

Modelo Relacional

martes, 6 de julio de 2021 04:52 p. m.

Introducción - Historia

En los años 70's Edgar Codd define:

- **Modelo Relacional**
- **Álgebra Relacional**

Estos dos conceptos son utilizados por los motores de bases de datos relacionales

- **RDBMS: *Relational Data base Management Systems.***

Luego se crean las primeras **RDBMS** (Oracle, Informix, IBM BD2, etc.) y el lenguaje SEQUEL (**SQL**).

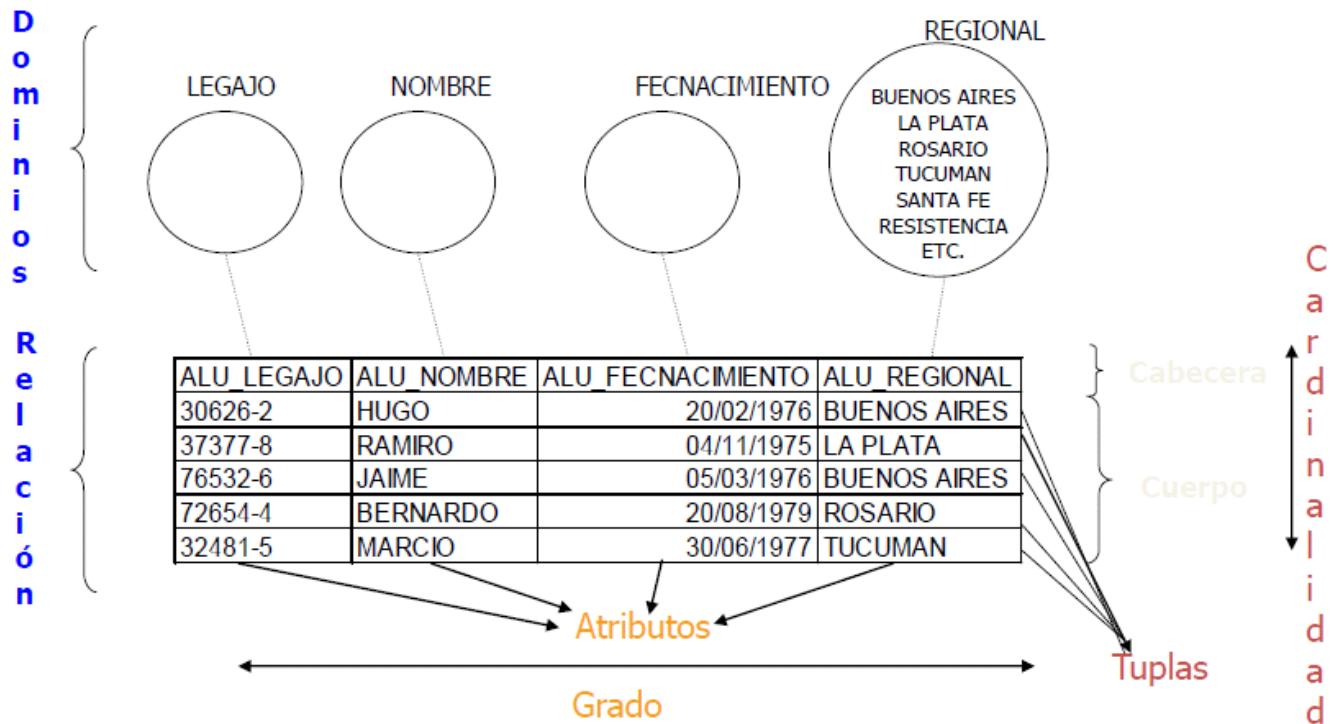
Modelo Relacional para una **base de datos** definido por Edgar Codd compuesto por:

- **Estructura:** Recopilación o conjunto de objetos y relaciones.
- **Manipulación de datos:** Definir operaciones sobre las relaciones para manipular los datos, como podría ser consultarlos, agruparlos, sumarlos, etc.
- **Integridad:** definir reglas para asegurar precisión y consistencia de los datos.

Estructura

martes, 6 de julio de 2021 05:59 p. m.

Edgar Codd definió las **relaciones** como una **estructura bidimensional** tabulada con una **cabecera** y un **cuerpo**, la **cabecera** está formada por un conjunto de **atributos** como "Legajo" y "Nombre", por otro lado el **cuerpo** es un **conjunto de N tuplas** compuesta por **M valores** como **cantidad de atributos** tenga la relación.



Cada relación tiene un nombre como por ejemplo en este caso podría ser "Alumnos".

- **Relación:** Tabla.
- **Tupla de la relación:** Fila o registro.
- **Cardinalidad:** Número de filas.
- **Atributo:** Columna o campo.
- **Grado:** Número de columnas.
- **Clave primaria:** Identificador único o clave primaria también.
- **Dominio:** valores posibles.

Dominio

martes, 6 de julio de 2021 06:00 p. m.

Dominio

- **Conjunto de valores posibles** de los cuales los **atributos** pueden **obtener sus valores reales**.
 - **Menor unidad semántica** de información, no son divisibles.
 - **Atómicos** (no se pueden descomponer sin perder significado).
 - Ej: no tiene sentido separar un DNI en sus dígitos.
 - Conjunto de **valores escalares** (un solo valor posible) de **igual tipo**.
 - Los dominios **NO contienen nulos**.
 - Todo **atributo** tiene un **dominio** definido.

Por ejemplo:

- El **dominio** del atributo **nombre** es un **conjunto de nombres** existentes que existirá en una lista, el **dominio** del atributo **fecha de nacimiento** será una **fecha valida menor que la fecha de ingreso a la facultad**, el atributo **regional** tomara valores de un **listado existente de las regionales** de la facultad. Si existiera un **atributo booleano** su **dominio** seria "si" o "no".

Implementación de Dominio en un DBMS

Las bases de datos implementan los dominios de distintas maneras, pueden implementar por ejemplo las siguientes herramientas:

- **Restricciones de Chequeo.**
 - Restringen los valores posibles dentro del tipo de dato
 - NULL / NOT NULL
 - DEFAULT
 - CHECK
- **Claves Foráneas.**
 - Utilizadas para indicar que un atributo referencia a un registro de otra tabla.
 - *FOREIGN KEY (provincia) REFERENCES state (cod_state)*
- **Definición de "tipos de dato de usuario".**
 - Ej: COD_PROVINCIA = numérico entero de dos posiciones entre 1 y 24.
- **Definición de convenciones de nombre para los atributos.**
 - Recomendable establecer naming convention para atributos definidos sobre el mismo dominio, en toda la BD se llama al mismo atributo con el mismo nombre.
- Al realizar una **consultas** sobre los **dominios** de los **atributos** solo **se realizan chequeos sintáticos y no semánticos**, por eso mismo la siguiente consulta se podría realizar sin problemas aunque no tenga sentido comparar el código de una materia con el código de un

alumno.

- El **chequeo** es **realizado** sobre los **tipos de datos comparados**, no se podrían comparar números con fechas por ejemplo.

```
SELECT ALUMNO.NOMBRE  
      FROM ALUMNO, CURSO, MATERIA  
     WHERE CURSO.ALU_LEGATO = ALUMNO.ALU_LEGATO  
       AND CURSO.MAT_CODIGO = MATERIA.MAT_CODIGO  
       AND MATERIA.MAT_CODIGO = '85-1346';
```

➡ Consulta bien formulada

```
SELECT ALUMNO.*  
      FROM ALUMNO, CURSO  
     WHERE CURSO.MAT_CODIGO = ALUMNO.ALU_LEGATO  
       AND CURSO.COD_CURSO = '3052';
```

➡ Consulta mal formulada
(aunque sin errores de sintaxis)

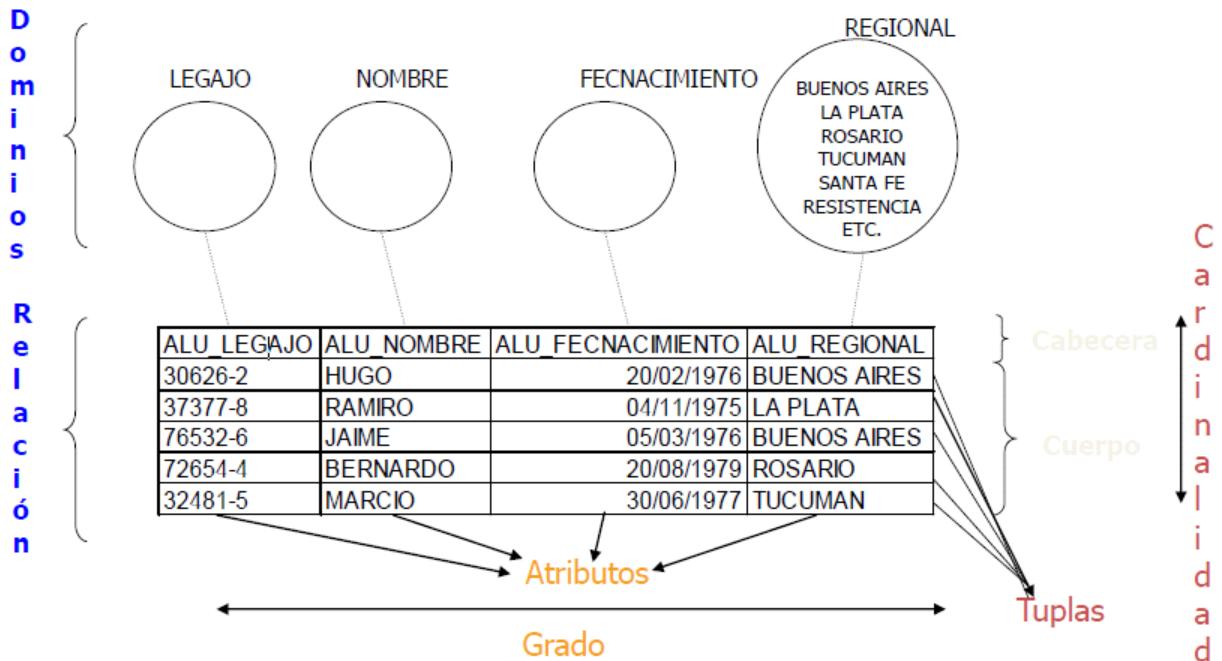
Relaciones

martes, 6 de julio de 2021 06:15 p. m.

Una **Relación** es un "subconjunto del producto cartesiano de los dominios de valores involucrados".

Una **Relación** se define sobre un **conjunto de dominios** D_1, D_2, \dots, D_n y se compone de una **cabecera** y un **cuerpo**.

- **Cabecera:** conjunto fijo de **pares atributo-dominio** (cada atributo con su respectivo dominio)
- **Cabecera** = $\{(A_1 : D_1), (A_2 : D_2), \dots, (A_n : D_n)\}$
- **Cuerpo:** conjunto de **tuplas** que varía con el **tiempo**.
 - **Tupla:** conjunto de **pares atributo-valor**.
 - **Tupla (i)** = $\{ (A_1 : V_{i1}) \dots (A_n : V_{in}) \}$ con $1 \geq i \geq m$
 - m = cardinalidad de la relación.
 - n = grado de la relación.



Propiedades de una Relación

- **No hay tuplas repetidas.**
 - Todas tienen **clave primaria** (atributo o conjunto de atributos que definen únicamente a una tupla).
- Las **tuplas no están ordenadas**.
- Los **atributos no están ordenados**.
- Los **valores de los atributos son atómicos**.

Tipos de Relaciones

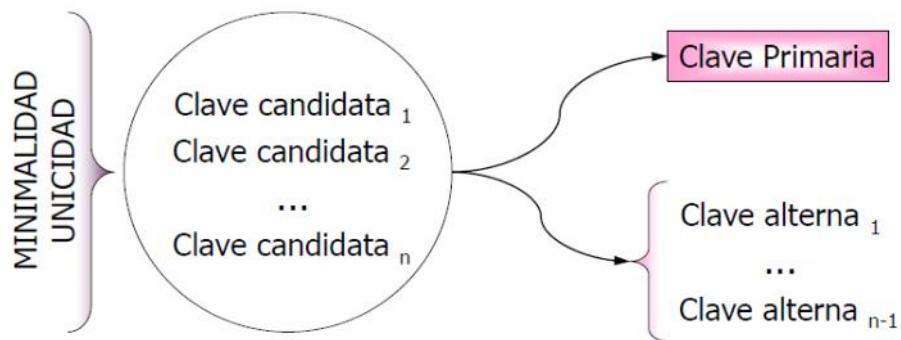
* Como se implementan las Relaciones definidas por Edgar Codd en una base de datos relacional:

- **Tablas** (relaciones Base).
- **Resultados de consultas.**
- **Resultados intermedios de consultas.**
- **Vistas.**
- **Snapshots.**
- **Tablas temporales.**

Claves CK - PK - FK

martes, 6 de julio de 2021 06:35 p. m.

Claves



Clave Candidata

- **Atributo (o conjunto de atributos)** que puede **identificar únicamente una tupla**.
- En el ejemplo podría ser el Legajo, TipoDoc+NroDoc o NroCuil.



Clave Primaria

- **Atributo (o conjunto de atributos) identificador único para la Relación.**
- **Identifican únicamente a cada tupla.**
- **Características:**
 - **Minimalidad:** menor combinación de atributos posible.
 - **Unidad:** no debe repetirse.
- La Clave Primaria de la Relación se selecciona de las Claves Candidatas.

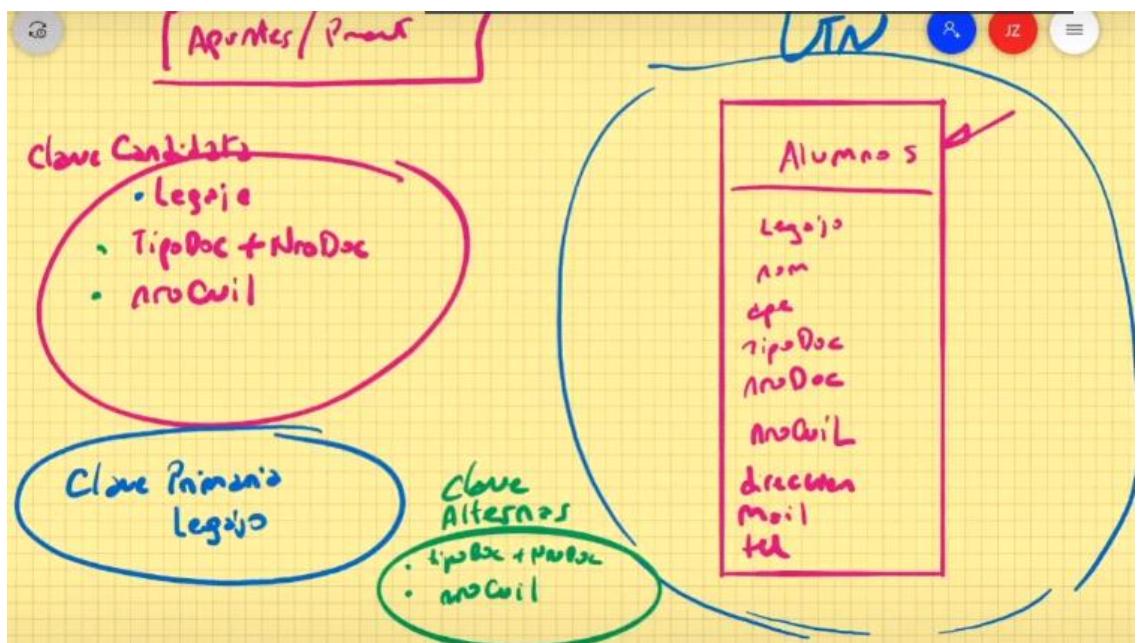
Obtención:

- 1. Obtengo **claves candidatas** que cumplan con **minimalidad y unicidad (CK = Candidate Keys)**

- 2. De las CK elijo una **clave primaria (PK)**
 - nota: cualquiera podría ser pero la elijo en base al contexto cual es más conveniente.
- 3. El resto representan de las CK no elegidas como PK son **claves alternas**

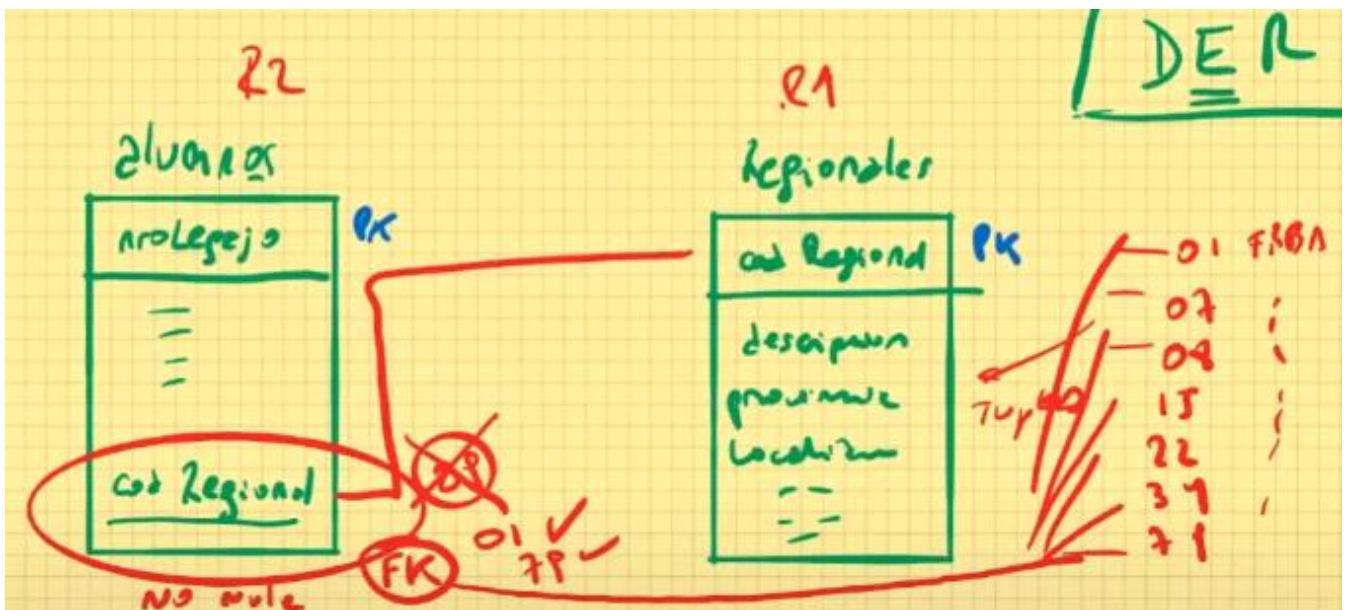
Ej:

- Tabla Alumnos
 - Claves candidatas: legajo, doc + tipo, cuil, mail
 - Clave primaria: en utn, es el legajo.



Clave Foránea

- **Atributo (o conjunto de atributos) de una relación (R2) cuyos valores (no nulos) deben coincidir con los de la clave primaria de una relación (R1).**
- Utilizada para **asociar entidades**.
- Propiedades:
 - La **clave primaria en R1 y foránea en R2** están definidas sobre el **mismo dominio**.
 - La **clave foránea** puede o no ser parte de la **clave primaria de R2**.
 - **R1 y R2 no necesariamente son distintas**.
 - Si $R1 = R2$ existe una **relación autorreferencial**.
 - Las **claves foráneas** deben en ciertas ocasiones aceptar **nulos**.



Reglas de Integridad (Codd)

martes, 6 de julio de 2021 07:10 p. m.

- **Reglas para asegurar la integridad de los datos y relacionar entidades.**
 - Ej: si tengo un alumno con 10 notas, no puedo borrar el alumno porque quedarían notas sueltas.
- **Reglas de integridad** definidas por Edgar Codd:
 - **Regla de integridad de las entidades**
 - **Regla de integridad referencial**

Regla de integridad de las entidades

- **No se permiten claves primarias (PK) nulas.**
- **Ningún componente de la clave primaria de una relación base puede aceptar nulos.**
 - Una **tupla** es un **elemento de una relación**, que representa al mundo real.
 - Las **claves primarias** son el **mecanismo de identificación** en el **modelo relacional**.
 - Una **base de datos relacional** no admite registrar información acerca de algo que no se puede identificar.

Regla de integridad referencial

- **La base de datos no debe contener valores no nulos de clave foránea para los cuales no existe un valor concordante de clave primaria en la relación referenciada.**
- Mantener un **estado consistente** de la **base de datos**.
- Determinar **acciones** a llevar a cabo **ante operaciones que puedan violar la integridad referencial**:
 - **Eliminación:** RESTRICT / CASCADE / SET NULL
 - **Modificación:** RESTRICT / CASCADE / SET NULL
 - **Inserción:** RESTRICT

A definir:

1. **¿Clave foránea acepta nulos o no?**

2. **¿Qué hacer si se intenta eliminar la clave primaria de un registro referenciado?**

- **RESTRICT:** no se permite la eliminación del registro “padre” referenciado.
- **CASCADE:** se eliminan también los registros que la referencian.
- **ANULACIÓN:** se le asigna nulo a todas las claves foráneas que referenciaban al registro padre (la clave foránea debe permitir nulos).

