar base de datos**.
- En esta última, se debe seleccionar la opción “Desde dispositivos”.
- Los datos son similares a los que acabamos de ver.

Ejercicio

- Modifiquen gráficamente su base de datos de agencia de viajes para que su tamaño máximo sea de 500 Mb.
- Creen un backup de dicha base de datos y la entregan.

2. Creación de la estructura de bases de datos.

Índices

- Un índice es una estructura de datos que contiene apuntadores (localizadores) a los registros de una tabla.
- Es parecido al índice de un libro que tiene apuntadores a cada capítulo.
- Con el índice podemos acceder de forma más rápida a los registros, de la misma forma que un índice de un libro permite acceder a un capítulo de forma más rápida que sin él.

Un índice de un libro

Libro: Moby Dick.
Índice (por capítulos).

- Capítulo 1.....página 3.
- Capítulo 2.....página 9.
- Capítulo 3.....página 17.
- Capítulo 4.....página 25
- Capítulo 5.....página 33.
- Capítulo 6..... página 37.
- Capítulo 7..... página 42.

Un índice de una tabla

Tabla: Clientes
Índice (sobre el campo "teléfono").

- Teléfono 220-7246..... registro 22.
- Teléfono 223-8293..... registro 97.
- Teléfono 243-2345.....registro 20.
- Teléfono 475-5663..... registro 5.
- Teléfono 545-8754..... registro 45.
- Teléfono 567-8922..... registro 12.
- Teléfono 866-2599..... registro 67.

Índices

- Como vemos, en un índice de un libro, los apuntadores se organizan según el capítulo. Sabiendo el capítulo, podemos saber en qué página buscarlo.
- En un índice de una tabla, los apuntadores se organizan según un campo. Por ejemplo, el índice de la tabla de clientes se organiza por el campo "Teléfono".
- Sabiendo el teléfono que contiene un cliente, podemos saber qué registro contiene los datos de ese cliente.

Se dice que el índice se ha definido sobre el campo "teléfono"

Tabla: Clientes
Índice (sobre el campo "teléfono").

- Teléfono 220-7246..... registro 22.
- Teléfono 223-8293..... registro 97.
- Teléfono 243-2345.....registro 20.
- Teléfono 475-5663..... registro 5.
- Teléfono 545-8754..... registro 45.
- Teléfono 567-8922..... registro 12.
- Teléfono 866-2599..... registro 67.

Un índice definido sobre el campo "DUI"

Tabla: Clientes
Índice (sobre el campo "DUI").

- DUI 19456875-3.....registro 57.
- DUI 27645836-5..... registro 5.
- DUI 29745378-1..... registro 62.
- DUI 34843534-1..... registro 14.
- DUI 39346325-1..... registro 44.
- DUI 44543532-1..... registro 21.
- DUI 52222334-1..... registro 02.

Mail: palasi@palasi.com Web: www.palasi.com

También puede definirse un índice sobre varios campos

Tabla: Clientes

Índice (sobre los campos "Apellido", "Nombre").

- Balmore, Pedro.....registro 95.
- Bonilla, Claudia..... registro 14.
- Bonilla, Claudia..... registro 64.
- Cruz, Ana ..... registro 25.
- Diego, Juan..... registro 87.
- Parada, José..... registro 12.
- Renderos, Mauricio..... registro 47.

Importante: que los apuntadores en 1 índice están ordenados

Tabla: Clientes

Índice (sobre el campo "teléfono").

- Teléfono 220-7246..... registro 22.
- Teléfono 223-8293..... registro 97.
- Teléfono 243-2345.....registro 20.
- Teléfono 475-5663..... registro 5.
- Teléfono 545-8754..... registro 45.
- Teléfono 567-8922..... registro 12.
- Teléfono 866-2599..... registro 67.

¿Para qué sirven los índices? (1)

- Utilidad 1: Para acceder más rápidamente a los datos.
- Por ejemplo, si buscamos muchas veces un cliente a partir de su teléfono, un índice sobre el campo "teléfono" mejora el rendimiento.
- Esto es análogo a la forma en que un índice de un libro nos permite acceder más rápidamente a los diferentes capítulos.

¿Para qué sirven los índices? (2)

- Utilidad 2: Para asegurar la unicidad de ciertos valores.
- Hay índices llamados "únicos" que no permiten valores duplicados sobre el campo que se definen.
- Así, si definimos un índice único en la tabla clientes sobre el campo DUI, esto quiere decir que no podrán haber dos clientes en la tabla con el mismo DUI.
- Como ven, esto es especialmente útil para campos que son claves primarias o cualquier otro campo que necesite ser único.

Tipos de índices

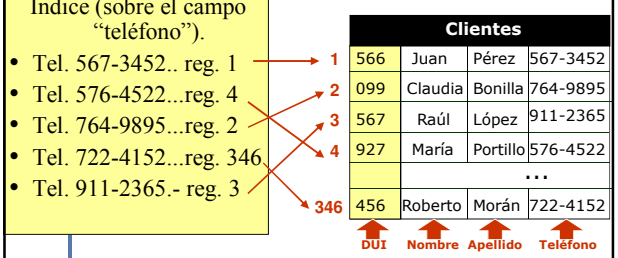
- Índices no agrupados (NONCLUSTERED).
- Son aquellos en que el índice y la tabla (es decir, los registros) se mantienen por separado.
- El índice lo único que tiene es una referencia a los registros de la tabla.
- Todos los índices que se han visto hasta ahora son no agrupados.

Un índice no agrupado

Tabla: Clientes

Índice (sobre el campo "teléfono").

- Tel. 567-3452.. reg. 1
- Tel. 576-4522...reg. 4
- Tel. 764-9895...reg. 2
- Tel. 722-4152...reg. 346
- Tel. 911-2365.- reg. 3



El índice y la tabla (los registros) están separados.

Mail: palasi@palasi.com Web: www.palasi.com

Tipos de índices

- Índices agrupados (CLUSTERED).
- Son aquellos en que el índice y la tabla (es decir, los registros) se mantienen en la misma estructura de datos.
- El índice no tiene referencia a los registros de la tabla, sino que contiene los mismos registros.
- La tabla y el índice agrupado son lo mismo.

Un índice agrupado

Tabla: Clientes y su índice agrupado (sobre el campo "DUI").

- DUI. 099..

099	Claudia	Bonilla	764-9895

- DUI. 456...

456	Roberto	Morán	722-4152

- DUI. 566...

566	Juan	Pérez	567-3452

- DUI. 567...

567	Raúl	López	911-2365

- DUI. 927...

927	María	Portillo	576-4522

La tabla y el índice agrupado son una sola cosa.

El índice no contiene referencias a los registros, contiene los propios registros.

Sólo puede haber un índice agrupado por tabla

Tabla: Clientes y su índice agrupado (sobre el campo "DUI").

- DUI. 099..

099	Claudia	Bonilla	764-9895

- DUI. 456...

456	Roberto	Morán	722-4152

- DUI. 566...

566	Juan	Pérez	567-3452

- DUI. 567...

567	Raúl	López	911-2365

- DUI. 927...

927	María	Portillo	576-4522

Ya que sólo podemos guardar los datos de los registros en un solo sitio.

Es por eso que se ha de elegir con cuidado cuál es el índice agrupado.

Pueden haber tantos índices no agrupados por tabla como se quiera

Tabla: Clientes

Índice (sobre el campo "teléfono").

- Tel. 567-3452.. reg. 1
- Tel. 576-4522...reg. 4
- Tel. 764-9895...reg. 2
- Tel. 722-4152...reg. 346
- Tel. 911-2365.- reg. 3

	Clientes			
1	566	Juan	Pérez	567-3452
2	099	Claudia	Bonilla	764-9895
3	567	Raúl	López	911-2365
4	927	María	Portillo	576-4522
		...		
346	456	Roberto	Morán	722-4152
	DUI	Nombre	Apellido	Teléfono

En SQL Server "sólo" pueden haber 249 índices no agrupados por tabla.

Índices únicos

- Si definimos un índice como único, el SGBD se asegurará de que no haya dos registros con el mismo valor del campo (o campos) sobre el que se define el índice.
- Así un índice de la tabla clientes sobre el campo DUI, debería definirse como único, para evitar que haya dos clientes con el mismo DUI.
- Un índice de la misma tabla sobre los campos apellidos y nombre **NO** debería definirse como único, pues puede haber dos clientes con el mismo nombre y apellidos.

Cómo crear un índice. (SQL estándar. DDL)

- Para crear un índice agrupado

```
CREATE CLUSTERED INDEX nombreindice ON
  nombretabla (nombrecampo1,...nombrecampon)
```

- Para crear un índice no agrupado

```
CREATE NONCLUSTERED INDEX
  nombreindice ON nombretabla
  (nombrecampo1,...nombrecampon)
```

(La palabra **NONCLUSTERED** es opcional, ya que por defecto se crea un índice no agrupado)

Si queremos que el índice sea único colocamos la palabra **UNIQUE** después de **CREATE**. Si no, será no único.

Mail: palasi@palasi.com Web: www.palasi.com

Ejemplos

```
CREATE UNIQUE CLUSTERED INDEX
 indiceDUI ON Clientes (DUI)
```

- Crea un índice agrupado y único en la tabla Clientes sobre el campo "DUI".

```
CREATE INDEX indiceTelefono ON
 Clientes (telefono)
```

- Crea un índice no agrupado y no único en la tabla Clientes sobre el campo "teléfono".

Si deseamos indicar el orden en que se van a ordenar los diferentes campos

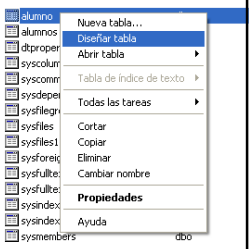
- Podemos hacerlo con las palabras **ASC** (orden ascendente) o **DESC** (orden descendente) después del nombre del campo. El valor por defecto es **ASC**.

```
CREATE INDEX indiceTelefono ON
 Clientes (telefono DESC)
```

- El índice sobre el campo "teléfono" está en orden descendente, es decir, los teléfonos más altos se guardan primero.
- Sin embargo, esto es algo que nos preocupa raras veces.

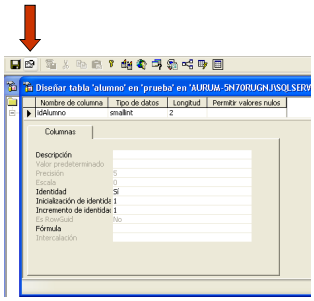
Cómo crear un índice con el Administrador corporativo

- Seleccionamos la tabla y hacemos clic derecho. Seleccionamos la opción "Diseñar tabla".



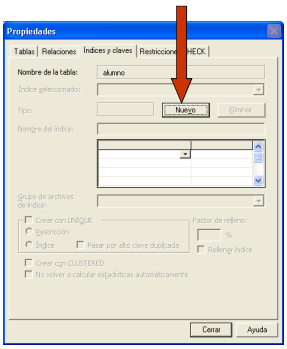
Aparecerá la ventana "Diseñar pantalla"

- Hacemos clic en el segundo icono de la barra de herramientas, llamado "Propiedades de tabla e índice".



En la ventana que sale, seleccionamos la pestaña "Índices y claves"

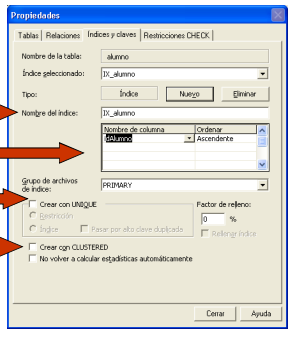
- En esta pestaña, hacemos clic en el botón "Nuevo" para crear un nuevo índice.



En la ventana podemos especificar todos los datos de creación de un índice

- Los otros datos los ignoraremos.

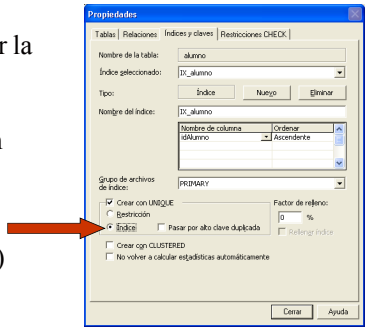
Nombre
Campos y orden
Si es único
Si es agrupado



Mail: palasi@palasi.com Web: www.palasi.com

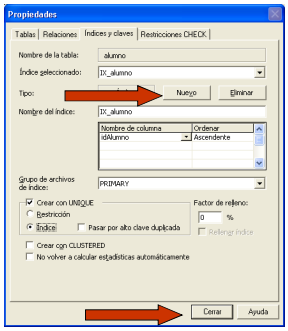
Si especificamos que un índice es único

- Debemos seleccionar la opción "Indice".
- (La opción restricción aún no la hemos estudiado.)



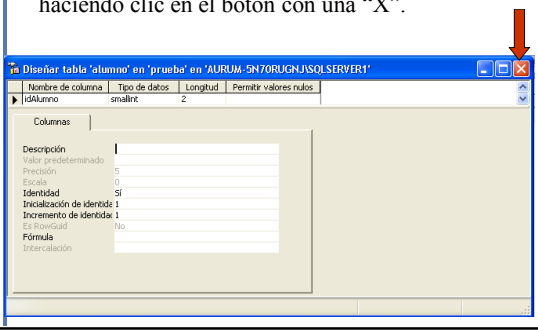
Cuando acabemos de especificar los datos del índice

- Podemos hacer clic en "Nuevo" para crear un nuevo índice o en "Cerrar" para acabar.



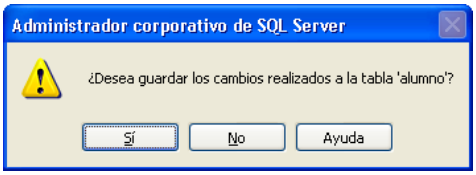
Después de hacer clic en "Cerrar"

- Debemos cerrar la ventana "Diseñar ventana" haciendo clic en el botón con una "X".



Nos aparecerá un cuadro de confirmación

- Deberemos contestar "Sí".



Eliminar un índice

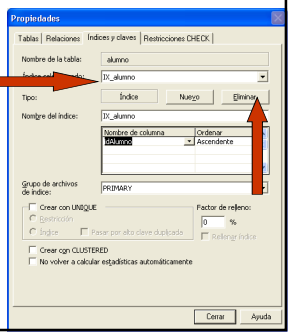
- Desde el analizador de consultas

```
DROP INDEX nombretabla.nombreindice
```

- Este comando es SQL estándar y DDL.
- No olvidar el punto del medio. Para ver el resultado en el examinador de objetos oprimiremos F5.

Eliminar un índice desde el administrador corporativo

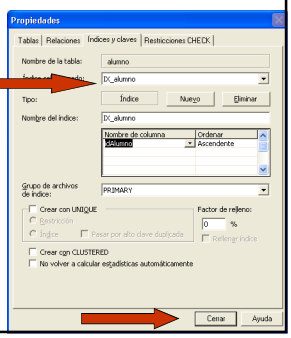
- Desde la ventana de índices que ya hemos visto, seleccionamos el índice que queremos borrar desde la lista desplegable marcada con la flecha en rojo. Después hacemos clic en el botón "Eliminar".



Mail: palasi@palasi.com Web: www.palasi.com

Modificar un índice desde el administrador corporativo

- Seleccionamos el índice que queremos modificar desde la lista desplegable marcada con la flecha en rojo. Después hacemos las modificaciones y hacemos clic en el botón "Cerrar".



Modificar un índice en el analizador de consultas

- No se puede. Debe eliminarse el índice con **`DROP INDEX`** y debe volver a crearse con **`CREATE INDEX`**.

¿Cuándo definir índices?

- Los índices hacen más rápidas las consultas aunque ocupan espacio en disco y hacen más lentas las actualizaciones (inserción, borrado y modificación) de registros.
- Por ello, es importante ser cuidadoso en la creación de índices y sólo definirlos cuando las ventajas superan a los inconvenientes.
- Esto plantea la cuestión, ¿cuándo definimos índices en una de nuestras tablas?

¿Cuándo definimos índices en nuestras tablas?

- Buenos candidatos a índices son:
  - Campos de claves primarias y claves foráneas.
  - Campos que aparecen frecuentemente en la cláusula WHERE, en la forma WHERE campo=valor.
  - Campos que aparecen frecuentemente en las cláusulas ORDER BY o GROUP BY.
- En general, no se tienen más de tres o cuatro índices en una tabla. Los índices no deben ser campos muy grandes, ni ser de los tipos **`bit`**, **`image`** o **`text`**.

¿Qué tipo de índices?

- Índice agrupado. Sólo hay uno y es bueno crearlo antes que los índices no agrupados. Algunos criterios.
  - Cuanto más corto sea mejor (si no, aumenta el espacio en disco de la tabla).
  - Es mejor que sus valores no se actualicen frecuentemente.
  - Las claves primarias hacen buenos índices agrupados.
  - Los índices únicos también hacen buenos índices agrupados.
- Índices no agrupados. Todos los demás.

Mantenimiento de índices

- Con el tiempo los índices se fragmentan (parecido a la fragmentación de disco) y pierden efectividad.
- Por eso es bueno, regenerarlos para que se desfragmenten y tengan un mayor rendimiento.

Regeneración de índices

DBCC DBREINDEX(*nombretabla, nombreindice*)

Regenera un índice de una tabla.

DBCC DBREINDEX(*nombretabla*)

Regenera todos los índices de una tabla.

Estos comandos no son SQL estándar. Son propios de SQL Server.

¿Cuándo hay que regenerar los índices?

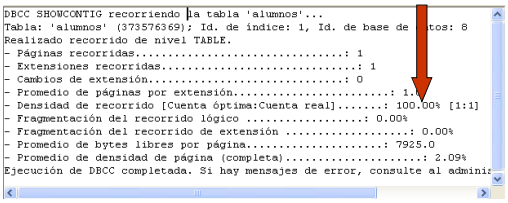
- Para saber cuando debemos regenerar los índices, obtenemos algunas estadísticas con la función.

DBCC SHOWCONTIG(*nombretabla, nombreindice*)

- Este comando no es SQL estándar. Es propio de SQL Server.
- Aparecerá una lista de datos. No los veremos todos.

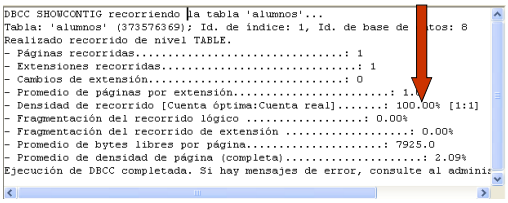
¿Cuándo hay que regenerar índices?

- De todos los datos que aparecen, sólo nos fijamos en "Densidad de recorrido" ("Scan density" en las versiones en inglés).



¿Cuándo hay que regenerar índices?

- Cuando más alto sea el porcentaje, mejor se encontrará el índice.
- Si vemos que el porcentaje es bajo, probablemente sea un buen momento para regenerar los índices.



Ejercicio para hacer ya

- Tenemos una tabla de empresas con los siguientes campos: clave de empresa(autonumérico), nombre de empresa, código de la empresa cliente, nombre del dueño, apellidos del dueño, teléfono, fax, correo electrónico y monto del activo.
- Las consultas más frecuentes son por el nombre de empresa, código de la empresa cliente y conjuntamente con el nombre y apellidos del dueño.
- Creen la tabla en SQL Server y definan los índices más convenientes, indicando si son agrupados o no agrupados.

Solución (1)

- Uno de los propósitos de un índice es el acceso más rápido a los datos. Por ello, las consultas más frecuentes son buenas candidatas para un índice. En nuestro caso, sería un índice para el nombre de empresa, otro para el código de la empresa cliente y otro índice con los dos campos de nombre y apellidos del dueño.
- Además otro de los propósitos de un índice es asegurar la unicidad de los datos. Parece lógico, pues, asociar índices únicos para la clave de la empresa, nombre de empresa y código de la empresa cliente.

Mail: palasi@palasi.com Web: www.palasi.com

Solución (2)

- Tenemos, pues los siguientes índices:
  - Clave de empresa. ÚNICO.
  - Nombre de empresa. ÚNICO.
  - Código de empresa cliente. ÚNICO.
  - Nombre del dueño+Apellidos del dueño.
- El último índice no debería ser el agrupado porque es largo y puede variar. El nombre de la empresa es también largo.
- De los dos que quedan, la clave de empresa es más difícil que varíe, es único y es clave primaria. Parece el candidato ideal.

Ejercicio para ya. Decidan qué índices, créenlos y entreguen el respaldo de la BD.

Asistentes
• CodAsist. A(6).
• Nombre. T(50).
• Apellidos T(50).
• DUI N(20).
• CodPais N(3)
• CodTarifa N(3)
• Telefono T(7).
• Direccion T(150).
• Hotel. T(50).

TiposTarifa
• CodTarifa A(3)
• TipoTarifa T(10)
• Importe N(3)

Paises
• CodPais A(3)
• Pais T(50)
• Avion L(1)

Hagan las suposiciones que crean convenientes.

Ejercicio

- Añadan índices a su base de datos de "Agencia de viajes".

Temario del curso

- 1. Introducción a las bases de datos relacionales.
- 2. Creación de la estructura de bases de datos.
- 3. Recuperación y actualización de datos.
- 4. Vistas.
- 5. Seguridad. Permisos de usuario.
- 6. OLAP (Data warehousing)

3. Recuperación y actualización de datos.

- 3.1. Recuperación de datos.
- 3.2. Actualización de datos.
- 3.3. Aspectos procedimentales.
- 3.4. Lotes y transacciones.
- 3.5. Restricciones de integridad.
- 3.6. Procedimientos almacenados.
- 3.7. Triggers.

3. Recuperación y actualización de datos.

- 3.1. Recuperación de datos.
- 3.2. Actualización de datos.
- 3.3. Aspectos procedimentales.
- 3.4. Lotes y transacciones.
- 3.5. Restricciones de integridad.
- 3.6. Procedimientos almacenados.
- 3.7. Triggers.

Mail: palasi@palasi.com Web: www.palasi.com

Recuperación de datos

- Hasta ahora habíamos visto las bases de datos como algo estático: un almacén de datos donde guardábamos datos con una cierta estructura.
- Para crear las estructuras de BDs usábamos el subconjunto DDL del lenguaje SQL.
- Sin embargo, el objetivo de las bases de datos es poder recuperar y actualizar estos datos de forma dinámica.
- A partir de ahora, nos dedicaremos a la recuperación y actualización de esos datos, usando el subconjunto DML de SQL.
- Comenzaremos con la recuperación.

La recuperación de datos es la operación más crítica de un SGBD

- Hay más operaciones de recuperación de datos que de actualización (inserción, borrado y modificación), de ahí la importancia de su eficiencia.
- Las operaciones de recuperación gestionan más información que las de actualización. Esto tiene implicaciones en el desempeño y el tráfico de red.
- Las operaciones de recuperación tienen muchas más variantes que las de actualización: podemos recuperar por criterios muy diversos.

3.1. Recuperación de datos.

- 3.1.1. Instrucciones **SELECT** simples.
- 3.1.2. Campos calculados.
- 3.1.3. Selección de registros.
- 3.1.4. Agrupación de registros.
- 3.1.5. Selección de varias tablas.
- 3.1.6. Subconsultas, **UNION** y **SELECT INTO**.

3.1. Recuperación de datos.

- 3.1.1. Instrucciones **SELECT** simples.
- 3.1.2. Campos calculados.
- 3.1.3. Selección de registros.
- 3.1.4. Agrupación de registros.
- 3.1.5. Selección de varias tablas.
- 3.1.6. Subconsultas, **UNION** y **SELECT INTO**.

Instrucciones SELECT

- La instrucción SELECT es la instrucción que nos permite recuperar datos en SQL.
- Probablemente, es la instrucción más compleja, flexible y potente que haya tenido nunca un lenguaje de programación.
- Tiene tantas posibilidades que es difícil explicarla.
- Aquí se explicará poco a poco: de los ejemplos más sencillos a los más complicados.

Instrucción SELECT. Recupera datos de los registros de 1 BD

```
SELECT listacampos1      Qué datos se recuperan
FROM listatablas         De qué tablas son los reg.
WHERE condiciones1       Qué condic. cumplen los reg.
ORDER BY listacampos2    Cómo se ordenan los reg.
GROUP BY campo           Cómo se agrupan los registros
HAVING condiciones2      Qué condic. cumplen los grupos
```

- Dentro del WHERE puede haber subconsultas.
- Dentro del FROM se pueden definir combinaciones (JOIN)

Mail: palasi@palasi.com Web: www.palasi.com

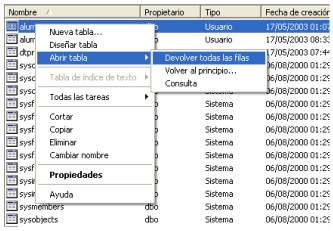
Para escribir los ejemplos, crearemos la siguiente tabla

```
CREATE TABLE Facturas (
 IdFactura smallint IDENTITY NOT NULL,
 NumFactura smallint NOT NULL,
 TipoFactura tinyint NOT NULL,
 Cliente char(50) NOT NULL,
 Descripcion char(200),
 VtasExentas smallint NOT NULL,
 VtasGravadas smallint NOT NULL,
 TipoIva tinyint NOT NULL
)
```

La instrucción anterior creará la siguiente tabla

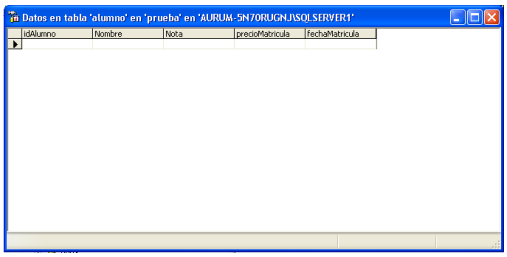
Facturas		
IdFactura	smallint	IDENTITY
NumFactura	smallint	
TipoFactura	tinyint	
Cliente	char(50)	
Descripcion	char(200)	
VtasExentas	smallint	
VtasGravadas	smallint	
TipoIva	tinyint	

Añadiendo registros de prueba



- En el administrador corporativo se hace clic derecho en la tabla. Se selecciona la opción "**Abrir tabla**" y en el menú que se despliega "**Devolver todas las filas**".

Aparece una cuadrícula

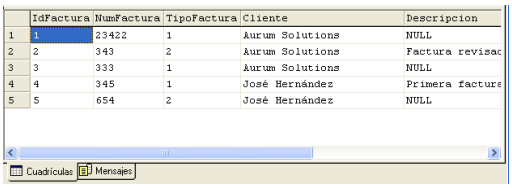


- Aquí podemos introducir los registros de nuestra tabla.

Ejercicio

- Añadan unos registros de prueba a la tabla que acaban de crear.
- Repitan el nombre del mismo cliente en varias facturas.
- Hagan que algunas facturas tengan tipo IVA 0 y otras 13.
- Llenen el campo "Descripción" de algún registro y en otros lo dejan en blanco.

Nosotros usaremos el siguiente ejemplo



	IdFactura	NumFactura	TipoFactura	Cliente	Descripcion
1	1	23422	1	Aurum Solutions	NULL
2	2	343	2	Aurum Solutions	Factura revisac
3	3	333	1	Aurum Solutions	NULL
4	4	345	1	José Hernández	Primera factura
5	5	654	2	José Hernández	NULL

Cuadrículas | Mensajes

Recuperando todos los registros de una la tabla

- Para recuperar todos los registros de una tabla con todos sus campos se utiliza

```
SELECT * FROM nombretabla
```

- Si no queremos recuperar todos los campos sino sólo unos cuantos, utilizamos

```
SELECT nombrecampo1,…,nombrecampon
FROM nombretabla
```

Hagan la siguiente recuperación

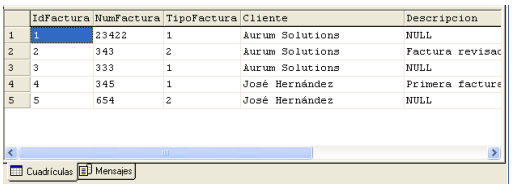
- Recuperar todos los registros de la tabla Facturas.
- Escriban la sentencia SQL en el administrador de consultas y ejecútenla.

Deberán obtener algo así

La instrucción:

```
SELECT * FROM Facturas
```

- Los resultados aparecen en la ventana de cuadrículas, debajo de la instrucción SQL.



	IdFactura	NumFactura	TipoFactura	Cliente	Descripcion
1	1	23422	1	Aurum Solutions	NULL
2	2	343	2	Aurum Solutions	Factura revisac
3	3	333	1	Aurum Solutions	NULL
4	4	345	1	José Hernández	Primera factura
5	5	654	2	José Hernández	NULL

Cuadrículas | Mensajes

Hagan más recuperaciones

- Recuperen todos los registros pero sólo con los campos “Cliente” y “VtasGravadas”.
- Recuperen todos los registros pero sólo con el campo “Cliente”

Resultados

```
SELECT Cliente, VtasGravadas FROM
Facturas
```

	Cliente	VtasGravadas
1	Aurum Solutions	200
2	Aurum Solutions	123
3	Aurum Solutions	233
4	José Hernández	200
5	José Hernández	0

```
SELECT Cliente FROM Facturas
```

	Cliente
1	Aurum Solutions
2	Aurum Solutions
3	Aurum Solutions
4	José Hernández
5	José Hernández

Fíjense que en la última recuperación se repiten los clientes

	Cliente
1	Aurum Solutions
2	Aurum Solutions
3	Aurum Solutions
4	José Hernández
5	José Hernández

- Esto es porque hay un resultado por cada registro de la tabla original.
- Si simplemente queremos saber todos los clientes que hay en la tabla, es decir, queremos eliminar los resultados repetidos, podemos poner la palabra **DISTINCT** después del **SELECT**.

Uso de DISTINCT

- Resultado de

```
SELECT DISTINCT Cliente FROM Facturas
```

	Cliente
1	Aurum Solutions
2	José Hernández

- Como vemos la palabra DISTINCT elimina los resultados repetidos.
- Pruébenlo en su copia de SQL server.

¿Qué pasa si queremos los resultados ordenados?

- Para ello utilizamos la cláusula **ORDER BY**
- La sintaxis de esta cláusula es

```
SELECT...FROM...ORDER BY
campo1, campo2, ..., campon
```

- Los resultados se ordenan por ***campo1***. Los resultados con el mismo ***campo1*** se ordenan por ***campo2*** y así sucesivamente.

Ejemplo

- Recuperen todos los registros con todos los campos, pero ordenados por el monto de las ventas gravadas.

Solución

```
SELECT * FROM Facturas ORDER BY
  VtasGravadas
```

	IdFactura	Num...	T	Cliente	Descripcion	V...	VtasGravadas	T
1	5	654	2	José ...	NULL	100	0	1
2	2	343	2	Aurum...	Factura revisada por...	234	123	0
3	1	23422	1	Aurum...	NULL	100	200	1
4	4	345	1	José ...	Primera factura de J...	0	200	1
5	3	333	1	Aurum...	NULL	122	233	1

- Como vemos, los registros aparecen en orden ascendente: los que tienen menos ventas gravadas al principio y los que más al final.

¿Qué pasa si queremos que estén en orden descendente?

- Podemos incluir después del campo de orden la palabra **DESC**.

```
SELECT * FROM Facturas ORDER BY
  VtasGravadas DESC
```

	IdFactura	Nu...	T..	Cliente	Descrip...	VtasExentas	VtasGravadas
1	3	333	1	Aurum Solutio...	NULL	122	233
2	4	345	1	José Hernánde...	Primer...	0	200
3	1	23422	1	Aurum Solutio...	NULL	100	200
4	2	343	2	Aurum Solutio...	Factur...	234	123
5	5	654	2	José Hernánde...	NULL	100	0

Hay varios campos con las mismas ventas gravadas

- Para ordenar estos campos que tienen las mismas ventas gravadas, podemos dar otro campo de ordenación

```
SELECT * FROM Facturas ORDER BY
  VtasGravadas, VtasExentas
```

- Los registros con las mismas ventas gravadas se ordenarán en orden (ascendente, pues no hemos puesto DESC de ventas exentas).

	I...	N...	T..	Cliente	Descri...	VtasExentas	VtasGravadas	TipoIva
1	5	654	2	José ...	NULL	100	0	13
2	2	343	2	Aurum...	Factu...	234	123	0
3	4	345	1	José ...	Prime...	0	200	13
4	1	...	1	Aurum...	NULL	100	200	13
5	3	333	1	Aurum...	NULL	122	233	13

ik

3.1. Recuperación de datos.

- 3.1.1. Instrucciones SELECT simples.
- 3.1.2. Campos calculados.
- 3.1.3. Selección de registros.
- 3.1.4. Agrupación de registros.
- 3.1.5. Selección de varias tablas.
- 3.1.6. Subconsultas, **UNION** y **SELECT INTO**.

¿Qué pasa si queremos saber el IVA que se cobró en la factura?

- Esta información no está en la tabla, pero es fácilmente deducible de ella, ya que el IVA es el producto de las ventas gravadas por el tipo de IVA partido por 100.
- Podemos hacer

```
SELECT *, (VtasGravadas*TipoIVA)/100
  FROM Facturas
```

En los resultados aparece un nuevo campo

- Que tiene el IVA de cada factura, es decir, el producto de las ventas gravadas por el IVA dividido por 100. Pruébenlo ustedes.

	I...	N...	T...	C.	Des...	V..	VtasGravadas	TipoIva	(Sin nombre de columna)
1	1	...	1	...	NULL	100	200	13	26
2	2	343	2	...	Fa...	234	123	0	0
3	3	333	1	...	NULL	122	233	13	30
4	4	345	1	...	Pr...	0	200	13	26
5	5	654	2	...	NULL	100	0	13	0

- Fíjense que el campo que se ha calculado no tienen nombre. En vez de su nombre aparece el texto "Sin nombre de columna".

Poniendo nombre al nuevo campo

- Para ello se utiliza la cláusula **AS**, seguida del nombre que queremos poner al nuevo campo calculado.

```
SELECT *,(VtasGravadas*TipoIVA)/100
  AS Iva FROM Facturas
```

El nuevo campo aparecerá con el nombre "Iva".

	I...	NumF...	T.	C...	Desc...	VtasExentas	VtasGravadas	TipoIva	Iva
1	1	23422	1	...	NULL	100	200	13	26
2	2	343	2	...	Fac...	234	123	0	0
3	3	333	1	...	NULL	122	233	13	30
4	4	345	1	...	Pri...	0	200	13	26
5	5	654	2	...	NULL	100	0	13	0

AS también puede servir para poner nombre a campos no calculados

- **AS** puede servir para cambiar el nombre a un campo ya existente.
- Por ejemplo, suponemos que el campo **NumFactura** queremos que aparezca como **Numero**, y descripción como **Comentarios** para mayor claridad. Podemos usar.

```
SELECT NumFactura As Numero,
  TipoFactura,Cliente, Descripcion
  AS Comentarios FROM Facturas
```

Resultado

- Como vemos, el campo de número de factura aparece como "Numero" y la de "Descripción" como "Comentarios".

	Numero	TipoFactura	Cliente	Comentarios
1	23422	1	Aurum Solutions	NULL
2	343	2	Aurum Solutions	Factura revisada por...
3	333	1	Aurum Solutions	NULL
4	345	1	José Hernández	Primera factura de J...
5	654	2	José Hernández	NULL

Mail: palasi@palasi.com Web: www.palasi.com

Recapitulando

- La instrucción

```
SELECT *,(VtasGravadas*TipoIVA)/100
  AS Iva FROM Facturas
```

- Nos ha demostrado que, al hacer una consulta, no tenemos porqué limitarnos a los campos que hay en la tabla, sino que podemos hacer operaciones y obtener nuevos campos calculados.
- En este ejemplo, tenemos dos operaciones * y /.
- ¿Hay más operaciones que podemos utilizar?

Funciones aritméticas que podemos usar (e es una expresión numérica)

`+,-,*,/,%`	Op. aritm. División es entera si los dos son enteros y, si no, real. % es el resto.
`ABS(e)`	Valor absoluto
`SIGN(e)`	Signo de la expresión: positivo (1), negativo (-1) o cero (0).
`CEILING(e)`	El entero por bajo.
`FLOOR(e)`	El entero por alto.
`ROUND(e,d)`	e redondeado a d decimales.
`POWER(e,y)`	Potencia de e elevado a y.
`SQUARE(e)`	e elevado al cuadrado
`SQRT(e)`	Raíz cuadrada de e

Más funciones aritméticas que podemos utilizar (cálculos científicos)

`SIN(e)`, `COS(e)`, `TAN(e)`, `COT(e)`, `ASIN(e)`, `ACOS(e)`, `ATAN(e)`, `ATN2(e)`, `RADIANS(e)`, `DEGREES(e)`, `PI()`	Funciones trigonométricas: seno, coseno, tangente, cotangente, arco seno, arco coseno, arco tangente, arco cotangente en radianes, conversión de grados en radianes, conversión de radianes a grados, número π
`LOG(e)`, `LOG10(e)`, `EXP(e)`	Logaritmo natural, logaritmo en base 10 y exponencial.

Funciones de cadena que podemos usar (e es una expresión de cadena)

`UPPER(e)`	Convierte a e en sólo mayúsculas
`LOWER(e)`	Convierte a e en sólo minúsculas
`LTRIM(e)`	Suprime los blancos a la izquierda.
`RTRIM(e)`	Suprime los blancos a la derecha.
`STR(n)`	Convierte una expresión numérica n en una expresión de caracteres
`+`	Concatena dos cadenas (infijo)
`LEFT(e,n)`	Obtiene los n primeros caracteres a la izquierda.
`RIGHT(e,n)`	Lo mismo pero a la derecha.
`SUBSTRING(e,i,l)`	Obtiene l caracteres empezando desde el carácter de inicio i

Funciones de cadena que podemos usar (e es una expresión de cadena)

`ASCII(e)`	Valor en ASCII del 1er. carácter.
`CHAR(n)`	Carácter con código ASCII n.
`UNICODE(e)`	Valor en Unicode del 1er. cará.
`NCHAR(n)`	Carácter con código Unicode n
`CHARINDEX(e,e2)`	1era posición en que la cadena e se encuentra dentro de c2. 0 si no se encuentra.
`REPLICATE(e,n)`	Repite n veces e.
`REPLACE(e,e2,e3)`	Reemplaza en la cadena e las ocurrencias de e2 por e3.
`STUFF(e,i,l,e2)`	En la cadena e, borra los l caracteres que comienzan en i, insertando e2 en ese punto.

Otras funciones de cadena

`SPACE(n)`
`PATHINDEX`
`DIFFERENCE`
`SOUNDEX`
`PATINDEX`
`QUOTENAME`
	No las veremos aquí. Consultar la ayuda.

Funciones de fecha que podemos usar

`GETDATE()`	Obtiene la fecha actual.
`DATEPART(pf,f)`	Obtiene de la fecha f, la parte de fecha pf.
`DATEADD(pf,n,f)`	Añade n partes de fecha pf a la fecha f
`DATEDIFF(pf,f1,f2)`	Indica cuantas partes pf hay entre f1 y f2
`DATENAME(pf,f)`	Devuelve el nombre (en inglés) de la parte de fecha pf en la fecha f.

- Las partes de fecha se indican así: dd(día), mm(mes), yy(año), hh(hora), mi(minuto),ss(segundo), wk(semana), dw(día de la semana).

Otras funciones

`ISNULL(e,v)`	Si la expresión e es NULL, devuelve v. Si no, devuelve e.
`ISNUMERIC(e)`	Verifica si es un formato numérico válido (0-válido, 1-inválido)
`ISDATE(e)`	Verifica si es un formato de fecha válido.

Otras (ver ayuda): `COALESCE, COL_NAME, COL_LENGTH, DATALENGTH, DB_ID, DB_NAME, GETANSINULL, HOST_ID, HOST_NAME, IDENT_INCR, IDENT_SEED, INDEX_COL, NULLIF, OBJECT_ID, OBJECT_NAME, STATS_DATE, SUSER_ID, SUSER_NAME, USER_ID, USER_NAME`

3.1. Recuperación de datos.

Selección de registros

- Hasta ahora, hemos visto que las instrucciones devuelven todos los registros. Pero muchas veces lo que queremos es obtener sólo unos registros específicos. Esto lo conseguimos con la cláusula `WHERE`.

```
SELECT .. FROM .. WHERE condición
```

Sólo se seleccionarán los registros que cumplen la condición. Así:

```
SELECT * FROM Facturas WHERE
  VtasGravadas>4000
```

Sólo se recuperarán los registros con ventas gravadas superiores a 4000 dólares.

Condiciones de la cláusula WHERE

- c es un nombre de campo, e es una expresión.

`c > e`	El valor del campo c es mayor que el de la expresión e
En vez de `>` : `>=, <, <=, =, <>`	Mayor o igual, menor, menor o igual, igual, diferente.
`c BETWEEN e AND e2`	El valor de c está entre e y e2.
`c NOT BETWEEN e AND e2`	El valor de c está entre e y e2.

Condiciones de la cláusula WHERE

`c IN (v1,..,v2)`	El valor de c está en la lista de valores v1..v2
`c NOT IN (v1,..,v2)`	El valor de c no está en la lista de valores
`c IS NULL`	El valor de c es nulo.
`c IS NOT NULL`	El valor de c no es nulo
`c LIKE p`	El valor de c sigue el patrón p. Nota: c debe ser de tipo textual.