3. **¿Qué hacer si se intenta modificar la clave primaria de un registro referenciado?**

- **RESTRICT**: no se permite la modificación del registro “padre” referenciado.
- **CASCADE**: se modifican también las claves foráneas que la referencian.
- **ANULACIÓN**: se le asigna nulo a todas las claves foráneas que referenciaban al registro padre (la clave foránea debe permitir nulos)

4. ¿Qué hacer si se intenta insertar un registro con clave foránea que quiera referenciar una clave primaria inexistente?

- **RESTRICT**: no se permite la insertar el nuevo registro ya que no existe el “padre” con la PK especificada.

Independencia de Datos (Codd)

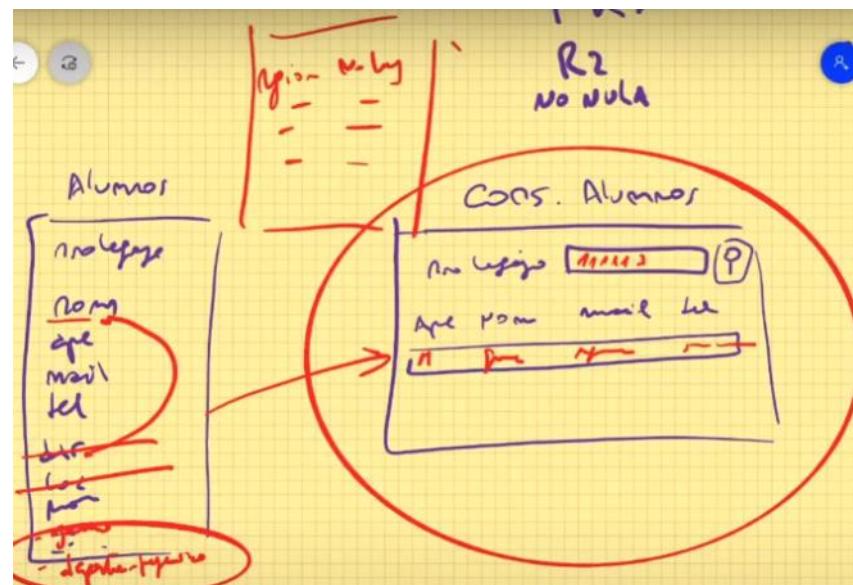
martes, 6 de julio de 2021 07:28 p. m.

En el pasado los programas estaban 100% acoplados lógica y físicamente a los archivos donde se almacenaba la información, como por ejemplo con el lenguaje COBOL.

En cambio, **Edgar Codd** estableció que las **bases de datos relacionales** debían ser **lógica y físicamente independientes** de los archivos que almacenan su información.

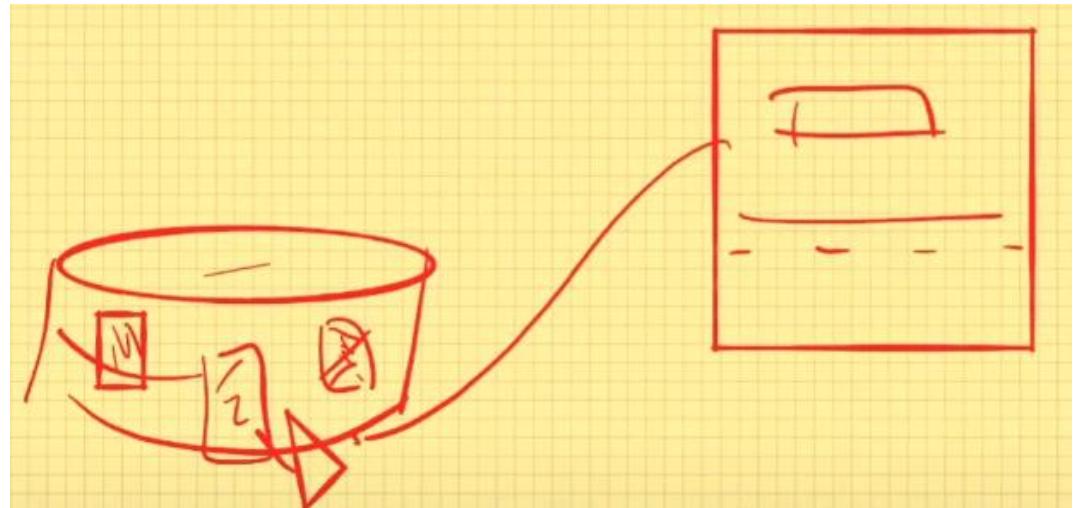
- **Independencia lógica**

- Pueden existir **aplicaciones** que **requieran una vista diferente de los mismos datos** y eso **no requiere saber todos los atributos de una tabla** o el **orden real** de dichos atributos, ni la **distribución de atributos** a través de las tablas.
- Se **podrían hacer cambios** en tablas **sin que afecten a las distintas aplicaciones** que no los requieren.



- **Independencia física**

- Es **posible modificar la estructura de almacenamiento, la distribución física o la técnica de acceso sin afectar las aplicaciones.**
- Por ejemplo:
 - Crear un **índice** por nombre para que se resuelva el listado más rápidamente
 - **Repartir** la información almacenada en **distintos discos**.



Reglas de Codd

martes, 6 de julio de 2021 07:55 p. m.

Reglas de Codd (aseguran que un motor de BD es relacional)

- **Independencia** entre el **motor de base de datos** y los **programas que acceden a los datos**
 - Es posible modificar el motor de base de datos o los componentes de aplicación en forma independiente.
- **Representar la información** de la **estructura de las tablas y sus relaciones** mediante el **modelo relacional**.
- Las **reglas de integridad** deben guardarse en el **catálogo de la base**, **NO solo en programas de aplicación**. Se **pueden** guardar y verificar las reglas de integridad en **ambos lugares**.
- **Soportar información faltante mediante valores nulos (NULL)**.
- Proveer lenguajes para:
 - **Definición de datos (DDL)**
 - Ej: CREATE, ALTER, DROP, etc.
 - **Manipulación de datos (DML)**
 - Donde debe haber operaciones de alto nivel para insertar, eliminar, actualizar o buscar.
 - **Definición de Restricciones**
 - Seguridad, integridad, autorización.

Álgebra Relacional

martes, 6 de julio de 2021 08:04 p. m.

Álgebra creado por Edgar Codd.

- Define un **conjunto de operaciones** que se pueden **realizar entre relaciones** de un modelo relacional.
- Estas operaciones producen como **resultado** una **nueva relación**.

nota: no se usa en el mundo laboral pero sirve para entender los motores de bases de datos.

Operadores

- **Tradicionales**: utilizados en álgebra anteriormente, Codd las incluyó en el Álgebra Relacional.
 - **Unión (U)**
 - **Intersección (\cap)**
 - **Diferencia (-)**
 - **Producto cartesiano (X)**
- **Especiales**: orientados al manejo de relaciones.
 - **Select (σ)**
 - **Project (π)**
 - **Join (\bowtie)**
 - **Division (%)**

Unión

martes, 6 de julio de 2021 08:24 p. m.

La **unión** de dos relaciones compatibles A y B es el **conjunto de todas las tuplas que pertenecen a ambas relaciones**.

A

A#	NomA	CiudadA
A1	Marina	París
A2	Martin	Londres
A3	Graciela	Bs. As.
X1	Angel	Toronto

B

B#	NomB	Ciudad
B1	José	Miami
B2	Jorge	Orlando
X1	Angel	Toronto

$A \cup B$

A#	NomA	CiudadA
A1	Marina	París
A2	Martin	Londres
A3	Graciela	Bs. As.
X1	Angel	Toronto
B1	José	Miami
B2	Jorge	Orlando

Intersección

martes, 6 de julio de 2021 08:33 p. m.

La **intersección** de dos **relaciones compatibles** entre A y B es el conjunto de todas las tuplas que **pertenecen tanto a A como a B**.

A

A#	NomA	CiudadA
A1	Marina	París
A2	Martin	Londres
A3	Graciela	Bs. As.
X1	Angel	Toronto

B

B#	NomB	Ciudad
B1	José	Miami
B2	Jorge	Orlando
X1	Angel	Toronto

$A \cap B$

A#	NomA	CiudadA
X1	Angel	Toronto

Diferencia

martes, 6 de julio de 2021 08:36 p. m.

La **diferencia** entre dos relaciones A y B es el **conjunto de las tuplas que pertenecen a A y no pertenecen a B**.

A

A#	NomA	CiudadA
A1	Marina	París
A2	Martin	Londres
A3	Graciela	Bs. As.
X1	Angel	Toronto

B

B#	NomB	Ciudad
B1	José	Miami
B2	Jorge	Orlando
X1	Angel	Toronto

A – B

A#	NomA	CiudadA
A1	Marina	París
A2	Martin	Londres
A3	Graciela	Bs. As.

Producto Cartesiano

martes, 6 de julio de 2021 08:37 p. m.

El producto cartesiano extendido de dos **relaciones A y B** es el conjunto de las tuplas **t** tales que **t** es la concatenación de una tupla **a** perteneciente a **A** y una tupla **b** perteneciente a **B**.

A

A#	NomA	CiudadA
A1	Marina	París
A2	Martin	Londres
A3	Graciela	Bs. As.
X1	Angel	Toronto

B

B#	NomB	Ciudad
B1	José	Miami
B2	Jorge	Orlando
X1	Angel	Toronto

A X B

A#	NomA	CiudadA	B#	NomB	Ciudad
A1	Marina	París	B1	José	Miami
A1	Marina	París	B2	Jorge	Orlando
A1	Marina	París	X1	Angel	Toronto
A2	Martin	Londres	B1	José	Miami
A2	Martin	Londres	B2	Jorge	Orlando
A2	Martin	Londres	X1	Angel	Toronto
A3	Graciela	Bs. As.	B1	José	Miami
A3	Graciela	Bs. As.	B2	Jorge	Orlando
A3	Graciela	Bs. As.	X1	Angel	Toronto
X1	Angel	Toronto	B1	José	Miami
X1	Angel	Toronto	B2	Jorge	Orlando
X1	Angel	Toronto	X1	Angel	Toronto

Select (σ)

martes, 6 de julio de 2021 08:39 p. m.

Operador SELECT (σ)

- Construye una **nueva tabla al tomar un subconjunto horizontal** de la tabla existente.
- Produce un **subconjunto horizontal** de una relación específica.
- El resultado de la selección es otra tabla con los mismos atributos que la tabla original.

A

A#	NomA	CiudadA
A1	Marina	París
A2	Martin	Londres
A3	Graciela	Bs. As.
X1	Angel	Toronto

B

B#	NomB	Ciudad
B1	José	Miami
B2	Jorge	Orlando
X1	Angel	Toronto

σ (A)

CiudadA='París'

A#	NomA	CiudadA
A1	Marina	París

σ (B)

NomB='Jorge'

B#	NomB	Ciudad
B2	Jorge	Orlando

Project (π)

martes, 6 de julio de 2021 08:43 p. m.

- Construye una nueva tabla al tomar un **subconjunto vertical** de la tabla existente.
- Produce un **subconjunto vertical de una relación** dada, es decir el **subconjunto** obtenido de **seleccionar los atributos especificados**.

A

A#	NomA	CiudadA
A1	Marina	París
A2	Martin	Londres
A3	Graciela	Bs. As.
X1	Angel	Toronto

B

B#	NomB	Ciudad
B1	José	Miami
B2	Jorge	Orlando
X1	Angel	Toronto

π (A)

NomA, CiudadA

NomA	CiudadA
Marina	París
Martin	Londres
Graciela	Bs. As.
Angel	Toronto

π (B)

Ciudad

Ciudad
Miami
Orlando
Toronto

División (%)

martes, 6 de julio de 2021 08:45 p. m.

Sea una **relación A** de **grado $m + n$** donde **A** puede definirse como un conjunto de pares de valores $\langle x, y \rangle$.

Sea una **relación B** de **grado n** , donde **B** puede definirse como un conjunto de valores $\langle y \rangle$ simples.

Al aplicar el operador **división A % B** el resultado será una relación **C** de **grado m** donde **C** puede definirse como el **conjunto de valores x tales que el par $\langle x, y \rangle$ aparece en A para todos los valores y que aparecen en B**.

Los atributos de la **relación resultado**, tienen los **mismos nombres que los primeros m atributos de A**.

A		B		A % B	
S#	P#	P#		S#	
S1	P1	P1		S1	
S1	P2	P2			
S1	P3	P3			
S1	P4	P4			
S1	P5	P5			
S1	P6	P6			
S2	P1				
S2	P2				
S3	P2				
S4	P2				
S4	P4				
S4	P5				
D		A % D		A % E	
		P#		S#	
		P1		S1	
		P2		S2	
E					
		P#		S#	
		P2		S1	
				S2	
				S3	
				S4	

Join (\bowtie)

martes, 6 de julio de 2021 08:48 p. m.

- El resultado de aplicar un **JOIN** sobre dos tablas es una **nueva tabla** donde **cada renglón se forma concatenando dos renglones que tengan el mismo valor de atributo**.
- Se puede definir un **JOIN mayor que** de la relación **A** sobre el atributo **X** con la relación **B** sobre el atributo **Y** como el conjunto de todas las **tuplas t tales que, t es la concatenación de una tupla a tal que a pertenece a A y una tupla b perteneciente a B donde x > y y x es el componente X de A e y es el componente Y de B**.
- **Esta operación es equivalente a tomar el producto cartesiano de las dos relaciones dadas y luego realizar una selección adecuada sobre ese producto.**

S: Proveedores

S#	NombreS	Estado	Ciudad
S1	Smith	30	Bs.As.
S2	Jones	10	Rosario
S3	Blake	30	Rosario
S4	Clark	20	Bs. As.
S5	Adams	30	Córdoba

P: Productos

P#	NombreP	Color	Stock	Ciudad
P1	Monitor Led 19"	Rojo	100	Bs.As.
P2	Notebook 14" I5	Verde	70	Rosario
P3	HDD 500 Gb SATA	Azul	200	Córdoba
P4	HDD 1 Tb SATA	Rojo	100	Bs.As.
P5	CD / DVD Externo	Azul	200	Rosario
P6	UltraBook Corei7	Rojo	150	Bs.As.

SP: Relaciones

S#	P#	Cantidad
S1	P1	20
S1	P2	10
S1	P3	40
S2	P3	20
S2	P4	30
S2	P3	50
S2	P6	60
S2	P3	40
S2	P6	80
S2	P5	10
S3	P3	20
S3	P4	50
S4	P6	30
S4	P6	50
S5	P2	20
S5	P2	10
S5	P5	50
S5	P5	10
S5	P6	20
S5	P1	9
S5	P3	9
S5	P4	8
S5	P5	4
S5	P6	5

SP \bowtie P

p#

S#	P#	Cantidad	NombreP	Color	Stock	Ciudad
S1	P1	20	Monitor Led 19"	Rojo	100	Bs.As.
S1	P2	10	Notebook 14" I5	Verde	70	Rosario
S1	P3	40	HDD 500 Gb SATA	Azul	200	Córdoba
S2	P3	20	HDD 500 Gb SATA	Azul	200	Córdoba
S2	P4	30	HDD 1 Tb SATA	Rojo	100	Bs.As.
S2	P3	50	HDD 500 Gb SATA	Azul	200	Córdoba
S2	P6	60	UltraBook Corei7	Rojo	150	Bs.As.
.....						

Diseño de Datos

martes, 6 de julio de 2021 17:15

En diseño lógico separamos

- Entidades: partes de un sistema
- Relaciones
- Otras Responsabilidades

En diseño de datos

- A las relaciones las denominamos entidades

Entidad

La relación definida por Codd (representación de objetos, cosas , o partes de un sistema.)

- Representan asociaciones del mundo real entre entidades.
- Se puede representar con un **verbo o preposición** que conecta dos entidades.



Formas de comunicar un Modelo de Datos

- Prosa, poca y puntual
- **Diagrama Entidad Relación (DER)**
- Código SQL - DDL Data Definition Language (*Metadata*)
 - *Metadata*: definición de los datos.
 - Un campo nombre varchar, un campo edad int, etc

Modelo Conceptual

martes, 6 de julio de 2021 17:39

Define las **entidades y sus relaciones** en base al **negocio** (fácil de validar por el usuario)

Es independiente de la tecnología.

Diseño lógico

Define el esquema a trabajar.

El **esquema definido NO depende** de una **implementación física** determinada.

Ej: trabajamos en un esquema relacional sin decidir si lo implementaremos en Oracle, MySQL o SQLServer.

Un ejemplo de diseño lógico sería plantear una relación de muchos clientes a muchos autos, esto en el diseño físico se resolverá creando una tabla intermedia que rompa esa relación muchos a muchos.

Diseño físico

Se **implementa en motor de base de datos**.

Soporta un **determinado esquema**.

Impone **restricciones en la implementación**

- Lenguaje de manipulación de datos (DML = Data Manipulation Language)
- Lenguaje para creación de entidades (DDL = Data Definition Language)

Diagrama de Entidad Relación

- **Entidad**

Cualquier cosa que tenga relevancia para nuestro sistema.

Todo lo que se quiera persistir.

Los proyectos, las tareas, los impuestos, las complejidades, etc.

- **Atributos**

Son las propiedades o características que describen una entidad.

Un cliente tiene nombre y cuit, un auto tiene modelo, marca y color, etc.

- **Atributos multivaluado**

Son atributos que pueden tomar más de un valor (direcciones de mail de un empleado, subordinados de un jefe, entre otros)

- **Instancia de una entidad**

Ocurrencia particular de una entidad.

Ej: de una tabla Cliente cierto registro representa una instancia de un cliente.

- **Relaciones entre entidades**

Representan asociaciones del mundo real entre entidades

Un alumno puede pertenecer a un curso, entonces un curso estará relacionado con el alumno

Se puede representar con un verbo o preposición que conecta dos entidades.

Características de las Relaciones

martes, 6 de julio de 2021 18:01

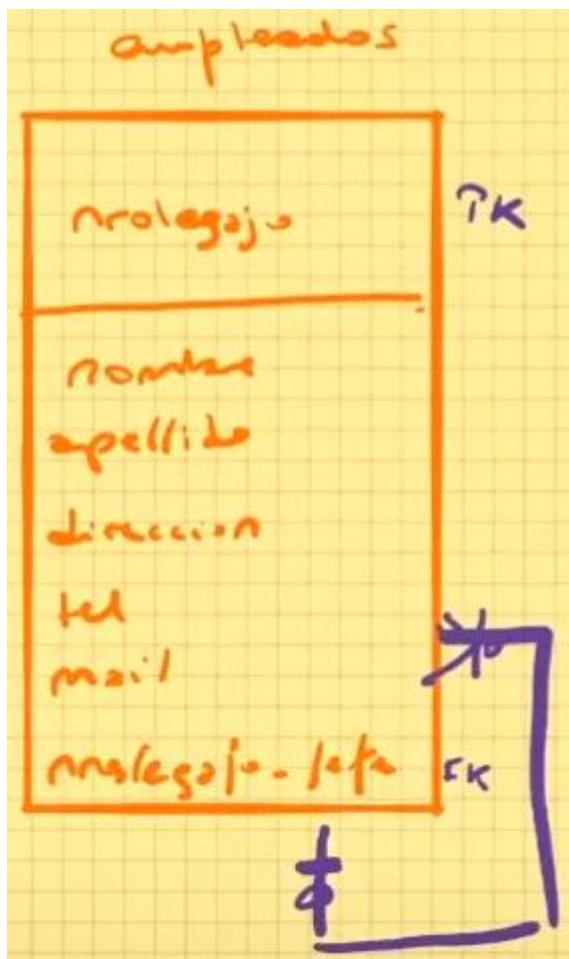
Grado

Yendo mas a fondo, **TODAS** las relaciones son binarias, pero luego se muestra.

• Unarias o Recursivas

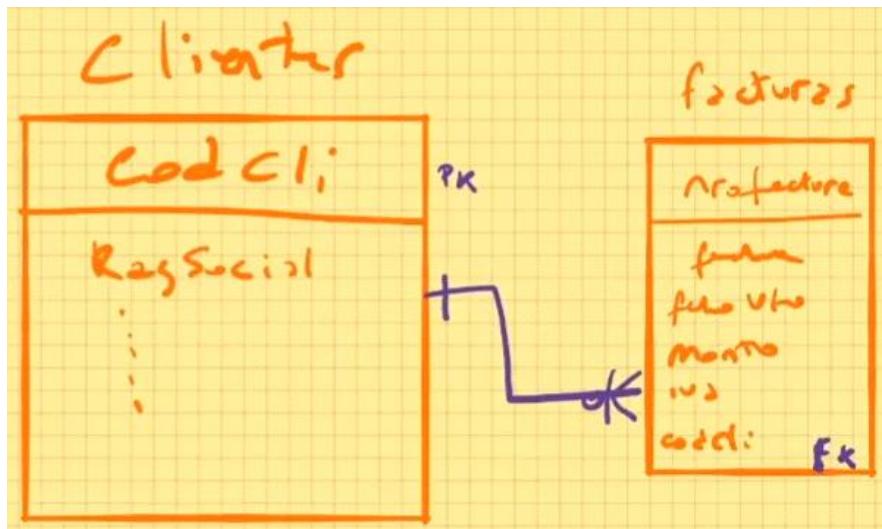
Son relaciones que asocian a una entidad con ella misma.