`p` es 1 cadena que puede contener caracteres y comodines:
%, cadena de cero o más caracteres. _, un solo carácter.
[], un carácter que está entre los corchetes.[abc]
[^], un carácter que no está entre los corchetes.[^abc]

Mail: palasi@palasi.com Web: www.palasi.com

Condiciones de la cláusula WHERE

`NOT con1`	No se cumple la condición con1
`con1 AND con2`	Se cumplen las condiciones con1 y con2
`con1 OR con2`	Se cumple una de las dos condiciones con1 y con2

con1 y con2 son condiciones: tanto las que se han visto anteriormente como las que están en esta tabla. Por ello, pueden haber complejas expresiones con **AND**, **OR** y **NOT**. Para expresar el orden de evaluación de estas expresiones complejas, se usan paréntesis.

Nota importante

- Todas las funciones que hemos visto antes: aritméticas, de cadena, de fecha y otras, pueden aplicarse también a la cláusula WHERE.

3.1. Recuperación de datos.

Funciones de agregado

- Hasta ahora, hemos visto cada registro de forma separada.
- Pero, a veces, necesitamos obtener información resumen de todos los registros o de grupos de registros.
- Por ejemplo, queremos sumar todas las ventas gravadas de las facturas del cliente "José Hernández". Esto se escribiría así:

```
SELECT SUM(VtasGravadas) FROM
Facturas WHERE cliente='José
Hernández'
```

- La función **SUM** se le llama función de agregado ("de resumen") pues resume varios registros.

Funciones de agregado

- Son funciones que resumen información de todos los registros o de grupos de registros (c es un nombre de campo). Entre el **SELECT** y el **FROM**.

`COUNT(*)`	Número de registros
`COUNT DISTINCT(c)`	Número de registros con diferente valor de c.
`MAX(c)`	Valor máximo del campo para los registros
`MIN(c)`	Valor mínimo
`SUM(c)`	Suma del valor del campo para los registros.
`AVG(c)`	Media del valor del campo para esos registros.

Funciones de agregado

`VAR(c), STDEV(c)`	Varianza y desviación típica

- `SUM DISTINCT(c), AVG DISTINCT(c)`. Suma los valores distintos del campo y media aritmética de los valores distintos.

GROUP BY

- Cuando queremos agrupar grupos de registros.
- Se suele utilizar con funciones de agregados.

```
SELECT Cliente, SUM(VtasGravadas)
FROM Facturas GROUP BY Cliente.
```

- Agrupa los registros por cliente y saca la suma para cada cliente de las ventas gravadas. Obtenemos una lista de clientes con sus sumas de ventas gravadas.

HAVING

- Se utiliza con **GROUP BY**. Sirve para seleccionar los grupos de registros. Es como **WHERE** pero para los grupos de registros, no para los registros.

```
SELECT Cliente, SUM(VtasGravadas)
AS Ventasgrav FROM Facturas GROUP
BY Cliente HAVING Ventasgrav>10000
```

- Agrupa los registros por cliente y saca la suma para cada cliente de las ventas gravadas. Sólo me muestra los clientes que tienen más de 10000 dólares en ventas gravadas (los importantes).

3.1. Recuperación de datos.

Varias tablas

- Hasta ahora, hemos recuperado registros de sólo una tabla.
- Pero frecuentemente necesitamos recuperar datos utilizando de varias tablas.
- Esto se realiza gracias el concepto de combinación ("join")

Combinación

- Podemos verla como una tabla temporal que se obtiene a partir de dos tablas preexistentes.
- La tabla condensa de alguna manera la información de las tablas preexistentes.

Ejemplo de combinación

Departa

001	Compras	566
014	Producción	927
015	Ventas	566
016	RRHH	653

Código Nombre Jefe

Empleados

566	Juan	Pérez
567	Raúl	López
927	María	Portillo

DUI Nombre Apellido

Combinación

001	Compras	566	566	Juan	Pérez
014	Producción	927	927	María	Portillo
015	Ventas	566	566	Juan	Pérez

Código Nombre Jefe DUI Nombre Apellido

Como vemos, una combinación es una "unión" de las dos tablas. Pero no una "unión" cualquiera. Tiene unas reglas.

Mail: palasi@palasi.com Web: www.palasi.com

Una regla de una combinación

Departa		
001	Compras	566
014	Producción	927
015	Ventas	566
016	RRHH	653

Código Nombre Jefe

Empleados		
566	Juan	Pérez
567	Raúl	López
927	María	Portillo

DUI Nombre Apellido

Combinación					
001	Compras	566	566	Juan	Pérez
014	Producción	927	927	María	Portillo
015	Ventas	566	566	Juan	Pérez

Código Nombre Jefe DUI Nombre Apellido

La tabla de combinación tiene todos los campos de las dos tablas.

La principal regla de una combinación

Departa		
001	Compras	566
014	Producción	927
015	Ventas	566
016	RRHH	653

Código Nombre Jefe

Empleados		
566	Juan	Pérez
567	Raúl	López
927	María	Portillo

DUI Nombre Apellido

Combinación					
001	Compras	566	566	Juan	Pérez
014	Producción	927	927	María	Portillo
015	Ventas	566	566	Juan	Pérez

Código Nombre Jefe DUI Nombre Apellido

Hay dos campos especiales en cada combinación: uno por cada tabla. Se llaman los campos de combinación (amarillo).

Se unen los registros con el mismo valor de los campos de combinación

Departa		
001	Compras	566
014	Producción	927
015	Ventas	566
016	RRHH	653

Código Nombre Jefe

Empleados		
566	Juan	Pérez
567	Raúl	López
927	María	Portillo

DUI Nombre Apellido

Combinación					
001	Compras	566	566	Juan	Pérez
014	Producción	927	927	María	Portillo
015	Ventas	566	566	Juan	Pérez

Código Nombre Jefe DUI Nombre Apellido

Lo vemos en este ejemplo. Juan Pérez, se empareja con los 2 registros de Departamentos con los que coincide los campos de combinación.

Los registros que no coinciden su campos de comb. con el de la otra tabla

NO SE INCLUYEN EN LA COMBINACIÓN

Ejemplo: Raúl López

Departa		
001	Compras	566
014	Producción	927
015	Ventas	566
016	RRHH	653

Código Nombre Jefe

Empleados		
566	Juan	Pérez
567	Raúl	López
927	María	Portillo

DUI Nombre Apellido

Combinación					
001	Compras	566	566	Juan	Pérez
014	Producción	927	927	María	Portillo
015	Ventas	566	566	Juan	Pérez

Código Nombre Jefe DUI Nombre Apellido

Esto es el tipo de combinación más común, llamada INNER JOIN.

SINTAXIS SQL PARA UN INNER JOIN

```
SELECT ..
FROM tabla1 INNER JOIN tabla2 ON
  tabla1.campo1 = tabla2.campo2
WHERE..
```

- Como se ve, se indican las tablas implicadas y los campos de combinación.
- Nota: en toda la sentencia **SELECT**, se diferencia de qué tabla viene cada campo, con la sintaxis ***nombretabla.nombrecampo***

INNER JOIN

Departa		
001	Compras	566
014	Producción	927
015	Ventas	566
016	RRHH	653

Código Nombre Jefe

Empleados		
566	Juan	Pérez
567	Raúl	López
927	María	Portillo

DUI Nombre Apellido

Departa INNER JOIN Empleados ON Departa.Jefe=Empleados.DUI

001	Compras	566	566	Juan	Pérez
014	Producción	927	927	María	Portillo
015	Ventas	566	566	Juan	Pérez

Código Nombre Jefe DUI Nombre Apellido

El INNER JOIN se destaca porque los registros que no se corresponden, no aparecen en la combinación.

OTRA SINTAXIS SQL PARA UN INNER JOIN

```
SELECT . .
FROM tabla1 INNER JOIN tabla2 ON
  tabla1.campo1 = tabla2.campo2
WHERE. .
```

Se prefiere la primera. Es más clara y es ANSI.

```
SELECT . .
FROM tabla1, tabla2
WHERE tabla1.campo1 = tabla2.campo2
  AND..
```

Se puede hacer una combinación con la misma tabla

- Esta combinación o "autojoin" usa alias para poder escribir bien la consulta

```
FROM tabla t1 INNER JOIN tabla
 t2 ON t1.campo1 = t2.campo2
```

¿Y si queremos incluir los registros cuyos campos de comb. no coinciden?

- Para ello, hay otros tipos de combinaciones diferentes del **INNER JOIN**.
- Se llaman **OUTER JOINs** y hay tres tipos.

LEFT OUTER JOIN

```
SELECT . .
FROM tabla1 LEFT OUTER JOIN tabla2
  ON tabla1.campo1 = tabla2.campo2
WHERE. .
```

- Se incluyen todos los registros que coinciden y también los registros de ***tabla1*** (la tabla de la izquierda) que no coinciden, rellenándolos con **NULLs**.

LEFT OUTER JOIN

Departa

001	Compras	566
014	Producción	927
015	Ventas	566
016	RRHH	653

Código Nombre Jefe

Empleados

566	Juan	Pérez
567	Raúl	López
927	María	Portillo

DUI Nombre Apellido

Como vemos, el departamento de RRHH no coincide pero como es de la primera tabla, se incluye RELLENÁNDOSE DE NULLs

Departa LEFT OUTER JOIN Empleados ON Departa.Jefe=Empleados.DUI

001	Compras	566	566	Juan	Pérez
014	Producción	927	927	María	Portillo
015	Ventas	566	566	Juan	Pérez
016	RRHH	653	NULL	NULL	NULL

Código Nombre Jefe DUI Nombre Apellido

RIGHT OUTER JOIN

```
SELECT . .
FROM tabla1 RIGHT OUTER JOIN tabla2
  ON tabla1.campo1 = tabla2.campo2
WHERE. .
```

- Se incluyen todos los registros que coinciden y también los registros de ***tabla2*** (la tabla de la derecha) que no coinciden, rellenándolos con **NULLs**.

Mail: palasi@palasi.com Web: www.palasi.com

RIGHT OUTER JOIN

Departa

001	Compras	566
014	Producción	927
015	Ventas	566
016	RRHH	653

Código Nombre Jefe

Empleados

566	Juan	Pérez
567	Raúl	López
927	María	Portillo

DUI Nombre Apellido

Departa RIGHT OUTER JOIN Empleados ON Departa.Jefe=Empleados.DUI

001	Compras	566	566	Juan	Pérez
014	Producción	927	927	María	Portillo
015	Ventas	566	566	Juan	Pérez
NULL	NULL	NULL	567	Raúl	López

Código Nombre Jefe DUI Nombre Apellido

Como vemos, Raúl López no coincide pero como es de la segunda tabla, se incluye RELLENÁNDOSE DE NULLs

FULL OUTER JOIN

```
SELECT ..
FROM tabla1 FULL OUTER JOIN tabla2
  ON tabla1.campo1 = tabla2.campo2
WHERE..
```

- Se incluyen todos los registros que coinciden y también los registros de ***tabla1*** y ***tabla2*** (las dos tablas) que no coinciden, rellenándolos con **NULLs**.

FULL OUTER JOIN

Departa

001	Compras	566
014	Producción	927
015	Ventas	566
016	RRHH	653

Código Nombre Jefe

Empleados

566	Juan	Pérez
567	Raúl	López
927	María	Portillo

DUI Nombre Apellido

Departa FULL OUTER JOIN Empleados ON Departa.Jefe=Empleados.DUI

001	Compras	566	566	Juan	Pérez
014	Producción	927	927	María	Portillo
015	Ventas	566	566	Juan	Pérez
016	RRHH	653	NULL	NULL	NULL
NULL	NULL	NULL	567	Raúl	López

Código Nombre Jefe DUI Nombre Apellido

Tanto Raúl López como RRHH se incluyen

Abreviaciones

INNER JOIN se abrevia **JOIN**
LEFT OUTER JOIN se abr. **LEFT JOIN**
RIGHT OUTER JOIN se abr. **RIGHT JOIN**
FULL OUTER JOIN se abr. **FULL JOIN**

¿Por qué hacer combinaciones?

- Hay veces que necesitamos obtener información de varias tablas.
- Para hacerlo, creamos una combinación y realizamos una sentencia **SELECT** sobre esa combinación.
- Ejemplo: queremos ver una lista de nombres de departamentos con los apellidos de sus jefes.

Nombres de departamentos con apellidos de sus jefes

Departa

001	Compras	566
014	Producción	927
015	Ventas	566
016	RRHH	653

Código Nombre Jefe

Empleados

566	Juan	Pérez
567	Raúl	López
927	María	Portillo

DUI Nombre Apellido

- Lo que queremos recuperar está en dos tablas, por lo tanto, haremos una combinación.
- Como nos interesan, sólo los departamentos y jefes que coinciden haremos un INNER JOIN.

Mail: palasi@palasi.com Web: www.palasi.com

Nombres de departamentos con nombres de sus jefes

```
SELECT Departa.Nombre,Empleados.Apellido
  FROM Departa JOIN Empleados
       ON Departa.Jefe = Empleados.DUI
```

- Esta es la combinación INNER JOIN.

Departa INNER JOIN Empleados ON Departa.Jefe=Empleados.DUI					
001	Compras	566	566	Juan	Pérez
014	Producción	927	927	María	Portillo
015	Ventas	566	566	Juan	Pérez
Código	Nombre	Jefe	DUI	Nombre	Apellido

Como sólo recuperamos dos campos de la combinación

```
SELECT Departa.Nombre,Empleados.Apellido
  FROM Departa JOIN Empleados
       ON Departa.Jefe = Empleados.DUI
```

- Este es el resultado de la consulta.

Resultado del SELECT	
Compras	Pérez
Producción	Portillo
Ventas	Pérez
Nombre	Apellido

3.1. Recuperación de datos.

- 3.1.1. Instrucciones SELECT simples.
- 3.1.2. Campos calculados.
- 3.1.3. Selección de registros.
- 3.1.4. Agrupación de registros.
- 3.1.5. Selección de varias tablas.
- 3.1.6. Subconsultas, **UNION** y **SELECT INTO**.

Subconsultas

- A veces para especificar la condición WHERE de una consulta utilizamos una consulta SELECT..FROM..WHERE.
- A esto se le llaman subconsultas.

Un tipo de subconsulta

```
SELECT
FROM
WHERE campo >
      (SELECT..
       FROM..
       WHERE..)
```

subconsulta

- La subconsulta debe devolver un único valor. En vez de >, se puede usar, <, >, <=, >=, <>.

Un tipo de subconsulta

```
SELECT
FROM
WHERE campo IN
      (SELECT..
       FROM..
       WHERE..)
```

subconsulta

- Sólo se devuelven los registros que tienen como valor del campo uno que está en los resultados de la subconsulta. También se usa **NOT IN**.

Mail: palasi@palasi.com Web: www.palasi.com

Otro tipo de subconsulta

```
SELECT
FROM
WHERE EXISTS
     (SELECT..
      FROM..
      WHERE..)
```

subconsulta

- Sólo se devuelven los registros para los cuales, la subconsulta devuelve resultados. También se usa **NOT EXISTS**.

Si la consulta y la subconsulta tienen la misma tabla

```
SELECT
FROM tabla t1
WHERE EXISTS
     (SELECT..
      FROM tabla t2
      WHERE..)
```

- Se pueden usar alias, para distinguir entre las dos. Sobre todo, si la subconsulta hace referencia a algun dato de la consulta.

Nota: Las subconsultas son costosas

- Degradan el rendimiento, pues consumen muchos recursos.
- Es mejor evitarlas cuando se puede.
- Las subconsultas con IN y EXISTS se pueden sustituir siempre con joins, lo que es más eficiente.

Ejemplo

UNION ALL

- Se puede combinar el resultado de dos consultas con el resultado **UNION ALL**.

```
SELECT .. FROM .. WHERE
UNION ALL
SELECT .. FROM .. WHERE
```

- Devuelve todos los resultados de las dos consultas (deben tener el mismo número, tipo y orden de los campos).

UNION

- Es lo mismo que UNION pero suprime los registros repetidos.

```
SELECT .. FROM .. WHERE
UNION
SELECT .. FROM .. WHERE
```

SELECT..INTO

- Si después del **SELECT** ponemos un **INTO**, se crea una nueva tabla y en ella se incluyen todos los resultados de la consulta.

```
SELECT ..
INTO nombreTablaNueva
WHERE ..
```

SELECT TOP

- Con TOP n podemos obtener los n primeros registros del resultado con TOP n PERCENT el n% registros primeros.

```
SELECT TOP n
INTO nombreTablaNueva
WHERE ..
```

Ejercicio

- Escriban las sentencias SELECT que permitan realizar las consultas que se les incluye en el documento adjunto.
- Para ello, deberán utilizar la base de datos de Planilla, que se les proporcionará y que tiene la siguiente estructura.

Tabla Empleados

IDENTIDAD

IdEmp	smallint	2	
Apellido	char	20	✓
Trabajo	char	20	✓
Jefe	smallint	2	✓
FechaInicio	datetime	8	✓
Salario	smallint	2	✓
Comision	smallint	2	✓
Departamento	tinyint	1	✓

Tabla Departamentos

IDENTIDAD

Nombre de columna	Tipo de datos	Longitud	Permitir valore
IdDep	tinyint	1	
NombreDep	char	14	✓
Ciudad	char	14	✓

Tabla Departamentos

IdDep	NombreDep	Ciudad
1	Contabilidad	San Salvador
2	Compras	Santa Ana
3	Ventas	Soyapango
4	Operaciones	Santa Ana

Mail: palasi@palasi.com Web: www.palasi.com

Tabla Empleados

IdEmp	Apellido	Trabajo	Jefe	FechaInicio	Salario	Comision	Depa
1	Portillo	Oficinista	13	17/12/2000	800	<NULL>	2
2	Melara	Vendedor	6	20/02/2001	1600	300	3
3	Flores	Vendedor	6	22/02/2001	1250	500	3
4	Bonilla	Gerente	9	02/04/2001	2975	<NULL>	2
5	Barrera	Vendedor	6	28/09/2001	1250	1400	3
6	Funes	Gerente	9	01/05/2001	2850	<NULL>	3
7	Ayala	Gerente	9	09/06/2001	2450	<NULL>	1
8	López	Analista	4	09/12/2002	3000	<NULL>	2
9	Murray	Presidente	<NULL>	17/11/2001	5000	<NULL>	1
10	Cubías	Vendedor	6	08/09/2001	1500	0	3
11	Aparicio	Oficinista	8	12/01/2003	1100	<NULL>	2
12	Cruz	Oficinista	6	03/12/2001	950	<NULL>	3
13	Gálvez	Analista	4	03/12/2001	<NULL>	<NULL>	2
14	Escobar	Oficinista	7	23/01/2002	1300	<NULL>	1

3. Recuperación y actualización de datos.

Instrucción INSERT

- Sirve para añadir nuevos registros a una tabla.
- Sintaxis simplificada:

```
INSERT INTO tabla (listacampos) VALUES
  (listavalores)
```

- Donde ***tabla*** es el nombre de la tabla a la que queremos insertar el nuevo registro.
- ***listacampos*** son los nombres de los campos que queremos especificar.
- ***listavalores*** es la lista de valores de esos campos

Instrucción INSERT: Ejemplo

```
INSERT INTO empleados
  (DUI,nombre,apellido,edad,
  teléfono) VALUES (566,'Aparicio',
  'Gómez',23,'226-3456')
```

- Añade un nuevo registro a la tabla de empleados especificando todos sus datos.

Otra sintaxis de la instrucción INSERT

```
INSERT INTO tabla SELECT
  instrucción_select
```

- Los registros que devuelve la ***instrucción_select*** son los que se insertan en la ***tabla***.
- Por supuesto, el número y tipo de los campos del resultado del **SELECT** deben concordar con los de la ***tabla.***

Instrucción INSERT: Ejemplo

```
INSERT INTO nuevosEmpleados
   SELECT *  FROM empleados WHERE
     anoInicio = 2003
```

- Incluye en la tabla **nuevosEmpleados los** resultados del **SELECT**, es decir, los empleados que iniciaron el 2003.

Instrucción UPDATE

- Sirve para modificar registros existentes en una tabla.
- Sintaxis simplificada:

```
UPDATE tabla SET campo = expresion
  WHERE condiciones
```

- Donde ***tabla*** es el nombre de la tabla que contiene los registros a modificar.
- ***condiciones*** son las condiciones que deben cumplir los registros a modificar.
- ***campo*** es el nombre del campo que se modificará.
- ***expresion*** es una expresión que devuelve el nuevo valor que tendrá el campo.

Instrucción UPDATE

- Si queremos actualizar varios campos al mismo tiempo:

```
UPDATE tabla SET campo1 = expresion1,
campo2 = expresion2, . . . , campon =
  expresionn WHERE condiciones
```

Instrucción UPDATE

```
UPDATE empleados SET edad =31
WHERE nombre = 'Juan' AND
apellido = 'Pérez'
```

- Cambia la edad a 31 a todos los empleados que se llaman Juan Pérez.

Instrucción DELETE

- Sirve para borrar registros de una tabla.
- Sintaxis simplificada:

```
DELETE FROM tabla WHERE condiciones
```

- Donde ***tabla*** es la tabla que contiene los registros a borrar.
- ***condiciones*** son las condiciones que deben cumplir los registros que se borrarán.

Instrucción DELETE: Ejemplo

```
DELETE FROM empleados WHERE edad
  > 60
```

- Borra todos los empleados de edad mayor de 60.

3. Recuperación y actualización de datos.

Mail: palasi@palasi.com Web: www.palasi.com

Aspectos procedimentales

- SQL estándar es un lenguaje que sólo permite hacer definición, recuperación y modificación de datos en una BD relacional.
- Esto no es suficiente para programar. Se necesitan aspectos como variables, control de flujo, etc.
- Para añadir estos aspectos, hay dos soluciones posibles.

Dos soluciones posibles

- Utilizar SQL desde otro lenguaje de programación (SQL incrustado o "embedded"). Así:
  - Programar desde Java y enviar comandos SQL desde el programa Java (mediante JDBC o un motor de persistencia).
  - Programar desde Visual Basic y enviar comandos SQL desde el programa VB (mediante ADO).
- Extender SQL para añadir los aspectos que faltan: variables, control de flujo, etc.

SQL Server sigue las dos soluciones

- Siempre se puede enviar comandos SQL desde otro lenguaje de programación, como Java o Visual Basic. El estudio de esta solución está más allá del alcance de este curso.
- Además SQL Server extiende SQL para añadir aspectos procedimentales, como variables y control de flujo, que es lo que veremos ahora.

Variables

- Transact-SQL nos permite definir variables locales para poder guardar datos interesantes.
- Estas variables se escriben con una arroba delante.
- Así, para declarar una variable, usaremos

```
DECLARE @nombrevariable tipo
```

donde ***tipo*** es uno de los tipos de SQL Server, que ya hemos visto.

Ejemplos de declaración de variables

```
DECLARE @apellido char(50)
DECLARE @edad tinyint
DECLARE @fecha smalldatetime
```

- Igual que en otros lenguajes, es obligatorio declarar las variables antes de poder usarlas.

Para asignar valor a las variables, usamos el comando SET

- El comando SET es la instrucción de asignación en Transact-SQL.

Su sintaxis es:

```
SET @nombrevariable = expresion
```

Ejemplos:

```
SET @variable=2
SET @variable=@variable+1
```

Para imprimir el valor de las variables por pantalla, usamos PRINT

- Su sintaxis es:

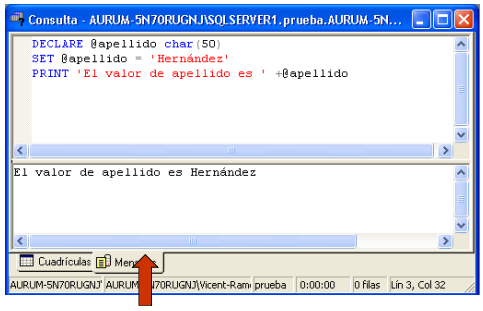
```
PRINT expresiónCadena
```

- Por ejemplo:

```
PRINT @apellido
PRINT 'Se imprimirá esta frase'
PRINT 'El valor de apellido es'
  +@apellido
```

El resultado de PRINT aparecerá en la pestaña de mensajes

- No en la pestaña de cuadrículas. Probar.



Podemos asignar el resultado de una sentencia SELECT a una variable

- Para ello, precedemos el resultado que queremos devolver con el nombre de la variable y un igual. Así, una sentencia SQL normal sería:

```
SELECT exp1, exp2, ..,expn FROM  ...
```

- Si quisiéramos guardar los resultados de las expresiones ***exp1*** y ***expn*** (pero no la expresión ***exp2***) en variables, escribiríamos

```
SELECT @var1=exp1, exp2, ..., @varn=expn FROM
  ...
```

Ejemplo de usar SELECT para guardar el valor de la variable

- La variable **@numeroRegistros** contendrá el número de registros de la tabla.

```
SELECT @numeroRegistros = COUNT(*)
  FROM tabla
```

- Después, podremos escribir:

```
PRINT 'El numero de registros de la
  tabla es'+ @numeroRegistros
```

- Importante: El **SELECT** debe devolver un único valor para poder asignar.

Control de flujo

- En un programa, el orden de ejecución predeterminado es ejecutar primero los comandos que hay arriba y después los que hay abajo.
- Este orden puede alterarse con las instrucciones de control de flujo.
- Transact-SQL tiene las instrucciones de control de flujo de todo lenguaje imperativo: el condicional **IF** y el ciclo **WHILE**.
- No explicaremos su funcionamiento pues ustedes ya las conocen de otros lenguajes. Sólo explicaremos su sintaxis.

Instrucción IF

- Sintaxis:

```
IF condición
    BEGIN
      instrucciones
    END
ELSE
    BEGIN
      instrucciones
    END
```

- La parte a partir de **ELSE** es opcional. Si sólo hay una instrucción se puede prescindir de **BEGIN** y **END**.

Ejemplo de instrucción IF

```
IF (SELECT COUNT(*) FROM tabla)>0
  PRINT 'La tabla tiene registros'
ELSE
  PRINT 'La tabla está vacía'
```

Fijémonos que una instrucción SELECT que devuelve un único resultado puede formar parte de una instrucción.

Esto lo habíamos visto para la asignación, pero puede aplicarse en cualquier expresión en Transact-SQL

Instrucción WHILE

- Sintaxis:

```
WHILE condición
    BEGIN
        instrucciones
    END
```

- Si sólo hay una instrucción se puede prescindir de **BEGIN** y **END**.

Ejemplo de instrucción WHILE

```
DECLARE @sueldo smallint
DECLARE @numemple smallint
SET @sueldo=0
WHILE @sueldo<10000
  BEGIN
   SELECT @numemple=COUNT(*) FROM
   empleados WHERE salario BETWEEN
   @sueldo AND @sueldo+499
   PRINT 'Hay '+LTRIM(STR(@numemple))+'empleados'
        +'entre'+ LTRIM(STR(@sueldo))+'dólares y'+
         LTRIM(STR(@sueldo+499))+'dólares'
    SET @sueldo = @sueldo+500
  END
```

Ejercicio

- Con la base de datos de Empleados y Departamentos, que hemos visto hasta ahora, guardar en una variable el número de Departamentos que no tienen empleados e imprimirlo con **PRINT**.

Solución

```
DECLARE @numDep tinyint

SELECT @numDep= COUNT (*) FROM
    Departamentos LEFT OUTER JOIN
    Empleados ON Departamentos.idDep =
    Empleados.Departamento
  WHERE Empleados.idEmp IS NULL

PRINT 'Hay '+LTRIM(STR(@numDep))+
    ' departamentos sin empleados'
```

3. Recuperación y actualización de datos.

Mail: palasi@palasi.com Web: www.palasi.com

Lotes

- Un lote es un conjunto de instrucciones Transact-SQL que acaban con la palabra clave GO. Ejemplo

```
USE prueba
INSERT INTO empleados (nombre,
  telefono) VALUES ('Juan','222-
  4532')
SELECT * FROM empleados
GO
```

- Las instrucciones se envían juntas al servidor de base de datos, donde se compilan conjuntamente, pero se ejecutan una a una.

¿Para qué los lotes?

- Como las instrucciones se envían conjuntamente y sus resultados se reciben conjuntamente se disminuye el tráfico de red.
- Como se compilan conjuntamente, esto es más eficiente.

Restricciones de los lotes

- En un lote, las instrucciones **CREATE DEFAULT**, **CREATE PROCEDURE**, **CREATE RULE**, **CREATE TRIGGER** y **CREATE VIEW** no se pueden combinar con otras instrucciones.
- En el mismo lote, no puede alterar una tabla y, a continuación, hacer referencia a los nuevos campos.

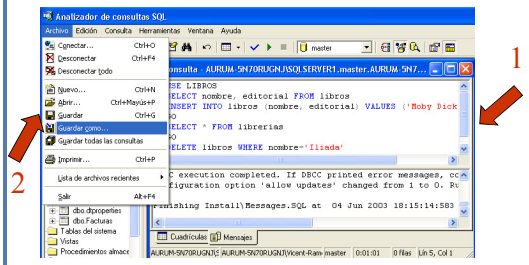
Secuencia de comandos

- Es un archivo de texto (extensión .SQL) con uno o más lotes. Por ejemplo:

```
USE LIBROS
SELECT nombre, editorial FROM libros
INSERT INTO libros (nombre,
  editorial) VALUES ('Moby
  Dick','Planeta')
GO
SELECT * FROM librerias
GO
DELETE libros WHERE nombre='Iliada'
GO
```

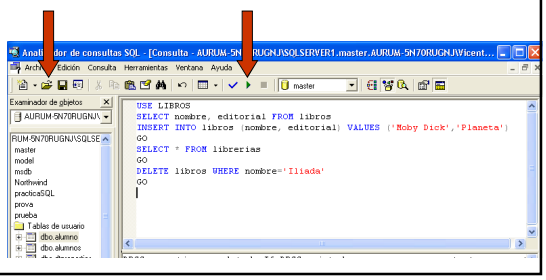
Cómo crear una secuencia de comandos

- Se escribe en el analizador de consultas y se graba con la opción "Guardar como". Es un archivo de texto común.



Cómo ejecutar una secuencia de comandos

- Se abre el archivo desde el analizador de consultas y después se ejecuta con el icono del triángulo verde.



Mail: palasi@palasi.com Web: www.palasi.com

Comentarios en las secuencias de comandos

- Si son de una línea van precedidos por `--`
- Si son de varias líneas van entre `/*` y `*/`

```
/*Este es un ejemplo de comentario de
  varias líneas*/
USE LIBROS
--Un comentario de una sola línea
SELECT nombre, editorial FROM libros
INSERT INTO libros (nombre,
  editorial) VALUES ('Moby
  Dick','Planeta')
GO
```

Integridad

- Hay integridad en una BD cuando sus datos son coherentes entre ellos mismos y con la realidad que modelan.
- Dicho de forma más sencilla: hay integridad cuando los datos son correctos.
- La integridad es lo más importante de todos los aspectos relacionados con BDs.
- La integridad se consigue:
  - Programación correcta.
  - Transacciones
  - Restricciones.
  - Triggers.

Transacción

- Conjunto de operaciones de la base de datos que se ejecutan como una unidad:
  - Se ejecutan todas.
  - O bien no se ejecuta ninguna.
- No puede ser que se ejecuten unas sí y otras no.
- Las transacciones sirven cuando tenemos operaciones que se deben ejecutar conjuntamente, si no dan error.

Ejemplo: Una transferencia bancaria

- Un señor X tiene dos cuentas bancarias y transfiere 1000 dólares de la cuenta bancaria 1 a la cuenta bancaria 2 .
- Esto son dos operaciones sobre la BD:
  - Restar 1000 del saldo de la cuenta 1 (Retirar).
  - Sumar 1000 al saldo de la cuenta 2 (Ingresar).
- Las dos deben realizarse de forma conjunta: o se realizan todas o ninguna. Si no, se pierde el dinero.

Supongamos que se realiza una y la otra no

- Las operaciones son:
  - Restar 1000 del saldo de la cuenta 1.
  - Sumar 1000 al saldo de la cuenta 2.
- Por ejemplo, si el sistema cae después de ejecutar la primera operación pero antes de la segunda, el cliente pierde 1000 dólares.
- De ahí la importancia de se ejecuten las dos operaciones o no se ejecute ninguna. Esto se consigue con transacciones.

¿Cómo hacer una transacción en SQL Server?

- Todas las operaciones de la transacción se encierran entre **`BEGIN TRAN`** i **`COMMIT TRAN`**.
- Es la forma de indicarle a SQL Server que todas las operaciones que hay entre **`BEGIN TRAN`** i **`COMMIT TRAN`** deben ejecutarse como una unidad (o todas o ninguna).

Mail: palasi@palasi.com Web: www.palasi.com

Una transacción en SQL Server

```
BEGIN TRAN
UPDATE cuentas
  SET saldo = saldo - 1000
  WHERE numCuenta='1'
UPDATE cuentas
  SET saldo = saldo +1000
  WHERE numCuenta ='2'
COMMIT TRAN
```

Si cae el sistema

```
BEGIN TRAN
UPDATE cuentas
  SET saldo = saldo - 1000
  WHERE numCuenta='1'
UPDATE cuentas
  SET saldo = saldo +1000
  WHERE numCuenta ='2'
COMMIT TRAN
```

No se hace ninguna operación

Todas se hacen

Cómo funciona esto

```
BEGIN TRAN
UPDATE cuentas
  SET saldo = saldo - 1000
  WHERE numCuenta='1'
UPDATE cuentas
  SET saldo = saldo +1000
  WHERE numCuenta ='2'
COMMIT TRAN
```

Las operaciones se guardan en un buffer

En el COMMIT se escriben en la BD

Si no se hace ninguna operación, se dice que "se cancela la transacción"

```
BEGIN TRAN
UPDATE cuentas
  SET saldo = saldo - 1000
  WHERE numCuenta='1'
UPDATE cuentas
  SET saldo = saldo +1000
  WHERE numCuenta ='2'
COMMIT TRAN
```

No se hace ninguna operación. Se cancela la tr.

Todas se hacen

La transacción se cancela si el sistema cae antes del COMMIT TRAN

```
BEGIN TRAN
UPDATE cuentas
  SET saldo = saldo - 1000
  WHERE numCuenta='1'
UPDATE cuentas
  SET saldo = saldo +1000
  WHERE numCuenta ='2'
COMMIT TRAN
```

Hasta ahora, hemos visto que el único problema era que el sistema podía caer

- Sin embargo, la primera instrucción puede fallar por otros problemas: que se viole una restricción de la base de datos, que se acabe el espacio en disco, etc.
- En estos casos, la transacción también debe cancelarse.
- No puede ser que una instrucción se realice y otra no. Esto haría que el dinero se evaporara.

Mail: palasi@palasi.com Web: www.palasi.com

Lo que necesitamos es cancelar la transacción sin que se caiga el sistema

- Esto se consigue con **ROLLBACK TRAN**.
- Es la instrucción que sirve para cancelar una transacción.

La transacción se cancela si hay error en la primera operación

```
BEGIN TRAN
UPDATE cuentas
  SET saldo = saldo - 1000
  WHERE numCuenta='1'
IF ha habido error
  ROLLBACK TRAN
UPDATE cuentas
  SET saldo = saldo +1000
  WHERE numCuenta ='2'
COMMIT TRAN
```

Si ha habido error, todas las operaciones desde el BEGIN TRAN

@@ERROR

- Variable que devuelve el número de error de la última instrucción Transact-SQL ejecutada.
- Cuando SQL Server completa con éxito la ejecución de una instrucción SQL, en @@ERROR se establece el valor 0.
- Si se produce un error, @@ERROR devuelve el número de error. Se puede ver el texto asociado a un número de error en la tabla de sistema **sysmessages** de la base de datos **master**

La secuencia de comandos quedaría así

```
BEGIN TRAN
UPDATE cuentas
  SET saldo = saldo - 1000
  WHERE numCuenta='1'
IF @@ERROR>0
  ROLLBACK TRAN
UPDATE cuentas
  SET saldo = saldo +1000
  WHERE numCuenta ='2'
COMMIT TRAN
```

¿Qué pasa si queremos deshacer sólo parte de la transacción?

- No queremos deshacer todas las operaciones sino sólo unas cuantas.
- Para ello, encerramos las operaciones que queremos deshacer entre dos operaciones:

```
SAVE TRAN nombre
ROLLBACK TRAN nombre
```

La secuencia de comandos quedaría así

```
BEGIN TRAN
INSERT INTO ...
SAVE TRAN borrar
DELETE movimientos
  WHERE numCuenta='1'
DELETE cuentas
  WHERE numCuenta='1'
IF @@ERROR > 0
 ROLLBACK TRAN borrar
(más operaciones)
COMMIT TRAN
```

Si falla el borrado de la cuenta, deshacemos el borrado de los movimientos. Sin embargo el INSERT INTO, no se deshace.

Mail: palasi@palasi.com Web: www.palasi.com

Estas "cancelaciones parciales" también se pueden hacer de otra forma

- Transacciones anidadas: son transacciones que están dentro de otras transacciones.

```
BEGIN TRAN
  instrucciones
  BEGIN TRAN
    instrucciones
    BEGIN TRAN
      instrucciones
    COMMIT TRAN
    instrucciones
  COMMIT TRAN
  instrucciones
COMMIT TRAN
```

La transacción en azul está anidada dentro de la transacción en verde.

La transacción en verde está anidada dentro de la transacción en rojo.

¿Para qué sirven las transacciones anidadas?

- No es por si se cae el sistema, pues en ese sentido se comportan como 1 sola transacción

```
BEGIN TRAN
  instrucciones
  BEGIN TRAN
    instrucciones
    BEGIN TRAN
      instrucciones
    COMMIT TRAN
    instrucciones
  COMMIT TRAN
  instrucciones
COMMIT TRAN
```

Si cae el sistema antes del último COMMIT TRAN, no se realiza ninguna de las operaciones.

Si cae después del último COMMIT TRAN, se realizan todas.

Esto es como si sólo existiera la transacción en rojo.

Entonces, ¿para qué sirven las transacciones anidadas?

- Sirven para deshacer parcialmente ciertas operaciones. Por ejemplo:

Entonces, ¿para qué sirven las transacciones anidadas?

```
BEGIN TRAN
INSERT INTO ...
BEGIN TRAN
DELETE movimientos
  WHERE numCuenta='1'
DELETE cuentas
  WHERE numCuenta='1'
IF @@ERROR > 0
  ROLLBACK TRAN
ELSE
  COMMIT TRAN
(más operaciones)
COMMIT TRAN
```

- Sirven para deshacer parcialmente ciertas operaciones, como en este ejemplo

Entonces, ¿para qué sirven las transacciones anidadas?

```
BEGIN TRAN
INSERT INTO ...
BEGIN TRAN
DELETE movimientos
  WHERE numCuenta='1'
DELETE cuentas
  WHERE numCuenta='1'
IF @@ERROR > 0
  ROLLBACK TRAN
ELSE
  COMMIT TRAN
(más operaciones)
COMMIT TRAN
```

Si falla el borrado de la cuenta, deshacemos el borrado de los movimientos. Sin embargo el INSERT INTO, no se deshace.

Esto se hacía de otra manera antes.

Ejercicio

- En la base de datos de Departamentos y Empleados que hemos visto hasta ahora.
- Crear una secuencia de comandos con dos transacciones.
  - La primera transacción borra el departamento 3 y todos sus empleados.
  - La segunda transacción borra el departamento 4 y todos sus empleados.
- Cada transacción debe ir en un lote diferente y debe estar compuesta por dos instrucciones SQL.

Mail: palasi@palasi.com Web: www.palasi.com

Todo son transacciones

- En SQL Server toda operación de actualización sobre la base de datos se ve como parte de una transacción.
- Si la operación no es parte de una transacción, se considera como una transacción de una sola operación.
- SQL Server trata toda la actualización de la base de datos como un conjunto de transacciones que están ejecutándose.

Concurrencia y transacciones

- ¿Qué pasa si dos transacciones sobre los mismos registros se ejecutan al mismo tiempo?
- Respuesta: En una transacción "pura", no se pueden ejecutar al mismo tiempo. La base de datos hace que primero se ejecute una y luego otra.
- Se dice que la BD secuencializa (o "serializa") las transacciones que afectan a los mismos datos.

Una transacción debe seguir la regla ACID

- **A**tómica. Se ejecuta todos los cambios de la transacción o no se ejecuta ninguno.
- **C**onsistente. Deja el estado de la BD de forma consistente (unos datos no se contradicen con otros).
- **I**solated ("aislada"). Se ejecuta de forma aislada de otras transacciones. Es decir, las otras transacciones no influyen en ella ni ella en las otras. Esto se consigue con la **serialización.**
- **D**uradera. Una vez la transacción acaba con COMMIT, sus cambios permanecen en la BD.

Centrémonos en la I

- ¿Cómo se consigue el aislamiento de las transacciones?
- Ya lo hemos dicho: haciendo que, para los mismos datos, una transacción se ejecute una detrás de la otra (serialización). Esto se implementa con bloqueos.
- Sin embargo, esto degrada el rendimiento, pues las transacciones no se pueden ejecutar paralelamente, sino una detrás de otra.

Es por eso que SQL Server permite suavizar el nivel de aislamiento

- En SQL Server, podemos especificar cuán aisladas pueden estar las transacciones.
- Esto nos permite gestionar un compromiso:
  - Cuánto menos aisladas, más eficiente la transacción.
  - Cuánto más aisladas, más se preserva la integridad de los datos.
- Hay que ver qué nivel de aislamiento se usa en cada caso.

Hay cuatro niveles de aislamiento en SQL Server

- Read uncommited. Ningún aislamiento.
- Read committed.
- Repeatable read.
- Serializable. Total aislamiento.

Mail: palasi@palasi.com Web: www.palasi.com

Esto se controla con la siguiente sentencia SQL

```
SET TRANSACTION ISOLATION LEVEL nivel de aislamiento
```

- Que fija el nivel de la conexión a la BD.
- Donde *nivel de aislamiento* puede ser uno de los siguientes:

```
READ UNCOMMITTED
READ COMMITTED
REPEATABLE READ
SERIALIZABLE
```

Si no se especifica nada, la opción predeterminada es **READ COMMITED**

Vamos a verlos en la práctica

- Restauren la base de datos "aislamiento" y examinen la tabla "aislamiento".

	clave	nombre
1	10	Primer registro
2	20	Segundo registro
3	30	Tercer registro
4	40	Cuarto registro
5	50	Quinto registro

Abran el analizador de consultas

- Hagan clic en el primer icono y después en "**Ventana|Mosaico horizontal**"

- Así conseguirán dos ventanas diferentes del analizador de consultas, es decir, dos conexiones diferentes a la base de datos.

Escriban en la primera ventana

```
BEGIN TRAN
UPDATE AISLAMIENTO
  SET nombre = 'Nuevo valor'
```

- Y ejecuten con la flecha verde.
- Fíjense que no hemos acabado la transacción y los cambios están pendientes del **COMMIT TRAN**.

Escriban en la segunda ventana

```
SET TRANSACTION ISOLATION LEVEL
  READ UNCOMMITTED
SELECT * FROM AISLAMIENTO
```

- Ejecuten. Y miren lo que ven

Ven los datos que actualizó el UPDATE

- Pero estos datos eran provisionales. No eran definitivos hasta el **COMMIT TRAN**. Si la primera transacción se cancela, esto serán datos erróneos.

	clave	nombre
1	10	Nuevo valor
2	20	Nuevo valor
3	30	Nuevo valor
4	40	Nuevo valor
5	50	Nuevo valor

Mail: palasi@palasi.com Web: www.palasi.com

Veámoslo en la práctica

- En la primera ventana, bórrenlo todo y escriban **ROLLBACK TRAN**. Ejecuten.
- En la segunda ventana, hagan clic en ejecutar nuevamente. ¿Qué recuperan?

	clave	nombre
1	10	Primer registro
2	20	Segundo registro
3	30	Tercer registro
4	40	Cuarto registro
5	50	Quinto registro

Tenemos un problema

- La primera vez que leímos obtuvimos datos no correctos, pues eran provisionales y se desecharon.
- Esto quiere decir que nuestro programa ha obtenido datos erróneos y esto puede tener malas consecuencias.
- A este fenómeno se le llama “lecturas no actualizadas”.

Lecturas no actualizadas

- Se leen datos provisionales de la BD.
- Estos datos aún no han sido actualizados a la BD por el **COMMIT TRAN** y, por lo tanto, pueden ser erróneos (si la transacción se cancela porque cae el sistema o se hace un **ROLLBACK**).
- Problema que da el nivel de aislamiento **READ UNCOMMITTED**

Pero hay otros problemas

- Ejecuten en la primera ventana

```
SET TRANSACTION ISOLATION LEVEL
   READ UNCOMMITTED
BEGIN TRAN
SELECT * from AISLAMIENTO
   WHERE nombre='Primer registro'
```

	clave	nombre
1	10	Primer registro

Ahora ejecuten en la segunda ventana

```
UPDATE AISLAMIENTO
   SET clave=110 WHERE clave=10
```

Vuelvan a la primera ventana y ejecuten

```
SELECT * from AISLAMIENTO
     WHERE nombre='Primer registro'
```

- ¿Qué ven?

	clave	nombre
1	110	Primer registro

Conclusión

- La primera transacción (la de la primera ventana) no está aislada del resto.
- Los datos que hemos leído, cuando los volvemos a leer pueden haber cambiado.
- A esto se le llama “lecturas no repetibles”.

Recapitulando

- **Lecturas no actualizadas.** Se recuperan datos provisionales antes de ser confirmados (COMMITT) por la transacción.
- **Lecturas no repetibles.** Si volvemos a leer datos que ya hemos leído, pueden obtenerse resultados diferentes a la primera vez.

Nota.

- Vuelvan a la situación inicial ejecutando en la primera ventana.

```
ROLLBACK TRAN
```

- Y ejecutando en la segunda ventana

```
UPDATE AISLAMIENTO
    SET clave=10 WHERE clave=110
```

Aún hay otro problema

- Hagan en la primera ventana

```
SET TRANSACTION ISOLATION LEVEL
        READ UNCOMMITTED
BEGIN TRAN
SELECT * FROM AISLAMIENTO
```

	clave	nombre
1	10	Primer registro
2	20	Segundo registro
3	30	Tercer registro
4	40	Cuarto registro
5	50	Quinto registro

- Hagan en la segunda ventana

```
INSERT INTO AISLAMIENTO (clave, nombre)
   VALUES (25, 'Nuevo registro')
```

Ahora en la primera ventana ejecuten

```
SELECT * FROM AISLAMIENTO
```

- Aparece un nuevo registro en la misma transacción que antes no existía.

1	10	Primer registro
2	20	Segundo registro
3	25	Nuevo registro
4	30	Tercer registro
5	40	Cuarto registro
6	50	Quinto registro

- A este registro se le llama "**fantasma**"

3 problemas que se pueden dar con el aislamiento READ UNCOMMITTED

- **Lecturas no actualizadas.** Se recuperan datos provisionales antes de ser confirmados (COMMITT) por la transacción.
- **Lecturas no repetibles.** Si volvemos a leer datos que ya hemos leído, pueden obtenerse resultados diferentes a la primera vez.
- **Fantasmas.** Si volvemos a consultar una tabla, pueden aparecer registros nuevos, que antes no existían

Nota.

- Vuelvan a la situación inicial ejecutando en la primera ventana.

```
ROLLBACK TRAN
```

- Y ejecutando en la segunda ventana

```
DELETE AISLAMIENTO WHERE clave=25
```

Si queremos controlar esos problemas, podemos usar otros niveles de aislamiento

- Como vemos es un compromiso entre el aislamiento (integridad de los datos) y la concurrencia (o rendimiento de la aplicación).

+ aislamiento

	Lecturas no actualizadas	Lecturas no repetibles	Fantasmas
`READ UNCOMMITED`	SI	SI	SI
`READ COMMITED`		SI	SI
`REPEATABLE READ`			SI
`SERIALIZABLE`			

+ concurrencia

Este cuadro anterior se puede comprobar

- Simplemente, usando el mismo código anterior pero cambiando en la instrucción **`SET TRANSACTION ISOLATION LEVEL`**, el nivel **`READ UNCOMMITTED`** por los otros niveles.
- Veremos que algunas operaciones se bloquean hasta que acaba la transacción. Esto implementa un mayor aislamiento al precio de un menor rendimiento o concurrencia.
- No se recomienda por ahora. De todos modos, si lo prueban, tengan cuidado de hacerlo bien.

Recordemos: Cómo establecer el nivel de aislamiento

`SET TRANSACTION ISOLATION LEVEL` *nivel de aislamiento*

- Fija el nivel hasta que se vuelva a cambiar explícitamente.
- Este nivel es propio de cada conexión a la BD.
- Donde *nivel de aislamiento* puede ser uno de los siguientes:

```
READ UNCOMMITTED
READ COMMITTED
REPEATABLE READ
SERIALIZABLE
```

Si no se especifica nada, la opción predeterminada es `READ COMMITED`

Decíamos que una transacción era aislada (la I de ACID)

- Esto sólo ocurre con **`SERIALIZABLE`**, con la que se da un aislamiento puro.
- Con **`READ UNCOMMITTED`** no hay ningún tipo de aislamiento.
- **`READ COMMITTED`** y **`REPEATABLE READ`** tienen un aislamiento "light", pero más "light" en la primera.

Para curiosos

- SQL Server implementa el aislamiento de las transacciones bloqueando los datos.
- Hay dos clases principales de bloqueo:
  - Compartido. Impide actualizar los datos, pero estos se pueden leer.
  - Exclusivo. Impide acceder a los datos, tanto para actualizarlos como para leerlos.
- ¿Qué pasa si una transacción no puede acceder a un registro porque está bloqueado? La transacción se espera a que se desbloquee.

Bloqueos y niveles de aislamiento

- En toda situación, siempre que alguien **actualiza** un registro, impone un bloqueo exclusivo sobre él. Esto no depende del nivel de aislamiento.
- En principio, cuando se **lee** un registro (**`SELECT`**), puede o no imponerse un bloqueo compartido. Esto depende del nivel de aislamiento.

Bloqueos compartidos y niveles de aislamiento

- **READ UNCOMMITED** no implementa ningún bloqueo compartido y, por tanto, ignora los bloqueos de todo tipo de las otras conexiones.
- **READ COMMITTED** implementa un bloqueo compartido para cada uno de los registros que se leen y lo conserva hasta el fin de la operación de lectura (**SELECT**).
- **REPEATABLE READ** implementa un bloqueo compartido para cada uno de los registros que se leen y lo conserva hasta el fin de la transacción.
- **SERIALIZABLE** implementa un bloqueo compartido para todo el rango de registros de las consultas de la transacción y lo conserva hasta el fin de la transacción, de forma que no se puedan modificar ni insertar recursos en esas consultas.

Granularidad de los bloqueos

- En SQL Server hay tres niveles de granularidad para los bloqueos:
- **Fila.** Sólo bloquean un registro de la tabla.
- **Página**. Bloquean toda una página de la tabla (recordemos que los registros de una tabla se guardan en páginas).
- **Tabla.** Bloquean toda la tabla.
- SQL Server decide el nivel de granularidad a partir de los registros que se deben bloquear. Así, por ejemplo, si hay muchos registros que se deben bloquear, SQL Server puede decidir bloquear toda la tabla.

Deadlock (bloqueo indefinido)

- Cuando hay dos transacciones A y B.
- A bloquea un registro que está esperando B para bloquear.
- B bloquea otro registro que está esperando A para bloquear.
- Las dos transacciones se quedan esperando para la eternidad.
- El problema de los filósofos chinos que comen espagueti.

En SQL Server no se produce deadlock

- Cuando la situación anterior ocurre, SQL Server elige una de las transacciones y la cancela dejando a la otra transacción continuar.
- La transacción cancelada se comporta como si se hubiera hecho un ROLLBACK y un mensaje de error se envía al usuario.
- Generalmente, se cancela la transacción que cuesta menos de cancelar.

No se vayan todavía… Aún hay más.

- Lo que hemos visto sólo es lo fundamental de bloqueos en SQL Server.
- En realidad, hay más tipos de bloqueos y el tema se hace muy complejo.
- Pero en realidad entrar en esas complejidad sólo es útil en casos muy específicos.
- Normalmente, lo único que debe saber el programador son los niveles de transacción.

3. Recuperación y actualización de datos.

Mail: palasi@palasi.com Web: www.palasi.com

Restricciones de integridad

- Sabemos que una base de datos es íntegra cuando los datos que hay en ella son coherentes con la realidad y entre ellos mismos.
- Hay cuatro formas de garantizar la integridad de una BD:
  - Con una programación correcta que la preserve.
  - Con transacciones.
  - Con restricciones de integridad.
  - Con triggers.

Restricciones de integridad

- Son comprobaciones que se definen para asegurar la integridad de los datos. Se definen cuando se crea la tabla (aunque pueden alterarse).
- Cuando hacemos una operación de actualización (inserción, modificación o borrado), el SGBD comprueba si alguna de estas restricciones se viola. Si es así, cancela la operación de actualización.
- Así nos protegemos contra la introducción accidental de datos incorrectos en nuestra BD.

Restricciones de integridad

- Hay varios tipos de restricciones de integridad en SQL Server:
  - Restricciones de clave primaria.
  - Restricciones de integridad referencial (clave foránea).
  - Restricciones de unicidad.
  - Restricciones de comprobación.
- Las veremos una a una.

Restricciones de integridad

- Hay varios tipos de restricciones de integridad en SQL Server:
  - Restricciones de clave primaria.
  - Restricciones de integridad referencial (clave foránea).
  - Restricciones de unicidad.
  - Restricciones de comprobación.

Restricciones de clave primaria

- Sólo hay una por tabla y se refiere al campo que es clave primaria.
- La restricción asegura que en ese campo no se introduzcan valores repetidos.
- En su forma más sencilla, simplemente se consiguen añadiendo la palabra **PRIMARY KEY** después de la definición del campo en **CREATE TABLE**.

Creen una BD "alumno" con esta tabla "alumno"

```
USE alumno
CREATE TABLE alumno (
 idAlumno smallint IDENTITY PRIMARY KEY ,
 nombre char(50) NOT NULL,
 nota tinyint,
 precioMatricula smallmoney,
 fechaMatricula smalldatetime
)
```

El campo idAlumno es la clave primaria de la tabla. No se le pueden introducir repetidos.

Mail: palasi@palasi.com Web: www.palasi.com

¿Cómo se implementa esto?

```
USE alumno

CREATE TABLE alumno (
 idAlumno smallint IDENTITY PRIMARY KEY ,
 nombre char(50) NOT NULL,
 nota tinyint,
 precioMatricula smallmoney,
 fechaMatricula smalldatetime
)
```

Por dentro de SQL Server, implementa un índice **agrupado** único, para evitar repetidos.

¿Y si queremos que el índice no sea agrupado?

```
USE alumno

CREATE TABLE alumno (
 idAlumno smallint IDENTITY PRIMARY KEY
  NONCLUSTERED,
 nombre char(50) NOT NULL,
 nota tinyint,
 precioMatricula smallmoney,
 fechaMatricula smalldatetime
)
```

Simplemente añadimos la palabra **NONCLUSTERED**

Otra forma

```
USE alumno

ALTER TABLE alumno
   ADD CONSTRAINT cp_idAlumno
       PRIMARY KEY(idAlumno)
```

O bien

```
ALTER TABLE alumno
   ADD CONSTRAINT cp_idAlumno
       PRIMARY KEY NONCLUSTERED(idAlumno)
```

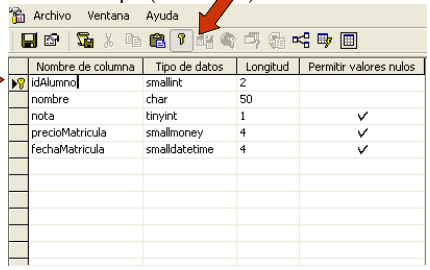
Esto nos permite añadir una restricción después de haber definido la tabla y se le puede dar un nombre.

Así se pueden definir claves primarias de varios campos

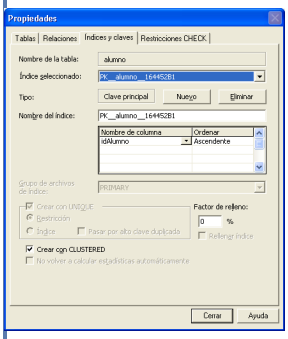
```
USE alumno

ALTER TABLE cliente
   ADD CONSTRAINT cp_cliente
     PRIMARY KEY(nombre, apellidos)
```

Desde el Administrador Corporativo

- Desde el cuadro de diálogo "Diseñar tabla" seleccionamos el campo (o campos) y hacemos clic en el icono de la llave. Nos aparecerá una dibujo de llave en el campo (o campos).



Otra forma



- En la pestaña de índices y claves, hacemos clic en el botón "Clave principal" y especificamos los campos.
- La ventaja es que aquí podemos dar un nombre a la restricción y decidir que no sea agrupada.

¿De verdad es útil esto?

- Por lo que hemos visto ahora, parece poco útil haber aprendido esto.
- Si queremos que los valores no se repitan podíamos haber creado un índice único, directamente. De todas maneras, la restricción de clave principal lo crea.
- Además, nuestras claves principales son IDENTITY por lo que tenemos asegurado que no se repitan.
- La única ventaja por ahora es saber que ese campo es la clave primaria, lo que puede tener alguna utilidad.
- La verdad es que parece poco útil.

La utilidad de la restricción de clave primaria

- No es tanto por ella misma.
- Es porque nos permite definir restricciones de clave foránea (restricciones de integridad referencial).
- Son éstas las verdaderamente importantes y las que vamos a ver ahora.

Ejercicio

- Creen una restricción de clave primaria en la tabla "alumno" de la base de datos "alumno" que se les proporcionará en formato de respaldo.
- La restricción debe tener un nombre y debe definirse con el Administrador corporativo.

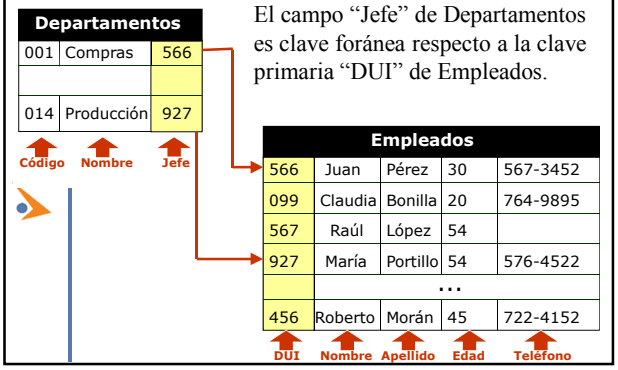
Restricciones de integridad

- Hay varios tipos de restricciones de integridad en SQL Server:
  - Restricciones de clave primaria.
  - Restricciones de integridad referencial (clave foránea).
  - Restricciones de unicidad.
  - Restricciones de comprobación.

Restricciones de integridad referencial

- O restricciones de clave foránea.
- Aseguran la integridad de los datos en una relación de clave foránea.
- Impiden que se rompa la conexión de una clave foránea con la clave primaria a la que apunta.
- Veamos dos ejemplos.

Tenemos una relación de clave foránea

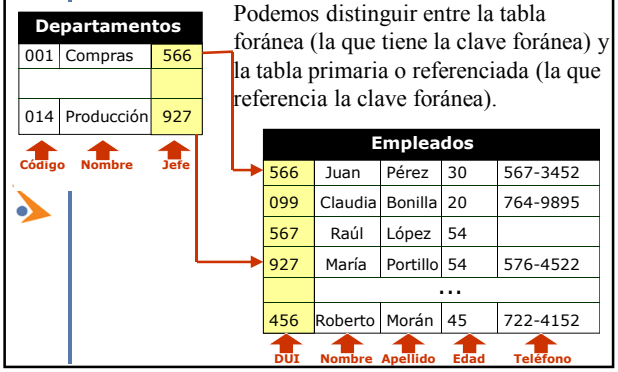


El campo "Jefe" de Departamentos es clave foránea respecto a la clave primaria "DUI" de Empleados.

Mail: palasi@palasi.com Web: www.palasi.com

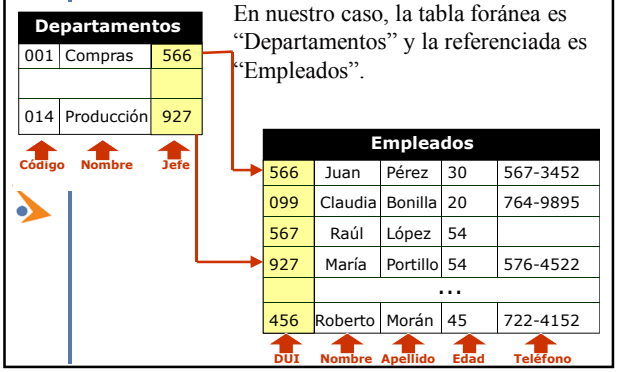
Tenemos una relación de clave foránea

Podemos distinguir entre la tabla foránea (la que tiene la clave foránea) y la tabla primaria o referenciada (la que referencia la clave foránea).



Tenemos una relación de clave foránea

En nuestro caso, la tabla foránea es "Departamentos" y la referenciada es "Empleados".

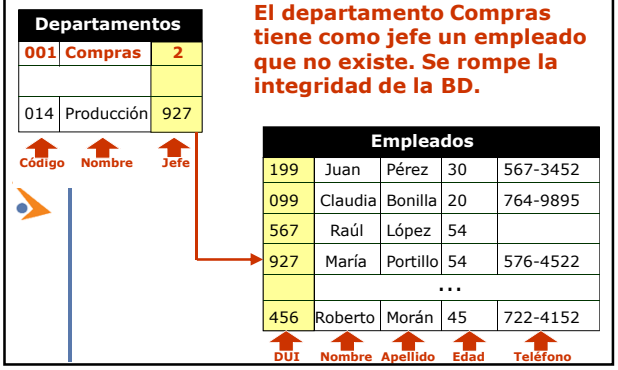


Situaciones conflictivas cuando tenemos una relación de clave foránea

- INSERT en tabla foránea. Que en la clave foránea introduzcamos un valor no válido.
- UPDATE en tabla foránea. Que modifiquemos la clave foránea a un valor no válido.
- UPDATE en tabla referenciada. Que modifiquemos a un valor no válido el campo al que referencia la clave foránea.
- DELETE en tabla referenciada. Que borremos un registro que tiene claves foráneas apuntando hacia él.

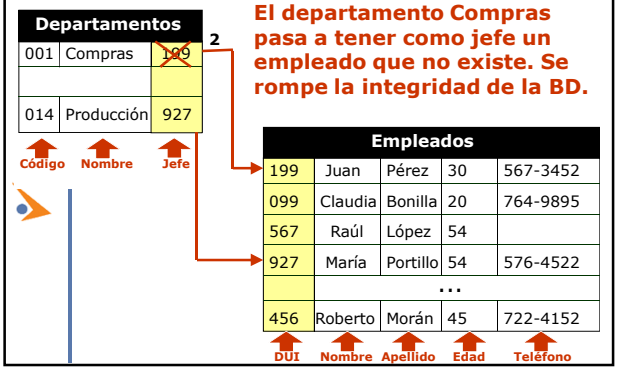
INSERT tabla foránea. ¿Si ponemos en "Jefe" un valor no adecuado?

El departamento Compras tiene como jefe un empleado que no existe. Se rompe la integridad de la BD.

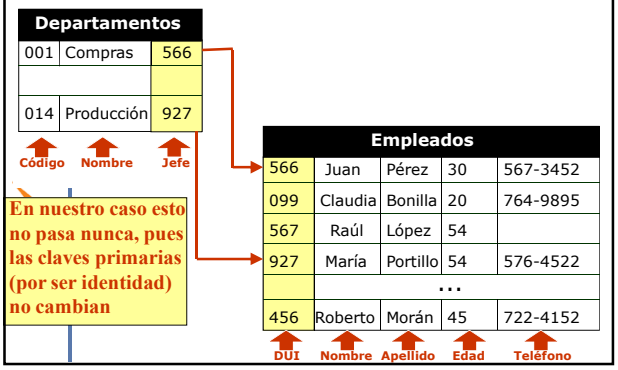


UPDATE tab foránea ¿Si actualizamos "Jefe" a un valor no adecuado?

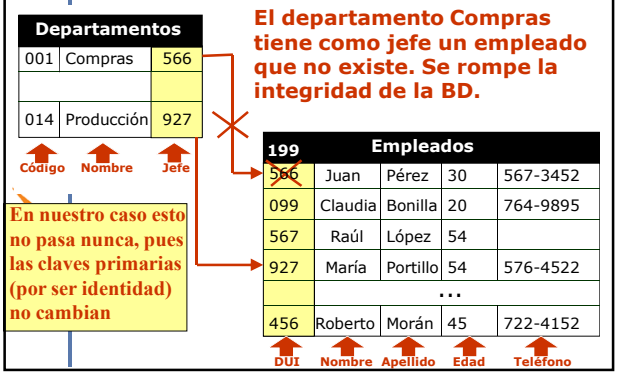
El departamento Compras pasa a tener como jefe un empleado que no existe. Se rompe la integridad de la BD.



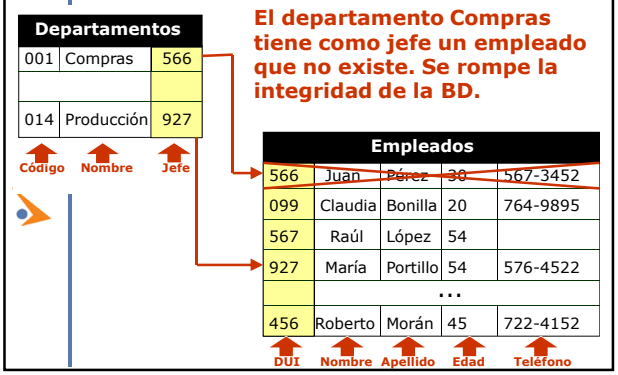
UPDATE tab referenciada ¿Si cambiamos el DUI de Juan Pérez?



UPDATE tab referenciada ¿Si cambiamos el DUI de Juan Pérez?



DELETE tab referenciada. ¿Si borramos el empleado Juan Pérez?



¿Cómo protegemos la integridad de la BD en estos casos?

- Una forma es programar de forma que esto no ocurra. Sin embargo, esto presenta algunos problemas:
  - No siempre es fácil, pues hay muchas claves foráneas en nuestra BD y, a veces, hay dependencias en cascada.
  - Si el programador comete un error, se pierde la integridad de nuestra BD.
  - Sería mejor que el SGBD controlara esto automáticamente y diera un mensaje de error si alguna de estas anomalías ocurren.

Para ello usamos la restricción de clave foránea

- En su forma más sencilla, simplemente se consiguen añadiendo, después de la definición del campo en **CREATE TABLE**, la expresión

```
FOREIGN KEY REFERENCES
tablareferenciada (camporeferenciado)
```

Así, por ejemplo

```
USE alumno

CREATE TABLE Departamentos (
 codigo tinyint IDENTITY PRIMARY KEY,
 nombre char(50) NOT NULL,
 jefe smallint FOREIGN KEY REFERENCES
  Empleados(DUI)
)
```

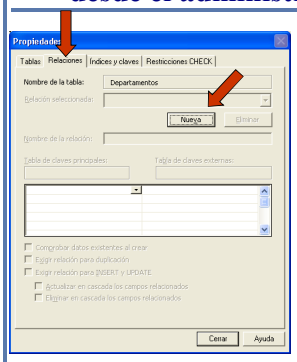
Esta definición, hará que el campo jefe siempre referencie al DUI de un empleado existente o bien contenga NULL (un Departamento sin jefe).

Otra forma

```
USE alumno

ALTER TABLE Departamentos
  ADD CONSTRAINT cf_jefe
    FOREIGN KEY (jefe) REFERENCES
      Empleados(DUI)
```

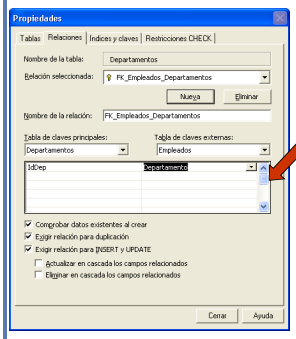
Esto permite agregar la restricción después de haber creado la tabla.

Todo esto también puede hacerse desde el administrador corporativo



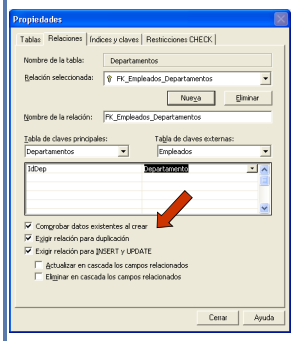
- Al lado de la pestaña de índices y claves hay una pestaña "Relaciones". Hacemos clic en el botón "Nueva".

Definiendo restricción de integridad referencial desde el adm. corporativo



- Se definen las tablas y los campos que definen la restricción de clave foránea.

Hay otras opciones



- Comprobar datos existentes al crear se asegura que los datos que hayamos introducido antes de crear la restricción, cumplan la misma.
- La otra opción no la veremos por ahora.

Una observación

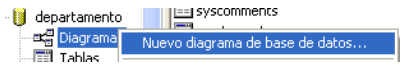
- No podremos definir restricciones de claves foráneas si el campo que es clave primaria no está definido con una restricción de clave primaria o una restricción de unicidad (ésta la veremos próximamente).
- Es ésta la importancia de las restricciones de clave primaria.

Ejercicio

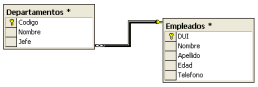
- Definan la anterior restricción de clave foránea en la base de datos de Departamentos que se les pasará en un respaldo y que se corresponde con la que hemos visto en las transparencias.

Una forma gráfica de ver las restricciones de clave foránea

- Clic derecho en Diagramas y seleccionen "Nuevo diagrama de bases de datos".



- En el asistente que aparece, añadan las tablas de Departamentos y Empleados. Les aparecerá un gráfico.



- Háganlo.

¿Qué pasa si intentamos violar una restricción de clave foránea?

Departamentos

001	Compras	566
014	Producción	927

Código Nombre Jefe

Se producirá un mensaje de error.

Empleados

566	Juan	Pérez	30	567-3452
099	Claudia	Bonilla	20	764-9895
567	Raúl	López	54	
927	María	Portillo	54	576-4522
		...		
456	Roberto	Morán	45	722-4152

DUI Nombre Apellido Edad Teléfono

Se producirá un mensaje de error en todos estos casos

- INSERT en tabla foránea. Que en la clave foránea introduzcamos un valor no válido.
- UPDATE en tabla foránea. Que modifiquemos la clave foránea a un valor no válido.
- UPDATE en tabla referenciada. Que modifiquemos a un valor no válido el campo al que referencia la clave foránea.
- DELETE en tabla referenciada. Que borremos un registro que tiene claves foráneas apuntando hacia él.

Ejercicio

- En la base de datos anterior, intenten violar la integridad referencial en los siguientes cuatro casos:
  - INSERT en tabla foránea.
  - UPDATE en tabla foránea.
  - UPDATE en tabla referenciada.
  - DELETE en tabla referenciada.
- Vean qué pasa.

Sin embargo, en los casos UPDATE y DELETE para tabla referenciada

- Hay otra opción a parte de obtener mensajes de error.
- Podríamos actualizar en cascada las claves foráneas (UPDATE) o bien borrar en cascada todos los registros (DELETE).
- Esto lo conseguiremos con

```
FOREIGN KEY REFERENCES
  tablareferenciada (camporeferenciado) ON
  DELETE CASCADE
 ON UPDATE CASCADE
```

(tanto si lo ponemos en la misma tabla o en un **ALTER TABLE**)

UPDATE tab referenciada ¿Si cambiamos DUI de Pérez con cascada?

Departamentos

001	Compras	566
014	Producción	927

Código Nombre Jefe

Ahora cambiamos el DUI para que, en vez de 566, sea 199.

Empleados

566	Juan	Pérez	30	567-3452
099	Claudia	Bonilla	20	764-9895
567	Raúl	López	54	
927	María	Portillo	54	576-4522
		...		
456	Roberto	Morán	45	722-4152

DUI Nombre Apellido Edad Teléfono

En nuestro caso esto no pasa nunca, pues las claves primarias (por ser identidad) no cambian

UPDATE tab referenciada ¿Si cambiamos DUI de Pérez con cascada?

Departamentos

001	Compras	~~566~~ 199
014	Producción	927

Código Nombre Jefe

El departamento de Compras también cambiará como clave foránea el nuevo DUI. Se mantendrá la integridad.

199 **Empleados**

~~566~~	Juan	Pérez	30	567-3452
099	Claudia	Bonilla	20	764-9895
567	Raúl	López	54	
927	María	Portillo	54	576-4522
		...		
456	Roberto	Morán	45	722-4152

DUI Nombre Apellido Edad Teléfono

En nuestro caso esto no pasa nunca, pues las claves primarias (por ser identidad) no cambian

Mail: palasi@palasi.com Web: www.palasi.com

DELETE tab referenciada. ¿Si borramos Pérez con cascada?

Ahora queremos borrar el empleado.

Departamentos

Código	Nombre	Jefe
001	Compras	566
014	Producción	927

Empleados

DUI	Nombre	Apellido	Edad	Teléfono
566	Juan	Pérez	30	567-3452
099	Claudia	Bonilla	20	764-9895
567	Raúl	López	54	
927	María	Portillo	54	576-4522
	...			
456	Roberto	Morán	45	722-4152

DELETE tab referenciada. ¿Si borramos Pérez con cascada?

El departamento Compras también se borrará. Se mantendrá la integridad.

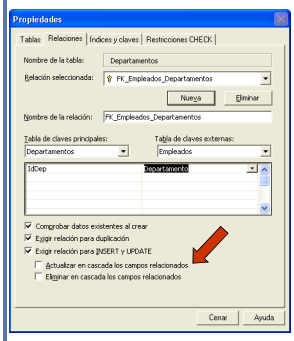
Departamentos

Código	Nombre	Jefe
~~001~~	~~Compras~~	~~566~~
014	Producción	927

Empleados

DUI	Nombre	Apellido	Edad	Teléfono
~~566~~	~~Juan~~	~~Pérez~~	~~30~~	~~567-3452~~
099	Claudia	Bonilla	20	764-9895
567	Raúl	López	54	
927	María	Portillo	54	576-4522
	...			
456	Roberto	Morán	45	722-4152

Desde el adm. corporativo



- Se definen si se quiere que se haga actualización y eliminación en cascada o no en unas casillas de verificación.

Ejercicio

- En la base de datos anterior, definan la restricción de forma que produzca actualizaciones en cascada. Después intenten violar la integridad referencial en los siguientes cuatro casos:
  - INSERT en tabla foránea.
  - UPDATE en tabla foránea.
  - UPDATE en tabla referenciada.
  - DELETE en tabla referenciada.
- Vean qué pasa.

Ejercicio

- En la base de datos de ejemplo de Planilla, que se les proporciona en respaldo de base de datos, definan las restricciones de clave primaria y clave foránea que se necesiten.

Restricciones de integridad

- Hay varios tipos de restricciones de integridad en SQL Server:
  - Restricciones de clave primaria.
  - Restricciones de integridad referencial (clave foránea).
  - Restricciones de unicidad.
  - Restricciones de comprobación.

Restricciones de unicidad

- Indican que un campo (o conjunto de campos) es único. Es decir, no se puede repetir en dos registros.
- Sirven para señalar las claves candidatas que no son clave primaria.

```
USE alumno

CREATE TABLE Departamentos (
 codigo tinyint IDENTITY PRIMARY
  KEY,
 nombre char(50) NOT NULL UNIQUE)
```

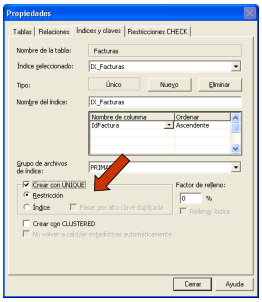
Otra forma

```
USE alumno

ALTER TABLE Departamentos
  ADD CONSTRAINT un_nombre
  UNIQUE (nombre)
```

También se pueden definir conjuntos de campos únicos, lo que va bien para claves candidatas de varios campos.

```
ALTER TABLE empleados
  ADD CONSTRAINT un_empleados
  UNIQUE (nombre, apellidos)
```

Definiendo restricción de unicidad desde el adm. corporativo



- Desde la pestaña de índices y claves, se oprime el botón "Nuevo" y después se eligen las opciones "Crear con UNIQUE" y "Restricción".

¿Para qué las restricciones de unicidad?

- Son una alternativa a definir un índice único.
- La única ventaja sobre el índice único es que se pueden definir restricciones de clave foránea sobre ella.
- Sin embargo, las restricciones de clave foránea se suelen definir sobre campos con restricciones de clave primaria (más que de unicidad).
- Tampoco nos hubiéramos muerto en el caso que no lo tuviéramos.

Ejercicio

- Creen una restricción de unicidad con el campo nombre de la tabla Departamento de la base de datos Departamento.
- Compruébenla intentando escribir dos valores repetidos en diversos registros de ese campo.

Restricciones de integridad

- Hay varios tipos de restricciones de integridad en SQL Server:
  - Restricciones de clave primaria.
  - Restricciones de integridad referencial (clave foránea).
  - Restricciones de unicidad.
  - Restricciones de comprobación.

Restricciones de comprobación

- También se llaman restricciones CHECK.
- Sirven para comprobar que un campo cumpla ciertas condiciones

Una restricción CHECK

```
USE alumno

CREATE TABLE alumno (
 idAlumno smallint IDENTITY PRIMARY KEY
  NONCLUSTERED,
 nombre char(50) NOT NULL,
 nota tinyint CHECK(nota BETWEEN 0 AND 10),
 precioMatricula smallmoney,
 fechaMatricula smalldatetime
)
```

Comprueba que la nota esté entre 0 y 10.

Otra forma

```
USE alumno

ALTER TABLE alumno
  ADD CONSTRAINT ck_nota
  CHECK (nota BETWEEN 0 AND 10)
```

Así se define después de crear la tabla.

Las expresiones que hay en una restricción de comprobación

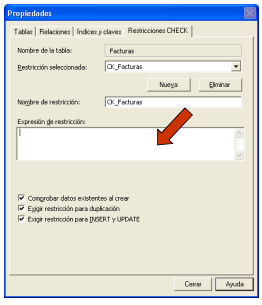
- Sólo pueden referirse a un registro de la tabla.
- Pueden referirse a varios campos. Así,

```
CHECK(fechafinal>fechainicio)
```

- Pueden usar funciones SQL

```
CHECK(fechafinal>GETDATE())
```

Definir restricción de comprobación desde el adm. corporativo



- Desde la pestaña de restricciones CHECK, contigua a la pestaña de índices y claves, se oprime el botón "Nueva", se le pone un nombre y después se escribe la expresión que la define en el cuadro llamado "Expresión de restricción".

Ejercicio

- Creen una restricción de comprobación para controlar la nota en la base de datos alumno.
- Comprueben que funciona introduciendo datos que la violan.

Mail: palasi@palasi.com Web: www.palasi.com

3. Recuperación y actualización de datos.

Procedimientos almacenados

- Recordemos que una secuencia de comandos era un conjunto de instrucciones Transact-SQL que se guardaba en un archivo de extensión .sql
- Una secuencia de comandos estaba formado por uno o varios lotes, que acababan con GO.
- Un procedimiento almacenado es casi igual que una secuencia de comandos: es un conjunto de instrucciones Transact-SQL que pueden agruparse en uno o más lotes.
- ¿Qué los diferencia de la secuencia de comandos?

La diferencia entre procedimientos almacenados y secuencias de comandos

- Es que las secuencias se almacenan fuera de la base de datos, en archivos .sql.
- En cambio, los procedimientos almacenados es lo mismo, pero se almacenan DENTRO de la base de datos.
- Esto tiene varias ventajas.

Las ventajas de los procedimientos almacenados

- Se compilan al momento de guardarlos en la base de datos. Esto quiere decir que cada vez que los ejecutamos no deben de compilarse. Esto los hace más rápido.
- Después de ejecutarse por primera vez, una copia compilada se almacena en RAM. Esto hace más rápida su ejecución.
- Con los procedimientos almacenados, se pueden pasar argumentos para obtener datos (como ya veremos).

Un procedimiento almacenado sencillo

- Para crear un procedimiento almacenado:

```
CREATE PROCEDURE nombre AS comandos-SQL
```

- Donde ***comandos-SQL*** son las sentencias que podríamos incluir en una secuencia de comandos.
- Para ejecutar este procedimiento almacenado se hace

```
EXEC nombre
```

EXEC es un comando SQL, por lo tanto dentro de un procedimiento almacenado podemos llamar a otro procedimiento almacenado.

Ejemplo

- Un ejemplo:

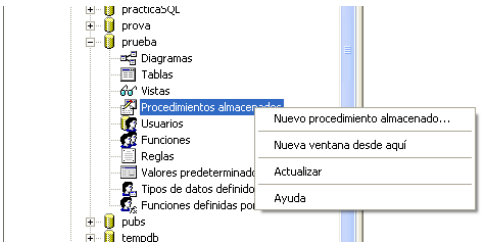
```
CREATE PROCEDURE insertaJuanyPedro AS
  INSERT INTO alumno (nombre)
     VALUES ('Juan')
  INSERT INTO alumno (nombre)
     VALUES ('Pedro')
   GO
```

- Para ejecutar este procedimiento almacenado se hace

```
EXEC insertaJuanyPedro
```

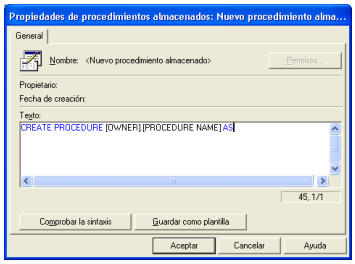

Mail: palasi@palasi.com Web: www.palasi.com

Desde el administrador corporativo



- Debajo de la base de datos, hay un icono "Procedimientos almacenados". Se hace clic derecho y se selecciona "Nuevo procedimiento almacenado" en la lista.