Un empleado tiene un jefe, un jefe puede tener muchos empleados



• Binarias

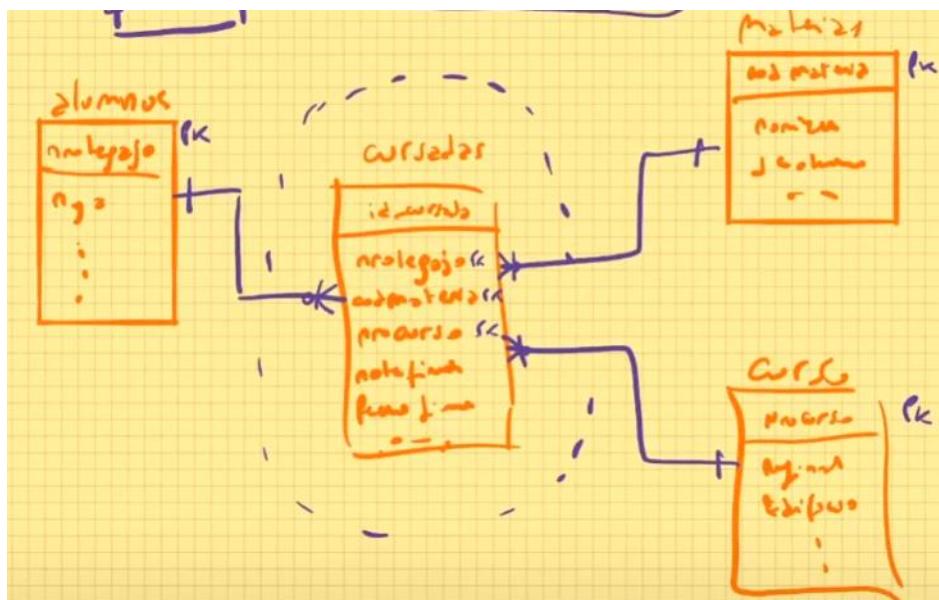
Relaciones que asocian a dos entidades.



- **N-arias**

Relaciones que asocian a N entidades

Por ejemplo, un alumno, curso y materia se relacionan a través de una entidad cursada



Cardinalidad

Describe la **cantidad de instancias de entidad permitidas en un relación entre dos entidades**.

- **1 a 1**

Un cliente tiene un domicilio y cada domicilio pertenece a un cliente.

- **1 a n**

Una empresa tiene muchas sucursales, cada sucursal pertenece a una empresa.

- **n a n**

Un profesor da clases a muchos alumnos, cada alumno tiene muchos profesores.

Modalidad

- Relación Mandatoria (**Obligatoria**)

Si la instancia de una entidad debe existir en la relación.

Ej: Una Factura debe contener como mínimo un ítem.

- Relación **Opcional**

Si la instancia de una entidad no necesita existir en la relación.

Ej.: Un cliente puede tener facturas asociadas.

Especialización o Generalización

- Entidad Supertipo:

Entidad padre.

- Entidad Subtipo:

Son las entidades hijas.

Son mutuamente excluyentes

Contienen distintos atributos



Notaciones de Relaciones

martes, 6 de julio de 2021 18:20

Notación	Uno a Uno	Uno a Muchos	Muchos a Muchos
Ross			
Bachman			
Martin (*)			
Chen			
IDEFX1			

Notación	Mandatoria	Opcional
Ross		
Bachman		
Martin (*)		
Chen		N/A
IDEFX1		

(*) Es la notación que utilizaremos.
También llamada Crow's foot notation.

Normalización / Desnormalización

martes, 6 de julio de 2021 18:20

Normalizar una base de datos significa **transformar un conjunto de datos** que tienen una **cierta complejidad** en su entendimiento y que su distribución en el modelo provoca problemas de lógica en las acciones de manipulación de datos, **en una estructura de datos que posea un diseño claro**, donde estos **datos guarden coherencia y no pierdan su estado de asociación**.

Normalizar **NO implica que sea mas performante**, si no que **la base de datos se vuelve menos performante pero a cambio otorga mucha mas consistencia a los datos almacenados**.

Objetivos de la normalización

- Reducir la **redundancia de datos**, y por lo tanto, las **inconsistencias**.
- Facilitar el **mantenimiento** de los **datos**.
- Evitar **anomalías** en la **manipulación de datos**.
- Reducir el **impacto** de los **cambios en los datos**.

Formas Normales

martes, 6 de julio de 2021 18:26

Son heurísticas o criterios que permiten resolver redundancias o inconsistencias.

Cada forma normal introduce restricciones nuevas, donde la primera restricción que aplica es cumplir la forma normal anterior.

Nosotros para implementar nos vamos a quedar con sólo las primeras 3 Formas Normales.

○ Primera Forma Normal

Una entidad que está en primera forma normal **No puede tener campos repetitivos** (arrays, mismo campo repetido n veces), o **campos multivaluados**.



No hay orden de arriba a abajo

ej: luego del ID 1 no está ID 2

Columnas no estan ordenadas de izquierda a derecha

ej: ID_Cliente no está antes de ID_Pedido

3. No hay filas duplicadas

4. No hay campos repetitivos, multivaluados

ej: arrays

5. Columnas regulares

Filas se identifican solo a través de claves candidatas.

Al normalizar:

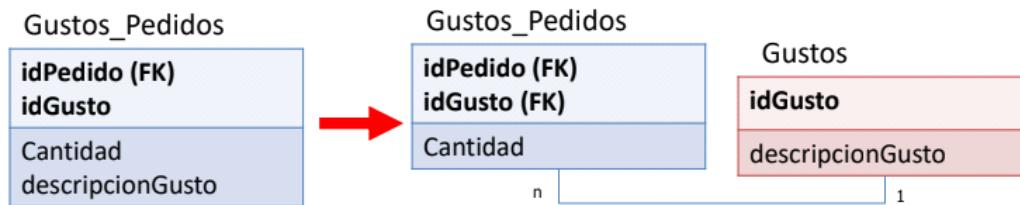
Descomponer atributos multivaluados en la misma tabla (genera multiples líneas con datos repetidos).

Encontrar PK nueva para evitar filas repetidas.

○ Segunda Forma Normal

Una estructura de datos está en 2FN si y sólo si **NO hay dependencias funcionales entre los atributos claves** (no hay dependencias parciales), y se satisface 1FN.

Atributos no clave deben depender de TODA la clave.



El atributo DescripcionGusto depende funcionalmente solo de idGusto, no tiene dependencia con el idPedido. Por lo que tiene una dependencia funcional parcial de la Clave Primaria.

Al Normalizar:

Dividir tabla en base a las claves de las que dependen los atributos.

ej:

Si en la misma relación id_pedido | id_gusto | cantidad | descripción:

id_gusto => descripción

id_pedido + id_gusto => Cantidad

En este caso id_gusto es determinante pero no clave candidata por si sola, también la compone

id_pedido.

Parto la relación en 2:

1. id_pedido | id_gusto | cantidad

nota: entidad-relacion o asociativa

2. id_gusto | descripción

y ahora todo determinante es clave candidata:

para Pedidos_Gustos: id_pedido + id_gusto => Cantidad

para Gustos: id_gusto => descripcion

Tercera Forma Normal

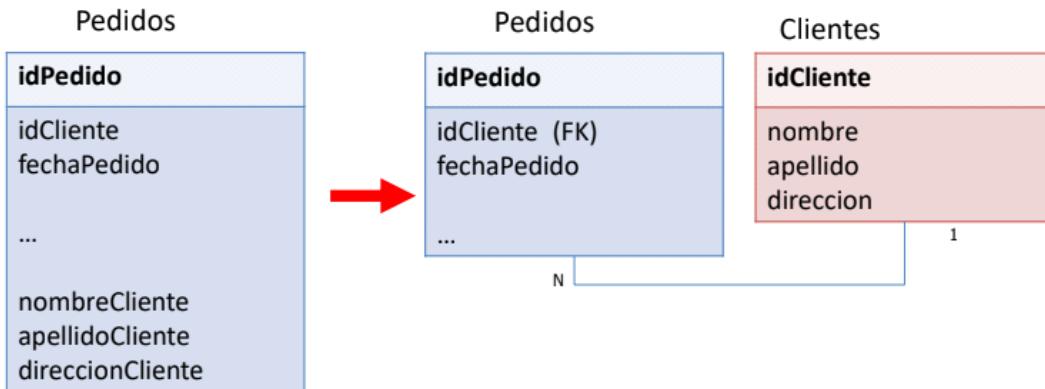
Una estructura de datos está en 3FN si y sólo si no hay dependencias funcionales entre los atributos no claves (y se satisface 2FN).

Atributos no clave NO deben depender de otros atributos.

La 3FN es una especialización de la 2FN .

Tercera Forma Normal

Una estructura de datos está en 3FN si y sólo si no hay dependencias funcionales entre los atributos no claves (y se satisface 2FN).



Al Normalizar:

Dividir tabla en base a las claves y atributos de las que dependen los atributos.

ej:

si en la misma relación id_pedido | id_cliente | nombre_cliente | domicilio_cliente | fecha_pedido:

id_pedido => fecha_pedido

id_cliente => nombre_cliente, domicilio_cliente

En este caso el atributo no-clave id_cliente es determinante pero no es clave candidata.

Parto la relación en 2 y elimino la redundancia:

1. id_pedido | id_cliente | fecha_pedido
2. id_cliente | nombre_cliente | domicilio_cliente

y ahora todo determinante es clave candidata:

para Pedidos_Gustos: id_pedido + id_gusto => Cantidad

para Gustos: id_gusto => descripcion

Ejemplo Normalización

martes, 6 de julio de 2021 18:42

1 FN

facturas

nroFactura
idCliente
nombreCliente
ciudadCliente
fechaEmision
fechaVto
productos [1..n]
cantidad [1..n]
precio [1..n]



Introducción en 1FN

facturas

nroFactura
idCliente
nombreCliente
ciudadCliente
fechaEmision
fechaVto
...

+

Items
nroFactura
idProducto
cantidad
precio

2 FN

Items

nroFactura
idProducto
cantidad
precio
descripcionProd
codFamilia
descriFamilia
codSubFamilia
descriSubFamilia



Items

nroFactura
idProducto
cantidad
precio

----- +

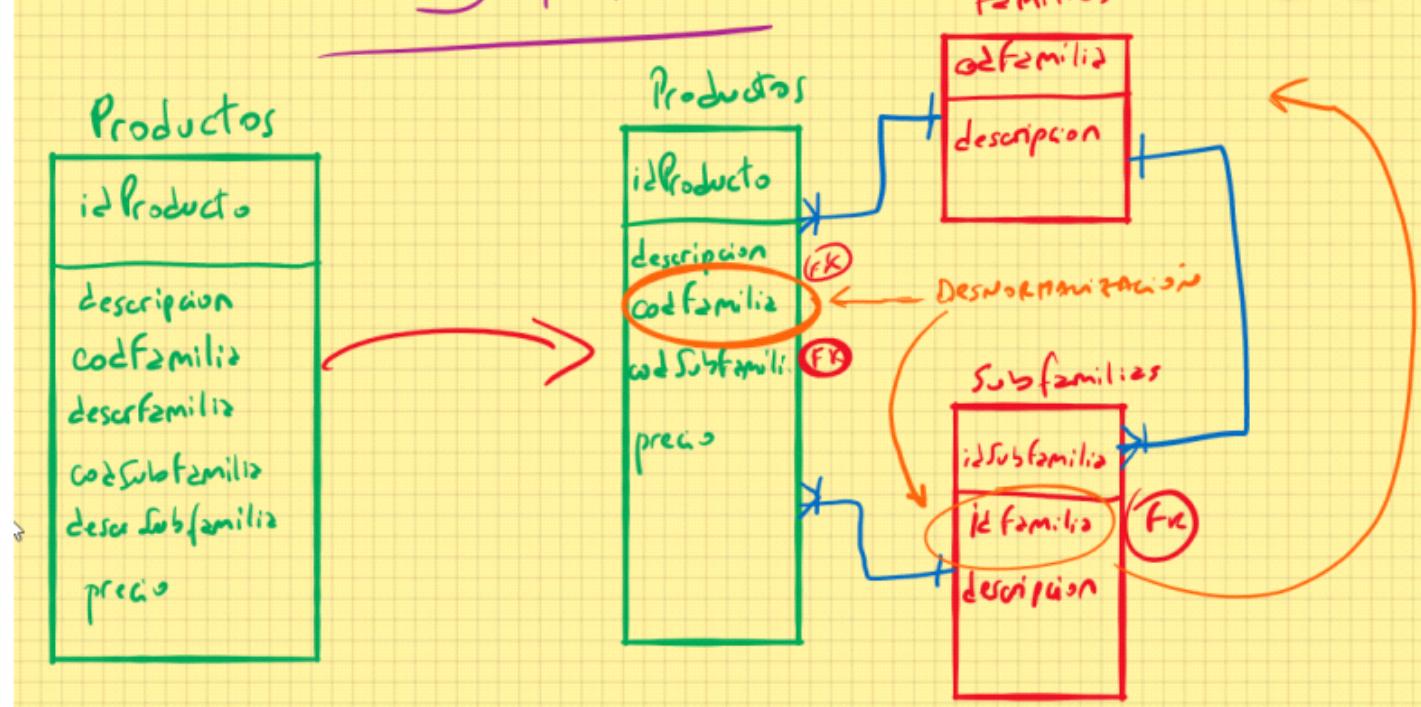
Productos

idProducto
descripcion
codFamilia
descriFamilia
codSubFamilia
descriSubFamilia
precio

Desnormalización

3 FN

R+ JZ



Practica en clase (Normalización)

martes, 6 de julio de 2021 18:46

Dadas las siguientes estructuras:

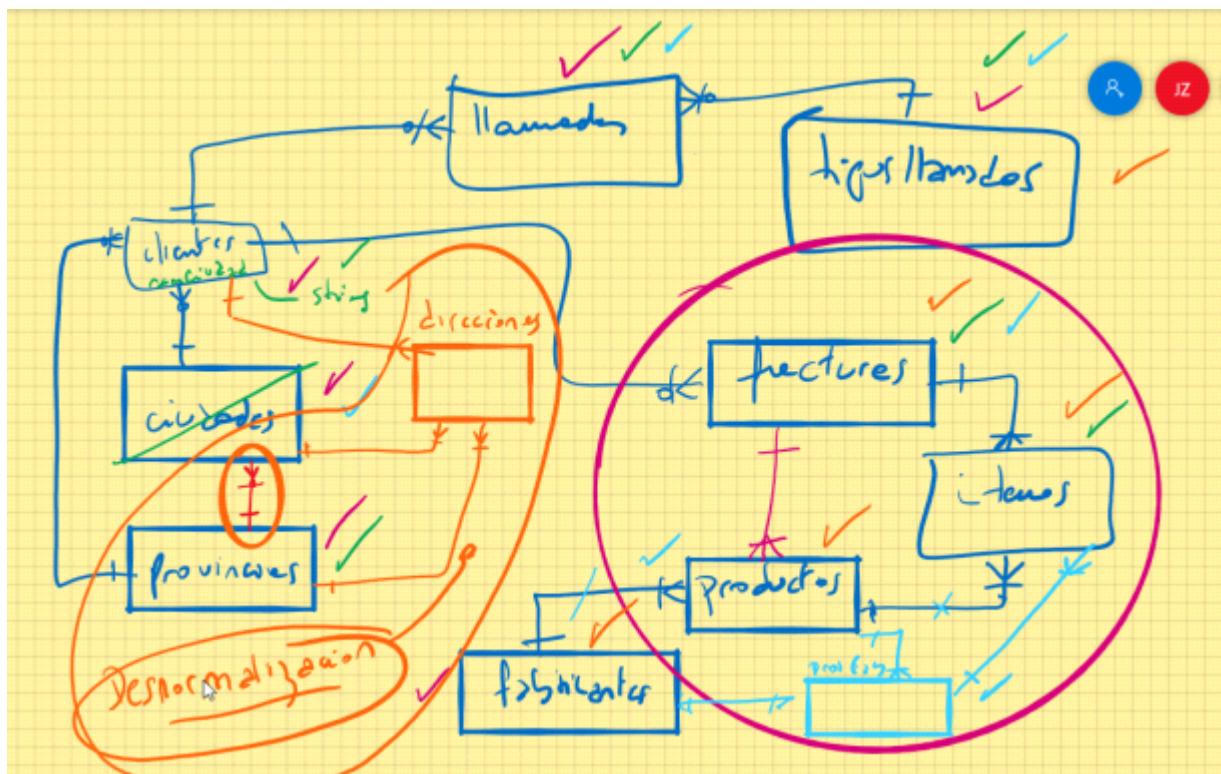
1. Normalice las entidades, aplique denormalización en los casos que crea convenientes.
2. Genere un Diagrama de Entidad Relación, indicando la cardinalidad y las claves primarias y foráneas.
3. Creación de Tablas y contraints ó restricciones (PK,FK, NOT NULL, UNIQUE, CHECK, DEFAULT)

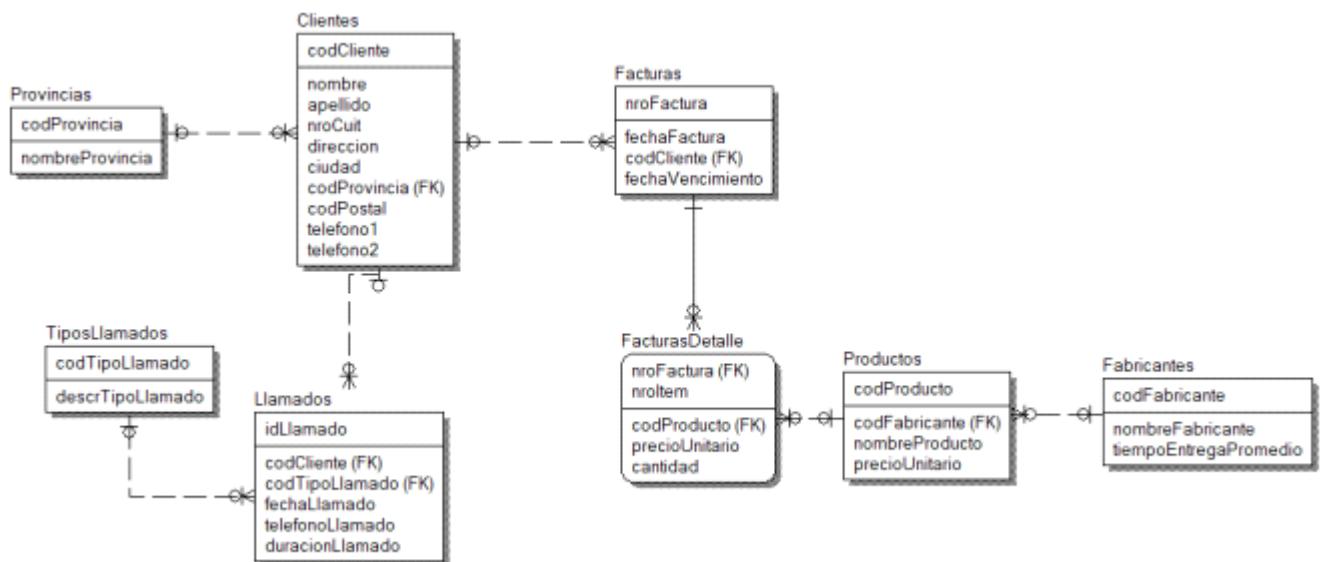
Ciudades

codCiudad, nombre, apellido, nroCuit, , teléfono1, teléfono2, dirección, ciudad, codProvincia, nombreProvincia, cod_postal, id_llamado1, fechallamado1, teléfonollamado1, codTipollamado1, descrTipollamado1,duracionllamado1, id_llamado2, fechallamado2, teléfonollamado2, codTipollamado2, descrTipollamado2,duracionllamado2, idLlamadoN, fechallamadoN, teléfonollamadoN, codTipollamadoN, descrTipollamadoN,duracionllamadoN,

Facturas

numeroFactura, fechaFactura, codCliente, nombre, apellido, nroCuit, fechaVencimiento, nroItem1, codProducto1, nomProducto1, codFabricante1, nombreFabricante1, tiempEntregaPromedio1, precioUnitario1, cantidad1, nroItem2, codProducto2, nomProducto2, codFabricante2, nombreFabricante2, tiempEntregaPromedio2, precioUnitario2, cantidad2,....., nroItem20, codProducto20, nomProducto20, codFabricante20, nombreFabricante20, tiempoEntregaPromedio20, precioUnitario20, cantidad20





DDL

miércoles, 7 de julio de 2021 15:08

Data Definition Language

Sublenguaje de SQL que sirve para **creación y definición de estructuras** de la BD.

- **CREATE TABLE**
 - Pueden tener **CONSTRAINTS**
- **CREATE VIEW**
- **CREATE FUNCTION**
- **CREATE STORED PROCEDURE**

Tablas

miércoles, 7 de julio de 2021 15:13

Unidad básica de almacenamiento de datos.

La **relación** especificada por **Codd** son lo que definimos como **tablas acá**.

Los **datos** están **almacenados en filas y columnas**.

Son de **existencia permanente** y poseen un **nombre identificatorio único** por **esquema o por base de datos** (dependiendo del motor de base de datos).

- No se borran hasta que decidamos borrarla nosotros
 - Caso especial son las tablas temporales (explicadas en Unidad 4)

Cada columna tiene entre otros datos un **nombre**, un **tipo de datos** y un **ancho** (este puede estar predeterminado por el tipo de dato).

```
CREATE TABLE nombreTabla(  
    NombreColumna1 INT NULL ,  
    NombreColumna2 INT NULL ,  
    NombreColumna3 VARCHAR(20) NULL )
```

Constraints opcionales aplicados a las columnas/campos de la tabla.

Los **tipos de datos de la columna** son constraints.

Constraints

miércoles, 7 de julio de 2021 15:21

Restricciones que se les puede imponer a los **valores** de cierta **columna / campo** de una tabla.

Todas las filas existentes en una tabla deben pasar un nuevo constraint creado para dicha tabla.

- En el caso de que alguna de las filas no cumpla, no se podrá crear dicho constraint o se creará en estado deshabilitado.

A nivel de Integridad de Entidades tenemos:

- **PRIMARY KEY CONSTRAINT**

- Atributo o conjunto de atributos que **identifican únicamente a una fila**.
- **No debe repetirse entre filas distintas.**
- **No debe permitir valores nulos.** (depende del motor de base de datos).
 - Semánticamente incorrecto
 - Si dos filas tendrían la PK en NULL, no se las podría identificar únicamente.
- Esta constraint es **utilizada** para asegurar la **Integridad de Entidades**
 - **Asegurar que los datos pertenecientes a una misma tabla tienen una única manera de identificarse**, o sea que **cada fila de cada tabla tenga una primary key** capaz de **identificar únicamente una fila** y esa no puede ser nula

A nivel referencial tenemos:

- **FOREIGN KEY CONSTRAINT**

- Atributo que referenciará a la **PRIMARY KEY de otra tabla**.
- **Debe permitir valores nulos.**
 - Caso de que un cliente no tenga asociada ninguna compra en una tabla compras
- Esta constraint es **utilizada** para asegurar la **Integridad Referencial**
 - **Asegurar la coherencia entre datos de 2 tablas.**

- Los constraints referenciales **permiten** a los usuarios **especificar claves primarias y foráneas** para **asegurar** una **relación PADRE-HIJO (MAESTRO-DETALLE)**.

A nivel de Integridad Semantica tenemos:

- **DATA TYPE**
 - Define el tipo de valor que se puede almacenar en una columna.
 - **INT**
 - **VARCHAR()**
- **DEFAULT**
 - Es el **valor insertado en una columna** cuando **al insertar un registro ningún valor fue especificado** para dicha columna.
 - El valor default por default es el NULL.
 - Se aplica a columnas no listadas en una sentencia INSERT.
 - Aplicado **sólo durante un INSERT (NO UPDATE)**.
- **UNIQUE**
 - Específica **sobre una o más columnas** que la **inserción o actualización de una fila contiene un valor único en esa columna o conjunto de columnas**.
 - Parecido a la PK, pero no necesariamente el campo con esta restricción debe ser PK.
 - Ejemplo
 - En la tabla Alumnos la PK sería el número de legajo y el número de CUIT debería tener como restricción UNIQUE.
- **NOT NULL**
 - Asegura que una **columna contenga un valor durante una operación de INSERT o UPDATE**.
 - Se considera el **NULL como la ausencia de valor**.
 - Uso en campos que sean obligatorios, que siempre deban ser cargados.

- **NULL**
 - Permite que el campo tenga valor nulo.
- **CHECK**
 - **Especifica condiciones para la inserción o modificación en una columna.**
 - Cada **fila insertada en una tabla** debe **cumplir con dichas condiciones**.
 - Actúa tanto **en el INSERT**, como en el **UPDATE**.
 - Es una **expresión** que **devuelve un valor booleano** de TRUE o FALSE.
 - Todas las columnas a las que referencia deben ser de la misma tabla.
 - No puede contener subconsultas, secuencias, funciones (de fecha, usuario) ni pseudocolumnas.

La **integridad semántica** es la que **nos asegura** que los **datos** que vamos a **almacenar**:

- **Tengan apropiada configuración**
- **Respeten las restricciones definidas sobre los dominios o sobre los atributos.**

Tipos de Constraints Referenciales

miércoles, 7 de julio de 2021 15:35

- **Ciclic Referential Constraint**

Asegura una relación de PADRE-HIJO entre tablas. Es el más común.

Ej: CLIENTE --> FACTURAS

- **Self Referencing Constraint**

Asegura una relación de PADRE-HIJO entre la misma tabla.

Son varios Ciclic Referential Constraint

Ej: EMPLEADOS --> EMPLEADOS

- **Multiple Path Constraint**

Se refiere a una PRIMARY KEY que tiene múltiples FOREIGN KEYS.

Este caso también es muy común.

Ej: CLIENTES --> FACTURAS

CLIENTES --> RECLAMOS

Ejemplo de SELF REFERENCING

miércoles, 7 de julio de 2021 16:29

- **Restricciones a nivel de Columna**

```
CREATE TABLE empleados (
    N_empleado INTEGER PRIMARY KEY,
    D_Apellido VARCHAR(60),
    D_nombres VARCHAR(60),
    N_cuil NUMERIC(11,0),
    N_jefe INTEGER,
    FOREIGN KEY (n_jefe) REFERENCES empleados (n_empleado) );
```

El motor no permitirá ingresar un empleado cuyo número de jefe no exista como número de empleado.

Lo que si permitirá es ingresar un número de jefe NULO.

Definicion de Constraints

miércoles, 7 de julio de 2021 16:04

Existen dos métodos para definir constraints:

- **Restricciones a nivel de Columna**

```
CREATE TABLE ordenes (
    N_orden INT PRIMARY KEY,
    N_cliente INT,
    F_orden DATE,
    C_estado SMALLINT,
    F_alta_audit DATE,
    D_usuario VARCHAR(20) );
```

- **Restricciones a nivel de Tabla**

Cuando la **restricción es sobre un grupo de columnas** se debe **utilizar una restricción a nivel de tabla**.

Cuando es **sobre sólo una columna** puede **utilizarse cualquiera de los dos modos**.

```
CREATE TABLE items_ordenes (
    N_orden INT,
    N_item SMALLINT,
    C_producto INT,
    Q_cantidad INT,
    I_precunit NUMERIC(9,3),
    PRIMARY KEY (n_orden, n_item) );
```

Ejemplos de FOREIGN KEY

miércoles, 7 de julio de 2021 16:08

• Restricciones a nivel de Columna

Cuando la clave foránea definida es simple.

```
CREATE TABLE ordenes (
    N_orden INTEGER PRIMARY KEY,
    N_cliente INTEGER,
    F_orden DATE,
    C_estado SMALLINT,
    F_alta_audit DATE,
    D_usuario VARCHAR(20)
);
```

```
CREATE TABLE items_ordenes (
    N_orden INT REFERENCES ordenes,
    N_item SMALLINT,
    C_producto INT,
    Q_cantidad INT,
    I_precunit NUMERIC(9,3),
    PRIMARY KEY (n_orden, n_item)
);
```

REFERENCES significa soy **clave foránea** de la **tabla items_ordenes** y **clave primaria** de la **tabla ordenes**, porque referencia a la PK de la tabla ordenes.

• Restricciones a nivel de Tabla

Cuando la clave foránea definida es compuesta.

```
CREATE TABLE items_ordenes (
    N_orden INT REFERENCES ordenes,
    N_item SMALLINT,
    C_producto INTEGER,
    Q_cantidad INTEGER,
    I_precunit DECIMAL ( 9,3 ),
    PRIMARY KEY (n_orden, n_item) );
```

```
CREATE TABLE mov_stock (
    N_stock INT,
```

```
C_movimiento SMALLINT,  
N_orden INT,  
N_item SMALLINT,  
C_producto INT,  
Q_cantidad INT,  
FOREIGN KEY (n_orden, n_item)  
REFERENCES items_ordenes  
);
```

Se podría poner también REFERENCES(n_orden, n_item), **siempre se trata de que todos los campos vayan teniendo los mismos nombres en las tablas.**

Ejemplos de Default

miércoles, 7 de julio de 2021 16:17

```
CREATE TABLE ordenes
(
    N_orden NUMBER NOT NULL,
    N_cliente NUMBER,
    F_orden DATE,
    C_estado NUMBER DEFAULT 1,
    F_alta_audit DATE DEFAULT CURRENT_DATE,
    D_usuario VARCHAR2(20) DEFAULT USER
);
```

Al ejecutar el siguiente comando

```
INSERT INTO ordenes (n_orden, n_cliente, f_orden) VALUES (117, 10, '12-JAN-2013')
```

El motor de base de datos insertará el siguiente registro en la tabla ordenes

N_orden 117
N_cliente 10
F_orden 12-JAN-2013
C_estado **1**
F_alta_audit Fecha del insert
D_usuario ID del usuario que realizó el insert

Ejemplos del NOT NULL

miércoles, 7 de julio de 2021 16:20

```
CREATE TABLE ordenes
(
    N_orden NUMBER NOT NULL,
    N_cliente NUMBER,
    F_orden DATE,
    C_estado NUMBER NOT NULL,
    F_alta_audit DATE,
    D_usuario VARCHAR2(20)
);
```

Ejemplos de UNIQUE

miércoles, 7 de julio de 2021 16:20

- **Restricciones a nivel de Columna**

```
CREATE TABLE empleados (
    N_empleado NUMERIC PRIMARY KEY,
    D_Apellido VARCHAR(60),
    D_nombres VARCHAR(60),
    N_cuil NUMERIC(11,0) UNIQUE,
    F_nacimiento DATE,
    F_ingreso DATE,
    N_jefe NUMERIC
);
```

- **Restricciones a nivel de Tabla**

```
CREATE TABLE empleados (
    N_empleado NUMERIC PRIMARY KEY,
    D_Apellido VARCHAR(60),
    D_nombres VARCHAR(60),
    T_docum NUMERIC(2,0),
    N_docum NUMERIC(11,0),
    F_nacimiento DATE,
    F_ingreso DATE,
    N_jefe NUMERIC,
    UNIQUE (t_docum, n_docum)
);
```

Ejemplos de CHECK

miércoles, 7 de julio de 2021 16:27

- **Restricciones a nivel de Columna**

```
CREATE TABLE ordenes
(
    N_orden NUMBER NOT NULL,
    N_cliente NUMBER,
    F_orden DATE,
    C_estado NUMBER CHECK (c_estado IN(1,2,3)),
    F_alta_audit DATE,
    D_usuario VARCHAR2(20)
);
```

- **Restricciones a nivel de Tabla**

```
CREATE TABLE empleados
(
    N_empleado NUMBER,
    D_Apellido VARCHAR2(60),
    D_nombres VARCHAR2(60),
    N_cuil NUMBER (11) UNIQUE,
    F_nacimiento DATE,
    F_ingreso DATE,
    N_jefe NUMBER,
    CHECK (F_nacimiento < F_ingreso)
);
```

SQL - DML

martes, 6 de julio de 2021 08:59 p. m.

Introducción

SQL - Lenguaje utilizado para interactuar con bases de datos.

- Al ser ejecutadas las sentencias SQL por el DBMS este retorna **Status** o **Datos**
 - **Status**
 - Ej: Al ejecutarse correctamente un **INSERT** el DBMS retorna **OK** como **Status**.
 - **Datos**
 - Ej: Al ejecutarse una sentencia **SELECT** sobre una tabla el DBMS devuelve los **Datos** solicitados.
- Sublenguajes de SQL
 - **DDL** - Data Definition Language: **creación y definición de estructuras** de la BD.
 - **CREATE TABLE**
 - Pueden tener **CONSTRAINTS**
 - **CREATE VIEW**
 - **CREATE FUNCTION**
 - **CREATE STORED PROCEDURE**
 - **DML** - Data Manipulation Language: **manipulación de datos** almacenados en la BD.
 - **SELECT**
 - **INSERT**
 - **UPDATE**
 - **DELETE**

SELECT

miércoles, 7 de julio de 2021 01:17 p. m.

Instrucción **SELECT** es utilizado para **consultar datos** de una o varias tablas.

FROM indica la tabla o tablas a consultar.

- **SELECT ***
 - Especificación de columnas o atributos a consultar de la tabla.
- **FROM nom_tabla**
 - Lista de tablas sobre la cual se realiza la consulta SELECT.
- **WHERE**
 - Condiciones ó filtros.
- **GROUP BY**
 - Columnas clave de agrupamiento.
- **HAVING**
 - Condiciones sobre lo agrupado.
- **ORDER BY**
 - Columnas clave de ordenamiento.

Ejemplos SELECT

miércoles, 7 de julio de 2021 01:24 p. m.

```
SELECT stock_num, manu_code, unit_price, unit_price*1.15  
FROM products
```

Consulta de distintas columnas con expresiones aritméticas.

```
SQLQuery1.sql - D...HQEMP6\zaffa (56)*
```

```
SELECT stock_num, manu_code, unit_price, unit_price*1.15  
FROM products
```

stock_num	manu_code	unit_price	No column name
1	HRO	250.00	267.5000
2	HSK	800.00	920.0000
3	SMT	450.00	517.5000
4	HRO	126.00	144.9000
5	HSK	240.00	276.0000
6	SHM	280.00	322.0000
7	HRO	480.00	552.0000
8	HSK	960.00	1104.0000
9	ANZ	19.80	22.7700
10	NRG	28.00	32.2000
11	SMT	25.00	28.7500
12	ANZ	48.00	55.2000
13	SMT	36.00	41.4000
14	HRO	600.00	690.0000
15	ANZ	840.00	966.0000
16	ANZ	20.00	23.0000
17	PRC	88.00	101.2000
18	SHM	68.00	78.2000
19	PRC	480.00	552.0000
20	SHM	220.00	253.0000

```
SELECT stock_num, manu_code, unit_price, unit_price*1.15 NewPrice  
FROM products
```

Alias de Columnas o Etiquetas

```
SQLQuery1.sql - D...HQEMP6\zaffa (56)*
```

```
SELECT stock_num, manu_code, unit_price,  
       unit_price*1.15 AS newPrice  
FROM products
```

stock_num	manu_code	unit_price	newPrice
1	HRO	250.00	267.5000
2	HSK	800.00	920.0000
3	SMT	450.00	517.5000
4	HRO	126.00	144.9000
5	HSK	240.00	276.0000
6	SHM	280.00	322.0000
7	HRO	480.00	552.0000
8	HSK	960.00	1104.0000
9	ANZ	19.80	22.7700
10	NRG	28.00	32.2000
11	SMT	25.00	28.7500
12	ANZ	48.00	55.2000
13	SMT	36.00	41.4000
14	HRO	600.00	690.0000
15	ANZ	840.00	966.0000
16	ANZ	20.00	23.0000
17	PRC	88.00	101.2000
18	SHM	68.00	78.2000
19	PRC	480.00	552.0000
20	SHM	220.00	253.0000

```
SELECT stock_num codTipoProducto, manu_code codFabricante,
unit_Price precioUnitario
FROM products
```

Alias de Columnas o Etiquetas

```
SQLQuery1.sql - D:\HOEMPO\zafra (56)* = X
SELECT stock_num codTipoProducto, manu_code codFabricante,
       unit_Price precioUnitario
  FROM products
```

	codTipoProducto	codFabricante	precioUnitario
1	1	HRO	250.00
2	1	HSK	800.00
3	1	SNT	450.00
4	2	HRO	126.00
5	3	HSK	240.00
6	3	SHM	280.00
7	4	HRO	480.00
8	4	HSK	960.00
9	5	ANZ	19.00
10	5	NPG	28.00
11	5	SNT	25.00
12	6	ANZ	48.00
13	6	SNT	36.00
14	7	HRO	600.00
15	8	ANZ	840.00
16	9	ANZ	20.00
17	101	PRC	68.00
18	101	SHM	68.00
...

Se puede utilizar también para mostrar columnas con otros nombres

```
SELECT lname + ', ' + fname apellidoYNombre, company, city
FROM customer
```

Concatenar columnas en una sola nueva columna de salida.

```
SQLQuery1.sql - D:\HOEMPO\zafra (56)* = X
SELECT lname + ', ' + fname apellidoYNombre, company, city
  FROM customer
```

	apellidoYNombre	company	city
1	Paul, Ludwig	All Sports Supplies	Sunnyvale
2	Sadler, Carla	Spots Spot	San Francisco
3	Curtis, Philip	Phil-a-Sports	Palo Alto
4	Higgins, Anthony	Play Ball!	Redwood City
5	Vetor, Raymond	Los Altos Sports	Los Altos
6	Watson, George	Watson & Son	Mountain View
7	Reeen, Charles	Athletic Supplies	Palo Alto
8	Quinn, Donald	Quinn's Sports	Redwood City
9	Miller, Jane	Spot Stuff	Sunnyvale
10	Jaeger, Roy	AA-Athletics	Redwood City
11	Keyes, Frances	Spots Center	Sunnyvale
12	Lawson, Margaret	Runnes & Others	Los Altos
13	Beatty, Lana	Spotstown	Menlo Park
14	Albertson, Frank	Spouting Race	Redwood City
15	Giant, Alfred	Gold Medal Sports	Menlo Park
16	Parmeleo, Joan	Olympic City	Mountain View
17	Spies, Arnold	Kids Kemer	Redwood City
18	Barker, Dick	Blue Ribbon Sports	Oakland
19	Shuster, Bob	The Tradition Club	Menlo Park

Además en el ejemplo le ponemos el alias apellidoYNombre

```

SELECT order_num, order_date, YEAR(order_date) anio,
       MONTH(order_date) mes, DAY(order_date) dia,
       USER_NAME()
FROM   orders

```

100 %

Results Messages

	order_num	order_date	anio	mes	dia	(No column name)
1	1001	2015-05-16 00:00:00.000	2015	5	16	dbo
2	1002	2015-05-17 00:00:00.000	2015	5	17	dbo
3	1003	2015-05-18 00:00:00.000	2015	5	18	dbo
4	1004	2015-05-19 00:00:00.000	2015	5	19	dbo
5	1005	2015-05-20 00:00:00.000	2015	5	20	dbo
6	1006	2015-05-26 00:00:00.000	2015	5	26	dbo
7	1007	2015-05-27 00:00:00.000	2015	5	27	dbo
8	1008	2015-06-03 00:00:00.000	2015	6	3	dbo
9	1009	2015-06-10 00:00:00.000	2015	6	10	dbo
10	1010	2015-06-13 00:00:00.000	2015	6	13	dbo
11	1011	2015-06-14 00:00:00.000	2015	6	14	dbo
12	1012	2015-06-14 00:00:00.000	2015	6	14	dbo
13	1013	2015-06-18 00:00:00.000	2015	6	18	dbo
14	1014	2015-06-21 00:00:00.000	2015	6	21	dbo
15	1015	2015-06-23 00:00:00.000	2015	6	23	dbo
16	1016	2015-06-28 00:00:00.000	2015	6	28	dbo
17	1017	2015-07-05 00:00:00.000	2015	7	5	dbo
18	1018	2015-07-06 00:00:00.000	2015	7	6	dbo
19	1019	2015-07-07 00:00:00.000	2015	7	7	dbo

Utilización de funciones especiales.

ARITMETICAS
TRIGONOMETRICAS
FINANCIERAS
DE FECHA
DE STRINGS
Entre otras..

WHERE

miércoles, 7 de julio de 2021 01:28 p. m.

Con la cláusula **WHERE** se aplican condiciones o filtros sobre los datos consultados de una tabla.

- **Operadores:**

AND, OR, NOT	(and, or, not lógicos)
=	(igualdad)
<> !=	(distinto)
> >=	(mayor, mayor igual)
< <=	(menor, menor igual)
LIKE	(validar substrings si son de determinada forma)
NOT LIKE	(validar substrings si NO son de determinada forma)
BETWEEN	(entre rango)
NOT BETWEEN	(no está entre rango)
IN	(en lista de valores)
NOT IN	(no está en la lista de valores)
IS NULL	(verificar si es NULL)
IS NOT NULL	(verificar si NO es NULL)

SELECT order_num, order_date, **YEAR**(order_date) anio,
MONTH(order_date) mes, **DAY**(order_date) dia,
USER_NAME()

FROM orders

Condiciones

WHERE condiciones

AND, OR, NOT

=

igualdad

<> !=

distinto

>, >=

mayor, mayor igual

<, <=

menor, menor igual

[NOT] LIKE

validar substrings

[NOT] BETWEEN

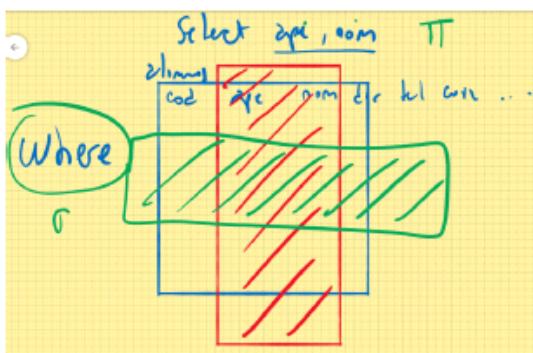
entre rango

[NOT] IN

en lista de valores

IS [NOT] NULL

es o no es nulo



```
SELECT order_num, order_date, customer_num, paid_date
FROM orders
WHERE customer_num=104
```

Condiciones por igualdad.

```
SQLQuery2.sql - D...HQEMP6\zaffa (55)* + X
SELECT order_num, order_date, customer_num, paid_date
FROM orders
WHERE customer_num = 104
```

Results

	order_num	order_date	customer_num	paid_date
1	1001	2015-05-16 00:00:00.000	104	2015-07-18 00:00:00.000
2	1003	2015-05-18 00:00:00.000	104	2015-05-10 00:00:00.000
3	1011	2015-06-14 00:00:00.000	104	2015-09-25 00:00:00.000
4	1013	2015-06-18 00:00:00.000	104	2015-07-27 00:00:00.000

```
SELECT order_num, order_date, customer_num, paid_date
FROM orders
WHERE customer_num=104
      AND order_num >1010
```

Varias Condiciones con AND.