En la ventana que aparece



- Escribimos la instrucción Transact-SQL del procedimiento almacenado en el cuadro de texto y hacemos clic en "Aceptar".

Una mala noticia

- No se puede ejecutar un procedimiento almacenado desde el administrador corporativo.

Ejercicio

- Creen este procedimiento almacenado y comprueben que funciona.

Procedimientos almacenados con parámetros

- Los procedimientos almacenados pueden tener parámetros, como si fueran un procedimiento, función o método.
- Esto los hace más útiles, ya que los podemos utilizar para tratar datos diferentes, dependiendo del valor de los parámetros.
- Es una de las ventajas que no tienen las secuencias de comandos.

Antes que nada, veamos un ejemplo

```
CREATE PROCEDURE insertaAlumno
   @nombre char(50)
    AS INSERT INTO alumno (nombre)
    VALUES (@nombre)
   GO
```

- Esto se ejecutaría así:

```
EXEC insertaAlumno 'Juan'
```

Hay un parámetro llamado @nombre

```
CREATE PROCEDURE insertaAlumno
   @nombre char(50)
    AS INSERT INTO alumno (nombre)
    VALUES (@nombre)
   GO
```

- Lo que hace este procedimiento es insertar un nuevo alumno con el nombre que se le pasa como parámetro.
- La ventaja es que ahora, con un solo procedimiento almacenado, podemos insertar todos los alumnos. Antes sólo podíamos insertar un alumno específico, lo que no era práctico.

Hay un parámetro llamado @nombre

```
CREATE PROCEDURE insertaAlumno
   @nombre char(50)
    AS INSERT INTO alumno (nombre)
    VALUES (@nombre)
   GO
```

- Se declara el nombre del parámetro y su tipo. A esto se le llama declaración de parámetro

Hay un parámetro llamado @nombre

```
CREATE PROCEDURE insertaAlumno
   @nombre char(50)
    AS INSERT INTO alumno (nombre)
    VALUES (@nombre)
   GO
```

- Se escriben los comandos Transact-SQL del procedimiento almacenado y en ellos se utiliza el parámetro declarado.

Para ejecutar este procedimiento alamacenado

- Se pasa el parámetro después del procedimiento separado por un espacio en blanco.

```
EXEC insertaAlumno 'Juan'
```

Generalizando, tenemos

```
CREATE PROCEDURE nombrePA
   declaraciónParámetro
    AS comandos
```

- Para ejecutar:

```
EXEC nombrePA valorParámetro
```

Ejercicio

- Ejecuten este procedimiento almacenado.

Pero puede haber más de un parámetro

```
CREATE PROCEDURE nombrePA
   declaraciónParam1,
   declaraciónParam2,
   ...
   declaraciónParamn
   AS comandos
```

- Para ejecutar:

```
EXEC nombrePA valorParám1, ..., valorParamn
```

La declaración de un parámetro tiene la siguiente sintaxis

@nombre tipo

@nombre tipo = predeterminado

- Al final se puede añadir la palabra **OUTPUT** para indicar que es un parámetro de salida, que nos sirve para retornar resultados.

Ejemplo de un parámetro de salida

```
CREATE PROCEDURE buscaAlumno
   @nombre char(50),
   @id smallint OUTPUT
   AS SELECT @id = idAlumno
   FROM alumno
   WHERE nombre=@nombre
   GO
```

- Este procedimiento obtiene una variable @id con el idAlumno del alumno que le pasamos como parámetro. Se ejecutaría así

```
EXEC buscaAlumno 'Juan', @idJuan
  OUTPUT
```

Ejemplo de ejecución de un procedimiento con parámetro de salida

```
/*Se declara la variable que usaremos
  para recoger el resultado*/
DECLARE @idJuan smallint

--Se llama al procedimiento
EXEC buscaAlumno 'Juan', @idJuan
  OUTPUT

--Se imprime el resultado
PRINT @idJuan
```

Ejercicio

- Ejecuten este código en la base de datos alumno.

Ejercicio

- Creen un procedimiento almacenado que, dado un nombre de Departamento, devuelva en una variable de salida el nombre del jefe de ese Departamento.
- Expliquen cómo se ejecutaría ese procedimiento.

Solución (1)

```
CREATE PROCEDURE buscaJefe
  @departamento char(50),
  @jefe char(50) OUTPUT
AS
SELECT @jefe=Empleados.nombre
FROM Departamentos JOIN Empleados
  ON
  Departamentos.jefe=Empleados.DUI
WHERE
  Departamentos.nombre=@departamento
```

Solución (2)

```
DECLARE @jefeVentas char(50)
EXEC buscaJefe 'Ventas',
  @jefeVentas OUTPUT
PRINT 'El jefe del Departamento
  de Ventas es '+@jefeVentas
```

3. Recuperación y actualización de datos.

Triggers (desencadenadores)

- Son una clase especial de procedimientos almacenados.
- Los procedimientos almacenados estándar, se ejecutan con una sentencia **EXEC**.
- Los triggers se ejecutan automáticamente **después de que** se actualiza una tabla.

"Tres tristes triggers"

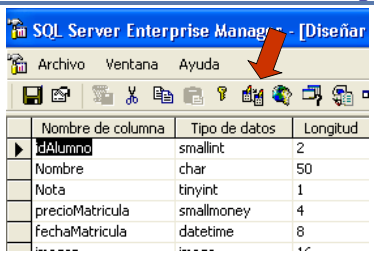
- Se pueden definir tres tipos de triggers por cada tabla:
- Hay triggers para las operaciones **INSERT**.
- Otros para las operaciones **UPDATE**.
- Otros para las operaciones **DELETE**.
- Cada trigger se ejecuta automáticamente **después de que** se realiza la operación correspondiente sobre una tabla.

Sintaxis de definición de triggers

```
CREATE TRIGGER nombreTrigger
ON nombreTabla
AFTER INSERT
AS comandosSQL
```

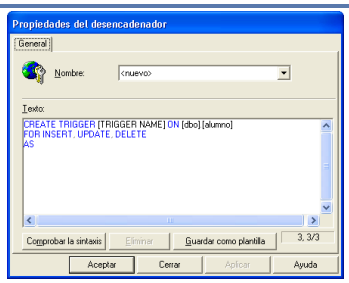
- En vez de **INSERT**, podemos poner **DELETE** o **UPDATE**.
- En vez de **AFTER** podemos poner **FOR**.

Definiendo triggers desde el Administrador corporativo



- En la ventana de "Diseñar tabla", se hace clic en el icono de "desencadenadores"

Aparece una ventana



- Donde podemos definir el desencadenador con sintaxis SQL.

Los triggers no pueden recibir parámetros

- En esto se diferencian de los procedimientos almacenados.
- Por el contrario, pueden usar dos tablas (**`inserted`** y **`deleted`**) que los procedimientos almacenados no pueden usar.

Los triggers muchas veces utilizan dos tablas

- Las tablas **`inserted`** y **`deleted`** son mantenidas por el sistema. **Tienen los mismos campos de la tabla sobre la que se define el trigger.**
- Cuando hacemos un **`INSERT`**, las filas insertadas están en la tabla **`inserted`**.
- Cuando hacemos un **`DELETE`**, las filas eliminadas están en la tabla **`deleted`**.
- Cuando hacemos un **`UPDATE`**, las filas "viejas" están en la tabla **`deleted`** y las nuevas en la tabla inserted. Es decir, un **`UPDATE`** se considera un **`DELETE`** seguido de un **`INSERT`**.

Ejemplo de un trigger (BD "Planilla")

```
CREATE TRIGGER trigerCascada
ON Departamentos
AFTER DELETE
AS DELETE FROM Empleados WHERE
  Empleados.Departamento IN
  (SELECT idDep FROM deleted)
```

- Cuando borra un Departamento, borra todos sus empleados (esto se consigue más fácilmente con una restricción de clave foránea).

Ejercicio

- Comprueben este trigger ejecutándolo.

Dos técnicas que no hacen exactamente lo mismo

- **Restricciones de clave foránea.** Conservan la integridad referencial de la BD, es decir, se aseguran que las claves foráneas apunten a un valor válido de clave primaria.
- **Triggers.** Permiten implementar la integridad referencial (como en el ejemplo anterior) pero también pueden implementar restricciones diferentes de la integridad referencial (por ejemplo, políticas de negocio).
- Los triggers son más potentes y más flexibles que las restricciones: **permiten hacer más cosas.**

Ejemplo

- Supongamos que en nuestra empresa hay una política que dice que un empleado no puede ganar más que su jefe.
- Esta restricción no la podemos implementar con una restricción de clave foránea, sin embargo, con triggers es posible.
- Decidiremos que, si un empleado gana más que el jefe, desharemos la transacción.
- Esto nos obliga a que las operaciones de inserción y modificación sobre empleados se hagan siempre dentro de una transacción.

Suponemos que insertamos nuevos registros en empleados

- La consulta que da el número de empleados insertados con más salario que el jefe es

```
SELECT COUNT(*)
FROM inserted JOIN Empleados ON
  inserted.Jefe = Empleados.idEmp
WHERE inserted.Salario >
  Empleados.Salario
```

- Es una combinación de los registros insertados con Empleados.
- Devuelve un número, que es el número de los empleados insertados que ganan más que su jefe

El trigger que necesitaríamos

```
CREATE TRIGGER TriggerInsert
ON Empleados
AFTER INSERT
AS IF (SELECT COUNT(*) FROM inserted JOIN
  Empleados ON inserted.Jefe =
  Empleados.idEmp WHERE inserted.Salario >
  Empleados.Salario)>0
    BEGIN
     PRINT 'El jefe debe ganar más'
    PRINT 'Deshaciendo la transacción'
    ROLLBACK TRAN
    END
```

También lo necesitaríamos para UPDATE

```
CREATE TRIGGER TriggerUpdate
ON Empleados
AFTER UPDATE
AS IF (SELECT COUNT(*) FROM inserted JOIN
  Empleados ON inserted.Jefe =
  Empleados.idEmp WHERE inserted.Salario >
  Empleados.Salario)>0
    BEGIN
     PRINT 'El jefe debe ganar más'
    PRINT 'Deshaciendo la transacción'
    ROLLBACK TRAN
   END
```

Ahora "inserted" contiene los registros nuevos

Como los dos son lo mismo, podemos ponerlo en un solo trigger

```
CREATE TRIGGER TriggerUpdate
ON Empleados
AFTER INSERT, UPDATE
AS IF (SELECT COUNT(*) FROM inserted
  JOIN Empleados ON inserted.Jefe =
  Empleados.idEmp WHERE
  inserted.Salario >
  Empleados.Salario)>0
    BEGIN
     PRINT 'El jefe debe ganar más'
     PRINT 'Deshaciendo la
  transacción'
     ROLLBACK TRAN
     END
```


Mail: palasi@palasi.com Web: www.palasi.com

Ejercicio

- Comprueben este trigger creándolo y realizando inserciones (¡Ojo! Dentro del analizador de consultas y de una transacción).
- Por ejemplo, pueden hacer

```
BEGIN TRAN
INSERT empleados
  (IdEmp,Apellido,Trabajo,Jefe,
  FechaInicio,Salario,Comision,
  Departamento) VALUES (15,
  'Barrera','Oficinista',1,
  '12/12/2004',1000, 500,2)
COMMIT TRAN
```

Como ven los triggers

- Permiten implementar restricciones sobre políticas de negocio, por muy complicadas y específicas que estas sean.
- En cambio, las restricciones de clave foránea sólo guardan la integridad referencial.
- Esto nos da una idea clara de cuando usar unas y otros.

Cuándo usar

- Si queremos imponer la integridad referencial, es mejor una restricción de clave foránea, pues es más simple.
- Si queremos imponer una restricción distinta de la integridad referencial, usaremos triggers, ya que son los únicos que lo pueden hacer.

Sin embargo

- Hay mucha polémica sobre si una restricción que no es de integridad referencial:
- Pertenece a la base de datos.
- Pertenece al programa (capa de negocio).
- Las BDs son para guardar datos. Sin embargo, el abuso de los triggers puede colocar reglas de negocio en la BD haciendo más difícil la programación y el mantenimiento.

Ejemplo

- Hay una política en nuestra empresa que, cuando nos compran más de 1000 dólares, entran en la categoría de cliente frecuente y reciben un descuento del 5%.
- Esto se puede hacer con un trigger. Cuando insertamos una factura, vemos si el total de las facturas del cliente es de 1000 dólares, cambiamos el campo "categoría" del registro del cliente y aplicamos un descuento de 5% a la factura que vamos insertando.

Esto es un pequeño desastre

- Es innecesariamente complejo.
- Tendremos que hacer triggers para el UPDATE y DELETE para tratar los casos en que el cliente se arrepiente y no nos compra tanto.
- Los triggers pueden desencadenar otros triggers.
- La base de datos, cuyo fin es almacenar datos, se ha convertido en una especie de pseudoprograma lamentable.
- Mejor déjenlo al programa.

Una norma orientativa

- Si lo que queremos hacer, se puede implementar como una restricción, no lo hagamos con un trigger, sino con una restricción.
- Si lo que queremos implementar puede verse como una regla del negocio y no como un asunto puramente de bases de datos, mejor usemos el programa y no un trigger.
- Si lo que queremos implementar sale muy complejo con triggers, seguramente hay que ponerlo en el programa.
- En caso contrario, podemos usar triggers.

Ejemplos

- Con una restricción. Ver que se cumple la integridad referencial.
- Con un trigger.
- Llevar un registro de las operaciones de una tabla en una tabla log.
- Rellenar los campos que no se especifican con un valor que se calcula en ese momento. Por ejemplo, campos con el último usuario que modificó el registro y la fecha de modificación.
- Llevar los registros borrados a una tabla de backup.
- Con el programa. Cambiar la categoría del cliente si compra más de 1000 dólares.

Ejercicio

- Crear un trigger que deshaga la transacción si se intenta borrar departamentos que tienen empleados. (Esto sería mejor implementarlo con una restricción de clave foránea, pero lo hacemos así para practicar).

Solución

```
CREATE TRIGGER triggerDepEmpleados
ON Departamentos
AFTER DELETE
AS
IF (SELECT COUNT(*) FROM Empleados
  WHERE Empleados.Departamento IN
  (SELECT idDep FROM deleted)) > 0
  BEGIN
    PRINT 'No se puede borrar Depar.'
    PRINT 'con empleados'
    PRINT 'Deshaciendo transacción'
    ROLLBACK TRAN
  END
```

Otra solución

```
CREATE TRIGGER triggerDepEmpleados2
ON Departamentos
AFTER DELETE
AS
IF (SELECT COUNT(*) FROM deleted
  INNER JOIN Empleados ON
  Empleados.Departamento =
  deleted.idDep) > 0
   BEGIN
    PRINT 'No se puede borrar
  Depar.'
    PRINT 'con empleados'
    PRINT 'Deshaciendo transacción'
    ROLLBACK TRAN
   END
```

Ejercicio

- Creen este trigger y compruébenlo

Mail: palasi@palasi.com Web: www.palasi.com

Pregunta

- Con esta solución, se deshace la transacción si sólo hay un único Departamento que tenga empleados.
- ¿Qué pasa si sólo quisiéramos que no se borraran los Departamentos que tienen empleados (pero los otros sí).

Solución

```
CREATE TRIGGER triggerDeshaceParte
ON Departamentos
AFTER DELETE
AS
INSERT INTO Departamentos
  SELECT DISTINCT deleted.* FROM
  deleted INNER JOIN Empleados ON
  Empleados.Departamento =
  deleted.idDep
```

Se vuelven a insertar los Departamentos borrados que tienen Empleados.

Fíjense que esto puede activar los triggers asociados con INSERT.

Ejercicio

- Creen este último trigger y compruébenlo.

Ejercicio

- Sea una tabla de productos.
  - idProd. Clave primaria. Autonumérico.
  - Nombre. Char(20).
  - Existencia. Smallint.
- Y una tabla de facturas (se supone que sólo hay un producto por factura).
  - idFac. Clave primaria. Autonuméricao.
  - Producto. Clave foránea a idProd.
  - Cantidad. Smallint.
- Escriban el trigger INSERT sobre Facturas que garanticen que no se pide más cantidad de un producto que las existencias.

El trigger para INSERT

```
CREATE TRIGGER triggerExistencia ON
  Facturas AFTER INSERT AS
DECLARE @erroneos int
SELECT @erroneos = COUNT(*)
FROM inserted JOIN Productos ON
  inserted.Producto=Productos.IdProd
WHERE
  inserted.cantidad>Productos.Existen
  cia
IF @erroneos >0
    BEGIN
        PRINT 'No hay existencias'
        ROLLBACK TRAN
    END
```

Curiosos: Los erróneos para UPDATE

```
SELECT @erroneos = COUNT(*)
FROM (inserted JOIN deleted ON
  inserted.IdFac = deleted.IdFac)
  JOIN Productos ON
  inserted.Producto=
  Productos.IdProd WHERE
```

hemos aumentado la cantidad de la factura menos que la existencia OR *hemos cambiado el producto y no hay existencias para él*

Mail: palasi@palasi.com Web: www.palasi.com

Curiosos: Los erróneos para UPDATE

```
SELECT @erroneos = COUNT(*)
FROM (inserted JOIN deleted ON
  inserted.IdFac = deleted.IdFac) JOIN
  Productos ON inserted.Producto=
  Productos.IdProd WHERE
  ((inserted.cantidad-deleted.cantidad) >
  Productos.Existencia) OR
  ((inserted.IdProd<>deleted.IdProd) AND
  (inserted.cantidad>Productos.Existencia)
  )
```

Curiosos: El trigger para UPDATE

```
CREATE TRIGGER triggerExistencia ON Facturas
  AFTER UPDATE AS
DECLARE @erroneos int
SELECT @erroneos = COUNT(*)
FROM (inserted JOIN deleted ON inserted.IdFac =
  deleted.IdFac) JOIN Productos ON
  inserted.Producto=Productos.IdProd
WHERE ((inserted.cantidad-deleted.cantidad) >
  Productos.Existencia) OR
  ((inserted.IdProd<>deleted.IdProd) AND
  (inserted.cantidad>Productos.Existencia))
IF @erroneos >0
    BEGIN
        PRINT 'No hay existencias`
        ROLLBACK TRAN
    END
```

Pregunta

- ¿Creen que sería bueno crear un trigger para actualizar las existencias a partir de las facturas?

Respuesta

- No. Sería hacer algo innecesariamente complejo.
- Estas cosas se dejan mejor para el programa.

Recordemos: Triggers

- Los triggers se ejecutan automáticamente **después de que** se actualiza una tabla.
- Estos son los triggers normales, que se llaman también triggers **AFTER.**
- SQL Server tiene unos triggers que se ejecutan EN VEZ de actualizar una tabla.
- Se llaman triggers **INSTEAD OF**

Los triggers INSTEAD OF no ejecutan la actualización

- En vez de la actualización, ejecutan el trigger.
- Esto los diferencia de los normales o AFTER que el trigger se ejecuta además de la actualización.

```
CREATE TRIGGER TablaSoloLectura ON
  tabla
 INSTEAD OF INSERT, UPDATE, DELETE
 AS PRINT 'No dejo actualizar nada'
```

Vean la diferencia entre AFTER e INSTEAD OF

```
CREATE TRIGGER TriggerInutil ON tabla
 AFTER INSERT, UPDATE, DELETE
 AS PRINT 'No hago nada'
```

- Este primero no hace nada, pero la tabla se actualiza

```
CREATE TRIGGER TablaSoloLectura ON
  tabla
 INSTEAD OF INSERT, UPDATE, DELETE
 AS PRINT 'No dejo actualizar nada'
```

- Este segundo ni hace nada ni deja actualizar.

Por lo tanto

- Cuando tenemos un trigger INSTEAD OF, tenemos que hacer la actualización a mano, dentro del trigger. Esto nos da más trabajo, pero más flexibilidad.

```
CREATE TRIGGER InutilInsertInstead
  ON tabla
 INSTEAD OF INSERT
  AS
  INSERT INTO tabla SELECT * FROM
 inserted
  PRINT 'No hago nada'
```

- Vemos que las tablas `inserted` y `deleted` también existen con este tipo de triggers.

Diferencia entre triggers AFTER e INSTEAD OF

	Triggers AFTER	Triggers INSTEAD OF
Se ejecutan	Después de la actualización	En vez de la actualización
Aplicables a	Tablas	Tablas y vistas
Cantidad por tabla o vista	Ilimitada	Uno para INSERT, otro para UPDATE y otro para DELETE
Tablas a las que es aplicable	Todas	Aquellas que no son destino de una restricción de clave foránea.

Temario del curso

Vistas

- Una vista es una tabla lógica o virtual, que muestra parte de la base de datos.
- Las vistas en realidad no son tablas verdaderas, sino consultas, pero se pueden programar y usar como si fueran tablas.
- Las vistas nos permiten ver la base de datos de la forma que más nos interese, con independencia de la forma en que esté guardada en disco.

Ejemplo de vistas

- Nuestra empresa es una empresa que da servicio a toda el área centroamericana. Tiene sucursales en cada país centroamericano.
- La base de datos de clientes está centralizada en El Salvador y se opera mediante Internet desde las diferentes sucursales.
- Los programas que hay en cada sucursal son diferentes ya que cada país tiene sus leyes y sus reglas de negocio.
- ¿Qué estructura de base de datos podríamos tener? Por ejemplo, respecto a los clientes.

Mail: palasi@palasi.com Web: www.palasi.com

Solución 1

- Tener todos los clientes en una misma tabla y, para los procesos que sólo se aplican a El Salvador, hacer una selección (**WHERE**) de las filas que nos interesan.

Clientes		
199	Simán	El Salvador
099	Prado	El Salvador
567	Lido	El Salvador
927	MiPreMi	Honduras
		...
456	Xapin	Guatemala

Código Nombre Sucursal

El problema es que la sucursal de El Salvador ve todas las tablas y, por mala programación, puede afectar registros de otros países, arruinando la integridad de la BD.

Además es incómodo añadir `WHERE sucursal = 'El Salvador'` a cada operación.

Solución 2

- Hacer una tabla por cada país. Esto hace compleja la BD y es incómodo de mantener.
- Además las operaciones que se realizan para todos los clientes son difíciles de programar.

Clientes Salvador	
199	Simán
099	Prado
567	Lido
	...

Código Nombre

Clientes Honduras	
927	MiPreMi
	...

Código Nombre

Clientes Guatemala	
456	Xapin
	...

Código Nombre

Solución 3

- Tener una única tabla de clientes pero permitir que la sucursal de El Salvador sólo vea los clientes de El Salvador. Para ello, definiremos una vista con sólo esos clientes.

Clientes		
199	Simán	El Salvador
099	Prado	El Salvador
567	Lido	El Salvador
927	MiPreMi	Honduras
		...
456	Xapin	Guatemala

Código Nombre Sucursal

- La vista se comporta como una tabla, pero no es una tabla verdadera: es una consulta.

Clientes Salvador		
199	Simán	El Salvador
099	Prado	El Salvador
567	Lido	El Salvador
	...	

Código Nombre Sucursal

`ClientesSalvador` es una consulta, pero se comporta como una tabla

- Es la consulta `SELECT * FROM Clientes WHERE sucursal ='El Salvador'`
- Por eso los datos no se guardan dos veces en disco: son los mismos datos de la tabla Clientes.

Clientes		
199	Simán	El Salvador
099	Prado	El Salvador
567	Lido	El Salvador
927	MiPreMi	Honduras
		...
456	Xapin	Guatemala

Código Nombre Sucursal

Clientes Salvador		
199	Simán	El Salvador
099	Prado	El Salvador
567	Lido	El Salvador
	...	

Código Nombre Sucursal

`ClientesSalvador` se comporta como una tabla.

- Se pueden hacer consultas sobre ella: `SELECT nombre FROM ClientesSalvador`
- Se le pueden asignar permisos de usuario, de forma que sólo la vea la sucursal salvadoreña. Evitamos el problema de modificar clientes de otros países.

Clientes		
199	Simán	El Salvador
099	Prado	El Salvador
567	Lido	El Salvador
927	MiPreMi	Honduras
		...
456	Xapin	Guatemala

Código Nombre Sucursal

Clientes Salvador		
199	Simán	El Salvador
099	Prado	El Salvador
567	Lido	El Salvador
	...	

Código Nombre Sucursal

¿Cómo se crearía esta vista?

```
CREATE VIEW ClientesSalvador AS
SELECT *
FROM Clientes
WHERE sucursal = 'El Salvador'
WITH CHECK OPTION
```

Clientes		
199	Simán	El Salvador
099	Prado	El Salvador
567	Lido	El Salvador
927	MiPreMi	Honduras
		...
456	Xapin	Guatemala

Código Nombre Sucursal

Clientes Salvador		
199	Simán	El Salvador
099	Prado	El Salvador
567	Lido	El Salvador
	...	

Código Nombre Sucursal

Mail: palasi@palasi.com Web: www.palasi.com

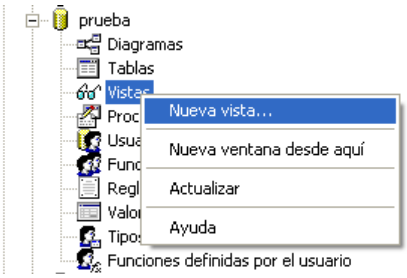
Sintaxis para crear una vista

```
CREATE VIEW nombreVista AS
SELECT instrucciónSelect
WITH CHECK OPTION
```

- Una vez creada la vista, puede usarse en instrucciones como si fuera una tabla verdadera.
- **WITH CHECK OPTION** nos asegura que todas las operaciones de actualización que hagamos sobre la vista sean coherentes con su definición.

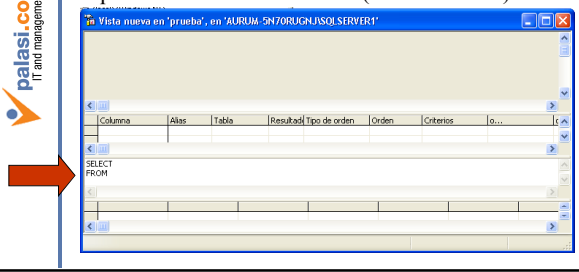
Crear una vista desde el administrador corporativo

- Expandiendo el nodo de la base de datos, aparece un nodo llamado "Vistas". Se hace clic derecho y se selecciona nueva vista.



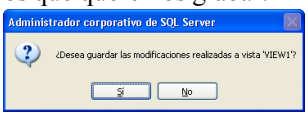
Aparece una ventana

- En el medio podemos escribir la instrucción SELECT que crea la vista. Alternativamente, en la parte superior, se pueden definir de la misma manera que las consultas de Access (no lo veremos).

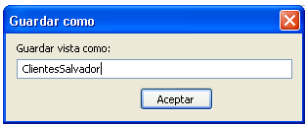


Cerramos la ventana cuando acabamos de definir la vista

- Decimos que queremos grabar.

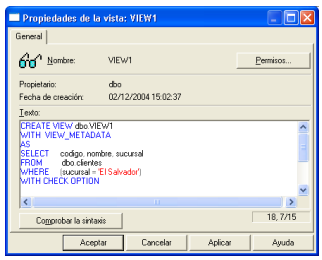


- Escribimos el nombre en el que queremos guardar la vista.



Sin embargo

- En el diseñador gráfico de vistas no puede ponerse el "**WITH CHECK OPTION**". Debe hacerse doble clic sobre la vista y escribirlo a mano.



Crear las vistas anteriores

- A partir de la tabla de clientes que se proporcionará.
- Crear las vistas para El Salvador y Guatemala.
- Comprobar que se pueden tratar como tablas, haciendo sentencias SELECT que las traten como tales.

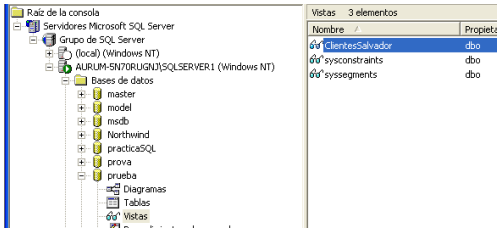
Borrar una vista

```
DROP VIEW nombreVista
```

- Esto desde el analizador de consultas, ¿cómo sería con el administrador corporativo?

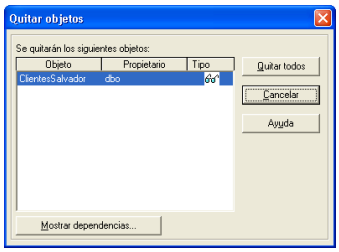
Borrar una vista desde el administrador corporativo

- Se hace clic en el nodo "Vistas". A la derecha aparecerán las vistas existentes. Se selecciona la que desea borrar y se oprime la tecla "Supr".



Borrar una vista desde el administrador corporativo

- En la ventana que aparece, se hace clic en "Quitar todos".

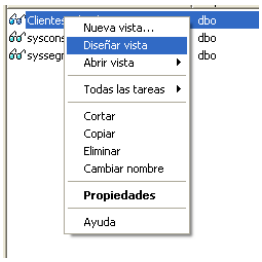


Modificar una vista

```
ALTER VIEW nombreVista AS
SELECT instrucciónSelect
WITH CHECK OPTION
```

- La instrucción **ALTER VIEW** debe tener la misma sintaxis que tenía **CREATE VIEW**. Lo único que cambia es la instrucción **ALTER**.
- La ventaja de usar la instrucción **ALTER VIEW** es que se conservan todos los permisos correspondientes.

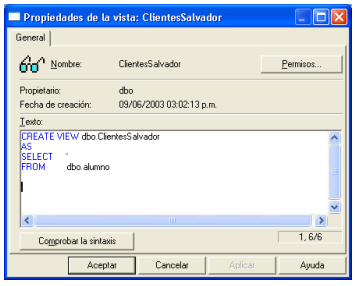
Modificar una vista desde el administrador corporativo



- Con clic derecho sobre la vista, seleccionamos la opción "Diseñar vista"
- Aparecerá la misma ventana que usábamos al crear la vista.

¿Cómo saber la instrucción con la que se definió la vista?

- Desde el administrador corporativo podemos hacer doble clic en el nombre de la vista.



Desde el analizador de consultas

- Podemos ejecutar el procedimiento almacenado **sp_helptext**, pasándole como parámetro el nombre de la vista.

```
EXEC sp_helptext ClientesSalvador
```

Tipos de vistas

- Hay tantos tipos de vistas como tipos de instrucción SELECT pueden haber (es decir, muchas).
- Destacamos los siguientes tipos (pero hay muchos más):
  - **Vistas de selección.**
  - **Vistas de proyección.**
  - **Vistas de combinación.**
  - **Vistas de agregados.**
  - **Vistas con campos calculados.**
- Estos tipos no son excluyentes. Una vista puede pertenecer a más de uno de estos tipos.

Vistas de selección

- Es las que hemos visto hasta ahora. Se selecciona un subconjunto de registros de la tabla. Los otros registros no son visibles.
- Ejemplo:

```
CREATE VIEW Directores AS
 SELECT * FROM Empleados
 WHERE (Cargo = 'Director de
 Departamento') OR (Cargo =
'Director de Sección')
 WITH CHECK OPTION
```

Vistas de proyección

- Son las que seleccionan unos campos de la tabla original. Los otros campos no son visibles.
- Ejemplo:

```
CREATE VIEW NombresEmpleados AS
 SELECT nombre, apellido FROM
 Empleados
 WITH CHECK OPTION
```

Vistas de combinación

- Son las que combinan información de varias tablas.
- Ejemplo:

```
CREATE VIEW EmpleadosDepartamentos
 AS SELECT Empleados.*,
 Departamentos.* FROM
 Empleados JOIN Departamentos ON
 Empleados.Departamento =
 Departamentos.IdDep
 WITH CHECK OPTION
```

Vistas de agregados

- Utilizan funciones de agregados para definir la vista.

```
CREATE VIEW
  PlanillaDepartamentos
  AS SELECT Departamento, SUM(Sueldo)
 AS Planilla
  FROM Empleados
  GROUP BY Departamento
  WITH CHECK OPTION
```

Vistas de campos calculados

- Utilizan campos calculados a partir de otros

```
CREATE VIEW
  FacturasCompletas AS
  SELECT Facturas.*,
    (Exentos+Gravados*1.13) As
     Total FROM Facturas
```

Algunas restricciones para crear una vista

- No puede incluir cláusulas **ORDER BY**, **COMPUTE** o **COMPUTE BY**.
- No puede utilizar **SELECT INTO** en una vista.
- No puede utilizar tablas temporales.
- Se debe especificar los nombres de todos los campos derivados (agregados, campos calculados y campos con funciones) con la cláusula **AS**.
- En una definición de vista no se pueden crear índices, restricciones de clave ni triggers.

¿Qué se puede hacer con las vistas?

- En cuanto a operaciones de consulta (**SELECT**) las vistas pueden tratarse como una tabla más, sin ninguna restricción.
- En cuanto a operaciones de actualización, hay unas cuantas restricciones que deben seguirse.
- Sobre las vistas con **CHECK OPTION** no se pueden definir triggers.
- Sobre las vistas sin **CHECK OPTION** se pueden definir triggers **INSTEAD OF** pero no **AFTER**.

¿Qué pasa cuando se actualiza una vista?

- Lo que se actualiza es la tabla original.
- Los campos que no están en la definición de vista se llenan con NULL.

Supongamos que nuestra vista sólo tiene código y nombre

```
CREATE VIEW ClientesSalvador2 AS
SELECT codigo, nombre
FROM Clientes
WHERE sucursal = 'El Salvador'
```

Clientes		
199	Simán	El Salvador
099	Prado	El Salvador
567	Lido	El Salvador
927	MiPreMi	Honduras
		...
456	Xapin	Guatemala
Código	Nombre	Sucursal

Clientes Salvador	
199	Simán
099	Prado
567	Lido
	...
Código	Nombre

Si insertamos un nuevo registro en la vista, la sucursal será NULL

```
CREATE VIEW ClientesSalvador2 AS
SELECT codigo, nombre
FROM Clientes
WHERE sucursal = 'El Salvador'
```

Clientes		
199	Simán	El Salvador
099	Prado	El Salvador
567	Lido	El Salvador
927	MiPreMi	Honduras
		...
456	Xapin	Guatemala
Código	Nombre	Sucursal

Clientes Salvador	
199	Simán
099	Prado
567	Lido
	...
Código	Nombre

Fíjense que esto no lo pudieran haber hecho si hubieran puesto WITH CHECK OPTION

¿Qué pasaría si yo quisiera que se insertara "El Salvador"?

- Lo podría hacer con un trigger **INSTEAD OF**.
- Pero no lo veremos.

```
CREATE TRIGGER TriggerEjemplo
  ON ClientesSalvador2 INSTEAD
  OF INSERT AS
INSERT INTO Clientes (codigo,
  nombre, sucursal) SELECT
  codigo, nombre, 'El Salvador'
  FROM inserted
```

Si lo prueban, tengan en cuenta

- Lo anterior funciona sólo en el analizador de consultas, ya que parece que el administrador corporativo no funciona bien con los triggers de vistas.
- Cuando insertemos debemos especificar los valores de los campos que no pueden ser nulos pero que son determinados por el motor (autonumérico, calculados, timestamp), aunque éstos valores sean desechados después.
- **En este caso, no se da.** Pero, si el campo código fuera IDENTITY, deberíamos especificar un valor para él cuando lo insertamos (al revés que hacemos normalmente).
- Esto parece una mala programación de Microsoft para los triggers INSTEAD OF pero es así.

No todas las vistas son actualizables

- Para que una vista se pueda actualizar necesita cumplir una serie de requisitos

Requisitos para que una vista sea actualizable

- Si una vista tiene información de varias tablas, la modificación sólo puede afectar una de las tablas originales.
- No se pueden modificar campos calculados, campos agregados ni campos con funciones integradas.
- Sólo puede modificar o agregar registros que se correspondan con la definición de vista (esto es debido al **WITH CHECK OPTION**).
- Los campos **NOT NULL** de la tabla original que NO aparecen en la vista deben tener un valor predeterminado.
- Si la vista es de combinación, no se puede usar para una actualización.

Sin embargo

- Con los triggers **INSTEAD OF** podemos hacer que se actualicen vistas no actualizables, es decir, que no cumplan los criterios anteriores.
- ¡¡¡Pero debemos programar estas actualizaciones a mano!!!

Ejercicio

- Dada la base de datos de Planilla, crear una vista con información de cada empleado junto con el nombre y trabajo de su jefe.
- Nota: Tengan en cuenta que los nombres de los campos no pueden repetirse.

Solución

```
CREATE VIEW EmpleadoJefe AS
SELECT e.*, j.Apellido AS
ApellidoJefe, j.Trabajo AS
TrabajoJefe FROM Empleados e
INNER JOIN Empleados j ON
j.IdEmp=e.Jefe
```

Temario del curso

5. Seguridad. Permisos de usuario.

- 5.1. Autenticación de Windows.
- 5.2. Acceso a las bases de datos.
- 5.3. Permisos sobre los objetos de BD.
- 5.4. Roles de base de datos.
- 5.5. Roles de servidor.
- 5.6. Autenticación de SQL Server.

5. Seguridad. Permisos de usuario.

- 5.1. Autenticación de Windows.
- 5.2. Acceso a las bases de datos.
- 5.3. Permisos sobre los objetos de BD.
- 5.4. Roles de base de datos.
- 5.5. Roles de servidor.
- 5.6. Autenticación de SQL Server.

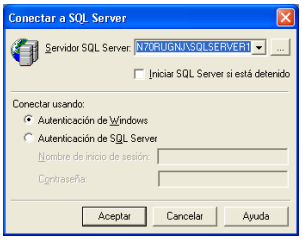
Permisos

- SQL Server es una base de datos que pone énfasis en la seguridad en el acceso a los datos.
- Se puede definir para cada operación y objeto de la base de datos cuáles son los usuarios que están autorizados a acceder a ellos. Se dice que los usuarios tienen permiso para esa operación y ese objeto.
- Los permisos son importantes para evitar accesos no autorizados a los datos.

Usuarios

- Cada persona cuando se conecta a la base de datos, se le da una identidad como usuario. A eso se le llama autenticación.
- Autenticación no es más que dar una clave de usuario y una contraseña, para que SQL Server sepa quienes somos.
- Hay dos tipos de autenticación en SQL Server:
  - Autenticación de SQL Server. Esta la veremos más adelante.
  - **Autenticación de Windows. En ésta, el usuario es identificado por su nombre y contraseña a la hora de entrar en Windows NT, 2000 o XP. Esto elimina la necesidad de autenticarse en Windows y después en SQL Server.**

Cada vez que uno inicia el analizador de consultas, debe autenticarse



- La opción de autenticación de Windows nos permite no tener que repetir el usuario y contraseña que hemos dado al entrar a Windows.

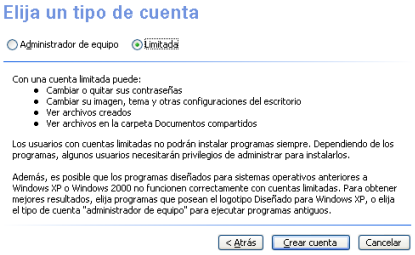
Si queremos definir o borrar usuarios, tenemos que hacerlo desde Windows

- En el Panel de control, se hace doble clic sobre el icono "Cuentas de usuario".

- En Windows XP, aparece una pantalla, los menús guían sin ninguna dificultad para añadir usuarios, modificarlos o borrarlos.

Un punto importante cada vez que definimos una cuenta de usuario

- Es decidir si el tipo de cuenta es de "Administrador de equipo" o "Limitada". Esto tendrá importancia más tarde.



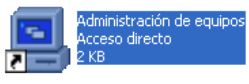
La importancia del tipo de cuenta

- Las cuentas definidas como **"Administradores" tienen acceso a todo** lo de SQL Server (servidor, bases de datos, objetos de bases de dados) mientras no se especifique lo contrario.
- Las cuentas definidas como **"Limitadas" no tienen acceso a nada**, mientras no se especifique lo contrario.

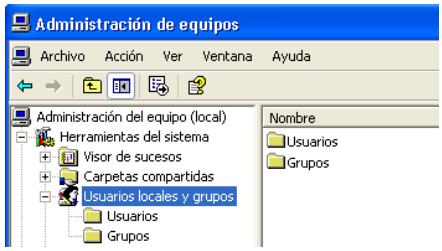
También podemos definir grupos de usuarios

- En Panel de control, hacemos clic en "Herramientas administrativas".

- En la ventana que se abre hacemos clic en "Administración de equipos"



Dentro de administración de equipos

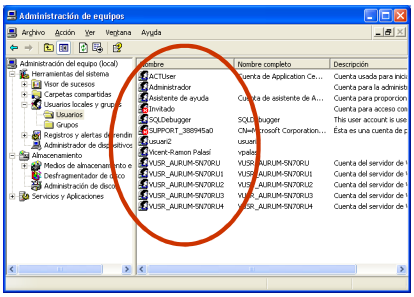


- Hacemos clic en "Usuarios locales y grupos".

Mail: palasi@palasi.com Web: www.palasi.com

Primero abrimos el nodo de Usuarios

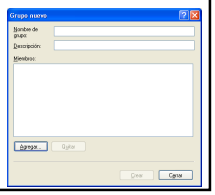
- Así veremos el nombre de los usuarios.