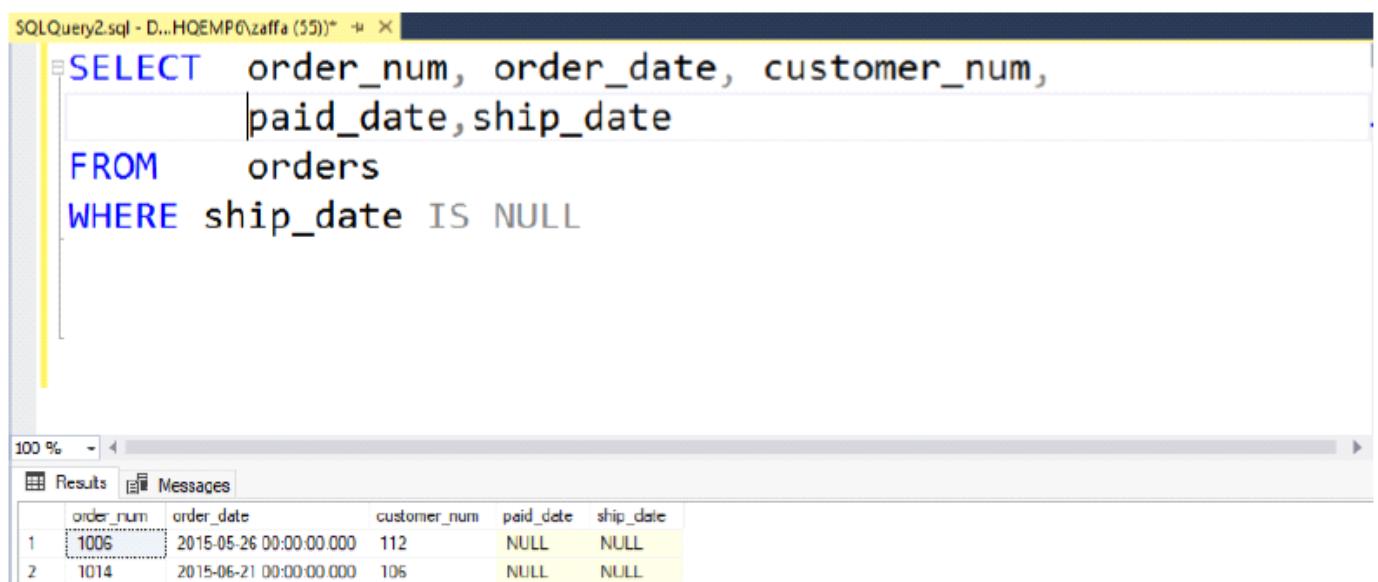
```
SQLQuery2.sql - D...HQEMP6\zaffa (55)* + X
SELECT order_num, order_date, customer_num, paid_date
FROM orders
WHERE customer_num = 104
AND order_num >1010
```

Results

	order_num	order_date	customer_num	paid_date
1	1011	2015-06-14 00:00:00.000	104	2015-08-25 00:00:00.000
2	1013	2015-06-18 00:00:00.000	104	2015-07-27 00:00:00.000

SELECT order_num, order_date, customer_num, paid_date, ship_date
FROM orders
WHERE ship_date IS NULL

Condición por columnas con Nulos.

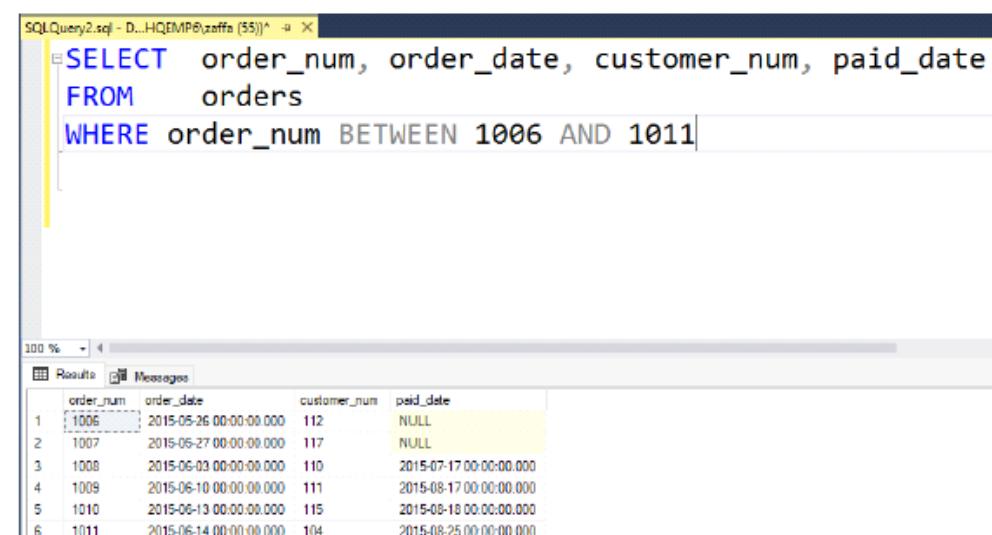


```
SQLQuery2.sql - D...HQEMP0\zaffa (55)* - μ ×
SELECT order_num, order_date, customer_num,
       paid_date, ship_date
FROM   orders
WHERE  ship_date IS NULL
```

	order_num	order_date	customer_num	paid_date	ship_date
1	1006	2015-05-26 00:00:00.000	112	NULL	NULL
2	1014	2015-06-21 00:00:00.000	106	NULL	NULL

SELECT order_num, order_date, customer_num, paid_date
FROM orders
WHERE order_num BETWEEN 1004 AND 1020

Condición por un rango de valores para una columna.



```
SQLQuery2.sql - D...HQEMP0\zaffa (55)* - μ ×
SELECT order_num, order_date, customer_num, paid_date
FROM   orders
WHERE  order_num BETWEEN 1006 AND 1011
```

	order_num	order_date	customer_num	paid_date
1	1006	2015-05-26 00:00:00.000	112	NULL
2	1007	2015-05-27 00:00:00.000	117	NULL
3	1008	2015-06-03 00:00:00.000	110	2015-07-17 00:00:00.000
4	1009	2015-06-10 00:00:00.000	111	2015-08-17 00:00:00.000
5	1010	2015-06-13 00:00:00.000	115	2015-08-18 00:00:00.000
6	1011	2015-06-14 00:00:00.000	104	2015-08-25 00:00:00.000

SELECT order_num, order_date, customer_num, paid_date
FROM orders
WHERE customer_num IN (104,110,127) Condición de una columna por una lista de valores

```
SQLQuery2.sql - D...HQEMP6\zafra (55)*  X
SELECT order_num, order_date, customer_num, paid_date
FROM orders
WHERE customer_num IN (104,110,127)
```

	order_num	order_date	customer_num	paid_date
1	1001	2015-05-16 00:00:00.000	104	2015-07-18 00:00:00.000
2	1003	2015-05-18 00:00:00.000	104	2015-06-10 00:00:00.000
3	1008	2015-06-03 00:00:00.000	110	2015-07-17 00:00:00.000
4	1011	2015-06-14 00:00:00.000	104	2015-08-25 00:00:00.000
5	1013	2015-06-18 00:00:00.000	104	2015-07-27 00:00:00.000
6	1015	2015-06-23 00:00:00.000	110	2015-08-27 00:00:00.000
7	1023	2015-07-20 00:00:00.000	127	2015-08-18 00:00:00.000

SELECT order_num, order_date, customer_num, paid_date
FROM orders
WHERE (order_num >= 1010 AND customer_num = 104)
OR (customer_num = 101) Condiciones con AND y OR.

```
SQLQuery2.sql - D...HQEMP6\zafra (55)*  X
SELECT order_num, order_date, customer_num, paid_date
FROM orders
WHERE (order_num >= 1010 AND customer_num = 104)
OR (customer_num = 101)
```

	order_num	order_date	customer_num	paid_date
1	1002	2015-05-17 00:00:00.000	101	2015-05-30 00:00:00.000
2	1011	2015-06-14 00:00:00.000	104	2015-08-25 00:00:00.000
3	1013	2015-06-18 00:00:00.000	104	2015-07-27 00:00:00.000

LIKE

miércoles, 7 de julio de 2021 03:36 p. m.

Operador LIKE

- % se reemplaza por 0 o más caracteres
- _ se reemplaza por 1 carácter
- [] utiliza un carácter del set

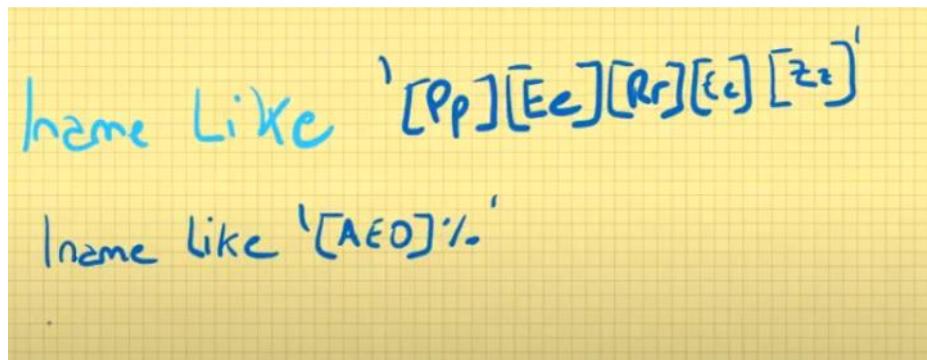
nota: LIKE es case sensitive.

nota2: Es costoso para la performance.

```
lname LIKE 'A%' -- campos comienzan con 'A'.
lname LIKE '%th%' -- campos contenga 'th' en cualquier parte.
lname LIKE 'A__' -- campos comienzan con 'A' y tengan 4 letras
lname LIKE '[AE]%' -- campos comienzan con 'A' ó 'E'.
lname LIKE '[A-E]%' -- campos comienzan entre la 'A' y la 'E'.
```

Otros ejemplos:

1. Iname tal que sea alguna combinación posible del apellido Pérez.
2. Iname tal que empiece con la letra A, E ó O



SELECT customer_num, lname, fname, company, city

FROM customer

WHERE condiciones

Condiciones con operador LIKE

Iname LIKE 'A%' apellidos que comienzan con 'A'.

Iname LIKE '%th%' apellidos que contenga 'th' en cualquier parte.

Iname LIKE 'A__' apellidos que comienzan con 'A' y tengan 4 letras

Iname LIKE '[AE]%' apellidos que comienzan con 'A' ó 'E'.

Iname LIKE '[A-E]%' apellidos que comienzan entre la 'A' y la 'E'.

```
SELECT customer_num, lname, fname, company, city
```

```
FROM customer
```

```
WHERE lname LIKE 'A%'
```

% reemplaza a 0 o más caracteres.

```
SQLQuery2.sql - D:\HQEMP6\zaffa (55)* -> x
SELECT customer_num, lname, fname, company, city
FROM customer
WHERE lname LIKE 'A%'
```

customer_num	lname	fname	company	city
114	Alberto	Frank	Sporting Place	Redwood City

```
SELECT customer_num, lname, fname, company, city
```

```
FROM customer
```

```
WHERE lname LIKE '%i%'
```

% reemplaza a 0 o más caracteres.

```
SQLQuery2.sql - D:\HQEMP6\zaffa (55)* -> x
SELECT customer_num, lname, fname, company, city
FROM customer
WHERE lname LIKE '%i%'
```

En este caso muestra todos los clientes cuyo apellido contenga una i, en cualquier lado.

customer_num	lname	fname	company	city
101	Pauli	Ludwig	All Sports Supplies	Sunnyvale
103	Cume	Philip	Phil's Sports	Palo Alto
104	Higgins	Anthony	Play Ball!	Redwood City
108	Quinn	Donald	Quinn's Sports	Redwood City
109	Miler	Jane	Sport Stuff	Sunnyvale
117	Stes	Arnold	Kids Komers	Redwood City
122	O'Brian	Cathy	The Sporting Life	Princeton
126	Neelie	Eileen	Neelie's Discount Sp	Denver
127	Sattler	Kim	Big Blue Bike Shop	Blue Island

```
SELECT customer_num, lname, fname, company, city  
FROM customer  
WHERE lname LIKE 'P_____'
```

Reemplaza a 1 sólo caracteres.

SQLQuery2.sql - D...HQEMP\zaffa (55)*

```
SELECT customer_num, lname, fname, company, city
FROM customer
WHERE lname LIKE 'P____'
```

Results Messages

	customer_num	lname	fname	company	city
1	101	Paul	Ludwig	All Sports Supplies	Sunnyvale

En este caso muestra todos los clientes cuyo apellido comience con P y tenga 5 letras.

```
SELECT customer_num, lname, fname, company, city  
FROM customer  
WHERE lname LIKE 'P%'
```

Diferencia entre _ y %

SQLQuery2.sql - D...HQEMP\zaffa (55)*

```
SELECT customer_num, lname, fname, company, city
FROM customer
WHERE lname LIKE 'P%'
```

Results Messages

	customer_num	lname	fname	company	city
1	101	Paul	Ludwig	All Sports Supplies	Sunnyvale
2	116	Parmelee	Jean	Olympic City	Mountain View
3	124	Putnum	Chris	Putnum's Putters	Bartleville

En este caso muestra todos los clientes cuyo apellido comience con P, sin importar la cantidad de letras.

```
SELECT customer_num, lname, fname, company, city
```

```
FROM customer
```

```
WHERE lname LIKE '[AB]%'
```

[] Evalúa un sólo carácter.

```
SQLQuery2.sql - D...HQEMP6\zaffa (55)* - X
SELECT customer_num, lname, fname, company, city
FROM customer
WHERE lname LIKE '[AB]%'
```

En este caso muestra todos los clientes cuyo apellido comience con A ó con B.

customer_num	lname	fname	company	city
1 113	Betty	Lana	Sportstown	Menlo Park
2 114	Alberton	Frank	Sporting Place	Redwood City
3 118	Baxter	Dick	Blue Ribbon Sports	Oakland

```
SELECT customer_num, lname, fname, company, city
```

```
FROM customer
```

```
WHERE lname LIKE '[A-G]%'
```

[] Evalúa un sólo carácter.

```
SQLQuery2.sql - D...HQEMP6\zaffa (55)* - X
SELECT customer_num, lname, fname, company, city
FROM customer
WHERE lname LIKE '[A-G]%' |
```

En este caso muestra todos los clientes cuyo apellido comience en el rango de A hasta la G inclusive.

customer_num	lname	fname	company	city
1 103	Currie	Philip	Phil's Sports	Palo Alto
2 113	Betty	Lana	Sportstown	Menlo Park
3 114	Alberton	Frank	Sporting Place	Redwood City
4 115	Grant	Alfred	Gold Medal Sports	Menlo Park
5 118	Baxter	Dick	Blue Ribbon Sports	Oakland

Verificar la segunda letra de un String

```
32  SELECT
33      manu_code AS Cod_Fabricante,
34      stock_num AS Nro_Producto,
35      SUM(quantity) AS Cantidad_Vendida,
36      SUM(quantity * unit_price) AS Total_Vendido
37  FROM items
38  WHERE manu_code LIKE '_[rR]%' -- fabricantes cuyo código tiene una "r" o una "R" como segunda letra
39  GROUP BY manu_code, stock_num
40  ORDER BY manu_code, stock_num
41
```

ORDER BY

miércoles, 7 de julio de 2021 03:24 p. m.

Ordenamiento del resultado de la consulta por una clave o múltiples claves.

nota: costoso para la performance

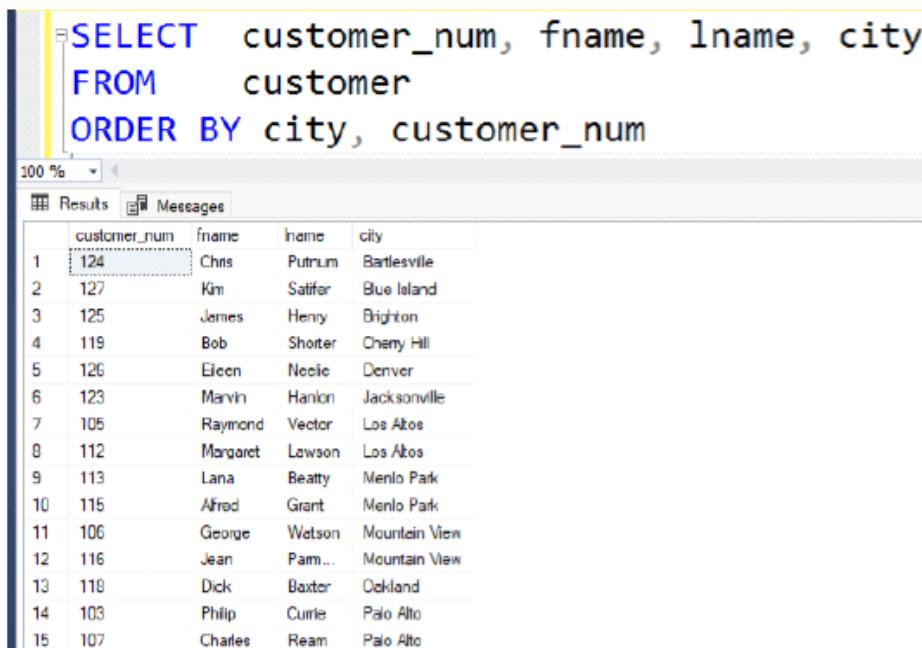
Tipos

- **ASC** (default): *Ascendente*
- **DESC** : *Descendente*

SQL – Operador SELECT

```
SELECT customer_num, fname, lname, city  
FROM customer
```

```
ORDER BY city, customer_num
```



The screenshot shows a SQL query results window. The query is:

```
SELECT customer_num, fname, lname, city
FROM customer
ORDER BY city, customer_num
```

The results table has four columns: customer_num, fname, lname, and city. The data is ordered by city (ascending) and customer_num (ascending). The cities listed are Bartlesville, Blue Island, Brighton, Chevy Hill, Denver, Jacksonville, Los Altos, Menlo Park, Mountain View, Oakland, Palo Alto, and Ream.

	customer_num	fname	lname	city
1	124	Chris	Putnum	Bartlesville
2	127	Kim	Satifer	Blue Island
3	125	James	Henry	Brighton
4	119	Bob	Shoiter	Chevy Hill
5	126	Eileen	Neele	Denver
6	123	Marvin	Hanlon	Jacksonville
7	105	Raymond	Vector	Los Altos
8	112	Margaret	Lawson	Los Altos
9	113	Lana	Beatty	Menlo Park
10	115	Alfred	Grant	Menlo Park
11	106	George	Watson	Mountain View
12	116	Jean	Parm...	Mountain View
13	118	Dick	Baxter	Oakland
14	103	Philip	Curle	Palo Alto
15	107	Charles	Ream	Palo Alto

Ordenamiento del resultado de la consulta por una clave o múltiples claves.

Observamos que las filas están ordenadas por ciudad ascendente y a igual ciudad ordena por customer_num también ascendente.

SQL – Operador SELECT

```
SELECT customer_num, fname, lname, city  
FROM customer  
ORDER BY city, customer_num DESC
```

```
SELECT customer_num, fname, lname, city  
FROM customer  
ORDER BY city, customer_num DESC
```

	customer_num	fname	lname	city
1	124	Chris	Putnum	Bartlesville
2	127	Kim	Satifer	Blue Island
3	125	James	Henry	Brighton
4	119	Bob	Shuter	Cherry Hill
5	126	Eileen	Neelie	Denver
6	123	Marvin	Hanlon	Jacksonville
7	112	Margaret	Lawson	Los Altos
8	105	Raymond	Vector	Los Altos
9	115	Alfred	Grant	Menlo Park
10	113	Lana	Beatty	Menlo Park
11	116	Jean	Parmelee	Mountain View
12	106	George	Watson	Mountain View
13	118	Dick	Baxter	Oakland
14	107	Charles	Rream	Palo Alto
15	103	Philip	Currie	Palo Alto

Ordenamiento del resultado de la consulta por una clave o múltiples claves.

Observamos que las filas están ordenadas por ciudad ASCENDENTE (default) y a igual ciudad ordena por customer_num DESCENDENTE.

SQL – Operador SELECT

```
SELECT customer_num, fname, lname, city  
FROM customer  
ORDER BY city, customer_num DESC
```

```
SELECT customer_num, fname, lname, city  
FROM customer  
ORDER BY 4, 1 DESC
```

	customer_num	fname	lname	city
1	124	Chris	Putnum	Bartlesville
2	127	Kim	Satifer	Blue Island
3	125	James	Henry	Brighton
4	119	Bob	Shuter	Cherry Hill
5	126	Eileen	Neelie	Denver
6	123	Marvin	Hanlon	Jacksonville
7	112	Margaret	Lawson	Los Altos
8	105	Raymond	Vector	Los Altos
9	115	Alfred	Grant	Menlo Park
10	113	Lana	Beatty	Menlo Park
11	116	Jean	Parmelee	Mountain View
12	106	George	Watson	Mountain View
13	118	Dick	Baxter	Oakland
14	107	Charles	Rream	Palo Alto
15	103	Philip	Currie	Palo Alto
16	128	Frank	Lessor	Phoenix
17	120	Fred	Jewell	Phoenix

Ordenamiento del resultado de la consulta por una clave o múltiples claves.

Se puede observar en este ejemplo que en el ORDER BY se pueden poner números que indican la posición de la columna en la consulta, en lugar del nombre.

SQL – Operador SELECT

```
SELECT customer_num  
FROM orders  
ORDER BY 1
```

VS.

```
SELECT DISTINCT customer_num  
FROM orders  
ORDER BY 1
```

customer_num
101
104
106
104
104
106
110
110

customer_num
101
104
106
110
111
112
115
116
117

GROUP BY y ORDER BY

```
SELECT YEAR(order_date) anio, MONTH(order_date) mes,  
       COUNT(order_num) cantOrdenes,  
       MIN(order_date) primerCompra, MAX(order_date) ultCompra  
  FROM orders  
 GROUP BY YEAR(order_date), MONTH(order_date)  
 ORDER BY cantOrdenes DESC
```

	anio	mes	cantOrdenes	primerCompra	ultCompra
1	2015	6	9	2015-06-03 00:00:00.000	2015-06-25 00:00:00.000
2	2015	7	7	2015-07-05 00:00:00.000	2015-07-20 00:00:00.000
3	2015	5	7	2015-05-16 00:00:00.000	2015-05-27 00:00:00.000

Las filas están ordenadas por cantidad de ordenes en orden Descendente.

Observamos que rápidamente obtenemos la cant. De ordenes, la primer y ultima compra pero ahora agrupado por año y mes.

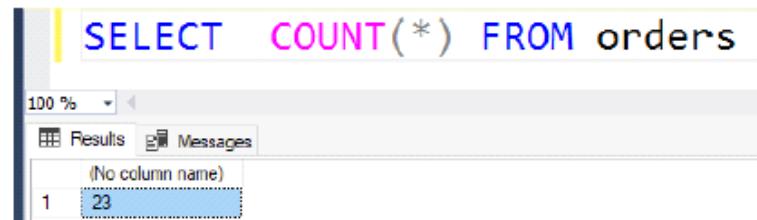
FUNCIONES AGREGADAS

miércoles, 7 de julio de 2021 03:40 p. m.

Son **funciones** que dado un **conjunto de datos** realizan **operaciones agregadas** devolviendo un **único valor** como resultado.

- **SUM(columna)**
- **COUNT(*)**
 - nota: cuenta todas las filas de la tabla
- **COUNT(columna)**
 - nota: cuenta filas con dicha columna No Nula
- **COUNT(DISTINCT columna)**
 - nota: cuenta solo una vez cada valor distinto de la columna.
- **MIN(columna)**
- **MAX(columna)**
- **SUM(columna)**
- **AVG(columna)**

Función COUNT (*) – Mostrar cantidad de Ordenes de Compra.



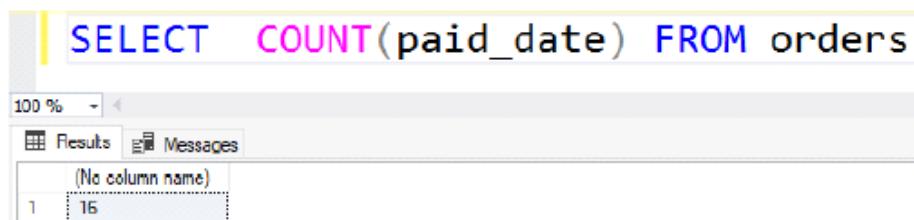
The screenshot shows a SQL query window with the following content:

```
SELECT COUNT(*) FROM orders
```

The results pane displays a single row with the value 23.

(No column name)
1 23

Función COUNT (paid_date) – Mostrar cantidad de Ordenes de Compra con fecha de pago No Nula.



The screenshot shows a SQL query window with the following content:

```
SELECT COUNT(paid_date) FROM orders
```

The results pane displays a single row with the value 16.

(No column name)
1 16

Función COUNT (DISTINCT customer_num) – Mostrar cuantos clientes nos pusieron Ordenes de Compra.

```
SELECT COUNT(DISTINCT customer_num) FROM orders
```

(No column name)
17

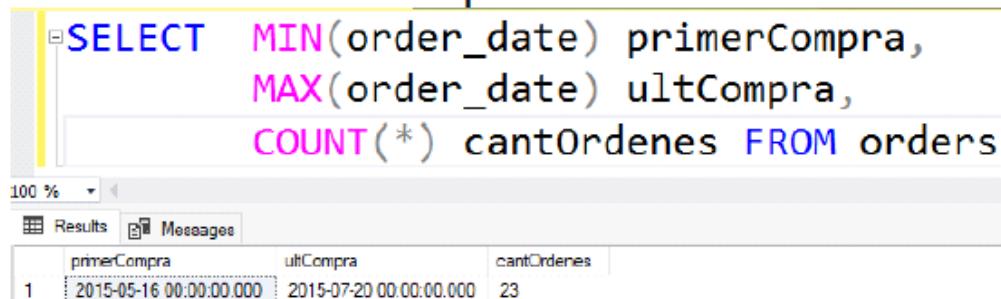
Función SUM (total_price) – Mostrar cuanto dinero nos ingreso por los ítems de todas las Ordenes de compra.

```
SELECT SUM(total_price) FROM items
```

(No column name)
18154.77

Varias funciones combinadas. Informar primer fecha de Orden de Compra, ultima fecha de orden de compra y cantidad de ordenes de compra

```
SELECT MIN(order_date) primerCompra,  
       MAX(order_date) ultCompra,  
       COUNT(*) cantOrdenes FROM orders
```



primerCompra	ultCompra	cantOrdenes
2015-05-16 00:00:00.000	2015-07-20 00:00:00.000	23

Función agregada de formula matemática. Informar promedio de precios unitarios de los ítems vendidos

```
SELECT AVG(total_price/quantity) promedioUnitPrice  
FROM items
```

promedioUnitPrice
207.13417910

GROUP BY

miércoles, 7 de julio de 2021 03:43 p. m.

- Esta cláusula de agrupamiento es muy poderosa en conjunto con la utilización de funciones agregadas.
- Utilizado cuando tengo funciones agregadas en los campos que no las utilizan.
- Las funciones van a ejecutarse por cada uno de los elementos singulares de la columna a la que se le aplica el GROUP BY.
- Debe agruparse por todos los campos NO agregados.
 - Un campo es "agregado" cuando se utiliza una función agregada sobre esta columna.

Ej:

```
SELECT YEAR(order_date) anio,      -- No agregado
       MONTH(order_date) mes,        -- No agregado
       COUNT(order_num) cantOrdenes, -- Agregado
       MIN(order_date) primerCompra, -- Agregado
       MAX(order_date) ultimaCompra, -- Agregado
FROM orders
GROUP BY YEAR(order_date), MONTH(order_date)      -- Group by campos no agregados
ORDER BY 2
```

The screenshot shows a MySQL command-line interface. At the top, there is a SQL query:

```
SELECT customer_num, COUNT(order_num) cantOrdenes,
       MIN(order_date) primerCompra, MAX(order_date) ultCompra
FROM   orders
GROUP BY customer_num
```

Below the query, there is a results tab showing the output of the query:

	customer_num	cantOrdenes	primerCompra	ultCompra
1	101	1	2015-05-17 00:00:00.000	2015-05-17 00:00:00.000
2	104	4	2015-05-16 00:00:00.000	2015-06-18 00:00:00.000
3	106	2	2015-05-10 00:00:00.000	2015-06-21 00:00:00.000
4	110	2	2015-06-03 00:00:00.000	2015-06-23 00:00:00.000
5	111	1	2015-06-10 00:00:00.000	2015-06-10 00:00:00.000
6	112	1	2015-05-26 00:00:00.000	2015-05-26 00:00:00.000
7	115	1	2015-06-13 00:00:00.000	2015-06-13 00:00:00.000
8	116	1	2015-05-20 00:00:00.000	2015-05-20 00:00:00.000
9	117	2	2015-05-27 00:00:00.000	2015-06-14 00:00:00.000
10	119	1	2015-06-25 00:00:00.000	2015-06-25 00:00:00.000
11	120	1	2015-07-05 00:00:00.000	2015-07-05 00:00:00.000
12	121	1	2015-07-06 00:00:00.000	2015-07-06 00:00:00.000
13	122	1	2015-07-07 00:00:00.000	2015-07-07 00:00:00.000
14	123	1	2015-07-07 00:00:00.000	2015-07-07 00:00:00.000
15	124	1	2015-07-19 00:00:00.000	2015-07-19 00:00:00.000
16	126	1	2015-07-20 00:00:00.000	2015-07-20 00:00:00.000
17	127	1	2015-07-20 00:00:00.000	2015-07-20 00:00:00.000

Observamos los 17 clientes que compraron, de cada uno tenemos la cantidad de Ordenes de compra, la fecha de la primer y ultima compra.

```

SELECT YEAR(order_date) anio, MONTH(order_date) mes,
       COUNT(order_num) cantOrdenes,
       MIN(order_date) primerCompra, MAX(order_date) ultCompra
FROM   orders
GROUP BY YEAR(order_date), MONTH(order_date)
ORDER BY 2

```

Results

	anio	mes	cantOrdenes	primerCompra	ultCompra
1	2015	5	7	2015-05-16 00:00:00.000	2015-05-27 00:00:00.000
2	2015	6	9	2015-06-03 00:00:00.000	2015-06-25 00:00:00.000
3	2015	7	7	2015-07-05 00:00:00.000	2015-07-20 00:00:00.000

Observamos que rápidamente obtenemos la cant. De ordenes, la primer y ultima compra pero ahora agrupado por año y mes.

GROUP BY y ORDER BY

```

SELECT YEAR(order_date) anio, MONTH(order_date) mes,
       COUNT(order_num) cantOrdenes,
       MIN(order_date) primerCompra, MAX(order_date) ultCompra
FROM   orders
GROUP BY YEAR(order_date), MONTH(order_date)
ORDER BY cantOrdenes DESC

```

Results

	anio	mes	cantOrdenes	primerCompra	ultCompra
1	2015	6	9	2015-06-03 00:00:00.000	2015-06-25 00:00:00.000
2	2015	7	7	2015-07-05 00:00:00.000	2015-07-20 00:00:00.000
3	2015	5	7	2015-05-16 00:00:00.000	2015-05-27 00:00:00.000

Las filas están ordenadas por cantidad de ordenes en orden Descendente.

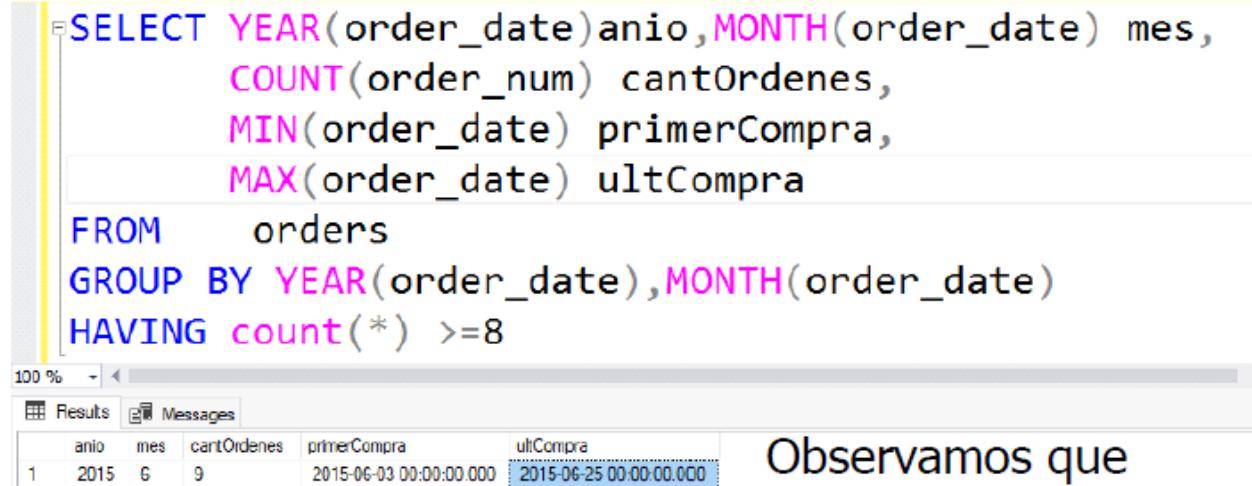
Observamos que rápidamente obtenemos la cant. De ordenes, la primer y ultima compra pero ahora agrupado por año y mes.

HAVING

miércoles, 7 de julio de 2021 03:51 p. m.

La cláusula **HAVING** actúa sobre los datos de las filas ya agrupadas y en general se le ponen condiciones con funciones agregadas.

GROUP BY y HAVING



```
SELECT YEAR(order_date) anio, MONTH(order_date) mes,
       COUNT(order_num) cantOrdenes,
       MIN(order_date) primerCompra,
       MAX(order_date) ultCompra
  FROM   orders
 GROUP  BY YEAR(order_date), MONTH(order_date)
 HAVING count(*) >= 8
```

	anio	mes	cantOrdenes	primerCompra	ultCompra
1	2015	6	9	2015-06-03 00:00:00.000	2015-06-25 00:00:00.000

La Cláusula HAVING actúa sobre los datos de las filas ya agrupadas y en general se le ponen condiciones con funciones agregadas.

Observamos que rápidamente obtenemos la cant. De ordenes, la primer y ultima compra pero ahora agrupado por año y mes, pero solo las que la cantidad sea ≥ 8 .

Ejercicio

miércoles, 7 de julio de 2021 03:52 p. m.

SQL – Operador SELECT

Ejercicio en clase

Listar por cada código de fabricante, cantidad de ordenes de compra (ante repetidos contar solo una), Suma de quantity, suma de unit_price, para los fabricantes que tengan mas de 5 items comprados (cantidad de filas en table items > 5).

Ordenado por la suma de unit_price*quantity.

items		
PK,FK	order_num	smallint
PK	item_num	smallint
FK	stock_num	smallint
FK	manu_code	char(3)
	quantity	smallint
	unit_price	decimal

```
SELECT manu_code,
       COUNT(DISTINCT order_num) cantOrdenes,
       SUM(quantity) CantidadComprada,
       SUM(unit_price*quantity) TotalComprado
  FROM   items
 GROUP  BY manu_code
 HAVING count(*) >=5
 ORDER BY 4 DESC
```

Results Messages

	manu_code	cantOrdenes	CantidadComprada	TotalComprado
1	AN2	11	59	4701.00
2	HSK	5	7	3748.00
3	HRO	6	10	2882.00
4	SHM	4	10	2337.97
5	KAR	4	13	1373.00
6	SMT	7	15	1269.00
7	FRC	4	13	1198.00

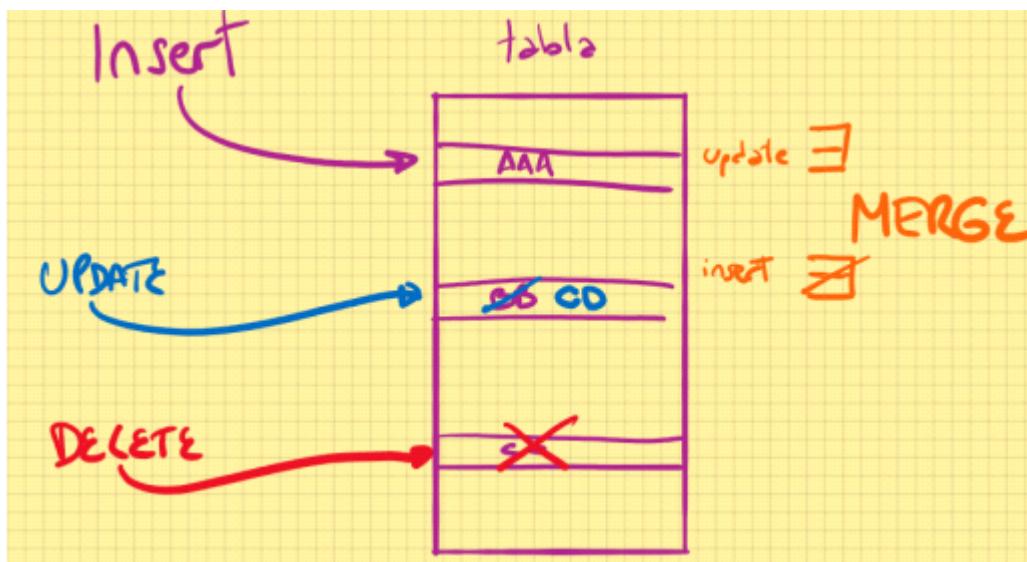
DML

martes, 6 de julio de 2021 18:55

Data Manipulation Language

4 Operadores

- INSERT
- UPDATE
- DELETE
- MERGE



Operador INSERT

martes, 6 de julio de 2021 19:05

Las columnas obviadas al insertar tienen por default NULL si no tiene el constraint DEFAULT.

Inserta una sola fila

INSERT INTO nom_tabla [(lista de columnas)]

VALUES (lista de valores)

nota: Los corchetes [] indican que es opcional

Ejemplo

INSERT INTO product_types

VALUES ('Short Baño', 375)

Al no poner nombres de columnas, el orden y cantidad de atributos debe cumplirse, acoplando la consulta.

Al agregar la lista de columnas,
podemos insertar filas con una
cantidad menor de valores a los
que tiene la definición de la tabla
y sin necesitar respetar el orden
de los mismos en la tabla real.

Siempre y cuando la lista de
columnas y la lista de valores
coincidan

Ejemplo desacoplado:

INSERT INTO customer (stock_num, lname, fname, address1, Company, address2, city)

VALUES (266,'Godoy','Estela','Pintos 3325', 'Ituza SA',null,'Buenos Aires')

Insertar múltiples filas

INSERT INTO nom_tabla [(lista de columnas)]

SELECT ...

nota: el select debe tener una estructura que espere el insert into

Ejemplo de Alto Acoplamiento

- **INSERT** acoplado a def de closed_orders
- **SELECT** acoplado a orders

INSERT INTO closed_orders

SELECT * FROM orders

WHERE paid_date IS NOT NULL

Ejemplo Desacoplado

INSERT INTO closed_orders

(order_num, order_date, customer_num,

ship_instruct, backlog, po_num, ship_date,

ship_weight, ship_charge, paid_date)

SELECT order_num, order_date, customer_num,

ship_instruct, backlog, po_num, ship_date,

ship_weight, ship_charge, paid_date

FROM orders

WHERE paid_date IS NOT NULL

Operador UPDATE

martes, 6 de julio de 2021 19:12

Modifica una o varias columnas de las filas que cumplan la condición.

UPDATE nom_tabla

SET columna=valor[, columna=valor...])

[WHERE condiciones]

NOTA: si no se pone where se hace en todas las filas.

Ejemplo

UPDATE customer

SET company = 'UTN',

phone = '5555-5555',

date = getdate() -- Uso de fórmula

WHERE customer_num = 112

Otro Ejemplo

Modificar ciertas filas cambiando el valor de una columna por el resultado de una formula.

UPDATE products

SET unit_price= unit_price*1,20

WHERE manu_code='ANZ'

Operador DELETE

martes, 6 de julio de 2021 19:26

Se eliminan las filas que cumplan con la condición del DELETE.

```
DELETE FROM nom_tabla  
[WHERE condiciones] (*)
```

(*) La cláusula WHERE es opcional, pero CUIDADO si no se pone condición el DELETE borrará la TOTALIDAD DE LAS FILAS DE LA TABLA.

Si los registros están referenciados a otra tabla NO te deja borrarlo (Integridad Referencial en las PKs y FKs)

Operador MERGE

martes, 6 de julio de 2021 20:07

Utilizada para **realizar procedimientos batch** de tablas.