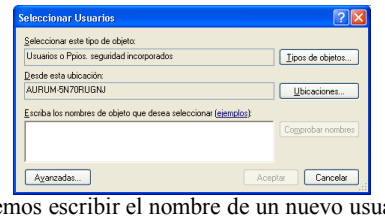
Después creamos un grupo nuevo

- Haciendo clic derecho en el nodo de "Grupos" y seleccionando "Grupo nuevo...".

- Aparece una ventana, donde podemos escribir el nombre y la descripción de grupo. Para agregar miembros, hacemos clic en "Agregar".



Al hacer clic en "Agregar"



- Podemos escribir el nombre de un nuevo usuario que queramos añadir al grupo (Recuerden que los nombres de los usuarios se obtienen mirando en el nodo "Usuarios").
- Hacemos clic en "Aceptar" y así añadimos todos los usuarios.

Con esto hemos definido los diferentes usuarios

- Lo primero que tenemos que hacer es autorizar a estos usuarios para que puedan entrar al SQL Server.
- De forma predeterminada, sólo pueden entrar al SQL Server el usuario **sa** (que ya veremos) y las cuentas de usuario del equipo en que se instaló SQL Server definidas como "Administradores de equipo" (es decir, el grupo de Windows "Administradores").
- Las cuentas limitadas que necesiten acceso a SQL Server, se lo tendremos que dar explícitamente.

Recordemos

- Las cuentas definidas como **"Administradores" tienen acceso a todo** (y el usuario "sa" que veremos más adelante) lo de SQL Server (servidor, bases de datos, objetos de bases de dados) mientras no se especifique lo contrario.
- Las cuentas definidas como **"Limitadas" no tienen acceso a nada**, mientras no se especifique lo contrario.

Autorizando el acceso a SQL Server a un usuario o grupo

- Se usa el procedimiento almacenado **`sp_grantlogin`**, que concede ("grant") el acceso.
- Para conceder el acceso a un usuario:

```
EXEC sp_grantlogin 'idUsuario'
```

- Para conceder el acceso a un grupo de Windows.

```
EXEC sp_grantlogin 'idGrupo'
```


Mail: palasi@palasi.com Web: www.palasi.com

¿Qué es esto de *idUsuario*?

- ***idUsuario*** es una cadena formada por ***nombreEquipo\nombreUsuario***
  - Donde ***nombreEquipo*** es el nombre de la computadora o dominio donde está instalado el servidor (se puede ver en **Panel de control|Sistema|Nombre de equipo**)
  - Donde ***nombreUsuario*** es el nombre de usuario en Windows (puede verse en Administrador de equipos, como se ha visto antes).
- Si ***idUsuario*** tiene algún carácter extraño debe ir entre corchetes.
- ***idGrupo*** es lo mismo, pero en vez del nombre de usuario aparece el nombre del grupo.

Retirando el acceso a SQL Server a un usuario o grupo

- Se usa el procedimiento almacenado **`sp_revokelogin`**, que retira (revoca) un permiso ya concedido.
- Para revocar el acceso a un usuario:

```
EXEC sp_revokelogin 'idUsuario'
```

- Para revocar el acceso a un grupo de Windows.

```
EXEC sp_revokelogin 'idGrupo'
```

¿Qué pasa si quisiéramos conceder permisos de entrada?

- A todos los miembros de un grupo menos a unos pocos.
- Podríamos ir especificando estos permisos para cada uno de los miembros que sí tienen permiso pero esto es muy tedioso.
- Lo que podemos hacer es usar dos procedimientos almacenados.

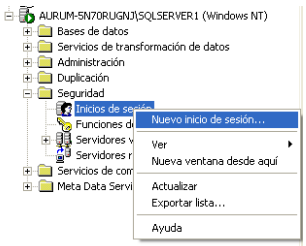
```
EXEC sp_grantlogin 'idGrupo'
EXEC sp_denylogin 'idUsuarioExcluido'
```

- Es decir, **`sp_denylogin`** se utiliza para excluir usuarios de la concesión de derechos.

En general, en los permisos de SQL Server

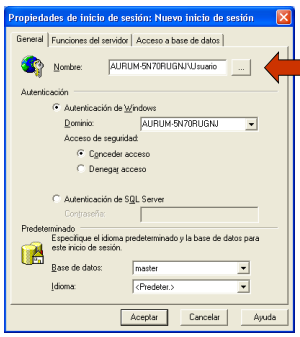
- Se tratan de la misma forma los usuarios que los grupos de usuarios.
- Las mismas operaciones que sirven para un usuario, sirven para grupos de usuarios definidos en Windows.
- Por eso, a partir de ahora, cuando ponga "idUsuario" en una instrucción, se supondrá que también puede ser un grupo.

Desde el Administrador corporativo



- Se expande el nodo de "Seguridad" y se hace clic derecho en "Inicios de sesión". Se selecciona "Nuevo inicio de sesión..."

En la ventana que aparece

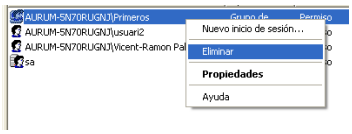


- Ponemos el **idUsuario** o el **idGrupo** en el cuadro texto "Nombre". Hacemos clic en "Aceptar".
- Con esto concedemos un permiso.

Mail: palasi@palasi.com Web: www.palasi.com

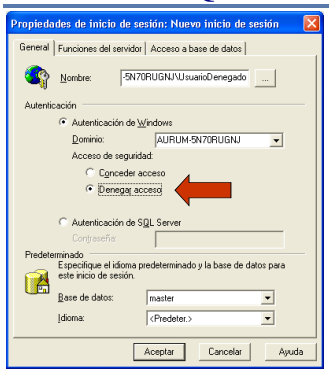
Para revocar el acceso a SQL Server de una cuenta

- Se hace clic derecho en el inicio de sesión correspondiente y se elige la opción "Eliminar".



- Si se hace clic en "Propiedades", se podrá modificar el inicio de sesión, en vez de borrarlo.

Para denegar el acceso a SQL Server de una cuenta



- Se hace lo mismo que para autorizarla, sólo que se elige la opción "Denegar acceso".

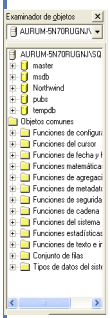
5. Seguridad. Permisos de usuario.

- 5.1. Autenticación de Windows.
- 5.2. Acceso a las bases de datos.
- 5.3. Permisos sobre los objetos de BD.
- 5.4. Roles de base de datos.
- 5.5. Roles de servidor.
- 5.6. Autenticación de SQL Server.

Ejercicio

- Creen un usuario en Windows con tipo de cuenta "Limitada". Lo llamaremos "limitado".
- Concédanle permiso para entrar en SQL Server.
- Ingresen a Windows con ese usuario y entren al analizador de consultas.
- ¿Qué bases de datos ven?

¿Qué es lo que se ve?



- Sólo ve las bases de datos predefinidas (master, msdb, Northwind, pubs, tempdb).
- No ve las bases de datos que han definido otros usuarios.
- ¿Por qué?

El usuario que crea una base de datos se llama el propietario de esa BD

- En principio, sólo él tiene acceso a la base de datos.
- Por eso, desde la nueva cuenta de usuario, no hay acceso a las bases de datos, ya que las definió el otro usuario.
- ¿Cómo hacemos para dar acceso a un usuario a bases definidas por otro?

Desde el analizador de consultas

- Para conceder el permiso de acceso a la base de datos.

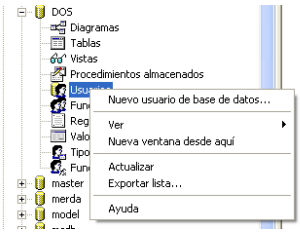
```
USE nombreBD
EXEC sp_grantdbaccess 'idUsuario'
```

- Para revocar el permiso concedido.

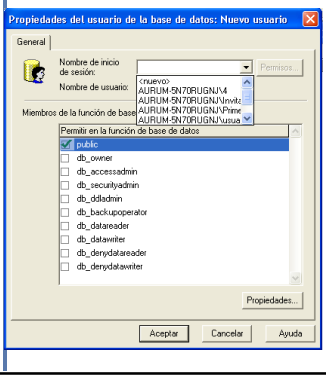
```
USE nombreBD
EXEC sp_revokedbaccess
 'idUsuario'
```

Desde el Administrador corporativo

- Expandimos el nodo de la BD a la que queremos dar acceso y hacemos clic derecho en el nodo "Usuarios".
- Seleccionamos la opción "Nuevo usuario de base de datos..."



En la ventana que aparece



- En la ventana que aparece, desplegamos la lista que aparece en el cuadro "Nombre de inicio de sesión". Allí aparecen todos los usuarios definidos.
- Escogemos un usuario y hacemos clic en "Aceptar". El usuario quedará autorizado para el acceso a la BD.

¿Qué pasa si queremos que una base de datos sea accesible a todo el mundo?

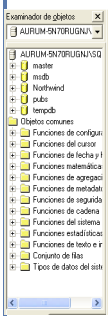
- Para esto está el usuario "guest" (invitado). Si concedemos el permiso a "guest", todo el mundo podrá acceder a nuestra base de datos.
- Ejemplo:

```
EXEC sp_grantdbaccess 'guest'
```

- Cuando nos cansemos de que todo el mundo vea nuestra BD.

```
EXEC sp_revokedbaccess 'guest'
```

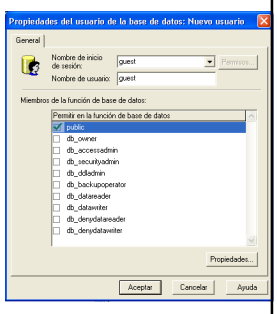
Las bases de datos "predefinidas" de las que hablábamos antes



- Pueden verse desde todos los usuarios.
- Esto no es ningún misterio: simplemente tienen han concedido acceso a "guest".
- Si revocamos el acceso a "guest", dejaran de estar disponibles a todos los usuarios.

Dando acceso a "guest" desde el Administrador corporativo

- Se siguen los mismos pasos que cualquier otro usuario: se selecciona la opción "Nuevo usuario de base de datos...", en la BD que queremos.
- En la ventana que aparece se escribe guest en el primer cuadro de texto y se hace clic en "Aceptar"



Mail: palasi@palasi.com Web: www.palasi.com

5. Seguridad. Permisos de usuario.

- 5.1. Autenticación de Windows.
- 5.2. Acceso a las bases de datos.
- 5.3. Permisos sobre los objetos de BD.
- 5.4. Roles de base de datos.
- 5.5. Roles de servidor.
- 5.6. Autenticación de SQL Server.

Ejercicio

- Creen una base de datos con una tabla de prueba desde su cuenta de usuario normal.
- Denle acceso a esa base de datos al usuario "limitado".
- Ingresen a Windows como usuario "limitado".
- Intenten hacer un **`SELECT * FROM`** ***tablaDePrueba***
- ¿Qué es lo que sucede?

Da un mensaje de error

- ¿No es esto ilógico? ¿No habíamos permitido el acceso a la base de datos al usuario "limitado"?
- En realidad, permitir el acceso a la BD, es sólo permitir que el usuario la vea, pero no da derecho a ver ni modificar ningún objeto dentro de la base de datos.
- Se deben dar permisos para que el usuario vea los objetos de la base de datos.

Recordemos

- Las cuentas definidas como **"Administradores" tienen acceso a todo** (y el usuario "sa" que veremos más adelante) lo de SQL Server (servidor, bases de datos, objetos de bases de dados) mientras no se especifique lo contrario.
- Las cuentas definidas como **"Limitadas" no tienen acceso a nada**, mientras no se especifique lo contrario.

Para un objeto de base de datos, hay los siguientes permisos

SELECT. Ver datos en una tabla, vista o campo
INSERT. Añadir datos a una tabla o una vista.
UPDATE. Modificar datos existentes en una tabla, vista o campo.
DELETE. Eliminar datos de una tabla o de una vista.
EXECUTE. Ejecutar un procedimiento almacenado.
REFERENCE. Hacer referencia a una tabla con una restricción de clave foránea.

- Aunque se pueden establecer permisos sobre campos, esto no es recomendable y no lo veremos. Es más recomendable crear una vista con las campos y establecer permisos sobre ella.

Letra pequeña: ¿Por qué no es recomendable permisos sobre campos?

- Mucho trabajo. Por cada usuario, deberemos determinar sus permisos sobre cada campo (en cambio, con una vista, sólo se define un permiso sobre la vista)
- Más flexible. Si queremos prohibir o permitir campos deberemos hacerlo con cada usuario. Con una vista, sólo debemos añadir o retirar campos sobre la vista.

Nota: Procedimientos almacenados

- Si el usuario tiene el permiso EXEC puede ejecutar un procedimiento almacenado.
- Sin embargo, el procedimiento almacenado accede a otros objetos de la base de datos.
- ¿Qué derechos tiene el procedimiento almacenado sobre los objetos que accede?
- Fácil. Los mismos que el usuario que los ejecuta.

Para conceder derechos

```
GRANT listaPermisos ON objeto TO
  idUsuario
```

Concede los permisos sobre el objeto al usuario (o grupo). **ALL** significa todos los permisos.

Ejemplos:

```
GRANT SELECT, INSERT ON
  empleados TO MIDOMINIO\Pedro
GRANT EXECUTE ON procAlmacenado
  TO MIDOMINIO\Claudia
GRANT ALL ON alumnos TO
  MIDOMINIO\Juan
```

Para revocar y denegar derechos sobre un objeto

```
REVOKE listaPermisos ON objeto TO
  idUsuario
```

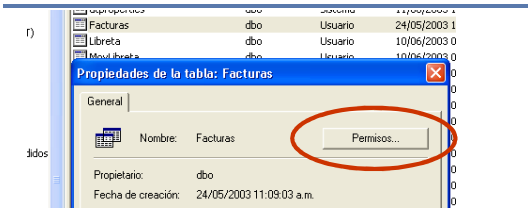
La sintaxis es parecida.

```
DENY listaPermisos ON objeto TO
  idUsuario
```

¿Qué pasa si queremos dar un mismo permiso a todos los usuarios?

- Por ejemplo, queremos que todos los usuarios que puedan acceder a nuestra base de datos consulten una cierta tabla.
- Para ello, en el ***idUsuario*** colocaremos **PUBLIC**.

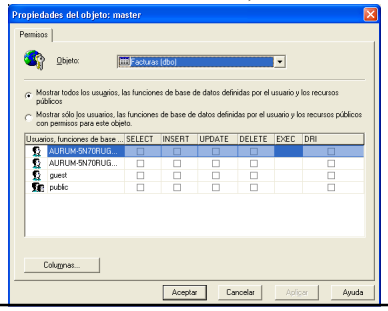
Desde el administrador corporativo



- Se hace doble clic en la tabla o vista sobre la que queremos definir los permisos. Aparece una ventana y hacemos clic en el botón "Permisos...".

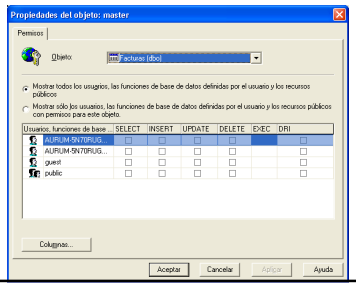
Aparece una ventana con todos los usuarios con acceso a la BD

- Hay columnas con los permisos SELECT, INSERT, UPDATE, DELETE, EXEC y DRI (que quiere decir REFERENCES).



Haciendo clic en las casillas de verificación

- Podemos conceder o denegar esos permisos sobre ese objeto a los diferentes usuarios y grupos.
- Recordemos que "public" significa todos los usuarios.



Igual que permisos de objeto también hay permisos de instrucción

- Igual que los permisos de objeto se usan para permitir acceder a un objeto de una BD, los permisos de instrucción se utilizan para poder ejecutar las siguientes instrucciones.

```
CREATE DATABASE
CREATE TABLE
CREATE PROCEDURE
CREATE DEFAULT
CREATE RULE
CREATE VIEW
BACKUP DATABASE
BACKUP LOG
```

Para conceder, revocar y denegar derechos sobre instrucciones

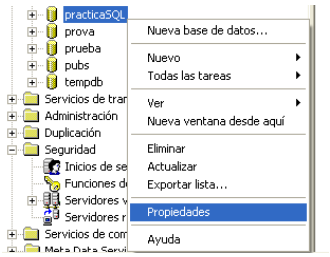
```
GRANT listaPermisos TO idUsuario
REVOKE listaPermisos TO idUsuario
DENY listaPermisos TO idUsuario
```

- Debemos estar en la misma base de datos en que queremos fijar los permisos. **ALL** significa los permisos para todas las instrucciones anteriores. **PUBLIC** significa permisos a todos los usuarios.
- Ejemplo:

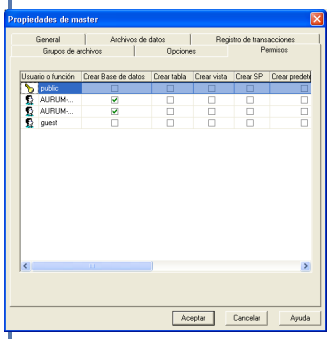
```
GRANT CREATE TABLE, CREATE VIEW TO
  MIDOMINIO\Pedro
```

Desde el administrador corporativo

- Hacemos clic derecho sobre la BD y seleccionamos "Propiedades".



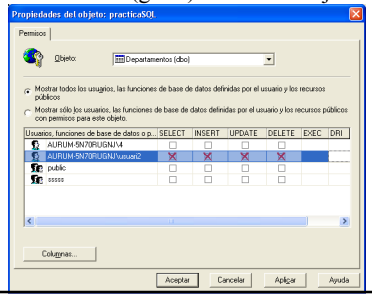
En la ventana que aparece



- Seleccionamos la pestaña "Permisos", donde podremos fijar los permisos de instrucción para cada uno de los usuarios de nuestra BD.

Código de color de las casillas

- Si la casilla está en blanco, el permiso no se ha concedido. Si está en verde el persmiso se ha concedido (grant). Si está en rojo está denegado



5. Seguridad. Permisos de usuario.

- 5.1. Autenticación de Windows.
- 5.2. Acceso a las bases de datos.
- 5.3. Permisos sobre los objetos de BD.
- 5.4. Roles de base de datos.
- 5.5. Roles de servidor.
- 5.6. Autenticación de SQL Server.

Manejar la seguridad en SQL Server (por lo que sabemos hasta ahora)

- 1. Definimos los usuarios y grupos desde Windows.
- 2. Definimos el acceso a SQL Server con sp_grantlogin y similares.
- 3. Definimos el acceso a las bases de datos a los usuarios no propietarios con sp_grantdbaccess y similares.
- 4. Definimos los permisos sobre los objetos e instrucciones con GRANT y similares.

- **¿No es demasiado trabajo?**

Es demasiado trabajo

- Sobre todo, el punto 4, ya que hay muchos objetos en cada una de las bases de datos.
- Una de las soluciones son los llamados "roles (o funciones) de bases de datos".
- Los roles son grupos de usuarios y grupos definidos en SQL Server.
- A un rol se le definen permisos y todos los usuarios del rol tienen esos permisos. Esto es mucho menos pesado que asignar los permisos por cada usuario.

¿Qué ventajas tienen los roles respecto a los grupos de Windows?

- Un rol no tiene que ver con la configuración de Windows. Además puede usar usuarios que no se autentican con Windows (lo veremos después).
- Los grupos son específicos de cada base de datos.
- Un usuario puede pertenecer a varios roles.
- Se pueden incluir roles dentro de roles, etc.

Usando un rol

- Se siguen los siguientes pasos.
- 1. Se crea un nuevo rol en la BD.

```
EXEC sp_addrole 'nombreRol'
```

- 2. Se añaden usuarios y grupos a ese rol.

```
EXEC sp_addrolemember
'nombreRol', 'idUsuario'
```

- 3. Se añaden permisos a ese rol con **GRANT, REVOKE, DENY.**

Ejemplo

- Supongamos que hay una BD "planilla" y 10 usuarios llamados "Usuario1"... "Usuario10" a los que queremos dar derechos a crear tablas, modificarlas y a actualizar la tabla "empleados", pero no a leerla (son administradores del sistema y no queremos que lean los datos).

Mala solución

```
USE planilla
GRANT CREATE TABLE TO Usuario1
GRANT ALTER TABLE TO Usuario1
GRANT INSERT, UPDATE, DELETE ON
  empleado TO Usuario1
```

(repetir esto diez veces por los 10 usuarios)

¿Qué pasaría si quisiéramos revocar el permiso de crear tabla a estos usuarios? Tendríamos que escribir 10 instrucciones.

Buena solución

```
USE planilla
EXEC sp_addrole 'CreaTablas'
EXEC sp_addrolemember 'CreaTablas',
'Usuario1'
```

(así 10 veces con todos los usuarios)

```
GRANT CREATE TABLE TO CreaTablas
GRANT ALTER TABLE TO CreaTablas
GRANT INSERT, UPDATE, DELETE ON
  empleado TO CreaTablas
```

¿Qué pasaría si quisiéramos revocar el permiso de crear tabla a estos usuarios? Sólo necesitaríamos:

```
REVOKE CREATE TABLE TO CreaTablas
```

Si queremos quitar un usuario de un rol

```
EXEC sp_droprolemember
  'nombreRol', 'idUsuario'
```

- Para borrar un rol

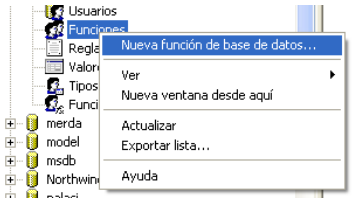
```
EXEC sp_droprole 'nombreRol'
```

Para saber qué usuarios tiene un rol

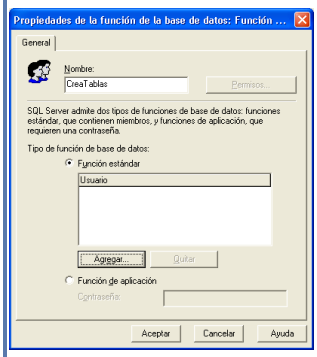
```
EXEC sp_helprolemember 'nombreRol'
```

Desde el administrador corporativo

- Los roles en la versión en español de SQL Server se llaman “funciones”.
- Se hace clic derecho en el nodo “Funciones” debajo de la BD. Se selecciona la opción “Nueva función de bases de datos”.

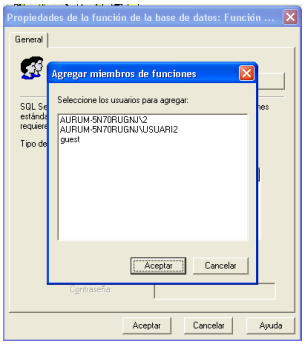


En la ventana que aparece



- Se pone nombre al rol y se hace clic en “Agregar” para añadirle usuarios y grupos.
- Quitar se usa para quitar usuarios de un rol preexistente.

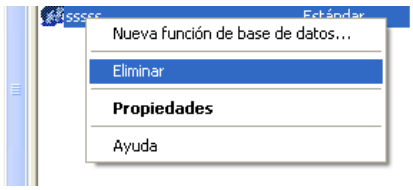
Al hacer clic en “Agregar”



- Aparece una lista de usuarios con acceso a la BD. De aquí seleccionamos los que formen parte del rol y hacemos clic en “Aceptar”.

Si queremos eliminar un rol

- Simplemente lo borramos desde el Administrador corporativo



Pero aún hay más

- Hay unos roles que ya están predefinidos para todas las bases de datos.
- Esto quiere decir que, para esos roles, ya se le han asignado permisos. Son como "paquetes predefinidos" de permisos listos para ser aplicados.
- Lo único que debemos hacer es añadir el usuario al rol y adquirirá todos los permisos de ese rol.

Roles predefinidos de bases de datos (1)

- **db_owner**. Los usuarios con este rol, puede hacer casi todo sobre la BD, excepto definir qué usuarios son db_owner (esto sólo lo puede hacer el propietario de la BD).
- **db_accessadmin**. Los usuarios con este rol administran qué usuarios pueden acceder a una BD.
- **db_securityadmin**. Los usuarios con este rol administran los permisos dentro de una BD.
- **db_dlladmin.** Los usuarios con este rol pueden ejecutar instrucciones DDL y otras relacionadas.

Roles predefinidos de bases de datos (2)

- **db_backupoperator**. Los usuarios con este rol pueden hacer backups.
- **db_datareader**. Los usuarios con este rol sólo pueden leer los datos de la BD.
- **db_datawriter.** Los usuarios con este rol sólo pueden escribir los datos de la BD.
- **db_denydatareader.** Los usuarios con este rol tienen prohibido leer los datos de la BD.

Para saber exactamente qué permisos tiene cada rol predefinido de BD

```
EXEC sp_dbfixedrolepermission
  'nombreRol'
```

El rol public

- Ya lo hemos visto. Pero ahora podemos comprenderlo mejor.
- Es un rol predefinido en cada base de datos, donde están todos los usuarios que pueden acceder a la base de datos.
- Añadir un permiso a public es añadirlo a todo el mundo (en esa base de datos).

Nota

- Esto son los roles predefinidos. Para decir que un usuario sigue un rol predefinido, simplemente se hace

```
EXEC sp_addrolemember
'nombreRol''idUsuario'
```

La ventaja de los roles predefinidos

- Nos ahorran mucho trabajo.
- Por ejemplo, si tenemos un usuario que sólo puede hacer leer la BD, es mucho más fácil asignarle el rol de **db_datareader** que ir objeto por objeto e instrucción por instrucción asignando permisos.

```
EXEC sp_addrolemember
  'db_datareader',
  'MIDOMINIO\Juan'
```

Los roles definidos por el usuario dan un poco más de trabajo

- Porque hay que definir los permisos para cada uno de ellos.
- De todas maneras, siempre es mejor definir permisos para un rol que para cada uno de los usuarios.

Ejemplo

- Supongamos que hay una BD “planilla” y 10 usuarios llamados “Usuario1”... “Usuario10” a los que queremos dar derechos a actualizar todas las tablas, pero no a leerlas (son administradores del sistema y no queremos que lean los datos).

Solución

```
EXEC sp_addrolemember
  'db_datawriter', 'Usuario1'

...

EXEC sp_addrolemember
  'db_datawriter', 'Usuario10'
```

Ejercicio

- Definir una base de datos de prueba con varias tablas, una de las cuales se llame “crítica”. Hay diez usuarios, llamados Lector1... Lector5 y Escritor1..Escritor5. Los lectores pueden leer todas las tablas excepto “crítica”. Los escritores pueden escribir todas las tablas excepto “crítica”. Definir los permisos para esa base de datos. Pista: usen roles predefinidos y combínenlos con instrucciones de permiso sobre objetos.

Mail: palasi@palasi.com Web: www.palasi.com

5. Seguridad. Permisos de usuario.

- 5.1. Autenticación de Windows.
- 5.2. Acceso a las bases de datos.
- 5.3. Permisos sobre los objetos de BD.
- 5.4. Roles de base de datos.
- 5.5. Roles de servidor.
- 5.6. Autenticación de SQL Server.

Hasta ahora hemos hablado de roles de bases de datos

- Lo que hacen es agrupar usuarios que tienen los mismos permisos para hacer operaciones sobre una base de datos.
- Pero, a veces nos interesa definir roles generales sobre todas las bases de datos e incluso sobre operaciones que se hacen en el servidor y no están asignadas a ninguna base de datos.
- Esto nos lleva a definir los llamados "roles de servidor".

Roles predefinidos de servidor (1)

- **sysadmin.** Los usuarios con este rol pueden hacer TODO en SQL Server. No tienen restricción en ninguna BD ni fuera de ella.
- **serveradmin.** Los usuarios con este rol son los administradores de servidor que no van a estar administrando bases de datos u otros objetos.
- **setupadmin.** Los usuarios con este rol son administradores que están configurando servidores remotos.

Roles predefinidos de servidor (2)

- **securityadmin**. Los usuarios con este rol pueden hacer cualquier operación relacionada con la seguridad en SQL Server.
- **processadmin**. Los usuarios con este rol pueden manejar los procesos que se ejecutan en el servidor.
- **dbcreator**. Los usuarios con este rol pueden realizar operaciones relacionadas con la creación y modificación de la estructura de BD.

Para saber exactamente qué permisos tiene cada rol de servidor

```
EXEC sp_srvrolepermission
 'nombreRol'
```

Operaciones para roles de servidor

- Para añadir un usuario a un rol de servidor.

```
EXEC sp_addsrvrolemember
  'idUsuario', 'rol'
```

- Para quitar un usuario de un rol de servidor.

```
EXEC sp_dropsrvrolemember
  'idUsuario', 'rol'
```

- Para saber qué usuarios tiene un rol

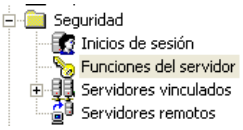
```
EXEC sp_helpsrvrolemember 'nombreRol'
```

- Estas operaciones sólo las puede realizar un miembro del rol sysadmin.

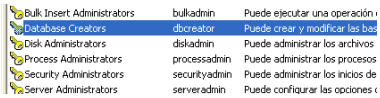
Mail: palasi@palasi.com Web: www.palasi.com

Desde el administrador corporativo

- Se expande el modo Seguridad y se hace clic en "Funciones del servidor".

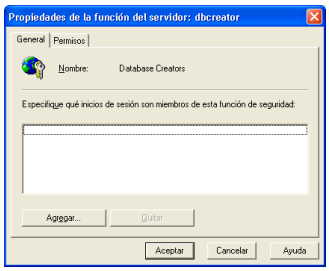


- Aparece todos los roles del servidor, hacemos doble clic en uno:



En la ventana que aparece

- Podemos añadir o quitar usuarios de ese rol. Podemos ver qué permisos tiene asociados.



El rol sysadmin

- Permite hacer cualquier cosa con SQL Server.
- Por defecto, tienen este rol:
  - Las cuentas de usuario definidas en Windows como "Administrador de equipo". Estas pueden eliminarse del rol.
  - La cuenta llamada sa. Esta no puede eliminarse del rol.

Manejar la seguridad en SQL Server

- 1. Definimos los usuarios y grupos desde Windows.
- 2. Definimos el acceso a SQL Server con sp_grantlogin y similares.
- 3. Definimos el acceso a las bases de datos a los usuarios no propietarios con sp_grantdbaccess y similares.
- 4. Definimos los roles definidos por el usuario.
- 5. Para cada usuario, grupo o rol, tenemos dos alternativas:
  - a. Definimos los permisos sobre los objetos e instrucciones con GRANT y similares.
  - b. Definimos estos permisos usando roles.

Algunas recomendaciones

- Se asignan los permisos que necesitan todos los usuarios al rol public.
- Se asignan los permisos que necesitan todos los miembros de un grupo de personas a un rol (o a un grupo de Windows).
- Sólo se asignan permisos individuales a los usuarios si los permisos que necesitan no se pueden asignar a un rol o grupo de Windows.

5. Seguridad. Permisos de usuario.

- 5.1. Autenticación de Windows.
- 5.2. Acceso a las bases de datos.
- 5.3. Permisos sobre los objetos de BD.
- 5.4. Roles de base de datos.
- 5.5. Roles de servidor.
- 5.6. Autenticación de SQL Server.

Mail: palasi@palasi.com Web: www.palasi.com

Modo de autenticación de SQL Server

- Sirve para dar acceso a computadoras con sistema operativo diferente de Windows NT, 2000 o XP.
- En este modo, definimos los usuarios de forma independiente del Windows.
- SQL Server define los usuarios, en vez de importarlos desde Windows.

Para definir un usuario con modo de autenticación SQL Server

- Hay que pertenecer al rol sysadmin.
- Desde el analizador de consultas, se hace

```
EXEC sp_addlogin 'login', 'contraseña'
```

- Para quitar a este usuario, se hace

```
EXEC sp_droplogin 'login'
```

- Para cambiar la contraseña, se hace

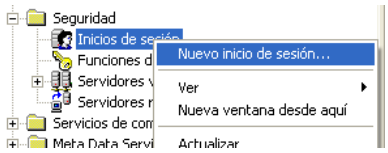
```
EXEC sp_password 'vieja', 'nueva', 'login'
```

En modo de autenticación de SQL Server

- No se utilizan **sp_grantlogin, sp_revoke_login** y **sp_denylogin**.
- Pues se utilizan en cambio **sp_addlogin** y **sp_droplogin.**

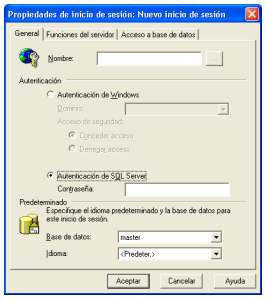
Desde el Administrador corporativo

- En el nodo "Inicio de sesion", que está debajo de "Seguridad", se hace clic derecho en "Nuevo inicio de sesión".



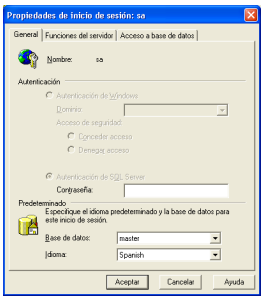
En la ventana que aparece

- Se ha de elegir "Autenticación de SQL Server", escribir el nombre (login) y la contraseña y hacer clic en Aceptar.



Para cambiar la contraseña

- Simplemente se entra a **Seguridad|Inicios de sesión** y doble clic en el inicio de sesión que queramos modificar. En el cuadro de diálogo que aparece, se puede cambiar la contraseña



Entrando con autenticación de SQL Server

- Simplemente se elige esa opción en la ventana de entrada del administrador de consultas y se provee un nombre y contraseña.

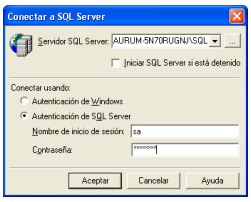
- En el administrador corporativo NO se puede entrar con autenticación de SQL Server.

Lo que hemos dicho hasta ahora

- Son los únicas diferencias entre el modo de autenticación SQL Server y Windows.
- Para todo lo otro, se comportan en forma igual.
- Los usuarios de autenticación SQL Server son unos usuarios normales, a los que se les puede dar acceso a BD, aplicar permisos, asignar roles, etc.

La cuenta sa

- Es de autenticación de SQL Server. Es siempre miembro del grupo sysadmin y, por tanto, puede hacerlo todo.



- Es importante ponerle una contraseña segura, ya que por defecto no tiene contraseña.

Manejar la seguridad en SQL Server

- 1. Definimos los usuarios y grupos con autenticación de Windows y les asignamos acceso con sp_grantlogin y similares.
- 2. Definimos los usuarios con autenticación de SQL Server.
- 3. Definimos el acceso a las bases de datos a los usuarios no propietarios con sp_grantdbaccess y similares.
- 4. Definimos los roles definidos por el usuario.
- 5. Para cada usuario, grupo o rol, tenemos dos alternativas:
  - a. Definimos los permisos sobre los objetos e instrucciones con GRANT y similares.
  - b. Definimos estos permisos usando roles.

Temario del curso

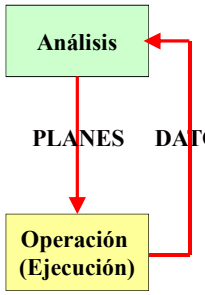
6. OLAP (Data warehousing).

Mail: palasi@palasi.com Web: www.palasi.com

6. OLAP (Data warehousing).

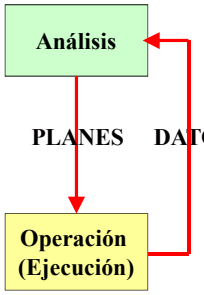
- 6.1. ¿Qué es OLAP?
- 6.2. Base de datos de prueba.
- 6.3. Microsoft Analysis Services.
- 6.4. Medidas y dimensiones.
- 6.5. Creación de un cubo.
- 6.6. Uso de un cubo.
- 6.7. Dimensiones, niveles y vistas.
- 6.8. Otros aspectos.
- 6.9. Integración con otros programas.

El funcionamiento de una empresa (y de cualquier actividad humana)



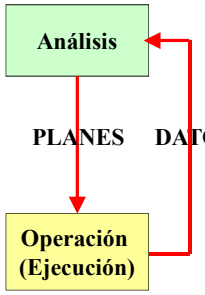
- Al análisis se le llama, a veces, planificación (si se produce antes de la operación) o evaluación (si se produce después de la operación).
- A la operación, se le llama, a veces, ejecución
- Ejemplo: una empresa, un curso de Universidad, el gobierno, los planes de vacaciones, etc.

Ejemplo: una empresa



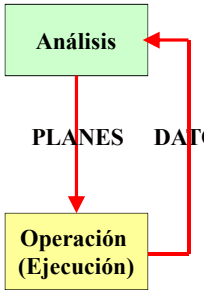
- El análisis determina qué debe producir la empresa, qué mercados se persiguen, cuáles son las decisiones estratégicas que hay que tomar
- La operación es cómo vendemos una cierta orden, cómo distribuimos un cierto pedido, etc, basándonos en los planes resultado del análisis.

Toda actividad humana necesita la toma de decisiones



- El análisis decidirá qué productos se producen, qué clientes se venden, cuál es la política de crédito, etc.
- En la operación, se decide si se debe dar crédito a un cliente, qué factura se sirve primero.

Toda toma de decisiones necesita información



- Tanto el análisis como la operación deben procesar información.
- La operación debe procesar, por ejemplo, los datos de una factura.
- El análisis debe procesar, por ejemplo, cuáles son los productos más vendidos el último año.

Procesamientos de la información muy diferentes (1)

- **Operación (Ejecución):**
  Se procesa información actual y se hace **de forma frecuente**. Proceso en tiempo real.
  - Procesar una factura que se acaba de vender.
- **Análisis:**
  Se procesa información presente, pasada y futura y se hace **de forma ocasional**. Proceso puede ser diferido (no se muere nadie si se retrasa)
  - Obtener la información de los clientes que más compraron el año pasado.

Mail: palasi@palasi.com Web: www.palasi.com

Procesamientos de la información muy diferentes (2)

- **Operación (Ejecución):**
  Se procesan datos específicos y primarios.
  Estos datos pueden ser de lectura y escritura.
  - El número de una factura que acabamos de vender
- **Análisis:**
  Se procesan datos generales y derivados de (muchos) otros datos. Suelen ser de lectura.
  Estos datos pueden ser pasados, presentes y futuros.
  - Qué productos son los más vendidos en el año pasado, lo que se deriva de muchas facturas.

Procesamientos de la información muy diferentes (3)

- **Operación (Ejecución):**
  La información se procesa con procedimientos preestablecidos, basándose en los planes obtenidos en el análisis.
  No siempre se necesita que un humano analice la información.
  - Para vender a un cliente, vemos si lo que nos debe no supera el límite de crédito. Esto puede ser programado.
- **Análisis:**
  La información se procesa de forma más flexible.
  Por ello, se necesita que un humano (gerente) analice la información.
  - Para decidir a qué clientes vender y qué limite de crédito tener, se consultan datos de venta, geográficos, de crédito. Con ellos, la Gerencia toma una decisión.

Resumen: procesamiento de datos en operación y análisis

Operación	Análisis
Tiempo real y frecuente	Diferido y ocasional
Más preestablecido	Más flexible.
No siempre necesita análisis de un humano.	Necesita análisis de un humano (gerente).
Datos presentes.	Datos pasados, presentes y futuros.
Lectura y escritura	Predominantemente lectura.
Datos específicos y primarios.	Generales y derivados de otros datos.
Necesita acceso rápido y concurrente a muchos datos primarios.	Necesita acceso razonablemente rápido a resúmenes de muchos datos primarios.

Son necesidades de acceso a datos muy diferentes

Operación	Análisis
Tiempo real y frecuente	Diferido y ocasional
Más preestab	
No siempre n un humano.	is de un humano
Datos present	presentes y
Lectura y esc	mente lectura.
Datos específicos y primarios.	Generales y derivados de otros datos.
Necesita acceso rápido y concurrente a datos primarios.	Necesita acceso razonablemente rápido a resúmenes de muchos datos primarios

Es lógico que haya diferentes soluciones adaptadas al acceso a datos en cada una de las situaciones

Hay dos tipos de soluciones para el acceso a datos

- En operación.
  - Una solución dominante. OLTP (On Line Transaction Processing) Programas tradicionales que acceden a BDs relacionales. Estándar desde hace 30 años.
- En análisis.
  - Las soluciones han recibido los nombres de data warehousing, business intelligence o soporte a la toma de decisiones.
  - Es un mercado más fragmentado y no tan estable.
  - En los 70, Decision Support Systems (DSS). En los 80, Enterprise Information Systems (EIS).
  - Actualmente: Hojas de cálculo, paquetes estadísticos, **OLAP** (On Line Analysis Processing) y data mining.

Estrategias actuales para el análisis de datos

- **Estrategias deductivas.** El usuario propone una hipótesis y el sistema realiza los cálculos para ver si es válida o no.
  - **OLAP.** La hipótesis se valida según una técnica de procesamiento de datos llamada análisis multidimensional.
  - **Paquetes estadísticos.** La hipótesis se valida según la ciencia estadística tradicional.
- **Estrategias inductivas. Minería de datos.** El programa, a partir de los datos, descubre relaciones inesperadas y desconocidas de forma automática.

Mail: palasi@palasi.com Web: www.palasi.com

Estrategias actuales para el análisis de datos

- **Estrategias deductivas.** Por ejemplo, el usuario supone la hipótesis que la venta de un auto depende del nivel de ingresos del comprador.
  - **OLAP.** Análisis multidimensional. Se obtiene la venta de autos según los rangos del nivel de ingreso de los clientes.
  - **Paquetes estadísticos.** Estadística tradicional. Se obtiene una regresión entre nivel de ingreso y compra de autos.
- **Estrategias inductivas. Minería de datos.** El programa descubre relaciones inesperadas (como, por ejemplo, que los bautistas compran más un tipo de pasta de dientes).

Aquí vamos a ver OLAP

- OLAP es una técnica que analiza hipótesis propuestas por el usuario a partir de los datos, usando un modelo conocido como análisis multidimensional.
- Sus características se resumen en las siglas FASMI (**F**ast **A**nalysis of **S**hared **M**ultidimensional **I**nformation) o Análisis Rápido de Información Multidimensional Compartida (ARIMC)

OLAP es FASMI

- **Fast.** (Razonablemente) rápido. El sistema debería dar una respuesta en menos de cinco segundos.
- **Analysis.** El sistema debería permitir que el usuario analice todas las hipótesis sin necesidad de programar.
- **of Shared.** Los datos deben ser compartidos de forma concurrente. Aunque el uso más frecuente es de sólo lectura, si hay escritura debe garantizarse la integridad.
- **Multidimensional.** Lo más importante. El usuario debe tener una vista conceptual de los datos completamente multidimensional.
- **Information.** Habilidad para gestionar grandes cantidades de datos de forma eficiente.

Pregunta. ¿Por qué hemos dicho (razonablemente) rápido?

- **Fast.** (Razonablemente) rápido. El sistema debería dar una respuesta en menos de cinco segundos.

Respuesta: Cinco segundos es rápido para análisis

Pues el análisis es ocasional, cinco segundos es razonablemente rápido.

Pero comparado con la operación es lento. Imaginen que una base de datos relacional tardara cinco segundos en procesar una transacción.

Operación	Análisis
Tiempo real y frecuente	**Diferido y ocasional**
Necesita acceso rápido y concurrente a muchos datos primarios.	**Necesita acceso razonablemente rápido a resúmenes de muchos datos primarios.**

Segunda pregunta.

- OLAP es una técnica. Pero ¿qué productos de software me permiten implementarla?
  - Hojas de cálculo.
  - Bases de datos relacionales con SQL.
  - Productos especializados OLAP.

Respuesta.

- Las tres.

Tercera pregunta.

- Si las hojas de cálculo y las BDs relacionales pueden implementar OLAP, ¿para qué necesitamos productos especializados OLAP?

Respuesta

- Que se pueda no quiere decir que sea fácil.
- Para modelos OLAP que no sean muy sencillos, implementarlo con hojas de cálculo o bases de datos puede ser un auténtico quebradero de cabeza, incluso para un programador.
- OLAP suele ser operado por un gerente (recordemos que lo decíamos cuando hablábamos de análisis), por lo que no le podemos pedir unas habilidades tan avanzadas.

6. OLAP (Data warehousing).

Metodología de aprendizaje

- Como en toda disciplina, hay que aprender la teoría y después cómo aplicarla a casos prácticos.
- Para aprender la teoría de OLAP, acompañaremos cada concepto teórico que se explique con un pequeño ejercicio para reforzar el concepto y ver cómo se implementa en la computadora.
- Así, el aprendizaje de la teoría será más divertido y efectivo.

Acompañaremos los conceptos teóricos con pequeños ejercicios de refuerzo

- Para ello, necesitamos un conjunto de datos de prueba, para hacer ejercicio.
- No olvidemos que OLAP es una técnica de análisis de datos. Sin datos, no podemos practicar OLAP.
- Hemos preparado una base de datos para hacer estos primeros ejercicios en OLAP. Es la base de datos ChepeInc

Chepe Inc

- Chepe el Pechito, de Intipucá, se fue a los Estados Unidos de mojado y, a base de trabajo duro, se ha convertido en un magnate al quien se le conoce como “Mr. Chepe, the rich guy”.
- Montó una tienda de alimentación para latinos (llamada Tienda Chepita), que fue creciendo hasta convertirse en una multinacional llamada Chepe, Inc.
- El crecimiento de Chepe, Inc. depende de la habilidad para una planificación en los inseguros tiempos que corren. Para ello, Mr. Chepe ha decidido utilizar OLAP.

Chepe, Inc.

- Algunos datos de las ventas de Chepe, Inc. durante el año 1998, se encuentran en la base de datos de Access “ChepeInc.mdb”.
- Esta base de datos se encuentra en **C:\Archivos de programa\Microsoft Analysis Services\Samples**
- Ábranla y examínenla.

La única tabla tiene una serie de ventas

- Cada venta tiene el día, mes, año en que se realizó, el producto que se vendió, la marca, el cliente, la promoción, el costo de ese producto para Chepe Inc y el precio por le que lo vendió (naturalmente más alto).

dia	mes	ano	producto	marca	cliente	promocion	tienda	precio	costo
8	January	1998	Tell Tale Ca	Tell Tale	Sweet	No Promotion	Store 3	19,10 €	6,49 €
9	January	1998	Blue Label	Blue Label	Sweet	No Promotion	Store 3	7,66 €	2,60 €
9	January	1998	Blue Label	Blue Label	Tusting	No Promotion	Store 3	15,32 €	7,51 €
10	January	1998	Golden Ora	Golden	Beltran	No Promotion	Store 7	15,52 €	6,05 €
10	January	1998	Club Low F	Club	Strambi	No Promotion	Store 3	8,79 €	2,99 €
10	January	1998	Green Ribb	Green Ribb	Manning	No Promotion	Store 8	2,62 €	1,21 €

Pregunta: ¿Esta tabla es correcta desde el punto de vista relacional?

- ¿Sí? ¿No? ¿Por qué?

No es correcta

- No está en tercera forma normal.
- Si recordamos, la 3FN dice “cualquier campo sólo depende de la clave”.
- Sin embargo, un campo como “marca” depende de producto.
- Por ello, esta tabla no es correcta.

¿Puede haber tablas no normalizadas?

- No se descarta que, en situaciones reales, a veces podamos recurrir a la desnormalización por cuestiones de eficiencia en algunos casos.
- Sin embargo, esto debe ser la excepción y no la norma. Sólo debe hacerse en casos muy especiales y críticos y siempre como último recurso.
- La desnormalización hace más difícil la programación y la actualización de datos. Es un lío. Mejor huir de ella.

Mail: palasi@palasi.com Web: www.palasi.com

En un caso real

- Mejor hubiéramos normalizado esa tabla y hubiéramos hecho una tabla a parte llamada "producto", ligada con ésta por una clave foránea. Otra tabla, llamada "marca" se ligaría con una clave foránea a "producto".
- Hay otras mejoras que podemos hacer: cliente, promoción y tienda merecen tabla a parte, pues seguramente querremos indicar características de estos conceptos.

Sin embargo…

- Se ha elegido esta tabla incorrecta por motivos didácticos.
- Los primeros ejercicios con OLAP no queremos complicarnos con el tema de claves foráneas, pues queremos estar pensando en los conceptos OLAP.
- Por ahora, no se preocupen que la tabla no sea normalizada. Ya lo arreglaremos más adelante. Además, Mr. Chepe aprendió Access en una academia y nunca entendió bien el modelo relacional.

6. OLAP (Data warehousing).

Aprendiendo OLAP

- El modelo relacional es uno solo, independientemente de la base de datos que se utilice.
- Si uno aprende el modelo relacional, puede aprender fácilmente cualquier base de datos relacional: Oracle, SQL Server, MySQL, PostgreSQL, Access y un largo etcétera.
- La técnica OLAP es una sola, independientemente del producto OLAP que usemos.
- Si aprendemos la técnica OLAP, podemos aprender fácilmente cualquier producto OLAP

Sin embargo …

- Igual que necesitamos una base de datos para hacer prácticas con el modelo relacional (en nuestro caso, habíamos seleccionado SQL Server)…
- … necesitamos un producto OLAP específico para practicar la técnica OLAP.
- En nuestro caso, usaremos Microsoft Analysis Services.

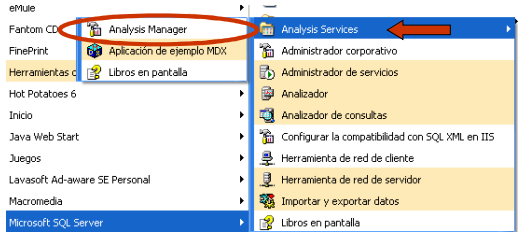
Microsoft Analysis Services

- Producto de Microsoft para el análisis de datos mediante OLAP y minería de datos.
- Puede trabajar con cualquier base de datos que pueda accederse con ODBC o OLE DB. Es decir, con todas las bases de datos comunes y con la mayoría de las que son no comunes.
- Es un producto independiente de SQL Server, pero se distribuye conjuntamente con él, con lo que puede llevar a confusión

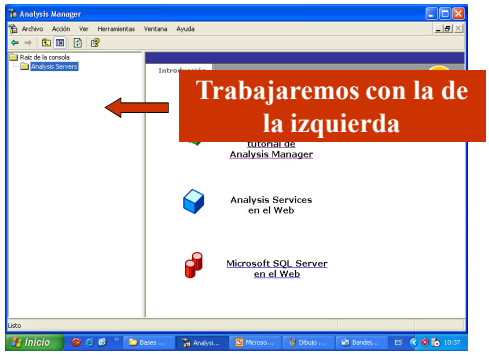
Mail: palasi@palasi.com Web: www.palasi.com

Iniciando Analysis Services

- **Inicio|Todos los programas|SQL Server|Analysis Services|Analysis Manager**



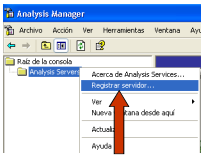
La pantalla principal se divide en dos zonas



Lo primero que debemos hacer es indicar con qué servidor de BD trabajaremos

- Miramos el nombre de nuestro servidor en la bandeja del sistema.

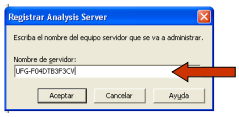
- Hacemos clic derecho en "Analysis Services" y seleccionamos "Registrar servidor".



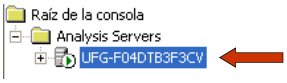
"Registrar servidor" es indicarle a Analysis Services que queremos trabajar con él.

Escribimos el nombre de nuestro servidor tal como lo vimos en la bandeja del sistema

- Lo ponemos en el cuadro de texto que aparece y hacemos clic en **Aceptar**.



- Aparecerá el servidor debajo de "Analysis Servers".



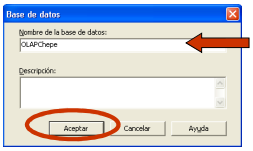
Indicamos a Analysis Services que queremos trabajar con una base de datos

- En clic derecho sobre el servidor, seleccionamos "Nueva base de datos …"

- Observen que el nombre de esta acción es confuso. No se crea una base de datos, pues Analysis Services sólo trabaja con BDs preexistentes.
- Simplemente se le dice a Analysis Services que se quiere trabajar con una BD.

Nos pedirá un nombre de la base de datos

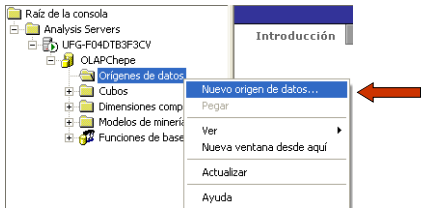
- También esto es confuso. No es un nombre de la base de datos, sino el nombre con el que queremos trabajar en Analysis Services con la BD.



- Ponemos un nombre (en este caso, OLAPChepe) y hacemos clic en **Aceptar**.

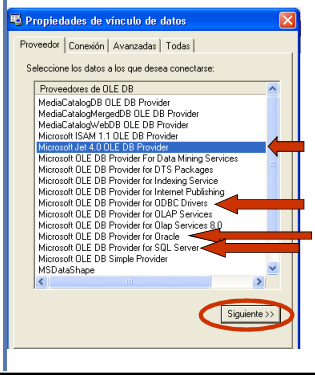
Mail: palasi@palasi.com Web: www.palasi.com

Aparecerá, bajo del nombre que hemos colocado, una serie de subcarpetas



- Hacemos clic derecho en "Orígenes de datos" y seleccionamos "Nuevo origen de datos…".
- Con esto le vamos a decir a Analysis Services qué base de datos vamos a usar para hacer el análisis.

Aparecerán interfaces de bases de datos (APIs de conexión).

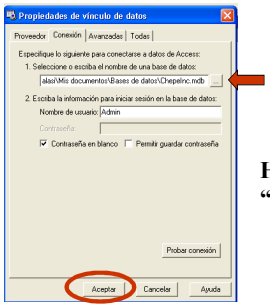


En nuestro caso, seleccionamos Microsoft Jet para Access y clic en Siguiente.

Microsoft Access

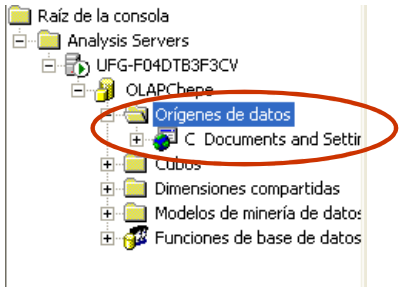
BDs con ODBC
Oracle
SQL Server

Hacemos clic en el botón con la flecha y abrimos la base de datos ChepeInc



Hacemos clic en "Aceptar".

La base de datos se ha incorporado en el apartado de orígenes de datos



Ya estamos preparados para comenzar a trabajar

6. OLAP (Data warehousing).

- 6.1. ¿Qué es OLAP?
- 6.2. Base de datos de prueba.
- 6.3. Microsoft Analysis Services.
- 6.4. Medidas y dimensiones.
- 6.5. Creación de un cubo.
- 6.6. Uso de un cubo.
- 6.7. Dimensiones, niveles y vistas.
- 6.8. Otros aspectos.
- 6.9. Integración con otros programas.

El concepto de hipercubo

- Para comprender OLAP, es fundamental tener claro el concepto de hipercubo de la misma forma que:
  - Para entender una base de datos, es fundamental conocer el concepto de tabla.
  - Para entender una hoja de cálculo, es fundamental conocer el concepto de hoja y celda.
- Nos acercaremos paulatinamente a este concepto.

Por ahora.

- Llamaremos hipercubo (y, por abreviar, cubo) a cualquier conjunto de datos.
- Más adelante, afinaremos más.

Cubos de cero dimensiones (datos)

- Supongamos que queremos analizar el monto de las ventas de ChepeInc durante el año 1998.
- La forma más sencilla de atacar este problema es con un dato que nos indique la venta total durante ese año. Por ejemplo,

$ 2,063,420 — **Monto total de las ventas de ChepeInc el año 1998**

Comencemos con la notación

- Al dato que queremos estudiar (normalmente numérico), se le llama "**medida**".
- En este caso, la medida es el monto de las ventas.

2063 — **A partir de ahora, lo expresaremos en miles de dólares**

Sin embargo, un único dato no es suficiente.

- Con un único dato tan general, un gerente no puede analizar mucho. No da muchos elementos para la toma de decisiones.
- Sería mejor idea entrar en un poco más de detalle para ver si podemos obtener una información más útil.

2063 — **Monto total de las ventas de ChepeInc el año 1998**

Cubos de una dimensión (arreglos)

- Una buena idea es desglosar ese dato global por meses.
- Así podemos ver la evolución a lo largo del tiempo y su carácter estacional, que son datos interesantes para un gerente.

2063 (en miles de dólares)

132	205	205	197	191	157	200	117	240	252	78	91
enero		marzo		mayo		julio		septiembre		noviembre	

Vemos que esto es más útil para hacer análisis que un simple dato

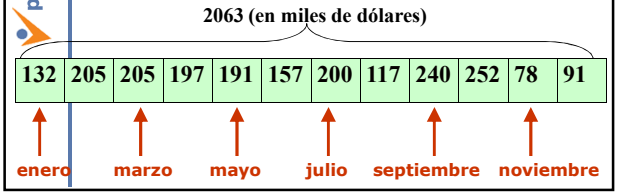
- Por ejemplo, se ve que noviembre y diciembre están muy flojos. Esto nos permite formular dos hipótesis:
- Quizás nuestra oferta de productos en la temporada de Navidad no es lo suficientemente atractiva (porque no están adaptadas a la época).
- Quizás no tenemos promociones atractivas en Navidad (o son menos atractivas que las de la competencia).

2063 (en miles de dólares)

132	205	205	197	191	157	200	117	240	252	78	91
enero		marzo		mayo		julio		septiembre		noviembre	

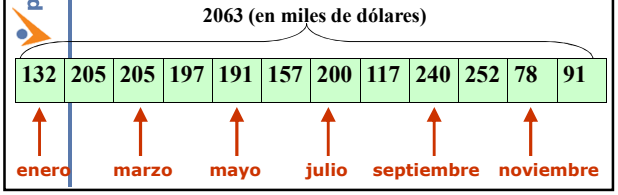
Por ahora, no tenemos suficiente información para decirlo

- Pero con sólo desglosar los datos, hemos podido formular hipótesis que, sin desglosar, hubieran sido imposibles de formular.
- **¡¡Viva el desglose!!**



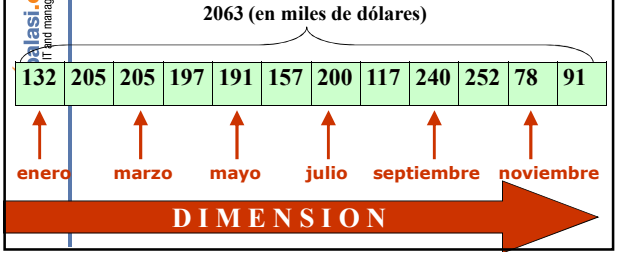
Más jerga

- Recordemos que "**medida**" es el dato que queremos analizar. En este caso, la medida es el monto de las ventas.
- "**Dimensión**" es el criterio por el cual desglosamos (la medida).



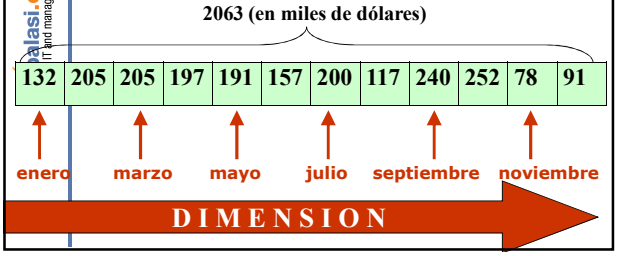
Dimensión es el criterio por el cual desglosamos la medida

- En este caso, la dimensión es el mes, pues desglosamos las ventas por mes. En este caso, es una dimensión temporal.



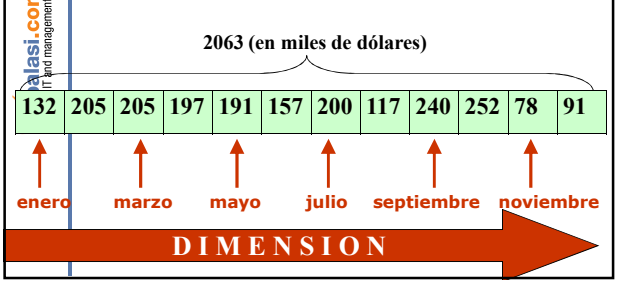
Recapitulando

- **Medida** es el dato que queremos analizar.
- **Dimensión** es el criterio por el cual lo desglosamos.



En este caso

- **Medida** es el precio (las ventas).
- **Dimensión** es el mes.

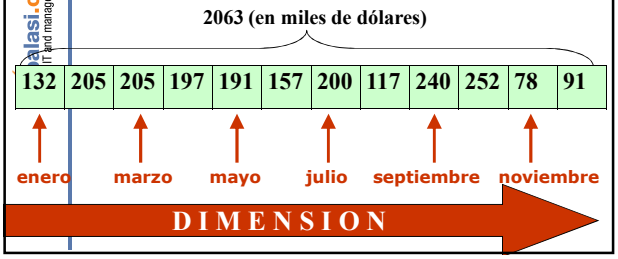


6. OLAP (Data warehousing).

Mail: palasi@palasi.com Web: www.palasi.com

Basta de teoría. ¡Queremos tocar máquina!

- Lo que vamos a hacer es obtener este cubo con Microsoft Analysis Services.



Abran la base de datos ChepeInc, ¿están los datos que queremos obtener?

- No exactamente. Están los datos en bruto, pero no están resumidos por meses.

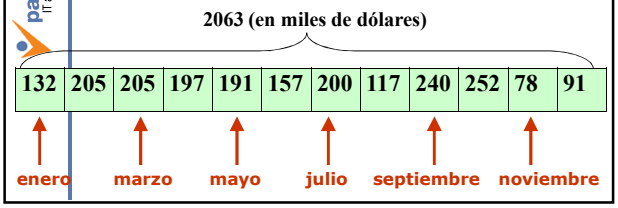
dia	mes	ano	producto	marca	cliente	promocion	tienda	precio	costo
8	January	1998	Tell Tale Ca	Tell Tale	Sweet	No Promotion	Store 3	19,10 €	6,49 €
9	January	1998	Blue Label	Blue Label	Sweet	No Promotion	Store 3	7,66 €	2,60 €
9	January	1998	Blue Label	Blue Label	Tusting	No Promotion	Store 3	15,32 €	7,51 €
10	January	1998	Golden Ora	Golden	Beltran	No Promotion	Store 7	15,52 €	6,05 €
10	January	1998	Club Low F	Club	Strambi	No Promotion	Store 3	8,79 €	2,99 €
10	January	1998	Green Ribb	Green Ribb	Manning	No Promotion	Store 8	2,62 €	1,21 €

- A esta tabla, que contiene los datos en bruto antes de analizarlo, se le llama **tabla de hechos** ("fact table").
- A la BD que contiene la tabla de hechos se le llama **almacén de datos** ("data warehouse").

¿Cómo obtenemos los datos que queremos?

- De la forma tradicional

```
SELECT mes, SUM(precio) AS ventas
FROM chepeinc
GROUP BY mes
ORDER BY mes
```



¿Cómo obtenemos los datos que queremos?

- De la forma tradicional

```
SELECT mes, SUM(precio) AS ventas
FROM chepeinc
GROUP BY mes
ORDER BY mes
```

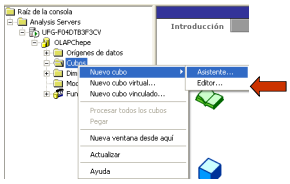
- Esto está muy bien, pero es factible porque es el caso más sencillo.
- Sin embargo, en casos un poco más complicados la sentencia SQL se hace endiabladamente compleja y es muy difícil de descubrir.
- Además, hacerlo con OLAP nos permite una presentación más adecuada y más posibilidad de manipulación de datos.

Probemos con Analysis Services

- Haremos la operación en Analysis Services. Queremos obtener un arreglo con medida precio y dimensión mes.
- Cómo estamos al inicio del conocimiento de esta herramienta y no quiero abrumarlos con detalles que por ahora no están preparados a entender, seguiré la siguiente estrategia
  - No les explicaré más que los pasos que considere necesarios en este punto.
  - Los otros tendrán que tener paciencia y refrenar su curiosidad por el momento.

En AS, hagan clic derecho en Cubos y seleccionen "Nuevo Cubo" y "Editor…"

- Recuerden que, por ahora, cubo quiere decir "conjunto de datos". Más adelante, seremos más rigurosos.



Mail: palasi@palasi.com Web: www.palasi.com

Crear un cubo en Analysis Services

- 1. Especificar la tabla de hechos.
- 2. Especificar las medidas.
- 3. Especificar las dimensiones.
- 4. Especificar las opciones de almacenamiento.

- Las opciones de almacenamiento las estudiaremos después. Por ahora las ignoramos.

Crear un cubo en Analysis Services

- 1. Especificar la tabla de hechos.
- 2. Especificar las medidas.
- 3. Especificar las dimensiones.
- 4. Especificar las opciones de almacenamiento.

- Lo probaremos con un cubo con precio como medida y mes como dimensión.

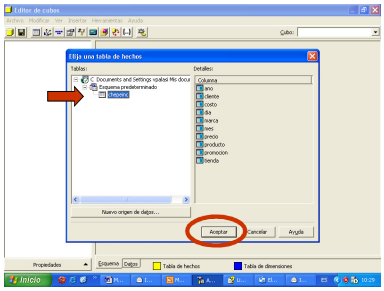
Crear un cubo en Analysis Services

- 1. Especificar la tabla de hechos.
- 2. Especificar las medidas.
- 3. Especificar las dimensiones.
- 4. Especificar las opciones de almacenamiento.

- Lo probaremos con un cubo con precio como medida y mes como dimensión.

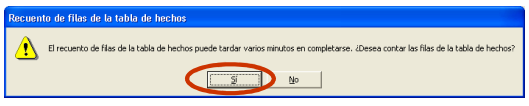
Una vez están en el editor, aparece una ventana que les pide la tabla de hechos

- En nuestro caso, es la tabla "chepeinc". Selecciónenla y hagan clic en "Aceptar".



Hacer clic en el cuadro siguiente

- Les aparecerá un cuadro de diálogo sobre el recuento de filas de la tabla de hechos. Hagan clic en Sí.



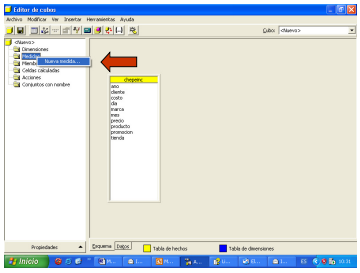
Crear un cubo en Analysis Services

- 1. Especificar la tabla de hechos.
- 2. Especificar las medidas.
- 3. Especificar las dimensiones.
- 4. Especificar las opciones de almacenamiento.

- Lo probaremos con un cubo con precio como medida y mes como dimensión.

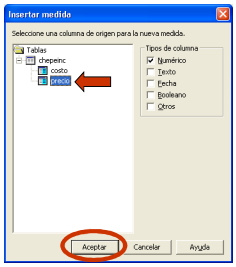
En la ventana que aparece

- Hagan clic derecho en “Medidas” y seleccionen “Nueva medida”.



En la ventana que aparece seleccionen la medida

- En nuestro caso, es el precio. Seleccionen precio y hagan clic en Aceptar.

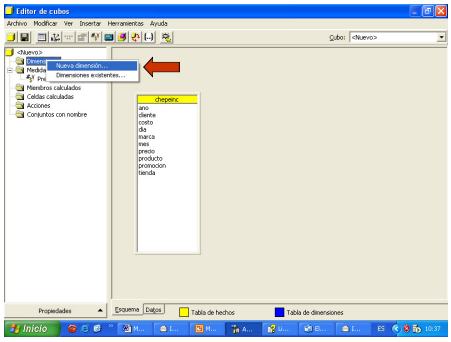


Crear un cubo en Analysis Services

- 1. Especificar la tabla de hechos.
- 2. Especificar las medidas.
- 3. Especificar las dimensiones.
- 4. Especificar las opciones de almacenamiento.

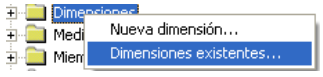
- Lo probaremos con un cubo con precio como medida y mes como dimensión.

Hacer clic derecho en Dimensiones y seleccionar “Nueva dimensión”



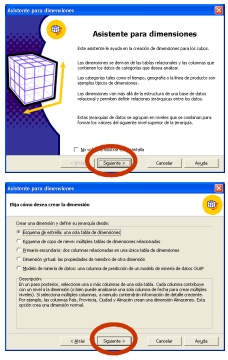
Nota importante.

- Si hubiéramos programado un cubo antes y, por lo tanto, hubiéramos definido dimensiones antes.
- Entonces, puede que nos interesara aprovechar alguna de las dimensiones que hemos definido (en vez de volver a definirla).
- Es para eso que existe la opción “Dimensiones existentes” que hay al lado de “Nueva dimensión”.



Aparecerá el Asistente para dimensiones

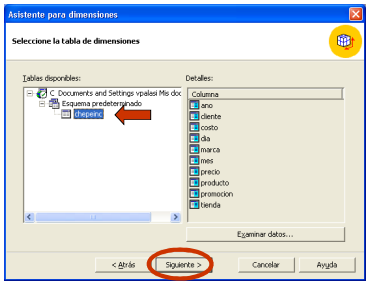
- Ignoren la ventana que aparece y hagan clic en Siguiente.
- Ignoren la siguiente ventana y hagan clic en Siguiente.



Mail: palasi@palasi.com Web: www.palasi.com

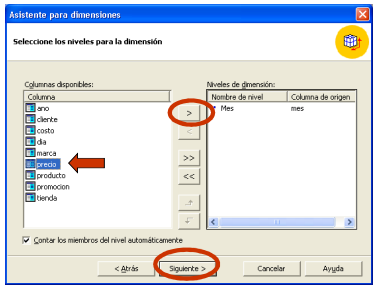
En la ventana, podemos elegir la tabla que definirá la dimensión que queremos

- En nuestro caso, es la tabla de hechos. Elegimos chepeinc y hacemos clic en Siguiente.



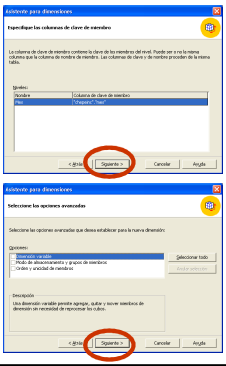
Podremos seleccionar el campo de la tabla que produce la dimensión

- En nuestro caso, mes.



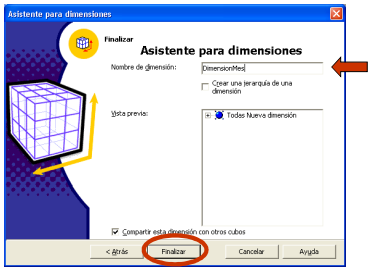
Ignoren las dos pantallas siguientes

- "La ignorancia es felicidad", Cypher, The Matrix.



Acabando la definición de dimensión

- Pongan un nombre a la dimensión y hagan clic en Finalizar. Aquí le ponemos "DimensionMes".



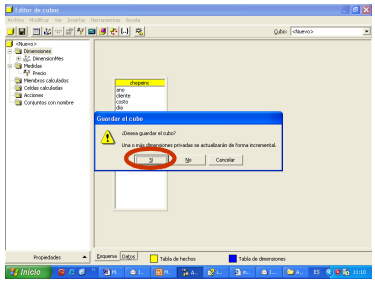
Crear un cubo en Analysis Services

- 1. Especificar la tabla de hechos.
- 2. Especificar las medidas.
- 3. Especificar las dimensiones.
- 4. Especificar las opciones de almacenamiento.

- Lo probaremos con un cubo con precio como medida y mes como dimensión.

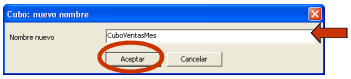
Cerramos la ventana principal del editor de cubos

- Nos preguntará si queremos guardar el cubo. Decimos que sí.

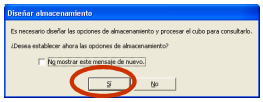


Pondremos un nombre al cubo y hacemos clic en "Aceptar"

- En este caso, se le llama CuboVentasPorMes

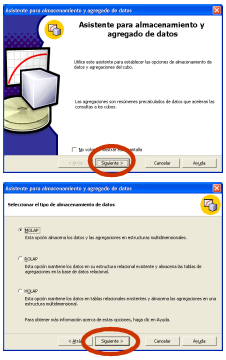


- Nos dirá si queremos especificar las opciones de almacenamiento. Decimos Sí.



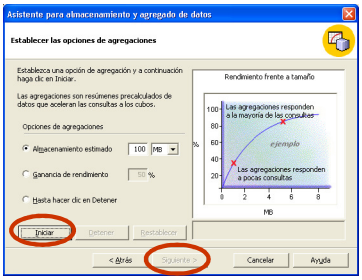
Las dos siguientes pantallas las ignoramos

- Simplemente, hacemos clic en Siguiente.



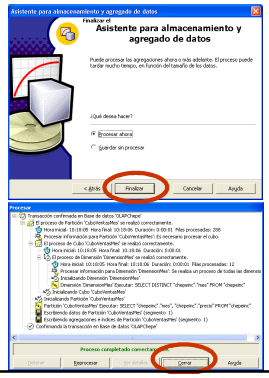
En la siguiente ventana, hacemos clic en "Iniciar" y después en "Siguiente"

- Tampoco nos fijamos en esto por ahora.



Otras que ignoramos son las siguientes pantallas

- Hacemos "Finalizar" en la primera y "Cerrar" en la segunda.

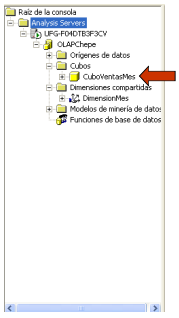


Crear un cubo en Analysis Services

- 1. Especificar la tabla de hechos.
- 2. [illegible]
- 3. Especificar las dimensiones.
- 4. Especificar las opciones de almacenamiento.

¡¡Ya hemos creado un cubo!!

Vemos el cubo a la izquierda de la pantalla principal, debajo de "Cubos"



- También vemos la dimensión que hemos creado

Mail: palasi@palasi.com Web: www.palasi.com

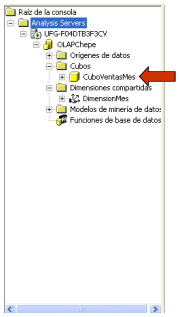
6. OLAP (Data warehousing).

6.6. Uso de un cubo

6.6. Uso de un cubo

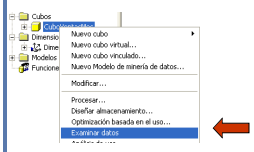
- 6.6.1. Examinar los datos de un cubo.
- 6.6.2. Actualizar un cubo.

Ya tenemos nuestro cubo



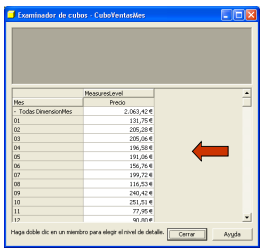
- Lo vemos a la izquierda de la ventana principal de Analysis Services.
- Ahora queremos examinar los datos.

Hacemos clic derecho en el cubo



- Y hacemos clic en "Examinar datos".

Obtenemos lo que queríamos

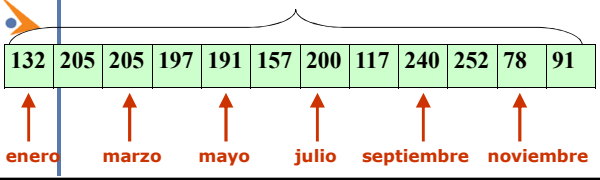


- Una lista de ventas desglosada por meses.
- En general, un análisis de la medida desglosada según la dimensión.

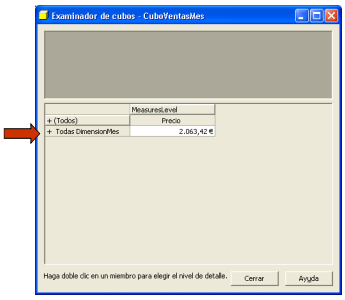
Mail: palasi@palasi.com Web: www.palasi.com

Fíjense que eso coincide con lo que vimos al principio.

- El arreglo que queríamos conseguir.



Haciendo doble clic en el primer registro, elegimos el nivel de detalle.



La buena noticia es que

- Una vez un cubo está programado podemos examinar sus datos tantas veces como queramos sin tener que volverlo a programar.
- Esto divide a las personas que usan OLAP en dos clases:
  - **Creadores de cubos.** Son los que crean los cubos como lo hemos visto ahora. Puede ser un programador o un usuario avanzado ("power user"), incluyendo algunos de los gerentes.
  - **Usuarios de cubos.** Son los que usan los cubos. Puede ser cualquier usuario con un mínimo conocimiento. Así, todos los gerentes entrarían en esta categoría.

¿Y qué hay de los programadores?

- Si hay una verdad en la Informática, es que "los programadores son de Marte y los usuarios son de Venus".
- Los programadores son tipos rudos y curtidos por la vida que matan por un código bien construido y una arquitectura robusta y flexible.
- Los usuarios son seres delicados y sensibles que se deleitan con la facilidad de uso y con el atractivo estético de las interfaces de usuario.

Lo que estamos viendo es manejar OLAP desde un administrador

- Es como manejar una base de datos SQL Server desde el analizador de consultas o una BD Access desde Microsoft Access.
- Esto te da mucha flexibilidad si eres un usuario avanzado, pero un usuario normal puede sentirse perdido.
- Para estos tipos de usuario mejor programar un programa que les simplifique las cosas. En el caso de OLTP, un programa que acceda a la BD. En el caso de OLAP, un programa que acceda a los cubos.
- El programador será el encargado de programar ese programa.

Entonces, también OLAP se puede programar

- Hay un lenguaje llamado MDX para programar OLAP y hay APIs para acceder a los cubos desde un programa.
- Esto es análogo a SQL y a las APIs que, en OLTP, nos permiten acceder a la BD desde un programa (ADO.NET, JDBC).
- Así podemos hacer un programa OLAP más adecuado para el usuario final.
- Esto no lo veremos por ahora.

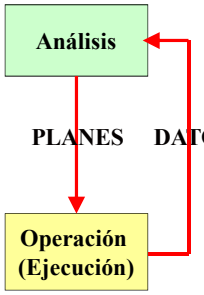
Ejercicio

- Creen un cubo de ventas por producto.

6.6. Uso de un cubo

- 6.6.1. Examinar los datos de un cubo.
- 6.6.2. Actualizar un cubo.

¿Qué pasa si actualizamos los datos de la tabla de hechos (del almacén de datos)?



- Esto no suele ser tan frecuente, pues normalmente tenemos datos pasados, que ya no se modifican.
- Eso se debe a que estamos en la etapa de análisis y no en la de operación, como pasaba en OLTP.

¿Qué pasa si actualizamos los datos de la tabla de hechos (del almacén de datos)?

- Supongamos que abrimos la tabla ChepeInc y ponemos $2,019.10 en el campo precio del primer registro.

dia	mes	ano	producto	marca	cliente	promocion	tienda	precio	costo
08	01	1998	Tell Tale Ca	Tell Tale	Sweet	No Promotion	Store 3	2.019,10 €	6,49 €
09	01	1998	Blue Label	Blue Label	Sweet	No Promotion	Store 3	7,66 €	2,60 €
09	01	1998	Blue Label	Blue Label	Tusting	No Promotion	Store 3	15,32 €	7,51 €

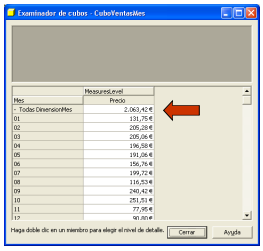
- Esto aumenta las ventas en 2000 (miles de dólares).
- Háganlo.

Examinen de nuevo los datos del cubo "CuboVentasMes"

- ¿Qué ven?

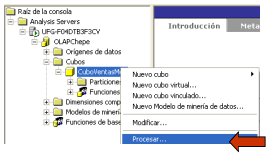
Nada ha cambiado

- Primera lección. La actualización de los cubos con nuevos datos no es automática.

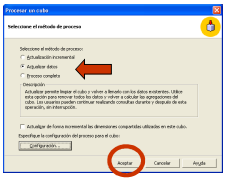


Para actualizar los cubos.

- Hay que hacer clic derecho en ellos y seleccionar "Procesar…".

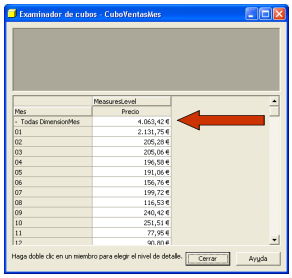


- Después seleccionar la opción "Actualizar datos" y "Aceptar".



Después podemos examinar de nuevo los datos

- Y estarán actualizados.



Nota importante

- Cuando actualicemos un cubo, elegimos la opción:
  - **Actualizar datos**, si sólo han cambiado los datos del almacén de datos.
  - **Proceso completo**, si ha cambiado la definición del cubo.

Ejercicio

- Modifiquen el cubo de ventas por producto para que sea un cubo de costos por producto.
- Atención: se trata de modificar el cubo existente y no de crear uno nuevo.

6. OLAP (Data warehousing).

6.7. Dimensiones, niveles y vistas

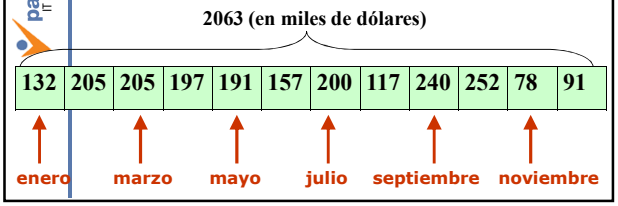
Mail: palasi@palasi.com Web: www.palasi.com

6.7. Dimensiones, niveles y vistas

- 6.7.1. Dimensiones de varios niveles.
- 6.7.2. Cubos de varias medidas.
- 6.7.3. Rebanadas.
- 6.7.4. Cubos de varias tablas.
- 6.7.5. Otros aspectos.