- Migraciones
- Carga de datos
- Apareos

```
MERGE <target table>
USING <table_source>
ON <merge search condition>
[ WHEN MATCHED [ AND <clause search_condition> ]
    THEN <merge_matched>
[ WHEN NOT MATCHED [ BY TARGET ] [ AND <clause_search_condition> ]
    THEN <merge_not_matched> ]
[ WHEN NOT MATCHED BY SOURCE [ AND <clause search_condition> ]
    THEN <merge_matched> ]
[ output_clause ] ;
```

Si la fila existe entonces realiza UPDATE

Si la fila no existe entonces realiza INSERT

Ejemplo Merge

Requerimientos:

Dada una tabla merge Fuente y una tabla merge Destino cuyas claves primarias en ambas es el atributo código:

- Si el código de la tabla merge Fuente existe en la tabla mergeDestino y las direcciones son diferentes entonces actualizar la dirección en la tabla merge Destino.
- Si el código de la tabla merge Fuente NO existe en la tabla mergeDestino entonces insertar en la tabla merge Destino el registro de la tabla merge Fuente.
- Si el código de la tabla merge Destino NO existe en la tabla merge Fuente entonces borrar el registro de la tabla merge Destino

Creación tablas

```
create table mergeFuente
(codigo smallint PRIMARY KEY,
nombre varchar(38) not null,
direccion varchar(50));
create table mergeDestino
```

```
(codigo smallint PRIMARY KEY,  
nombre varchar(30) not null,  
direccion varchar(50),  
estado char(1) default 'A',  
observaciones varchar(50));
```

Insertar filas en ambas

```
insert into mergeFuente (codigo, nombre, direccion) values  
(2, 'Ricardo Ruben', 'Paraguay 1888'), -- modifica la direccion  
(3, 'Juan Jose Jacinto', 'Terranova 765'), -- no modificada  
(8, 'Carola Sampietro', 'Arenales 1265'); -- nuevo  
insert into mergeDestino (codigo, nombre, dirección) values  
(1, 'Pepe', 'Venezuela 3456'),  
(2, 'Ricardo Ruben', 'Cucha Cucha 234'),  
(3, 'Juan Jose Jacinto', 'Terranova 765'),  
(4, 'Violeta Rivarola', "Quito 2112");
```

Sentencia merge

```
MERGE merge Destino d  
USING merge Fuente f  
ON d.codigo = f.codigo  
WHEN MATCHED AND d.direccion <> f.direccion THEN  
UPDATE  
SET d.direccion = f.direccion  
WHEN NOT MATCHED BY TARGET THEN  
INSERT (codigo, nombre, direccion, estado, observaciones) VALUES (f.codigo,  
f.nombre, f.  
WHEN NOT MATCHED BY SOURCE THEN  
DELETE;
```

```
Select * from mergeDestino;
```

codigo	nombre	direccion	estado	observaciones
2	Ricardo Ruben	Paraguay 1888	A	NULL
3	Juan Jose Jacinto	Terranova 765	A	NULL
8	Carola Sampietro	Arenales 1265	A	Nuevo

Podemos observar el resultado esperado.

Se actualizó la dirección de la fila de la tabla mergeDestino con código = 2;

Se insertó la fila con código = 8 y

Se borraron las filas con códigos 1 y 4.

Secuencias

martes, 6 de julio de 2021 19:30

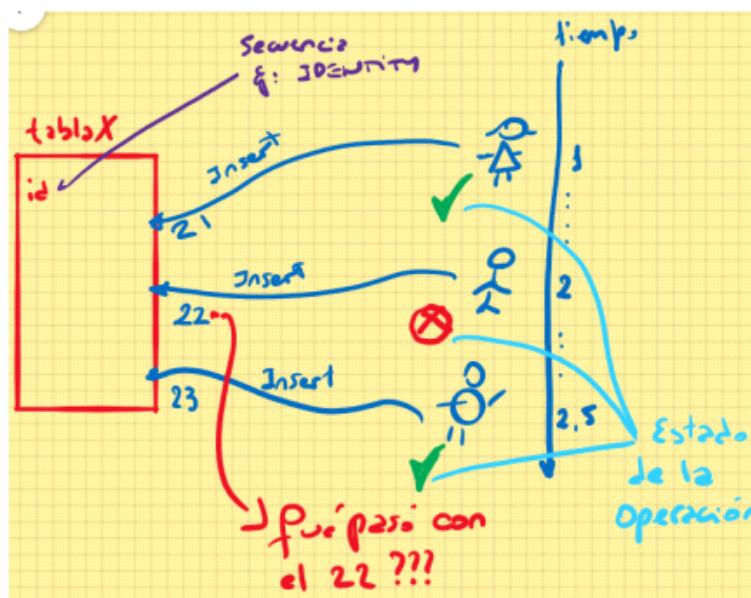
Generadores de secuencias

Proveen una serie de números secuenciales y únicos sin el overhead de I/O a disco o lockeo transaccional

nota: sirven para crear claves subrogadas.

Si por algún motivo se anula una transacción con una clave ya asignada, me queda un número sin usar.

No sirve para casos legales que requieran no tener huecos como números de factura, ejemplo del caso 22 que se pierde en el ejemplo.



Propiedades de una columna

Existen motores que poseen propiedades de columna que permite realizar lo mismo que una secuencia.

Al insertar una fila en dicha tabla, el motor va a buscar el próximo número del más alto existente en la tabla.

Ej. SQLServer IDENTITY

```
CREATE TABLE ordenes (
    N_orden int IDENTITY(1, 1),
    N_cliente int NULL,
    F_orden datetime NULL);
```

```
INSERT INTO ordenes
    (n_cliente, f_orden)
```

```
VALUES (114,'2020-03-03')
```

Motores de base de datos los implementan como:

- Tipo de dato de una columna
 - Infomix y PostgreSQL
- Propiedades de una columna
 - SqlServer, Mysql, DB2
- Object Sequence
 - Oracle, Informix, PostgreSQL, DB2, SqlServer.

Tablas Temporales

martes, 6 de julio de 2021 19:54

Son tablas creadas cuyos **datos son de existencia temporal, NO son persistentes.**

No son registradas en las tablas del **diccionario de datos.**

No es posible alterar tablas temporarias. **Si eliminarlas y crear.**

Pueden crearse índices temporales que necesiten las aplicaciones.

Las actualizaciones a una tabla temporal podrían no generar ningún log transaccional si así se configurara.

Tipos de Tablas

- De Sesión (locales)

Son visibles sólo para sus creadores durante la **misma sesión** (conexión) a una instancia del motor de BD.

Las tablas temporales locales **se eliminan cuando el usuario se desconecta o cuando decide eliminar la tabla durante la sesión.**

- Globales

Visibles para cualquier usuario y sesión una vez creadas. Su **eliminación depende del motor.**

Se crean con doble ##

Tipos de Creación

- Explícita

Este **tipo de creación se realiza mediante** la instrucción **CREATE.**

De manera explícita se deberá crear la tabla indicando el nombre, sus campos, tipos de datos y restricciones.

- Implícita

Se pueden **crear tablas temporales a partir** del resultado de una **consulta SELECT.**

Utilización de Tablas Temporales

- Almacenamiento intermedio de Consultas Muy Grandes
 - Dividir consultas gigantes en consultas mas pequeñas para mayor entendimiento.
- Optimizar accesos a una consulta varias veces en una aplicación
 - Cuando se acceden múltiples veces a datos que no suelen cambiar.

Ejemplos

-- "#" indica que es temporal de sesión

- Explícita:

```
CREATE TABLE #ordenes_pendientes (N_orden INTEGER,  
N_cliente INTEGER,  
F_orden DATE,  
I_Total DECIMAL(15 , 2),  
C_estado SMALLINT,  
F_alta_audit TIMESTAMP,  
D_usuario VARCHAR(20) )  
WITH NO LOG;
```

-- "NO LOG" indica que no genera logs

```
INSERT INTO #ordenes_Pendientes  
SELECT * FROM ordenes WHERE c_estado = 1
```

- Implícita:

```
SELECT *  
INTO #ordenes_Pendientes  
FROM ordenes
```

`WHERE c_estado = 1`

VIEW

miércoles, 7 de julio de 2021 04:13 p. m.

- Una **VISTA** toma la **salida** resultante de una **consulta** y la **trata como una tabla**.
- Una **VIEW** es un **conjunto de columnas**, ya sean **reales o virtuales** (función agregada), de una **misma tabla o no**, con **algún filtro determinado o no**.
- Una **VIEW** es una **presentación adaptada** de los **datos contenidos en una o más tablas**, o en **otras vistas**.

Características

- Tiene un **nombre específico**.
- **NO ocupa espacio de almacenamiento**, no guarda nada en disco, no contiene datos almacenados. Solo guarda metadata, guarda la consulta que se realiza en la vista.
- **Está definida por una query** que consulta datos de una o más tablas.

Usos

- Suministrar un **nivel adicional de seguridad**.
 - El usuario solo tiene acceso al conjunto predeterminado de datos de la vista, el usuario no sabe qué tablas se utilizan en la vista ni de qué manera.
- Ocultar **complejidad** del modelo.
 - El usuario no sabe cómo está compuesta la query que compone a la vista.
- **Simplifica las sentencias** para los desarrolladores.
 - Ya que es más simple realizar un `SELECT * FROM vista_clientes` que utilizar una query compleja cada vez que se necesita utilizar esa información.
- **Presentar los datos de una forma diferente**.
- **Aíslar a las aplicaciones** de los **cambios** en la **tabla base**.
 - **Desacoplamiento de apps**.
 - Ej: Si una app consulta algunos datos del modelo a través de una vista y este modelo sufriera cambios, solo deberíamos asegurarnos de que la vista siga devolviendo los datos solicitados y no es necesario realizar cambios en la app.

Restricciones

- **No se pueden crear índices** en vistas porque no es una tabla.
- Una **view depende de las tablas a las que hace referencia**.
 - Si se elimina una tabla todas las views que dependen de ella se borraran o se pasará a estado INVALIDO, dependiendo del motor. Lo mismo para el caso de borrar una view de la cual depende otra view.
- El uso principal de las views es poder consultar la información pero, a través de las **views también se pueden insertar y/o modificar las tablas utilizadas** por estas, pero existen ciertas restricciones y buenas prácticas (WITH CHECK OPTION) a tener en cuenta.
- Algunas views tienen **restringidas las operaciones INSERT, UPDATE y DELETE** cuando tienen:
 - **JOINS**
 - **Funciones Agregadas**
 - **Triggers INSTEAD OF**
- **Usuario** debe tener **permiso** de **SELECT** sobre las tablas que utiliza la vista.
- Algunos motores **no soportan UNION u ORDER BY** en views.
- **Restricciones** para el caso de **Actualizaciones**:
 - **Si en la tabla existieran campos que no permiten nulos y en la view no aparecen, los INSERT fallarían.**
 - **Si en la view no aparece la PK los INSERT podrían fallar.**
 - Pueden borrarse filas a través de la vista aunque tenga columnas virtuales, pero hay casos en que no.
 - Con la opción **WITH CHECK OPTION**, se puede actualizar siempre y cuando el checkeo de la opción en el **WHERE** sea verdadero.
 - Utilizado para **control de integridad**.

VIEW - Ejemplos

miércoles, 7 de julio de 2021 05:06 p. m.

- Al acceder a través de la vista debo usar los nombres de columnas definidos en ella, no los de la tabla
- Al hacer el select sobre la vista, el motor realiza la query definida en la creación para obtener los datos.

```
CREATE VIEW V_clientes_california  
(codigo, apellido, nombre)  
AS
```

```
SELECT customer_num, lname, fname  
  FROM customer  
 WHERE state='CA'
```

```
CREATE VIEW V_clientes_california  
(codigo, apellido, nombre, estado)  
AS  
  
SELECT customer_num, lname, fname, state  
  FROM customer  
 WHERE state='CA'
```

Messages
Command(s) completed successfully.

Se inserta una fila en la tabla clientes un cliente de FLORIDA a través de la vista v_clientes_california pero cuando se vuelve a consultar la vista dará el mismo resultado porque solo muestra clientes de CALIFORNIA.

Ejecutamos un inserta través de una vista de una fila que la vista no puede mostrar por sus condiciones en el WHERE.

A través de la vista manipulamos datos que actualmente no pueden ser accedidos a través de la vista.

Esto no es correcto, debería utilizarse la cláusula WITH CHECK OPTION.

Hacemos una consulta sobre la View v_clientes_california

```
select count(*) from v_clientes_california
```

Results

(No column name)

1 18

Insertamos a través de la View una fila en la tabla original customer los siguientes valores (999,'Martin', 'Mihura','FL').

```
INSERT INTO v_clientes_california  
(codigo, apellido, nombre, estado)  
VALUES (999, 'Martin', 'Mihura', 'FL')
```

1 row(s) affected

```
select count(*)  
from v_clientes_california
```

Results

(No column name)

1 18

Por Qué?

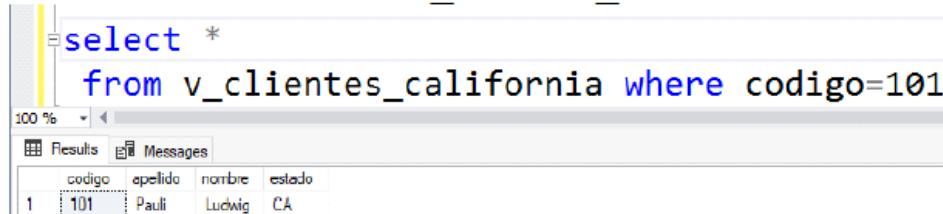
El cliente de código 101 se muestra en la vista v_clientes_california, pero al realizar una actualización a través de la vista del "estado" del cliente este ya no se muestra como resultado de la vista.

Ejecutamos un UPDATE en una fila a través de la vista cambiando un valor de un campo que es condicionen el WHERE.

A través de la vista manipulamos datos que actualmente no pueden ser accedidos a través de la vista.

Esto no es correcto, debería utilizarse la cláusula WITH CHECK OPTION.

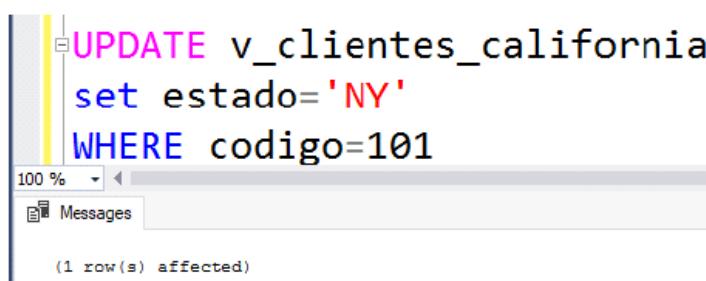
Hacemos una consulta sobre la View v_clientes_california



```
select *  
from v_clientes_california where codigo=101
```

	codigo	apellido	nombre	estado
1	101	Pauli	Ludwig	CA

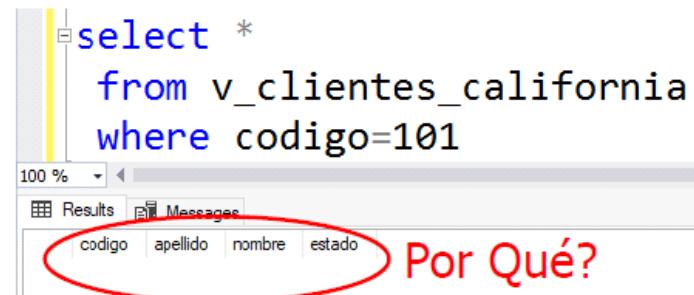
Hademos un Update sobre el cliente 101 de California cambiando su estado a NewYork.



```
UPDATE v_clientes_california  
set estado='NY'  
WHERE codigo=101
```

100 % ▶

Messages
(1 row(s) affected)



```
select *  
from v_clientes_california  
where codigo=101
```

	codigo	apellido	nombre	estado
1	101	Pauli	Ludwig	NY

100 % ▶

Results Messages

Por Qué?

VIEW - WITH CHECK OPTION

miércoles, 7 de julio de 2021 05:10 p. m.

WITH CHECK OPTION realiza un **chequeo de integridad** de los datos a **insertar o modificar**, los cuales deben cumplir con las **condiciones del WHERE** de la vista.

- *No permite que el resultado de una inserción o actualización sea tal que no pertenezca al dominio de la vista.*
- *Utilizado para control de integridad.*

Views

```
CREATE VIEW V_clientes_california_WCK  
(codigo, apellido, nombre,estado)  
AS
```

```
SELECT customer_num, lname, fname, state  
FROM customer  
WHERE state='CA'
```

WITH CHECK OPTION realiza un chequeo de integridad de los datos a insertar o modificar, los cuales deben cumplir con las condiciones del WHERE de la vista.

WITH CHECK OPTION

The screenshot shows a SQL query window with the following content:

```
INSERT INTO v_clientes_california_WCK  
(codigo, apellido, nombre, estado)  
VALUES (999, 'Martin', 'Mihura', 'FL')
```

Below the query, a message window titled "Messages" displays:

```
Msg 660, Level 16, State 1, Line 36  
The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.  
The statement has been terminated.
```

The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.

Views

```
CREATE VIEW V_clientes_california_WCK  
(codigo, apellido, nombre,estado)  
AS
```

```
SELECT customer_num, lname, fname, state  
      FROM customer  
     WHERE state='CA'
```

WITH CHECK OPTION

```
UPDATE v_clientes_california_WCK  
set estado='NY'  
WHERE codigo=101
```

00 %

Messages

Msg 560, Level 16, State 1, Line 15
The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.
The statement has been terminated.

The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.

WITH CHECK OPTION realiza un chequeo de integridad de los datos a insertar o modificar, los cuales deben cumplir con las condiciones del WHERE de la vista.

SNAPSHOT

miércoles, 7 de julio de 2021 05:14 p. m.

- Un **SNAPSHOT** es similar a una view pero realiza una copia de los datos.
 - Por ejemplo, la noche anterior siempre se guarda la ganancia por producto de ese día para consultar esta información al otro día, entonces se realiza un snapshot y se consulta este objeto.
 - Si se realizan ventas a la mañana el snapshot no se actualiza porque es una copia calculada de los datos.
- Los **SNAPSHOT**, también llamados **vistas materializadas** o **tablas summarizadas**.
 - **Snapshots / Materialized Views / Summarized Tables**
- Son **objetos** del esquema de una BD que pueden ser **usados para sumarizar, precomputar, distribuir o replicar datos**. Se **utilizan** sobre todo en **Data Warehouse**, sistemas para soporte de **toma de decisión**, y para **computación móvil y/o distribuida**.
- **Consumen espacio de almacenamiento** en disco.
- Deben ser **recalculadas o refrescadas cuando los datos de las tablas master cambian**. Pueden ser **refrescadas en forma manual o a intervalos de tiempo definidos** dependiendo el motor de BD.

Cláusulas:

- **BUILD DEFERRED**
 - No inserta los datos al momento en que se crea el snapshot, se actualiza al momento en que se ejecute el próximo refresh (crear de nuevo).
- **BUILD IMMEDIATE** (default)
 - Inserta los datos al momento en que se crea el snapshot.
- **REFRESH COMPLETE**
 - Actualiza toda la tabla. el método de refresco será completo, que ejecuta nuevamente la subconsulta.
- **REFRESH FAST**
 - Solo actualiza los campos que sufrieron cambios en la tabla master.

Ej. Objeto Materialized View en Oracle

```
CREATE MATERIALIZED VIEW ST_total_ordenes_clientes
BUILD DEFERRED
REFRESH COMPLETE
AS SELECT A.C_cliente, B.D_apellido, B.D_nombre,
           SUM(A.I_total) AS Total_ordenes
      FROM ordenes A, clientes B
     WHERE A.C_cliente = B.C_cliente
   GROUP BY A.C_cliente, B.D_apellido, B.D_nombre ;
```

JOIN

miércoles, 7 de julio de 2021 05:28 p. m.

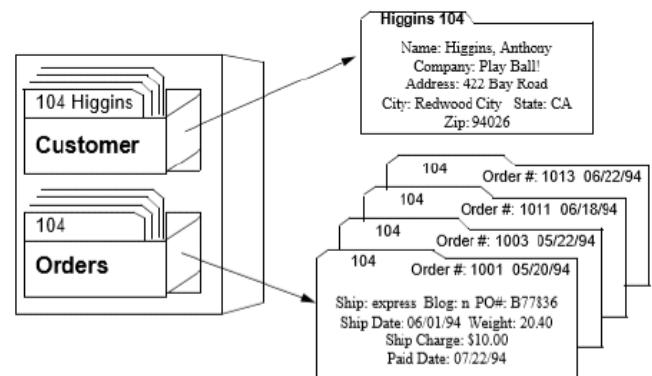
Matcheo de las filas de una tabla que coincidan a través de un atributo o combinación de atributos con filas de otras tabla.

Utilizado si queremos hacer **coincidir filas de dos o más tablas** a partir de un **atributo con valores comunes en las tablas**.

- DML – Data Manipulation Language
 - SELECT
 - INNER JOIN
 - OUTER JOIN

Si queremos hacer coincidir filas de dos o mas tablas a partir de un atributo con valores comunes, por ej. |

A partir del `customer_num` de la tabla `orders` obtener el `lname` y `fname` de cada cliente de la tabla `customer`.



INNER JOIN

miércoles, 7 de julio de 2021 05:36 p. m.

- Se realiza un **matcheo** ó **apareo** de las **filas** de una **tabla** que **coincidan a través de un atributo** o combinación de atributos con **filas** de **otras tablas**.
- **INNER JOIN** solo devolverá las filas que coincidan.
 - nota: si las tablas son muy grandes la query puede ser muy pesada.
 - nota: puedo agregarle condiciones adicionales **WHERE** para restringir y filtrar la consulta.
- **INNER JOIN** se puede utilizar también como **JOIN**.