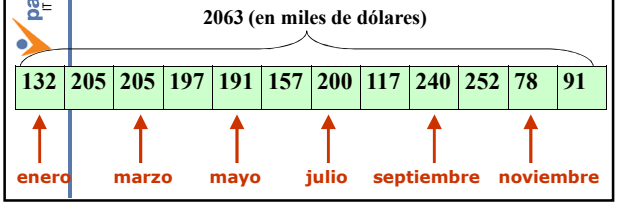
Lo que hemos hecho ha sido agrupar las ventas por meses

- Esto tiene ventajas para el análisis. Por ejemplo, hemos podido formular la hipótesis de que algo pasa en Navidad.



Sin embargo, para algunos propósitos esta información puede ser demasiado general

- Imaginemos que, después de formular la hipotesis, queremos comprobarla, viendo qué pasó la semana antes de Navidad.
- Esto no lo podemos ver con este arreglo: es demasiado general.



El problema es que a veces queremos mirar las cosas en detalle y otras de forma general

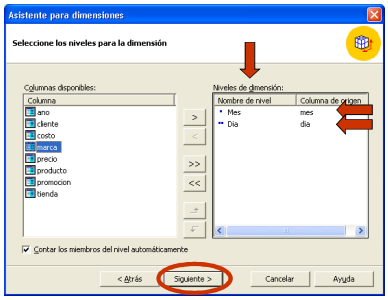
- A veces, con un mapa y, a veces, con una lupa.
- ¿No sería bonito que pudiéramos tener las dos cosas a la vez? Así podríamos elegir el nivel de detalle que queramos en cada momento.
- Así, por ejemplo, cuando queramos podemos ver las ventas por mes y otras veces podemos ver las ventas por día.

Para eso, están los niveles

- Tenemos una dimensión en la que queremos desglosar los datos. En nuestro caso, la dimensión del tiempo.
- Los niveles sirven para hacer este desglose en diferentes niveles de detalle. Así, en nuestro caso, podemos ver las ventas por mes y por día.
- Habría un nivel general de mes y un nivel específico de día.

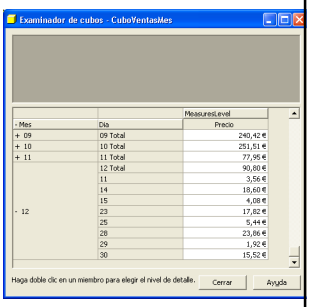
Creen un cubo exactamente como el que han hecho, pero cuando lleguen

- A la ventana de niveles, elijan mes y día (en ese orden)



Examinen los datos y verán algo así

- Como ven, los detalles se pueden ver en un nivel más general o más específico, contrayendo o expandiendo los nodos como si se tratara de carpetas en el Explorador de Windows.
- Estas son las ventajas de usar varios niveles en una dimensión.



Ya estamos llegando a algo

- Como vemos, los días alrededor de Navidad no hay un aumento espectacular de las ventas como debería ser.
- Esto es otra evidencia a favor de nuestra hipótesis, pero hay que estudiar más.

Borren el cubo que acabamos de hacer

- Vamos a enseñarles cómo añadir un nivel a una dimensión sin necesidad de crear todo el cubo desde el principio.
- Clic derecho en la dimensión del primer cubo (DimensionMes) y seleccionen "Modificar".

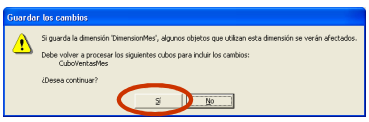
Seleccionen la dimensión y hagan clic en "Insertar|Nivel"

- En la pantalla que aparece, elijan día y hagan clic en Aceptar.

Verán que aparece un nuevo nivel a la izquierda

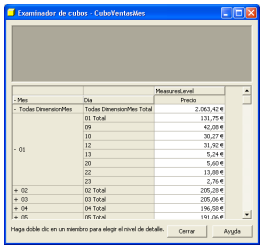
- Cerremos la ventana, guardemos los cambios.

- Nos avisa de que debemos volver a procesar el cubo. Hagamos Sí.



Volvamos a procesar el cubo (con Proceso completo) y a examinar datos.

- Y "voilà", ya hemos añadido un nivel a una dimensión sin necesidad de crear el cubo desde el principio.



- Queda como ejercicio quitar de nuevo ese nivel, para dejar el cubo como estaba en un principio.

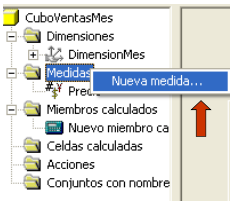
Ejercicio

- Creen un cubo de ventas por una dimensión que tenga dos niveles. El general es el nivel de marca y el específico es el nivel de producto.

6.7. Dimensiones, niveles y vistas

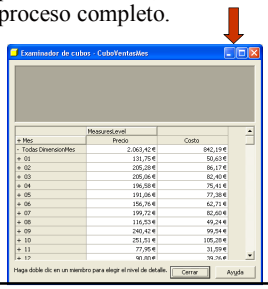
Hasta ahora hemos visto que existía una medida

- Un único aspecto que queremos estudiar.
- No hay problema en que haya varias medidas.
- Simplemente, las añadimos en el editor de cubos, haciendo clic derecho en Medidas y seleccionando "Nueva medida…".



Así podemos estudiar varios aspectos al mismo tiempo

- Háganlo, guarden los cambios y vuelvan a procesar el cubo, con proceso completo.



- Aparecerán las dos medidas al mismo tiempo, los que nos permite hacer un estudio conjunto.

Más difícil todavía

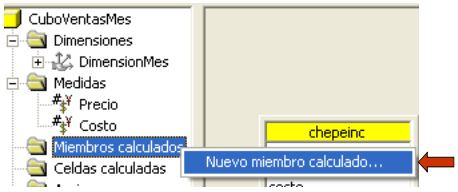
- Ya que tenemos el precio y el costo, ¿por qué no estudiar el beneficio?
- No tenemos el beneficio en la tabla de hechos, pero es fácil de calcular a partir del precio y del costo.
- Para ello, usamos una nueva característica, llamada "miembros calculados".

Miembros calculados

- Medidas que se calculan a partir de otras (o miembros de una dimensión que se calculan a partir de otros, como veremos después).
- No se encuentran en el almacén de datos sino que son cálculos sobre ellos.
- Lo veremos primero con las medidas.
- En nuestro caso, queremos definir una medida llamada beneficio, calculada a partir de las medidas "precio" y "costo", según la fórmula "precio-costo".

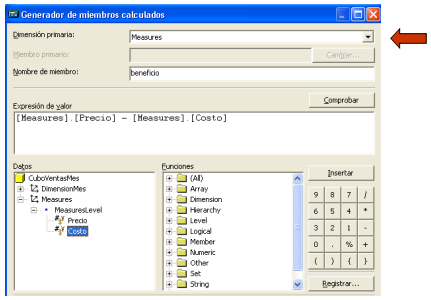
Desde el editor de cubos

- Hacemos clic derecho en “Miembros calculados” y seleccionamos “Nuevo miembro calculado…”.



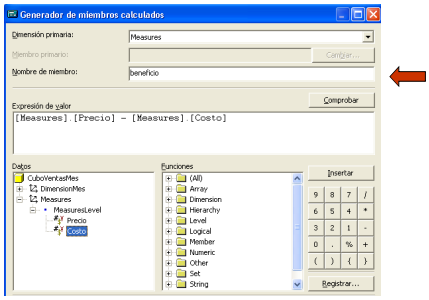
Ponemos dimensión primaria “Measures”

- Eso indica que el nuevo miembro calculado será una medida



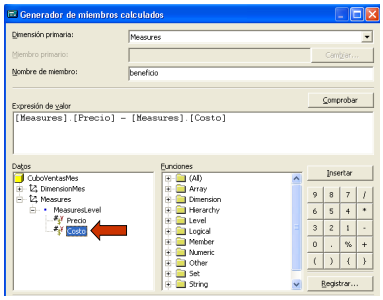
Le ponemos un nombre

- En este caso, le llamamos “beneficio”.



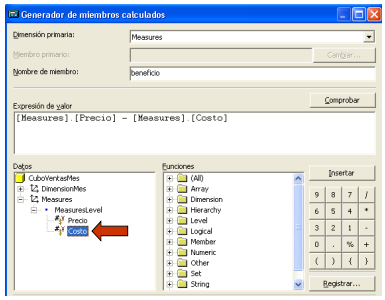
Expandimos el árbol de abajo y su nodo Measures y su subnodo MeasuresLevel

- Aparecerán precio y costo. Hacemos doble clic y sus expresiones aparecerán en el cuadro del medio.

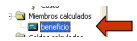


Ponemos un signo menos entre las dos expresiones

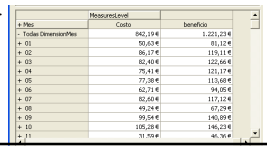
- Para indicar que se restan y hacemos clic en Aceptar.



Aparece que “beneficio” es un miembro calculado



- Salimos del editor de cubos guardando.
- Volvemos a procesar el cubo con “Proceso completo”.
- Examinamos los datos y ya está. Beneficio aparece como medidas.



Mail: palasi@palasi.com Web: www.palasi.com

Ejercicio

- Creen una nueva medida que indique el monto del IVA para las ventas.
- Se supone que todas las ventas son gravadas con IVA con un tipo del 13%.

Creando un miembro calculado para una dimensión

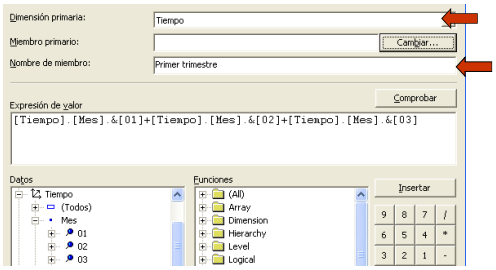
- Hasta ahora, los valores de una dimensión era los que estaban en la tabla como valores de campo.
- Así la dimensión "mes", tenía los valores 1,2,3…12, que son los que están en la tabla.
- Así la dimensión "producto", tenía los productos que hay en la tabla.

Por ej, queremos ver cómo se comportaron los productos el primer trimestre

- "Primer trimestre" no es un valor de la dimensión mes, pero se puede calcular fácilmente a partir de los valores del mes 1, 2 y 3.
- Necesitamos crear un valor nuevo de una dimensión que sea calculado a partir de los valores existentes.

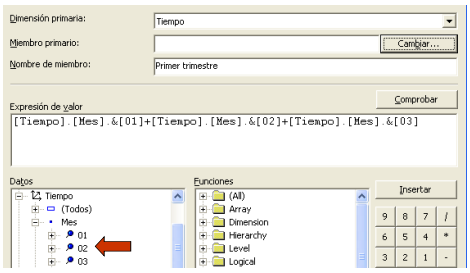
Creamos un nuevo miembro calculado

- Se selecciona la dimensión tiempo y como nombre de miembro se pone "Primer trimestre".



Creamos un nuevo miembro calculado

- Se seleccionan los meses 1, 2 y 3 en el árbol de abajo a la izquierda.

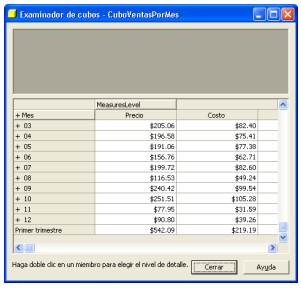


Creamos un nuevo miembro calculado

- En las expresiones que aparecen se suman para sacar el resultado del primer trimestre. Hagan clic en Aceptar y salgan del Diseñador de cubos.

Se reprocesa el cubo (proceso completo) y ya podemos ver primer trimestre

- Háganlo. ¿Ven qué fácil?



Ejercicio

- Crear miembros calculados de la dimensión para los otros tres trimestres.

Ejercicio de repaso

- Creen un cubo que analice las ventas por tienda. Queremos saber el precio de las ventas, el costo, el IVA, el beneficio y el impuesto sobre la renta (que es el 10% del beneficio).
- Las primeras cinco tiendas corresponden a la costa este y las segundas cinco tiendas corresponden a la costa oeste. Queremos ver los consolidados por costa.

6.7. Dimensiones, niveles y vistas

- 6.7.1. Dimensiones de varios niveles.
- 6.7.2. Cubos de varias medidas.
- 6.7.3. Rebanadas.
- 6.7.4. Cubos de varias tablas.
- 6.7.5. Otros aspectos.

Volvamos a casa

- Con las medidas nos hemos desviado de nuestro propósito que era analizar las ventas.
- Volvemos a nuestro propósito principal. Eliminen las medidas “costo” y “beneficio” desde el editor del cubo y reprocesen el cubo (proceso completo).
- Así volvemos al cubo inicial.

Teníamos una hipótesis

- Las ventas bajan porque Chepe, Inc. no tiene una buena estrategia navideña.
- Parece que esto se ha confirmado cuando hemos desglosado las ventas por días.
- Pero esto puede ser por dos cosas:
  - Los productos que ofrecemos no se venden en Navidad (en ese caso, deberíamos incluir otros productos en nuestra oferta).
  - Las promociones para Navidad no son efectivas o son menos efectivas que la competencia.

En otras palabras

- El problema está en los productos o el problema está en las promociones.
- Es decir, tenemos dos hipótesis y queremos validarlas.
- Pero para eso está OLAP para validar las hipótesis.
- Comencemos por ver si la primera hipótesis es válida.

La hipótesis de que los productos no son adecuados para Navidad

- Si validamos esta hipótesis, podemos hacer una alianza estratégica con Santa Klaus, S.A. de C.V. para que nos envíe algunos productos desde el Ártico.
- La verdad es que lo que deberíamos saber para validar esta hipótesis es cómo evoluciona la venta cada uno de los productos a través del tiempo.

Es decir, lo que querríamos es algo como lo siguiente

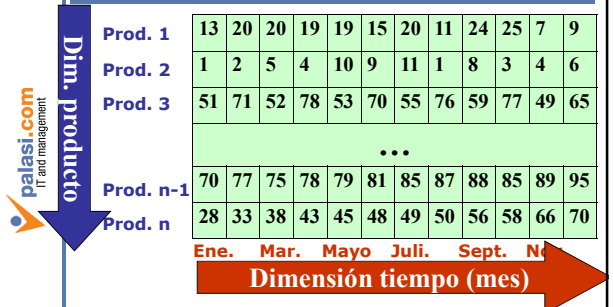
	Ene.			Mar.		Mayo		Juli.		Sept.		Nov.
Prod. 1	13	20	20	19	19	15	20	11	24	25	7	9
Prod. 2	1	2	5	4	10	9	11	1	8	3	4	6
Prod. 3	51	71	52	78	53	70	55	76	59	77	49	65
	...											
Prod. n-1	70	77	75	78	79	81	85	87	88	85	89	95
Prod. n	28	33	38	43	45	48	49	50	56	58	66	70

- Así podemos saber cómo evoluciona la venta cada uno de los productos a través del tiempo.

Esto es un cubo de dos dimensiones

- Hay dos formas diferentes de desglosar los datos (desglosar la medida).

A los cubos de dos dimensiones se les llama tablas



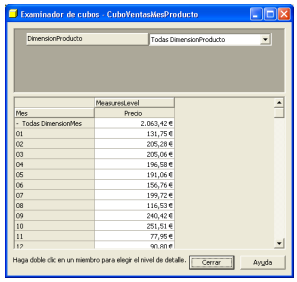
- No confundir con las tablas de modelo relacional.

¿Cómo se crea un cubo de dos dimensiones?

- Fácil. Igual que los cubos de una dimensión que se han hecho hasta ahora. Simplemente, en el editor de cubos, se especifica que son dos dimensiones.
- Creen el cubo anterior de ventas con dos dimensiones: mes y producto. Examinen los datos.

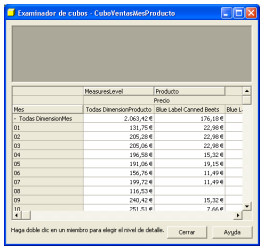
Lo que sale no parece lo que queríamos

- Queríamos una tabla.
- Para conseguirla, arrastramos el cuadro donde pone DimensionProducto a la tabla de precio que hay abajo.



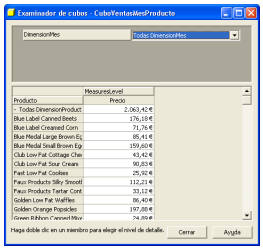
Ahora ya aparece una tabla

- Pero se visualiza de forma inversa a la que nosotros queríamos.
- Nosotros queríamos el mes en la dimensión horizontal y el producto en la dimensión vertical.



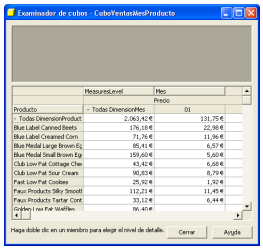
No hay problema

- Arrastren (la celda titulada como) Mes de nuevo al área color café.
- Aparecerá desglosado por producto.



Después, vuelvan a arrastrar DimensionMes al área de abajo

- Obtendremos la tabla que queremos.
- Como vemos, en el examinador de cubos:
- Como mínimo abajo debe haber una dimensión.
- La última dimensión que se añade se coloca horizontal y la primera vertical.



Ejercicio

- Creen un nuevo cubo para examinar las ventas por tienda. Debe tener dos dimensiones:
- La primera tendrá dos niveles: mes y día.
- La segunda será tienda.
- Muestren el cubo en el examinador de cubos.
- ¿Cuánto vendió la tienda "Store 1" en el mes de febrero?

Ejercicio

- Creen un nuevo cubo para examinar las ventas por productos y promociones.
- Las dos dimensiones son productos y promociones.
- Fíjense que este cubo no contiene información temporal, pero que no por eso deja de ser muy útil a Gerencia.

Mail: palasi@palasi.com Web: www.palasi.com

Recuperen el cubo de producto y mes y pongan producto horizont. y el mes vertical

	Producto				
Mes	Todas Dime	Blue Label Canned Beets	Blue Label Creamed Corn	ue Medal Large Brown Egg	lue Medal Small Brown Egg
- Todas	2.063,42 €	176,18 €	71,76 €	85,41 €	159,60 €
01	131,75 €	22,98 €	11,96 €	6,57 €	5,60 €
02	205,28 €	22,98 €	8,97 €	10,95 €	11,20 €
03	205,06 €	22,98 €	5,98 €	6,57 €	
04	196,58 €	15,32 €			19,60 €
05	191,06 €	19,15 €	11,96 €	17,52 €	19,60 €
06	156,76 €	11,49 €			22,40 €
07	199,72 €	11,49 €		10,95 €	8,40 €
08	116,53 €			13,14 €	11,20 €
09	240,42 €	15,32 €	20,93 €	6,57 €	14,00 €
10	251,51 €	7,66 €		6,57 €	28,00 €
11	77,95 €	26,81 €	11,96 €		5,60 €
12	90,80 €			6,57 €	14,00 €

- Esta es toda la información que tenemos sobre los productos por meses.

¿Qué es lo que pasa si decidimos que es demasiada información?

	Producto				
Mes	Todas Dime	Blue Label Canned Beets	Blue Label Creamed Corn	ue Medal Large Brown Egg	lue Medal Small Brown Egg
- Todas	2.063,42 €	176,18 €	71,76 €	85,41 €	159,60 €
01	131,75 €	22,98 €	11,96 €	6,57 €	5,60 €
02	205,28 €	22,98 €	8,97 €	10,95 €	11,20 €
03	205,06 €	22,98 €	5,98 €	6,57 €	
04	196,58 €	15,32 €			19,60 €
05	191,06 €	19,15 €	11,96 €	17,52 €	19,60 €
06	156,76 €	11,49 €			22,40 €
07	199,72 €	11,49 €		10,95 €	8,40 €
08	116,53 €			13,14 €	11,20 €
09	240,42 €	15,32 €	20,93 €	6,57 €	14,00 €
10	251,51 €	7,66 €		6,57 €	28,00 €
11	77,95 €	26,81 €	11,96 €		5,60 €
12	90,80 €			6,57 €	14,00 €

- Podemos quedarnos con parte de la información para analizarla.

Quedándonos con parte de la información

	Producto				
Mes	Todas Dime	Blue Label Canned Beets	Blue Label Creamed Corn	ue Medal Large Brown Egg	lue Medal Small Brown Egg
- Todas	2.063,42 €	176,18 €	71,76 €	85,41 €	159,60 €
01	131,75 €	22,98 €	11,96 €	6,57 €	5,60 €
02	205,28 €	22,98 €	8,97 €	10,95 €	11,20 €
03	205,06 €	22,98 €	5,98 €	6,57 €	
04	196,58 €	15,32 €			19,60 €
05	191,06 €	19,15 €	11,96 €	17,52 €	19,60 €
06	156,76 €	11,49 €			22,40 €
07	199,72 €	11,49 €		10,95 €	8,40 €
08	116,53 €			13,14 €	11,20 €
09	240,42 €	15,32 €	20,93 €	6,57 €	14,00 €
10	251,51 €	7,66 €		6,57 €	28,00 €
11	77,95 €	26,81 €	11,96 €		5,60 €
12	90,80 €			6,57 €	14,00 €

- Podemos quedarnos con una fila o una columna.

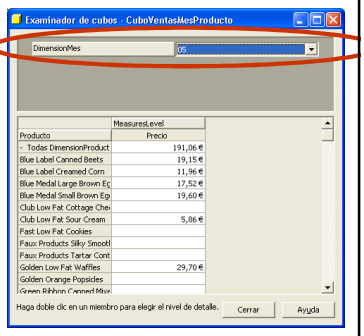
Nos podemos quedar sólo con una fila

	Producto				
Mes	Todas Dime	Blue Label Canned Beets	Blue Label Creamed Corn	ue Medal Large Brown Egg	lue Medal Small Brown Egg
- Todas	2.063,42 €	176,18 €	71,76 €	85,41 €	159,60 €
01	131,75 €	22,98 €	11,96 €	6,57 €	5,60 €
02	205,28 €	22,98 €	8,97 €	10,95 €	11,20 €
03	205,06 €	22,98 €	5,98 €	6,57 €	
04	196,58 €	15,32 €			19,60 €
05	191,06 €	19,15 €	11,96 €	17,52 €	19,60 €
06	156,76 €	11,49 €			22,40 €
07	199,72 €	11,49 €		10,95 €	8,40 €
08	116,53 €			13,14 €	11,20 €
09	240,42 €	15,32 €	20,93 €	6,57 €	14,00 €
10	251,51 €	7,66 €		6,57 €	28,00 €
11	77,95 €	26,81 €	11,96 €		5,60 €
12	90,80 €			6,57 €	14,00 €

- Ya sea la fila que tiene el resumen de todos los meses o bien la fila que tiene un mes en concreto (en la figura, mayo).

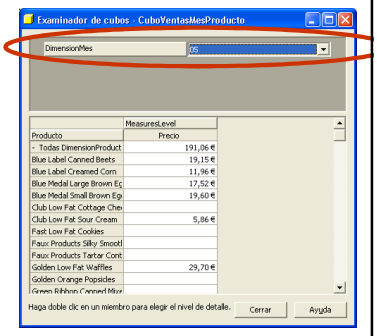
Para quedarnos sólo con una fila, pasamos la dimensión "mes" al área café

- Si es la fila de resumen la que queremos ver, ya está.
- Si es la fila de un mes (mayo), desplegamos el combo de DimensionMes y elegimos el mes.



Ahora con junio

- Quédense con el mes de junio.



¿Qué hemos hecho?

- Nos hemos quedado sólo con una parte (una fila) de la tabla inicial (del cubo).
- Se dice que hemos obtenido una **rebanada del cubo.**

Producto

Mes	Todas Dime	Blue Label Canned Beets	Blue Label Creamed Corn	ue Medal Large Brown Egg	lue Medal Small Brown Egg
- Todas	2.063,42 €	176,18 €	71,76 €	85,41 €	159,60 €
01	131,75 €	22,98 €	11,96 €	6,57 €	5,60 €
02	205,28 €	22,98 €	8,97 €	10,95 €	11,20 €
03	205,06 €	22,98 €	5,98 €	6,57 €	
04	196,58 €	15,32 €			19,60 €
05	191,06 €	19,15 €	11,96 €	17,52 €	19,60 €
06	156,76 €	11,49 €			22,40 €
07	199,72 €	11,49 €		10,95 €	8,40 €
08	116,53 €			13,14 €	11,20 €
09	240,42 €	15,32 €	20,93 €	6,57 €	14,00 €
10	251,51 €	7,66 €		6,57 €	28,00 €
11	77,95 €	26,81 €	11,96 €		5,60 €
12	90,80 €			6,57 €	14,00 €

También se dice que

- Hemos **rebanado** el cubo.
- Esta es terminología OLAP.

Producto

Mes	Todas Dime	Blue Label Canned Beets	Blue Label Creamed Corn	ue Medal Large Brown Egg	lue Medal Small Brown Egg
- Todas	2.063,42 €	176,18 €	71,76 €	85,41 €	159,60 €
01	131,75 €	22,98 €	11,96 €	6,57 €	5,60 €
02	205,28 €	22,98 €	8,97 €	10,95 €	11,20 €
03	205,06 €	22,98 €	5,98 €	6,57 €	
04	196,58 €	15,32 €			19,60 €
05	191,06 €	19,15 €	11,96 €	17,52 €	19,60 €
06	156,76 €	11,49 €			22,40 €
07	199,72 €	11,49 €		10,95 €	8,40 €
08	116,53 €			13,14 €	11,20 €
09	240,42 €	15,32 €	20,93 €	6,57 €	14,00 €
10	251,51 €	7,66 €		6,57 €	28,00 €
11	77,95 €	26,81 €	11,96 €		5,60 €
12	90,80 €			6,57 €	14,00 €

Rebanar (to slice)

- Acción por la cual nos quedamos sólo con una rebanada (una parte) de un cubo para analizarla.

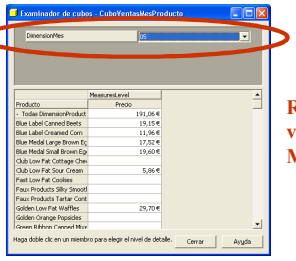
Producto

Mes	Todas Dime	Blue Label Canned Beets	Blue Label Creamed Corn	ue Medal Large Brown Egg	lue Medal Small Brown Egg
- Todas	2.063,42 €	176,18 €	71,76 €	85,41 €	159,60 €
01	131,75 €	22,98 €	11,96 €	6,57 €	5,60 €
02	205,28 €	22,98 €	8,97 €	10,95 €	11,20 €
03	205,06 €	22,98 €	5,98 €	6,57 €	
04	196,58 €	15,32 €			19,60 €
05	191,06 €	19,15 €	11,96 €	17,52 €	19,60 €
06	156,76 €	11,49 €			22,40 €
07	199,72 €	11,49 €		10,95 €	8,40 €
08	116,53 €			13,14 €	11,20 €
09	240,42 €	15,32 €	20,93 €	6,57 €	14,00 €
10	251,51 €	7,66 €		6,57 €	28,00 €
11	77,95 €	26,81 €	11,96 €		5,60 €
12	90,80 €			6,57 €	14,00 €

Rebanar sería quedarnos con esta fila (rebanada)

Rebanadas (slices)

- Partes de un cubo que examinamos por separado para analizar sus datos.



Rebanada tal como la vemos en el Analysis Manager

Hasta ahora hemos rebanado por meses (filas)

- Hemos visto las ventas de un mes determinado (o del resumen de todos los meses) desglosadas por productos. El criterio para rebanar ha sido un mes.

Producto

Mes	Todas Dime	Blue Label Canned Beets	Blue Label Creamed Corn	ue Medal Large Brown Egg	lue Medal Small Brown Egg
- Todas	2.063,42 €	176,18 €	71,76 €	85,41 €	159,60 €
01	131,75 €	22,98 €	11,96 €	6,57 €	5,60 €
02	205,28 €	22,98 €	8,97 €	10,95 €	11,20 €
03	205,06 €	22,98 €	5,98 €	6,57 €	
04	196,58 €	15,32 €			19,60 €
05	191,06 €	19,15 €	11,96 €	17,52 €	19,60 €
06	156,76 €	11,49 €			22,40 €
07	199,72 €	11,49 €		10,95 €	8,40 €
08	116,53 €			13,14 €	11,20 €
09	240,42 €	15,32 €	20,93 €	6,57 €	14,00 €
10	251,51 €	7,66 €		6,57 €	28,00 €
11	77,95 €	26,81 €	11,96 €		5,60 €
12	90,80 €			6,57 €	14,00 €

También podemos rebanar por productos (columnas)

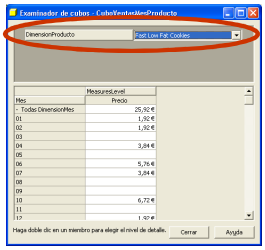
- Ver las ventas de un producto determinado (o del resumen de todos los productos) desglosadas por meses. El criterio para rebanar es el producto.

Producto

Mes	Todas Dime	Blue Label Canned Beets	Blue Label Creamed Corn	ue Medal Large Brown Egg	lue Medal Small Brown Egg
- Todas	2.063,42 €	176,18 €	71,76 €	85,41 €	159,60 €
01	131,75 €	22,98 €	11,96 €	6,57 €	5,60 €
02	205,28 €	22,98 €	8,97 €	10,95 €	11,20 €
03	205,06 €	22,98 €	5,98 €	6,57 €	
04	196,58 €	15,32 €			19,60 €
05	191,06 €	19,15 €	11,96 €	17,52 €	19,60 €
06	156,76 €	11,49 €			22,40 €
07	199,72 €	11,49 €		10,95 €	8,40 €
08	116,53 €			13,14 €	11,20 €
09	240,42 €	15,32 €	20,93 €	6,57 €	14,00 €
10	251,51 €	7,66 €		6,57 €	28,00 €
11	77,95 €	26,81 €	11,96 €		5,60 €
12	90,80 €			6,57 €	14,00 €

Para rebanar por productos en SQL Server (columnas)

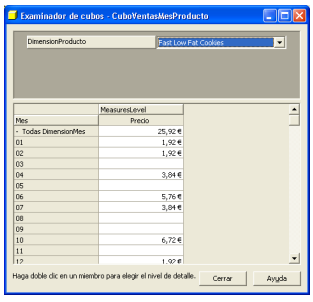
- Pasamos la dimensión de productos al área café y elegimos en el cuadro combinado qué producto queremos ver.
- Háganlo con el producto "Jeffers Corn Puffs".



Un poco de terminología

Panel del rebanador de datos (rebanador)

Panel de visualización de datos



Un poco de terminología

Sirve para rebanar (decir qué rebanada queremos)

Sirve para ver los datos de esa rebanada

Ejercicio

- En el cubo de ventas por tiendas, obtengan las siguientes rebanadas:
- La rebanada de la tienda "Store 1" desglosada por meses.
- La rebanada de todas las tiendas, desglosadas por meses.
- La rebanada del mes de diciembre, desglosado por tiendas.
- La rebanada de todos los meses, desglosado por tiendas.
- Fíjense que las rebanadas también se pueden sacar de todos los meses y tiendas. Es como si la dimensión mes o tienda no existiera.

¿Qué utilidad tiene rebanar?

- Hasta ahora, poca. Todo lo que podíamos ver en una rebanada podíamos verlo en la tabla (cubo) completo.
- A veces, era un poco pesado hacer mucho "scroll" pero no parece tan gran cosa.
- El problema es cuando un cubo tiene más de dos dimensiones.
- Veamos un ejemplo.

Recordemos: teníamos una hipótesis

- Las ventas bajan porque Chepe, Inc. no tiene una buena estrategia navideña.
- Esto parece haberse confirmado cuando hemos desglosado las ventas por días.
- Pero esto puede ser por dos cosas:
  - Los productos que ofrecemos no se venden en Navidad (en ese caso, deberíamos incluir otros productos en nuestra oferta).
  - Las promociones para Navidad no son efectivas o son menos efectivas que la competencia.

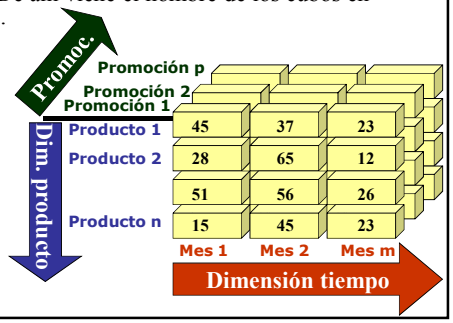
Mail: palasi@palasi.com Web: www.palasi.com

Necesitamos tener un desglose de las ventas por tres dimensiones

- Tiempo (para ver el efecto de la Navidad).
- Producto (para examinar los diferentes productos).
- Promoción (para ver si las promociones son las adecuadas).

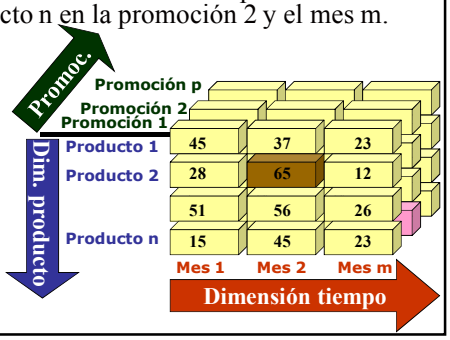
Este desglose de las ventas podría representarse así (simplificado)

- Esta estructura de tres dimensiones se llama un cubo. De ahí viene el nombre de los cubos en OLAP.



Por ejemplo, las ventas del producto 2 con la promoción 1 en el mes 2 están en café

- En rosa se encuentra el comportamiento del producto n en la promoción 2 y el mes m.

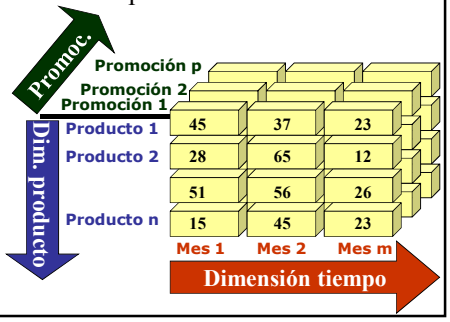


Ahora bien, tenemos un problema

- La pantalla sólo tiene dos dimensiones y el cubo tiene tres.
- Como vemos, la mayoría de datos están tapados por otros.
- ¿Cómo podemos examinar los datos del cubo?

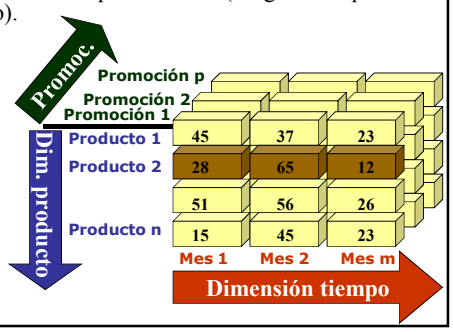
Sencillo: rebanándolo

- No veremos todo el cubo de golpe, sino sólo lo examinaremos por rebanadas.



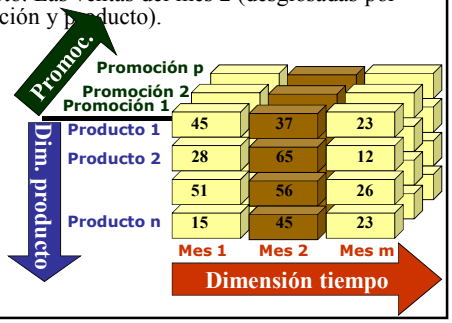
Las rebanadas pueden ser de una única dimensión

- Por ejemplo, la dimensión tiempo. Las ventas del producto 2 en la promoción 1 (desglosadas por tiempo).



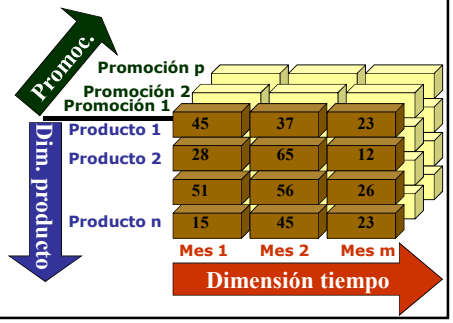
Las rebanadas pueden ser de dos dimensiónes

- Por ejemplo, las dimensiones promoción y producto. Las ventas del mes 2 (desglosadas por promoción y producto).



Las rebanadas pueden ser de dos dimensiónes

- Por ejemplo, tiempo y producto. Las ventas en la promoción 1 (desglosadas por tiempo y producto).



Fijémonos que lo que hacemos al rebanar

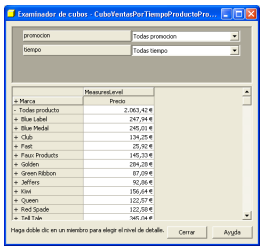
- Es quitar dimensiones.
- El cubo es de tres dimensiones, pero, rebanando hemos obtenido cachitos del cubo con una y dos dimensiones.
- Cuando rebanábamos tablas, éstas eran de dos dimensiones, pero obteníamos cachitos de una dimensión.

Véamoslo en la práctica.

- Creen un cubo con medida "precio" y tres dimensiones
- Una que se llama "tiempo", tendrá los niveles "mes" y "día" (por ese orden de prioridad).
- Otra que se llama "promocion", tendrá el nivel promocion.
- Otra que se llama "producto", tendrá el nivel "marca" y "producto" (por ese orden de prioridad).

Examinen los datos del cubo y encontrarán algo.

- Fíjense que ahora tienen dos dimensiones en el panel del rebanador y una en el panel de visualización.
- Pregunta. ¿Qué rebanada se está mostrando?



Respuesta

- La rebanada de ventas de "todas las promociones" y "todos los tiempos" (desglosada por producto).
- La dimensión es producto.
- Es una rebanada de una dimensión.

Consigan las siguientes rebanadas de una dimensión

- Las ventas para el 5 de marzo en todas las promociones desglosada por marca-producto.
- Las ventas para la promoción 4 en todos los tiempos desglosadas por marca-producto (miren cuánto se vendió del producto "Fast Low Fat Cookies"en la promoción) .
- Las ventas sin promoción para julio y desglosadas por marca-producto.
- Las ventas del producto "Club Low Fat Sour Cream" en todas las promociones desglosadas por tiempo (miren también cuanto se vendió de este producto el 10 de enero).
- Las ventas de la marca "Blue Label" en la promoción 1 y desglosada por tiempo.
- Las ventas de la marca "Faux Products" en diciembre desglosadas por promoción.

Consigan las rebanadas de dos dimensiones

- La promoción 4 desglosada por marca-producto y tiempo (mirar el comportamiento del producto "Kiwi Scallops" en esa promoción 4).
- La marca "Club" desglosada por promoción y tiempo (mirar cómo se comportó esa marca el 10 de enero).
- Las ventas del mes 1 desglosadas por marca-producto y promoción.
- Las ventas del 10 de enero desglosadas por marca-producto y promoción (mirar cómo se comporta el producto "Golden Orange Popsicles").

Ejercicio

- Creen un cubo de tres dimensiones:
- Marca-producto.
- Tiempo.
- Tienda.

- Consigan una rebanada de la tienda 1 y el mes 1 desglosada por producto.
- Otra rebanada del producto Kiwi Scallops desglosado por tiempo y tienda.

Mejorando el cubo

- Ahora queremos hacer un análisis también por tiendas.
- Lo que haremos es añadir una nueva dimensión al cubo llamada "Tienda" que será el campo "tienda".
- Así también podremos hacer desgloses por tienda.

Ahora tenemos 4 dimensiones

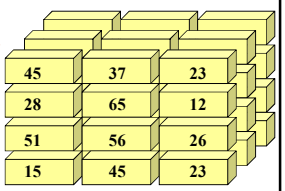
1 dimensión (arreglos) →

1	5	2	9	9	7	2	7	4	5	8	9

2 dimensiones (tablas) →

13	20	20	19
1	2	5	4
51	71	52	78

3 dimensiones (cubos) →



4 dimensiones → ?

Las figuras o estructuras de 4 o más dimensiones se llaman "hipercubos"

- Un hipercubo es a un cubo
  - Lo que un cubo es a un cuadrado.
  - Lo que una esfera es a un círculo.
- Desgraciadamente, aunque podemos definir un hipercubo, no lo podemos representar gráficamente (dibujar) como con los arreglos, tablas y cubos.
- El universo tiene sólo tres dimensiones espaciales (al menos, macroscópicamente) y no existen hipercubos en el espacio físico.

Mail: palasi@palasi.com Web: www.palasi.com

¿Nos vamos a detener sólo porque el Universo tiene sus limitaciones?

- Nosotros somos "muy machos": ni las leyes fundamentales de la Física nos detienen.
- La estructura de datos llamada hipercubo (un cubo OLAP con cuatro dimensiones) existe en nuestro servidor OLAP, aunque no se pueda representar gráficamente (dibujar) en nuestro Universo.
- Podemos tratarla aunque no podemos dibujarla.

¿Cómo tratamos un hipercubo?

- Son muchas dimensiones para tratarlas de un solo.
- Lo que debemos hacer es quitar dimensiones, es decir, rebanar.
- Rebanando obtenemos cachitos del hipercubo con una o dos dimensiones, que sí podemos tratar.

Ejercicio

- Hagan un hipercubo con cuatro dimensiones: marca-producto, mes-día, tienda y promoción.
- Para completar, que tenga tres medidas: ventas, costo y beneficio (ésta última es calculada).
- Obtengan una rebanada de los beneficios de la tienda 4 con todas las promociones desglosados por marca-producto y tiempo.
- (Nota. Deberán poner "Measures" en el rebanador y seleccionar "beneficio")

Pregunta importante

- ¿Qué es preferible?
  - Hacer un cubo con muchas dimensiones.
  - Hacer muchos cubos con pocas dimensiones cada uno.
- ¿Qué piensan?

Es preferible un cubo con muchas dimensiones

- Supongamos
  - un cubo por tiempo y otro por producto
  - contra un cubo por tiempo y producto.
- En el primer caso, hay rebanadas del segundo que no podemos ver (por ejemplo, la rebanada del producto X desglosada por tiempo).
- En el segundo caso, yo puedo obtener todas las rebanadas del primero (que son, ventas por tiempo y ventas por producto).

Es decir, añadir dimensiones añade posibilidades pero no quita ninguna

- Cuando creamos una dimensión,
  - Podemos desglosar los datos por esa dimensión.
  - O bien podemos no desglosarlos, eligiendo en el rebanador la opción "Todo ***NombreDimensión***".
- Es decir, una dimensión siempre se puede desactivar, si nos interesa, eligiendo esa opción.
- Por eso, es mejor muchas dimensiones, porque así tenemos todas las posibilidades.
  - Si queremos considerar una dimensión, lo hacemos.
  - Si no, la desactivamos y ya. Esto es como si no la hubiéramos definido.

Mail: palasi@palasi.com Web: www.palasi.com

6.7. Dimensiones, niveles y vistas

- 6.7.1. Dimensiones de varios niveles.
- 6.7.2. Cubos de varias medidas.
- 6.7.3. Rebanadas.
- 6.7.4. Cubos de varias tablas.
- 6.7.5. Otros aspectos.

Chepe, Inc. ha crecido

- Mr. Chepe quiere ampliar su mercado y no limitarse solamente a los inmigrantes latinos en Estados Unidos.
- Para ello, cambió el nombre del negocio a Foodmart, lo que es más gringo.
- Además, mandó a un experto a revisar la base de datos para ponerla a tono con los nuevos desafíos.

¿Qué tenía de malo la base de datos de Chepe, Inc.?

- Por ejemplo, que no estaba en forma normal.

dia	mes	ano	producto	marca	cliente	promocion	tienda	precio	costo
8	January	1998	Tell Tale Ca	Tell Tale	Sweet	No Promotion	Store 3	19,10 €	6,49 €
9	January	1998	Blue Label	Blue Label	Sweet	No Promotion	Store 3	7,66 €	2,60 €
9	January	1998	Blue Label	Blue Label	Tusting	No Promotion	Store 3	15,32 €	7,51 €
10	January	1998	Golden Ora	Golden	Beltran	No Promotion	Store 7	15,52 €	6,05 €
10	January	1998	Club Low F	Club	Strambi	No Promotion	Store 3	8,79 €	2,99 €
10	January	1998	Green Ribb	Green Ribb	Manning	No Promotion	Store 8	2,62 €	1,21 €

- Fíjense que "marca" depende funcionalmente de "producto". Esto viola la tercera forma normal.
- Además de no ser normal, hay aspectos que están muy simplificados. Por ejemplo, las tiendas deberían estar en una tabla aparte, para poder incluir más información en ellas.

Se ha creado una nueva base de datos llamada "foodmart_2000.mdf"

- Que resuelve esos problemas. Pueden verla en **C:\Archivos de programa\Microsoft Analysis Services\Samples**.
- Desgraciadamente, no se pudo convencer a Mr. Chepe que usara un motor de base de datos diferente de Access. Cuando se le propuso SQL Server dijo "¡Mucho pisto!" (La versión oficial de la compañía fue "We need to cut costs")

Examinen la base de datos "foodmart_2000"

- Vean que todos los nombres están ahora en inglés. Mr. Chepe se ha vuelto muy "fresón".
- La información de ventas está en la tabla "sales_fact_1998". Vean que hay más registros y que muchos campos son claves foráneas (como debe ser).
- También pueden ver las relaciones entre tablas haciendo **Herramientas|Relaciones** dentro de Access.

Ahora queremos programar un cubo como los de antes

- Concretamente un cubo con una dimensión: marca-producto.
- Como los de ChepeInc. ¡Ah, la nostalgia de los buenos viejos tiempos!
- Sin embargo, ahora la información está dispersa en varias tablas, debido a la normalización.
- La solución es programar un cubo con varias tablas.

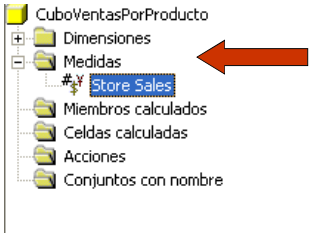
Mail: palasi@palasi.com Web: www.palasi.com

Creen una nueva base de datos en el Analysis Manager

- Pongan como origen de datos "foodmart_2000". Esto ya lo hemos visto.
- Ahora elijan crear un nuevo cubo.

Creando el nuevo cubo

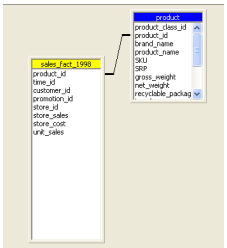
- Como tabla de hechos pongan "sales_fact_1998".
- Como medida, pongan "Store Sales". Esto es lo que antes llamábamos "precio".



Ahora creen una nueva dimensión

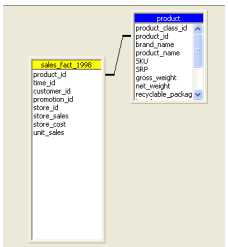
- Queremos crear el análogo de la dimensión "Marca-Producto" que teníamos antes.
- Para ello, creen una dimensión.
- Cuando le pida la tabla de dimensiones, elijan "product".
- De niveles de la dimensión, elijan "brand name" (nombre de marca) y "product name" (nombre de producto).

Fíjense que, cuando acaben la dimensión, aparecen dos tablas



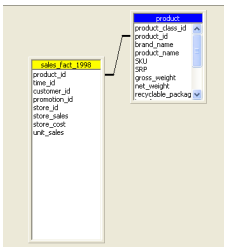
- La tabla amarilla es la tabla de hechos.
- La tabla azul es la tabla de dimensiones. Define la dimensión.
- Lo más importante: Las dos tablas están relacionadas por una relación de clave foránea. Es ésta la que nos permite tratarlas conjuntamente.

¿De dónde sale esa relación?



- En realidad, el Analysis Manager la ha deducido porque los dos campos tienen el mismo nombre y tipos compatibles.
- En todo caso, si no es la que queremos, la podemos cambiar en ese editor gráfico.
- En nuestro caso, está perfecta.

Otro detalle

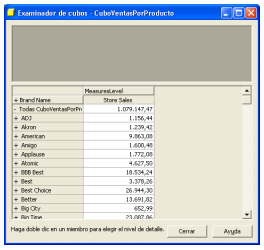


- El Analysis Manager supone por defecto que la tabla de hechos es la tabla que tiene claves foráneas en otras tablas.
- Es decir, en la relación, la tabla de hechos siempre tendrá el extremo de la clave foránea y la otra tabla la de la clave primaria.

Mail: palasi@palasi.com Web: www.palasi.com

Guarden el cubo y examinen datos

- ¿Qué ven?
- Este es el mismo cubo que habíamos definido antes para ChepeInc, pero ahora lo hemos hecho con dos tablas normalizadas relacionadas por clave foránea.
- Con ChepeInc era una sola tabla no normalizada.

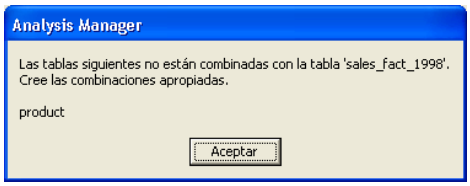


Hagan un experimento

- Borren la relación que existe entre las dos tablas.
- Intenten salir del cubo guardando.
- ¿Qué ven?

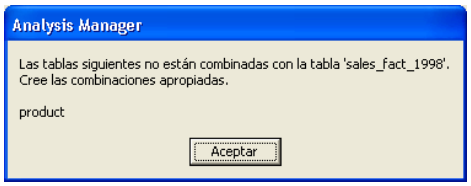
Nos da error

- El sistema sabe qué para hacer una dimensión con una tabla diferente a la de hechos, esta debe estar relacionada con la de hechos.



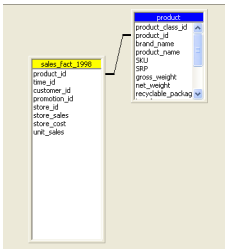
El sistema sabe que

- Necesitamos una relación entre la tabla de hechos y la tabla de dimensiones. Si no la tenemos, debemos crearla.



Creémosla, pues

- Arrastremos el campo "product-id" de la tabla de dimensión ("product") al campo "product-id" de la tabla de hechos ("sales_fact_1998").



Ahora vamos a crear la dimensión de tiempo

- En ChepeInc, la hacíamos con los campos día y mes.
- Sin embargo, aquí aprenderemos de un modo diferente.
- La haremos con un solo campo de fecha.

Abran Foodmart

- Abran la tabla sales_fact_1998. ¿Dónde está la fecha?

product_id	time_id	customer_id	promotion_id	store_id	store_sales	store_cost	unit_sa
304	819	3	0	15	8,40 €	3,70 €	
1056	819	3	0	15	11,72 €	4,22 €	
1418	819	3	0	15	6,44 €	3,16 €	
1063	819	3	0	15	9,54 €	4,58 €	
986	819	3	0	15	10,50 €	3,47 €	
803	819	3	0	15	10,53 €	4,21 €	
672	819	3	0	15	10,02 €	3,31 €	
747	838	3	0	15	5,49 €	2,58 €	

- Créanlo o no está en el campo "time_id". Este es una clave foránea a "time_by_day". Examinen los registros de esta última tabla.

La tabla "time_by_day"

- Cada registro es una fecha, "time_id" es la clave primaria, "the_date" tiene la fecha y otros días tienen otras propiedades de la fecha (día de la semana, del mes, etc.)

time_id	the_date	the_day	the_month	the_year	day_of_month	week_of_yea	month_of_y	
367	01/01/1997	Wednesday	January	1997	1	2	1	Q1
368	02/01/1997	Thursday	January	1997	2	2	1	Q1
369	03/01/1997	Friday	January	1997	3	2	1	Q1
370	04/01/1997	Saturday	January	1997	4	2	1	Q1
371	05/01/1997	Sunday	January	1997	5	3	1	Q1
372	06/01/1997	Monday	January	1997	6	3	1	Q1
373	07/01/1997	Tuesday	January	1997	7	3	1	Q1
374	08/01/1997	Wednesday	January	1997	8	3	1	Q1
375	09/01/1997	Thursday	January	1997	9	3	1	Q1
376	10/01/1997	Friday	January	1997	10	3	1	Q1

Esto es bastante común

- Separar las fechas en una tabla aparte es una práctica común en OLAP y se recomienda como una nueva práctica.

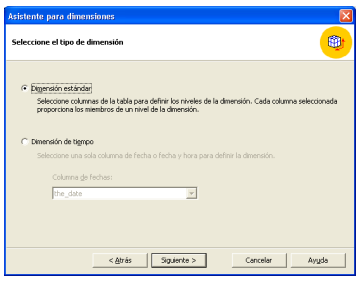
time_id	the_date	the_day	the_month	the_year	day_of_month	week_of_yea	month_of_y	
367	01/01/1997	Wednesday	January	1997	1	2	1	Q1
368	02/01/1997	Thursday	January	1997	2	2	1	Q1
369	03/01/1997	Friday	January	1997	3	2	1	Q1
370	04/01/1997	Saturday	January	1997	4	2	1	Q1
371	05/01/1997	Sunday	January	1997	5	3	1	Q1
372	06/01/1997	Monday	January	1997	6	3	1	Q1
373	07/01/1997	Tuesday	January	1997	7	3	1	Q1
374	08/01/1997	Wednesday	January	1997	8	3	1	Q1
375	09/01/1997	Thursday	January	1997	9	3	1	Q1
376	10/01/1997	Friday	January	1997	10	3	1	Q1

Basta de charla. ¡A trabajar!

- Modifiquen el cubo anterior para añadir la dimensión tiempo.
- Seguiremos los siguientes pasos.
- Crear una nueva dimensión.
- Decir que la tabla escogida para crear la dimensión es "time_by_day".

Sale algo raro

- ¿Qué es eso?



Analysis Manager detecta que el campo "the_date" es de tipo fecha

- Los campos tipo fecha son muy importantes en OLAP porque el tiempo es casi siempre una dimensión importante para analizar los datos.
- Las dimensiones definidas a partir de campos tipo fecha (que se llaman dimensiones temporales), tienen una serie de características particulares.
- Por ejemplo, una dimensión temporal puede definir varios niveles en un solo campo fecha (año, mes, día).

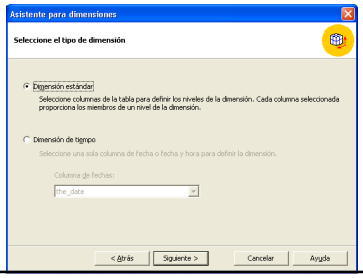
Mail: palasi@palasi.com Web: www.palasi.com

Cuando estamos creando una dimensión a partir de un campo de tipo fecha

- Analysis Manager nos da a elegir si queremos que esa dirección sea considerada
- **Como una dimensión temporal.** Por lo cual se deberán definir las diferentes características de una dimensión temporal.
- **Como una dimensión estándar.** Por lo cual el campo fecha se tratará como si fuera un campo de texto.

Es esto lo que aparece

- Analysis Manager nos pregunta si queremos una dimensión normal o una dimensión temporal, pues ha detectado que en la tabla de dimensiones hay campos de fecha.

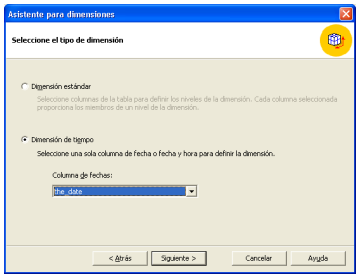


¿Por qué esto no ha aparecido hasta ahora?

- En ChepeInc el tiempo no se representaba por un campo tipo fecha, sino por dos campos numéricos: "mes" y "día".
- Así definíamos una dimensión estándar con dos niveles a partir de los dos campos.
- Ahora es un campo tipo fecha y tenemos que definir los niveles a partir de un único campo.

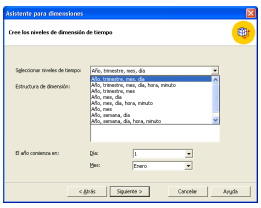
Seleccionen "Dimensión de tiempo" y "the_date" en el cuadro combinado

- Hagan clic en "Siguiente"



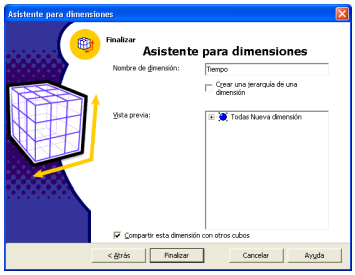
Aparece una pantalla que nos permite elegir los niveles de la dimensión temporal

- En nuestro caso, elegimos "Año, mes, día".



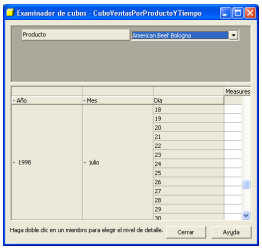
Elegimos un nombre para la dimensión temporal y ya está

- En este caso, llamaremos "Tiempo" a la dimensión temporal.



Examinen los datos

- Obtengan una rebanada con el producto "American Beef Bologna" y vean las ventas de este producto el 23 de julio de 1998.



Como ven, ahora el mes sale por su nombre

- Lo que está muy bien.
- Otra diferencia es que ahora sale el año 1997, que está a cero, pues no hay ventas ese año.

¿Por qué sale 1997?

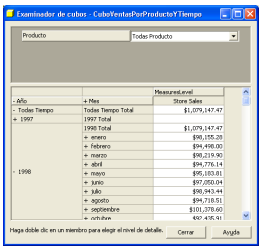
- La tabla "time_by_day" tiene las fechas del 1997, aunque la tabla de hechos, es decir, la de venta ("sales_fact_1998").
- ¿Qué pasaría si no queremos que salga el año 1997?
- Esto es un caso particular de un caso general, que es el de filtrar los datos.

Filtrando los datos

- A veces, queremos que en el cubo aparezcan sólo algunos datos de la tabla y no todos (como, por ejemplo, sólo el 1997).
- A esto se le llama en BD, filtrar los datos.
- ¿Cómo filtramos datos en OLAP?

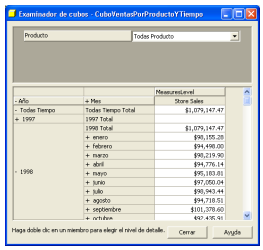
Hay dos maneras

- La fácil: Simplemente colapsamos los datos hasta que sólo ocupen una fila (o columna) del examinador de datos.
- Esto teóricamente no es filtrar, pero una fila no molesta mucho y podemos ignorarla.



Hay dos maneras

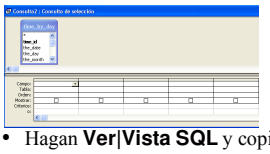
- La menos fácil: Definimos una vista en el almacén de datos que sólo tenga los datos que queremos.
- Definimos el cubo a partir de la vista y no de la tabla original.



Mail: palasi@palasi.com Web: www.palasi.com

En Access las vistas se llaman "consultas"

- Creen una consulta en vista diseño. Añadan la tabla "time_by_day".



- Hagan **Ver|Vista SQL** y copien esto

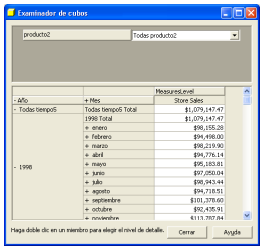
```
SELECT time_by_day.*
  FROM time_by_day
  WHERE (((time_by_day.the_date) Between
#1/1/1998# And #12/31/1998#));
```

Guarden la consulta con un nombre

- Por ejemplo, "Time_Year_1998"
- Modifiquen el cubo para que en vez de la dimensión "Tiempo" definida por "time_by_day", tengamos una dimensión definida por "Time_Year_1998".
- Examinen los datos.

Verán que no sale el año 1997

- Que es lo que queríamos.



¿Cuál es la mejor manera de filtrar los datos?

- Colapsar los datos que no nos interesan en el examinador de datos.
  - Es más fácil.
- Definir una vista en el almacén de datos que filtre los datos.
  - Es más flexible. Por ejemplo, podemos filtrar, por ejemplo, sólo los días hábiles, lo que no se puede hacer con la primera. Podemos filtrar los productos que sean de una clase, que valgan más de un cierto precio, etc.
- Cuando las dos son aplicables, es preferible la primera.
- Pero hay veces que sólo la segunda es aplicable.

Ejercicio

- Hagan un hipercubo con cuatro dimensiones: marca-producto, fecha, tienda y promoción.
- Para completar, que tenga tres medidas: ventas, costo y beneficio (ésta última es calculada).
- Sólo nos interesan las tiendas de la 1 a la 5, que son las de Estados Unidos (las otras son franquicias internacionales).
- Obtengan una rebanada de los beneficios de la tienda 4 con todas las promociones desglosadas por marca-producto y tiempo.
- (Nota. Deberán poner "Measures" en el rebanador y seleccionar "beneficio")

Consigan siguientes rebanadas. Recuerden que sólo interesan las tiendas de la 1 a la 5

- Las ventas para el 5 de marzo de 1998 en todas las promociones desglosada por marca-producto (para las tiendas 1 al 5).
- Las ventas de la marca "Blue Label" en la promoción "Bag Staffers" y desglosada por tiempo.
- La marca "Club" desglosada por promoción y tiempo (mirar cómo se comportó esa marca el 10 de enero).
- Las ventas del mes 1 desglosadas por marca-producto y promoción.

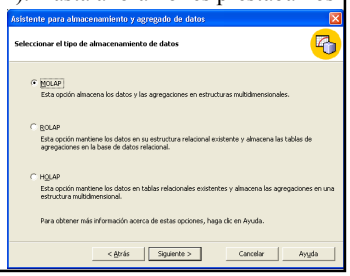
Mail: palasi@palasi.com Web: www.palasi.com

6.7. Dimensiones, niveles y vistas

- 6.7.1. Dimensiones de varios niveles.
- 6.7.2. Cubos de varias medidas.
- 6.7.3. Rebanadas.
- 6.7.4. Cubos de varias tablas.
- 6.7.5. Otros aspectos.

Opciones de almacenamiento

- ¿Dónde almacena Microsoft Analysis Services los datos de los diferentes cubos?
- Hay varias opciones que se pueden dar (MOLAP, ROLAP y HOLAP). Hasta ahora no les prestábamos atención.



Todos los datos de 1 cubo vienen del almacén de datos directamente o por cálculo

- Se pueden calcular a partir del almacén de datos cada vez que examinamos el cubo.
- Sin embargo, esto aunque es posible, no es práctico.
- Hay datos que es mejor tenerlos calculados ya, pues hacerlo cada vez que examinamos el cubo sería muy lento.
- Así, por ejemplo, las ventas totales o las ventas por mes.

Analysis Services tiene el poder en esto

- Analysis Services decide cuáles datos:
  - Calculamos a partir del almacén de datos cada vez que examinamos el cubo.
  - Qué datos se calculan antes y se guarda una copia para usarla cuando se examina el cubo.

Veamos un cubo

- Creemos una clave primaria autonumérica en ChepeInc si no existe.

Nombre del campo	Tipo de datos
clave	Autonumérico
dia	Texto
mes	Texto

- A partir de ChepeInc, creen un cubo de ventas con una única dimensión con los niveles mes, día y clave. Llámenlo "CuboClave1".
- Examinen el cubo y expandan el 9 de enero.

Hay tres tipos de datos en este cubo

- 1. En verde están los datos que estaban tal cual en el almacén de datos (compruébenlo). A estos datos se les llama **datos de origen.** Corresponden al nivel de clave, es decir, desglose por registros individuales.

- Mes	- Dia	Clave	Precio
- Todas DimensionClave	Todas DimensionClave Total		$2,063.42
	01 Total		$131.75
	- 09	09 Total	$42.08
		1	$19.10
		2	$7.66
		3	$15.32
- 01	+ 10	10 Total	$30.27
	+ 12	12 Total	$31.92
	+ 13	13 Total	$5.24
	+ 20	20 Total	$5.60
	+ 22	22 Total	$13.88
	+ 23	23 Total	$2.76
+ 02	02 Total		$205.28
+ 03	03 Total		$205.06

Hay tres tipos de datos en este cubo

- 2. En amarillo, están los datos que son fácilmente calculables a partir del almacén de datos. Se trata de los resúmenes por día. No hay que hacer muchas sumas para calcularlos.

- Mes	- Dia	Clave	Precio
- Todas DimensionClave	Todas DimensionClave Total		$2,063.42
	01 Total		$131.75
	- 09	09 Total	$42.08
		1	$19.10
		2	$7.66
		3	$15.32
- 01	+ 10	10 Total	$30.27
	+ 12	12 Total	$31.92
	+ 13	13 Total	$5.24
	+ 20	20 Total	$5.60
	+ 22	22 Total	$13.88
	+ 23	23 Total	$2.76
+ 02	02 Total		$205.28
+ 03	03 Total		$205.06

Hay tres tipos de datos en este cubo

- 3. En rojo están los datos que sería lentos de calcular a partir de los datos de origen. Se les llama **agregados**. En este caso, están los resúmenes por mes y la suma total.

- Mes	- Dia	Clave	Precio
- Todas DimensionClave	Todas DimensionClave Total		$2,063.42
	01 Total		$131.75
	- 09	09 Total	$42.08
		1	$19.10
		2	$7.66
		3	$15.32
- 01	+ 10	10 Total	$30.27
	+ 12	12 Total	$31.92
	+ 13	13 Total	$5.24
	+ 20	20 Total	$5.60
	+ 22	22 Total	$13.88
	+ 23	23 Total	$2.76
+ 02	02 Total		$205.28
+ 03	03 Total		$205.06

En cualquier cubo sólo hay tres tipos de datos

- Los **datos de origen**, que ya estaban en el almacén de datos (la tabla de datos original).
- Los datos fácilmente calculables, que son fácilmente calculables a partir de los datos de origen.
- Los **agregados**, que sería muy lento de calcular a partir de los datos de origen.

Analysis Services sigue esta política

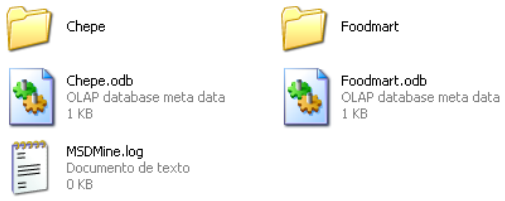
- Sólo almacenan los datos de origen y los agregados.
- Los datos fácilmente calculables se calculan a partir de los datos de origen cada vez que se examina el cubo.
- Así se optimiza el tiempo y el espacio.

Hay dos sitios donde almacenar los datos de origen y los agregados

- En el mismo almacén de datos (base de datos original).
- En el subdirectorio Data de la instalación de Microsoft Analysis Services.

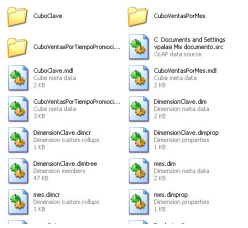
Entren en ese subdirectorio

- Verán un subdirectorio por cada uno de las bases de datos que han definido en el Analysis Manager.



Entren al subdirectorio de la base de datos ChepeInc

- Verán archivos y subdirectorios para cada uno de los cubos que hemos definido en esta base de datos.
- Y también para las dimensiones definidas.



Hay tres opciones de almacenamiento

- **1. MOLAP** (**M**ultidimensional **OLAP**). Tanto los datos de origen como los agregados se almacenan en el subdirectorio Data.
- Esto implica que
  - los datos de origen deben copiarse desde la base de datos hasta este directorio. Hay 2 copias.
  - los agregados se calculan y se copian al directorio.
- Los cubos con esa opción de almacenamiento ocupan mucho espacio pero son rápidos.
- Adecuado para cubos utilizados frecuentemente.

Hay tres opciones de almacenamiento

- **2. ROLAP** (**R**elational **OLAP**). Tanto los datos de origen como los agregados se almacenan en el almacén de datos.
- Esto implica que
  - Sólo hay una copia de los datos de origen.
  - los agregados se calculan y se copian al directorio.
- Los cubos con esa opción de almacenamiento ocupan poco espacio pero son lentos.
- Adecuado para cubos utilizados poco frecuentemente.

Hay tres posibilidades

- **3. HOLAP** (**H**ybrid **OLAP**). Los datos de origen se almacenan en el almacén de datos y los agregados en el subdirectorio Data.
- Esto implica que
  - Sólo hay una copia de los datos de origen.
  - los agregados se calculan y se copian al directorio.
- Los cubos con esa opción de almacenamiento ocupan un espacio mediano y son de una rapidez mediano.
- Adecuado para cubos utilizados con una frecuencia mediana.

Resumiendo

- Almacenamiento de los cubos.

	Datos de origen	Agregados
MOLAP	*Directorio Data*	*Directorio Data*
ROLAP	Almacén de datos	Almacén de datos
HOLAP	Almacén de datos	*Directorio Data.*

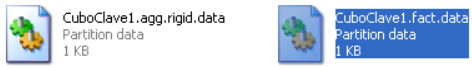
Viendo las diferencias

- El CuboClave1 está creado con MOLAP.
- Creemos CuboClave2 y CuboClave3 iguales que el anterior, pero almacenados con ROLAP y HOLAP respectivamente.

Mail: palasi@palasi.com Web: www.palasi.com

Veamos en el directorio Data el subsubdirectorio CuboClave1

- Se encuentran dos archivos. Uno fact.data, contiene los datos de origen. Otro agg.rigid.data, contiene los datos de la agregación.



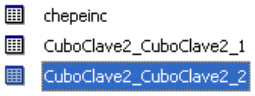
- Los dos tienen datos, pues su tamaño es mayor que cero.

	Datos de origen	Agregados
MOLAP	*Directorio Data*	*Directorio Data*

Veamos en el directorio Data el subsubdirectorio CuboClave2

- No hay esos archivos. Los datos no están en el directorio Data.

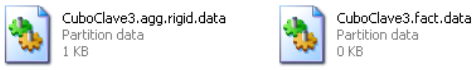
- Pero si abrimos ChepeInc, allá veremos los datos de origen (en la tabla de hechos) y las agregaciones en unas tablas que se han creado.



	Datos de origen	Agregados
ROLAP	Almacén de datos	Almacén de datos

Veamos en el directorio Data el subsubdirectorio CuboClave3

- Sólo hay los datos de agregaciones. El archivo fact.data tiene 0 bytes.

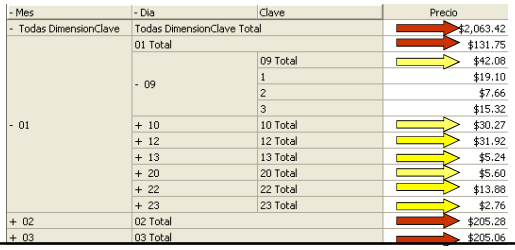


- Pero si abrimos ChepeInc, allá veremos los datos de origen (en la tabla de hechos).

	Datos de origen	Agregados
HOLAP	Almacén de datos	*Directorio Data.*

Otro tema: Veamos los tipos de datos.

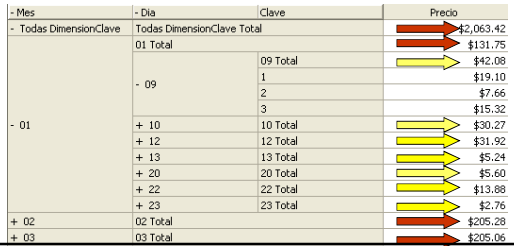
- Ahora no nos fijaremos en los datos de origen, sino en los otros, los que tienen flecha.



- Mes	- Dia	Clave	Precio
- Todas DimensionClave	Todas DimensionClave Total		$2,063.42
	01 Total		$131.75
	- 09	09 Total	$42.08
		1	$19.10
		2	$7.66
		3	$15.32
- 01	+ 10	10 Total	$30.27
	+ 12	12 Total	$31.92
	+ 13	13 Total	$5.24
	+ 20	20 Total	$5.60
	+ 22	22 Total	$13.88
	+ 23	23 Total	$2.76
+ 02	02 Total		$205.28
+ 03	03 Total		$205.06

Otro tema: ¿Cuántos datos se guardan como agregados?

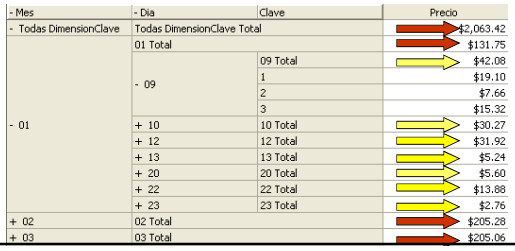
- Hemos visto que los agregados se guardan en una copia que sólo se lee cada vez que se examina el cubo. En rojo.



- Mes	- Dia	Clave	Precio
- Todas DimensionClave	Todas DimensionClave Total		$2,063.42
	01 Total		$131.75
	- 09	09 Total	$42.08
		1	$19.10
		2	$7.66
		3	$15.32
- 01	+ 10	10 Total	$30.27
	+ 12	12 Total	$31.92
	+ 13	13 Total	$5.24
	+ 20	20 Total	$5.60
	+ 22	22 Total	$13.88
	+ 23	23 Total	$2.76
+ 02	02 Total		$205.28
+ 03	03 Total		$205.06

Otro tema: ¿Cuántos datos se guardan como agregados?

- Hemos visto que hay unos datos (los llamábamos datos fácilmente calculables) deben ser calculados cada vez que se examina el cubo. En amarillo.



- Mes	- Dia	Clave	Precio
- Todas DimensionClave	Todas DimensionClave Total		$2,063.42
	01 Total		$131.75
	- 09	09 Total	$42.08
		1	$19.10
		2	$7.66
		3	$15.32
- 01	+ 10	10 Total	$30.27
	+ 12	12 Total	$31.92
	+ 13	13 Total	$5.24
	+ 20	20 Total	$5.60
	+ 22	22 Total	$13.88
	+ 23	23 Total	$2.76
+ 02	02 Total		$205.28
+ 03	03 Total		$205.06

¿Qué datos deben ser guardados como agregados y cuáles como calculables?

- Esta pregunta tiene implicaciones sobre el espacio y el rendimiento de nuestro rendimiento.

- Mes	- Dia	Clave	Precio
- Todas DimensionClave	Todas DimensionClave Total		$2,063.42
	01 Total		$131.75
	- 09	09 Total	$42.08
		1	$19.10
		2	$7.66
		3	$15.32
- 01	+ 10	10 Total	$30.27
	+ 12	12 Total	$31.92
	+ 13	13 Total	$5.24
	+ 20	20 Total	$5.60
	+ 22	22 Total	$13.88
	+ 23	23 Total	$2.76
+ 02	02 Total		$205.28
+ 03	03 Total		$205.06

Un extremo (100% rendimiento).

- Todos los datos que no son de origen se guardan como agregados. Esto es el máximo espacio y máximo rendimiento, pues no debe calcularse nada cuando se examina.

- Mes	- Dia	Clave	Precio
- Todas DimensionClave	Todas DimensionClave Total		$2,063.42
	01 Total		$131.75
	- 09	09 Total	$42.08
		1	$19.10
		2	$7.66
		3	$15.32
- 01	+ 10	10 Total	$30.27
	+ 12	12 Total	$31.92
	+ 13	13 Total	$5.24
	+ 20	20 Total	$5.60
	+ 22	22 Total	$13.88
	+ 23	23 Total	$2.76
+ 02	02 Total		$205.28
+ 03	03 Total		$205.06

Otro extremo (0% rendimiento).

- Ningún dato de entre los que no sean de origen se guarda como agregación. Todos son calculados cuando el cubo se examina. Esto es mínimo espacio (0) y mínimo rendimiento.

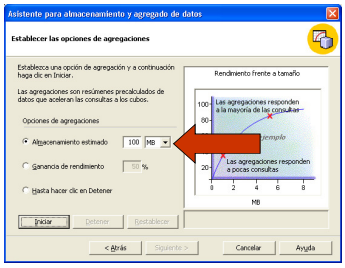
- Mes	- Dia	Clave	Precio
- Todas DimensionClave	Todas DimensionClave Total		$2,063.42
	01 Total		$131.75
	- 09	09 Total	$42.08
		1	$19.10
		2	$7.66
		3	$15.32
- 01	+ 10	10 Total	$30.27
	+ 12	12 Total	$31.92
	+ 13	13 Total	$5.24
	+ 20	20 Total	$5.60
	+ 22	22 Total	$13.88
	+ 23	23 Total	$2.76
+ 02	02 Total		$205.28
+ 03	03 Total		$205.06

Hay muchas opciones intermedias

- En el que parte de los datos no de origen son agregados y parte se calculan. Usan un espacio y rendimiento intermedio entre los dos extremos.

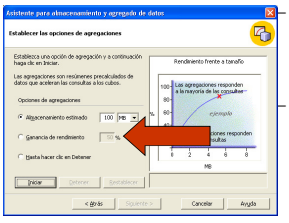
- Mes	- Dia	Clave	Precio
- Todas DimensionClave	Todas DimensionClave Total		$2,063.42
	01 Total		$131.75
	- 09	09 Total	$42.08
		1	$19.10
		2	$7.66
		3	$15.32
- 01	+ 10	10 Total	$30.27
	+ 12	12 Total	$31.92
	+ 13	13 Total	$5.24
	+ 20	20 Total	$5.60
	+ 22	22 Total	$13.88
	+ 23	23 Total	$2.76
+ 02	02 Total		$205.28
+ 03	03 Total		$205.06

Esto se especifica en la siguiente ventana.



- Podemos decir qué tamaño ocupan en disco en las agregaciones.
- Cuanto más tamaño, más agregaciones y más rápido.

Esto se especifica en la siguiente ventana.

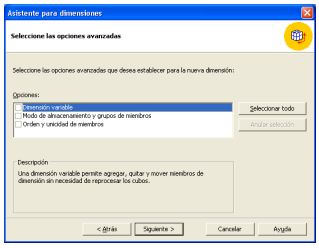


- Podemos decir qué ganancia de rendimiento
  - 0% es que no hay agregaciones (mínimo rendimiento) Tmax
  - 100% es que todos los datos no de origen son agregaciones (máximo rendimiento). Tmin
  - Un porcentaje intermedio indica el rendimiento. T=Tmax-Porcentaje/100(Tmax-Tmin)

Mail: palasi@palasi.com Web: www.palasi.com

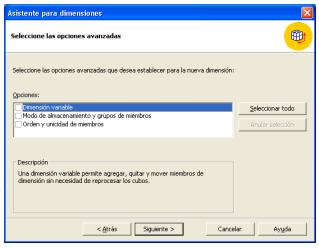
Opciones (raras) para dimensiones

- Dimensión variable es la que permite agregar, quitar y modificar los valores de una dimensión sin necesidad de reprocesar el cubo.
- Esto puede ser bueno si los datos de la dimensión varían frecuentemente, pero esto no es frecuente, pues el almacén de datos suelen ser datos pasados.



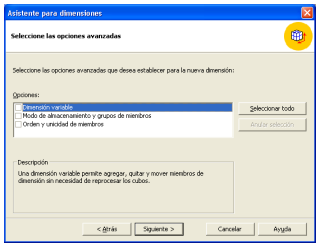
Opciones (raras) para dimensiones

- Modo de almacenamiento. Las dimensiones, como los cubos, pueden almacenarse en Analysis Services (MOLAP) o en el almacén de datos (ROLAP).
- Lo usual es lo primero. Sin embargo, para dimensiones con millones de valores, es más conveniente la segunda.

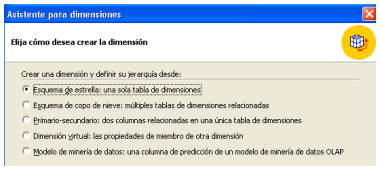


Opciones (raras) para dimensiones

- Orden y unicidad de los miembros. Con esta opción, se puede definir un orden diferente para los valores de un nivel de la dimensión. También se puede asegurar que estos sean únicos en el nivel o la dimensión.

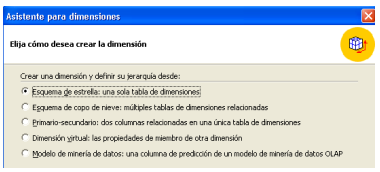


Opciones para la dimensión



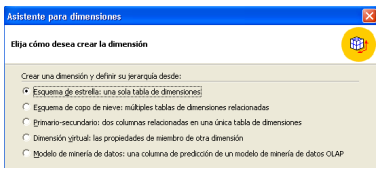
- Esquema de estrella. Los campos que definen la dimensión están en una misma tabla. Es el que hemos visto hasta ahora.

Opciones para la dimensión



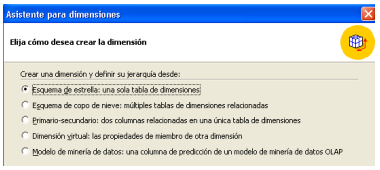
- Esquema de copo de nieve. Los campos que definen la dimensión están en tablas relacionadas por claves foráneas.
- Por ejemplo, una dimensión marca-producto, donde marca está en una tabla y producto en otra con clave foránea a la tabla de producto.

Opciones para la dimensión



- Primario-Secundario. Parecido al anterior, pero la clave foránea liga una tabla con ella misma.
- Por ejemplo, una dimensión "sueldo-sueldo jefe" en la base datos "planilla", donde el jefe es un campo de la tabla empleados que es clave foránea a la misma tabla.

Opciones para la dimensión



- Las otras opciones no las vamos a ver.
- Dimensión virtual es rara y avanzada.
- La otra es sobre minería de datos.

Exportar la información de los cubos a Excel

- Primero, se debe aprender cómo conectarse desde Excel al cubo y volcar su contenido en una tabla dinámica ("Pivot Table", algo parecido al "Examinador de cubos"). Esto se ve en:
  - http://www.databasejournal.com/features/mssql/article.php/10894_2175521_1
- Pero hay muchas más posibilidades: hacer gráficas a partir de esto, crear páginas web interactivas, etc. Esto se ve en
  - http://www.microsoft.com/latam/technet/articulos/200307/art05/ y hacer clic en "Demostración 3" (hay un error, pues la explicación de lo que se va a hacer aparece como Demostración 2).

MDX (MultiDimensional eXpressions)

- Igual que SQL es un lenguaje para la consulta de base de datos relacionales, MDX es un lenguaje para la consulta de bases de datos multidimensionales.
- Lo que SQL es para OLTP, MDX es para OLAP.
- SQL permite extraer conjuntos de registros de una tabla.
- MDX permite extraer estructuras multidimensionales de un cubo.
- Hay una completa serie de artículos sobre MDX en:
- http://www.databasejournal.com/news/article.php/1550061