Ejemplo:

```
SELECT c.customer_num, fname, lname, order_num, order_date
FROM customer c INNER JOIN orders o
          ON (c.customer_num = o.customer_num)
```

	customer_num	fname	lname	order_num	order_date
1	104	Anthony	Higgins	1001	2015-05-16 00:00:00.000
2	101	Ludwig	Pauli	1002	2015-05-17 00:00:00.000
3	104	Anthony	Higgins	1003	2015-05-18 00:00:00.000
4	106	George	Watson	1004	2015-05-18 00:00:00.000
5	116	Jean	Parmeleo	1005	2015-05-20 00:00:00.000
6	112	Margaret	Lavaron	1006	2015-05-26 00:00:00.000
7	117	Arnold	Sipes	1007	2015-05-27 00:00:00.000
8	110	Roy	Jaeger	1008	2015-05-03 00:00:00.000
9	111	Frances	Keyca	1009	2015-06-10 00:00:00.000

Ejemplo: código de producto, total vendido y la descripción de la unidad de medida. Para esto deberemos tomar la **unit_code** de la tabla **products** y el **unit_desc** de la tabla **units**.

	stock_num	manu_code	description	tot_product
1	5	ANZ	tennis racket	3404.00
2	6	ANZ	tennis ball	384.00
3	8	ANZ	volleyball	2520.00
4	9	ANZ	volleyball net	2609.00
5	201	ANZ	golf shoes	675.00
6	205	ANZ	3 golf balls	1248.00
7	304	ANZ	watch	170.00
8	1	HRO	baseball gloves	750.00
9	2	HRO	baseball	252.00
10	4	HRO	football	960.00
11	7	HRO	basketball	600.00
12	304	HRO	watch	280.00
13	309	HRO	ear phones	40.00
14	1	HSK	baseball gloves	800.00
15	3	HSK	baseball bat	720.00
16	4	HSK	football	1920.00
17	110	HSK	helmet	308.00
18	282	KAR	metal woods	2309.00
19	284	KAR	putter	100.00

Eg clave simple

```
SELECT c.customer_num, fname, lname, order_num, order_date
  FROM customer c -- uso alias por vagancia (y ta perfecto)
INNER JOIN orders o -- uso alias por vagancia
  ON (c.customer_num = o.customer_num)-- igualdad entre PK de customer y FK de orders
```

Los campos comunes debo especificar de donde quiero obtener el dato, sino tira error de referencia ambigua.

Eg **clave compuesta**: obtener unit_code de tabla products (+ otros datos)

```
SELECT i.stock_num, i.manu_code,
       SUM(i.unit_price * quantity) tot_producto, unit_code
  FROM items i INNER JOIN products p
    ON (i.stock_num = p.stock_num AND i.manu_code = p.manu_code)
 GROUP BY i.stock_num, i.manu_code, unit_code
```

Eg **múltiples tablas**: obtener también la descripción de la unit

```
SELECT i.stock_num, i.manu_code,
       SUM(i.unit_price * quantity) tot_producto, unit_descr
  FROM items i INNER JOIN products p
    ON (i.stock_num = p.stock_num AND i.manu_code = p.manu_code)
     INNER JOIN units u
    ON (p.unit_code = u.unit_code)
 GROUP BY i.stock_num, i.manu_code, unit_descr
```

INNER JOIN (mas de dos tablas)

Ejemplo 3: cod_cliente, apellido y nombre de cliente, orden de compra, fecha emisión, nro. de stock, código fabricante, nombre_fabricante, descripción tipo producto, nro de ítem, cantidad y precio unitario.

```
SELECT c.customer_num, fname, lname, o.order_num, order_date, item_num,
       i.stock_num, i.manu_code, manu_name, pt.description, item_num,
       quantity, i.unit_price
  FROM orders o JOIN customer c ON (o.customer_num = c.customer_num)
                  JOIN items i ON (o.order_num=i.order_num)
                  JOIN product_types pt ON (i.stock_num=pt.stock_num)
                  JOIN manufact m ON (i.manu_code=m.manu_code)
```

Results

customer_num	fname	lname	order_num	order_date	item_num	stock_num	manu_code	manu_name	description	item_num	quantity	unit_price	
1	104	Anthony	Higgins	1001	2015-05-16 00:00:00.000	1	1	HRO	Hero	baseball gloves	1	1	250.00
2	101	Ludwig	Paul	1002	2015-05-17 00:00:00.000	1	4	HSK	Husky	football	1	1	960.00
3	104	Anthony	Higgins	1003	2015-05-18 00:00:00.000	1	9	ANZ	Anza	volleyball net	1	1	20.00
4	106	George	Watson	1004	2015-05-18 00:00:00.000	1	1	HRO	Hero	baseball gloves	1	1	250.00
5	116	Jean	Pamellee	1005	2015-05-20 00:00:00.000	1	5	NRG	Norge	tennis racquet	1	10	280.00
6	112	Margaret	Lawson	1006	2015-05-26 00:00:00.000	1	5	SMT	Smith	tennis racquet	1	5	125.00
7	117	Arnold	Spies	1007	2015-05-27 00:00:00.000	1	1	HRO	Hero	baseball gloves	1	1	250.00
8	110	Roy	Jaeger	1008	2015-05-03 00:00:00.000	1	8	ANZ	Anza	volleyball	1	1	840.00
9	111	Frances	Keyes	1009	2015-06-10 00:00:00.000	1	1	SMT	Smith	baseball gloves	1	1	450.00
10	115	Alfred	Gundlach	1010	2015-06-19 00:00:00.000	1	6	SMT	Smith	baseball ball	1	1	20.00

INNER JOIN (mas de dos tablas con condiciones adicionales en WHERE)

Ejemplo 4: Mismo ejemplo anterior pero con Condiciones, cliente de nombre Husky, órdenes del mes de mayo del 2015 y precio unitario mayor que 126.

```
SELECT c.customer_num, fname, lname, o.order_num, order_date, item_num,
       i.stock_num, i.manu_code, manu_name, pt.description, item_num,
       quantity, i.unit_price
  FROM orders o JOIN customer c ON (o.customer_num = c.customer_num)
                  JOIN items i ON (o.order_num=i.order_num)
                  JOIN product_types pt ON (i.stock_num=pt.stock_num)
                  JOIN manufact m ON (i.manu_code=m.manu_code)
 WHERE manu_name='Hero' AND MONTH(order_date)=5 AND YEAR(order_date)=2015
       AND unit_price>126
```

Results

customer_num	fname	lname	order_num	order_date	item_num	stock_num	manu_code	manu_name	description	item_num	quantity	unit_price	
1	104	Anthony	Higgins	1001	2015-05-16 00:00:00.000	1	1	HRO	Hero	baseball gloves	1	1	250.00
2	106	George	Watson	1004	2015-05-18 00:00:00.000	1	1	HRO	Hero	baseball gloves	1	1	250.00
3	117	Arnold	Spies	1007	2015-05-27 00:00:00.000	1	1	HRO	Hero	baseball gloves	1	1	250.00
4	117	Arnold	Spies	1007	2015-05-27 00:00:00.000	4	4	HRO	Hero	football	4	1	480.00
5	117	Arnold	Spies	1007	2015-05-27 00:00:00.000	5	7	HRO	Hero	basketball	5	1	600.00

OUTER JOIN

miércoles, 7 de julio de 2021 05:56 p. m.

El **OUTER JOIN** mostrará **todas las filas de la tabla dominante** matcheas o no con la otra tabla.

- El **OUTER JOIN** puede ser:

- **LEFT**
 - (tabla izquierda dominante)
- **RIGHT**
 - (tabla derecha dominante)
- **FULL**
 - (ambas tablas dominantes)

OUR CUSTOMER table has :

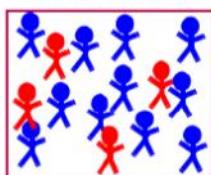


Customers that have placed orders



Customers that have NOT placed orders!

CUSTOMER



■ Join

```
SELECT c.customer_num, fname, lname, COUNT(order_num)
cantOrdenes
FROM customer c INNER JOIN orders o
ON (c.customer_num = o.customer_num)
GROUP BY c.customer_num, fname, lname
```



■ Outer Join

```
SELECT c.customer_num, fname, lname, COUNT(order_num)
cantOrdenes
FROM customer c LEFT JOIN orders o
ON (c.customer_num = o.customer_num)
GROUP BY c.customer_num, fname, lname
```



Ejemplo:

Si quiero obtener los clientes y la cantidad de órdenes de compra de cada uno, si el cliente no tiene ninguna orden de compra no lo va a mostrar con INNER JOIN, por lo que uso LEFT JOIN.

OUTER JOIN

Comparemos resultados.

The screenshot shows two SQL queries in the SSMS interface. Both queries select customer_num, fname, lname, and COUNT(order_num) as cantOrdenes from customer and orders tables. The first query is an INNER JOIN, resulting in 17 rows. The second query is a LEFT JOIN, resulting in 28 rows, including 11 rows where the count of orders is 0. The results are displayed in tables with columns: customer_num, fname, lname, and cantOrdenes. Rows 1 through 17 are identical between both queries. Rows 18 through 28 are highlighted in red boxes in the LEFT JOIN result set.

```
SELECT c.customer_num, fname, lname, COUNT(order_num) cantOrdenes
FROM customer c INNER JOIN orders o
ON (c.customer_num = o.customer_num)
GROUP BY c.customer_num, fname, lname
```

```
SELECT c.customer_num, fname, lname, COUNT(order_num) cantOrdenes
FROM customer c LEFT JOIN orders o
ON (c.customer_num = o.customer_num)
GROUP BY c.customer_num, fname, lname
```

Left Join 28 clientes, 17 con cantOrdenes mayor que 0 y 11 con cantOrdenes=0

Inner Join 17 clientes que compraron.

JOIN AUTO-REFERENCIADO

miércoles, 7 de julio de 2021 06:03 p. m.

JOIN AUTO REFERENCIADO (SELF REFERENCING JOIN)

- Realizar un **JOIN** relacionando a una tabla con sí misma.
- **Ejemplo:** La **tabla customer** tiene un **atributo customer_num_referredBy**, que indica quien fue el cliente que lo referenció. Podríamos armar una consulta que nos diga nombre y apellido del referido y de quien lo referenció.

The screenshot shows a SQL query window titled "SQLQuery1.sql - D...HQEMP6\zaffa (54)*". The query is:

```
SELECT c2.lname+', '+c2.fname Padrino, c1.lname+', '+c1.fname Referido
FROM customer c1 JOIN customer c2
ON (c1.customer_num_referredBy = c2.customer_num)
```

The results pane displays the output of the query:

	Padrino	Referido
1	Paul, Ludwig	Sader, Carole
2	Currie, Philip	Currie, Philip
3	Higgins, Anthony	Higgins, Anthony
4	Vector, Raymond	Vector, Raymond
5	Watson, George	Watson, George
6	Quinn, Donald	Quinn, Donald
7	Miller, Jane	Miller, Jane
8	Lawson, Margaret	Lawson, Margaret
9	Beatty, Lana	Beatty, Lana
10	Albertson, Frank	Albertson, Frank
11	Pamelee, Jean	Pamelee, Jean
12	Sipes, Arnold	Sipes, Arnold
13	Baxter, Dick	Baxter, Dick
14	Shorter, Bob	Shorter, Bob
15	Wallack, Jason	Wallack, Jason
16	O Brian, Cathy	O Brian, Cathy
17	Hanlon, Marvin	Hanlon, Marvin

At the bottom left, there is a message: "Query executed successfully." At the bottom right, there are status bars: "DESKTOP-MHQEMP6\SQLEXPRESS ... | DESKTOP-MHQEMP6\zaffa ... | stores7NewVersion | 00:00:00 | 21 rows".

PRODUCTO CARTESIANO

miércoles, 7 de julio de 2021 06:12 p. m.

- El producto cartesiano es **muy costoso** para el **motor de base de datos**. **RECOMENDACIÓN NO UTILIZARLO**.

- En el caso que se deba realizar un producto cartesiano se debe **tratar de achicar el working set lo máximo posible**.

- **Proyectando solo las columnas que necesiten y condicionando con WHERE lo que pudiesen.**

- **Ejemplo:**

- **Si la tabla customer tiene 28 filas y 12 columnas; y la tabla orders 23 filas y 10 columnas tendría como resultado 644 (o sea $28 \times 23 = 644$) filas de 22 (o sea $22 = 12 + 10$) columnas.**

The screenshot shows a SQL query window with the following content:

```
SELECT *
FROM customer c, orders o
```

Handwritten notes on the right side of the screen:

Clientes (12 cols)+Ordenes(10 cols)= 22 columnas
28 clientes * 23 ordenes =644 filas
Que pasaría si fuesen 1000 clientes con 100000 ordenes?

The results grid displays data from the customer and orders tables. The columns are:

	customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone	customer_num_referredBy	status	order_num	order_date
1	101	Ludwig	Feuli	All Sports Supplies	213 Erdwid Court		Sunnyvale	CA	94086	408-789-8075	NULL	NULL	1001	2015-05-16 00:00:00.000
2	102	Carole	Sedler	Sports Spot	785 Geary St		San Francisco	CA	94117	415-822-1289	101	NULL	1001	2015-05-16 00:00:00.000
3	103	Philip	Cunne	Phil's Sports	654 Poplar	P. O. Box 3498	Palo Alto	CA	94303	415-328-4543	101	NULL	1001	2015-05-16 00:00:00.000
4	104	Anthony	Higgins	Play Ball!	East Shopping Cntr.	422 Bay Road	Redwood City	CA	94026	415-368-1100	103	NULL	1001	2015-05-16 00:00:00.000
5	105	Raymond	Vector	Lee Atnos Spotts	1899 La Loma Drive		Los Altos	CA	94022	415-776-3249	103	NULL	1001	2015-05-16 00:00:00.000
6	106	George	Watson	Watson & Son	1143 Carver Place		Mountain View	CA	94063	415-389-8789	103	NULL	1001	2015-05-16 00:00:00.000
7	107	Charles	Rream	Athletic Supplies	41 Jordan Avenue		Palo Alto	CA	94304	415-356-9076	NULL	NULL	1001	2015-05-16 00:00:00.000
8	108	Donald	Gunn	Quinn's Sports	587 Avenida		Redwood City	CA	94063	415-544-8729	107	NULL	1001	2015-05-16 00:00:00.000
9	109	Jane	Miller	Sport Stuff	Mayfair Mart	7345 Ross Blvd.	Sunnyvale	CA	94086	408-723-8789	107	NULL	1001	2015-05-16 00:00:00.000
10	110	Roy	Jaeger	AA Athletics	520 Topaz Way		Redwood City	CA	94062	415-743-3611	NULL	NULL	1001	2015-05-16 00:00:00.000
11	111	Frances	Keyes	Sports Center	3199 Sterling Court		Sunnyvale	CA	94085	408-277-7245	NULL	NULL	1001	2015-05-16 00:00:00.000
12	112	Margaret	Lawson	UTN	234 Wyndotte Way		Los Altos	CA	94022	5555-5555	111	NULL	1001	2015-05-16 00:00:00.000
13	113	Lana	Beatty	Sportstown	654 Oak Grove		Menlo Park	CA	94025	415-356-9982	111	NULL	1001	2015-05-16 00:00:00.000
14	114	Frank	Albertson	Sporting Place	947 Waverly Place		Redwood City	CA	94062	415-886-6577	111	NULL	1001	2015-05-16 00:00:00.000
15	115	Alfred	Grant	Gold Medal Sports	776 Gary Avenue		Menlo Park	CA	94025	415-356-1123	NULL	NULL	1001	2015-05-16 00:00:00.000
16	116	Jean	Farnelle	Olympic City	1104 Spinosa Drive		Mountain View	CA	94040	415-534-8522	115	NULL	1001	2015-05-16 00:00:00.000
17	117	Ansel	Ornitz	Vista Marinas	9801 Lomas Circle		Mountain View	CA	94035	415-546-8279	116	NULL	1001	2015-05-16 00:00:00.000

Query executed successfully.

Transactions

miércoles, 7 de julio de 2021 06:20 p. m.

- **Conjunto de sentencias SQL que se ejecutan atómicamente** (indivisibles) en una unidad lógica de trabajo, sentencias que **se ejecutan en bloque**.
- Los DBMS cuentan con distintos **mecanismos** para poder **asegurar la consistencia de los datos** existentes en las Bases de Datos. Las **Transacciones** son uno de ellos.
 - Partiendo de que **una transacción lleva la base de datos de un estado correcto a otro estado correcto**, el **motor** posee mecanismos de manera de **garantizar** que **la operación completa se ejecute o falle**, no permitiendo que queden datos inconsistentes.

Conceptos relacionados

- **Logs Transaccionales**
 - Registro donde se almacena información de cada operación ejecutada.
- **Recovery**
 - Mecanismo para recuperación de datos ante caídas.

Propiedades de las RDBMS en base a las transacciones A-C-I-D

- **Atomicidad**
 - Una transacción se ejecuta completa o no se ejecuta
- **Consistencia**
 - Parte de un estado consistente y termina en un estado consistente.
- **Isolation (Aislamiento)**
 - Si diferentes transacciones utilizan por separado la misma tabla de forma concurrente, no se chocan sino que una se realiza antes que la otra.
- **Durabilidad**
 - Al hacer un commit el dato dura eternamente, se persiste en la base de datos hasta que se modifique o se borre.

Sentencias de una transacción

- **BEGIN TRANSACTION**
 - Marca inicio de las sentencias que se ejecutarán de forma atómica. El final se indica con END.
- **COMMIT TRANSACTION**
 - Actualiza los datos en la BD. Confirma la actualización de los datos en disco.
- **ROLLBACK TRANSACTION**
 - Deshace la transacción. Hace las operaciones opuestas a las realizadas en la transacción.
 - nota: debe indicarse de forma explícita.

Mecanismos de recuperación (RECOVERY)

- Es un **mecanismo provisto** por los **DBMS** que **se ejecuta en cada inicio del motor de forma automática** como dispositivo de tolerancia a fallas.
- Sus objetivos son los siguientes:
 - **Retornar al motor al punto consistente más reciente (checkpoint).**
 - (checkpoint = punto en el que el motor sincronizó memoria y disco)
 - Utiliza los **logs transaccionales** para **retornar el motor de base de datos a un estado lógico consistente.**
 - *Realizando un “rolling forwards” de las transacciones ocurridas con éxito luego del checkpoint más reciente y realizar un “rolling back” de las transacciones que no hayan sido exitosas.*

Transactions - Ejemplos

miércoles, 7 de julio de 2021 06:51 p. m.

Para nuestro ejemplo, se realiza una **inserción en la tabla de entregas** con la **factura entregada** y luego se **calcula el importe** a partir de una función.

Si el **importe** es **mayor a 0** la **factura se inserta en la tabla de facturas** y la **transacción se confirma** con un **COMMIT**.

En **caso contrario** se **imprime el mensaje de error** y la **transacción se vuelve al estado inicial** mediante un **ROLLBACK**.

Ejemplo:

```
--Declaración de variables
DECLARE @IMPORTE REAL;
DECLARE @FACTURA INT;
DECLARE @FECHA VARCHAR(8);
DECLARE @CLIENTE INT;

--Harcoddeo de inicalización
SET @FACTURA = 353;
SET @FECHA = '20120601';
SET @CLIENTE = 15;

BEGIN TRANSACTION --Comienzo de transacción

INSERT INTO entregas(numero,fecha,cliente)
VALUES (@FACTURA,@FECHA,@CLIENTE)

SET @IMPORTE = Calcular_Importe(@FACTURA)

IF @IMPORTE > 0 THEN
    INSERT INTO facturas(n_factura,imp_total)
    VALUES (@FACTURA,@IMPORTE)
    COMMIT TRANSACTION --Se insertan los datos en la base
ELSE
    PRINT 'ERROR! IMPORTE NO VALIDO'
    ROLLBACK TRANSACTION --Se vuelven al punto de inicio
END
```

Transacción anidada

miércoles, 7 de julio de 2021 06:57 p. m.

- Algunos motores de base de datos permiten el manejo de transacciones anidadas.
- En nuestro caso, la transacción T2 finaliza correctamente dentro de T1; pero en el momento de confirmar la transacción T1 se produce un rollback que deshace también a la T2 volviendo a la tabla a un estado inicial sin valores cargados.

Ejemplo:

```
CREATE TABLE #numeros (num int)

BEGIN TRAN T1

    INSERT INTO #numeros VALUES (1)

    BEGIN TRAN T2
        INSERT INTO #numeros VALUES (2)
    COMMIT TRAN --Confirma T2

ROLLBACK TRAN--Deshace T1 que contiene a T2; entonces también la deshace
```

Save Transaction

miércoles, 7 de julio de 2021 07:00 p. m.

Algunos motores de base de datos permiten **establecer puntos intermedios de guardado de información**. Solo se rollbackea una porción de la transacción.

Es posible realizar más de un **SAVE TRAN** en cada transacción y se puede elegir a cual se desea volver en cual momento.

Ejemplo:

```
CREATE TABLE #numeros (num int)

BEGIN TRAN

INSERT INTO #numeros VALUES (2)

SAVE TRAN N2 --Guardo estado actual a N2

BEGIN TRAN
    INSERT INTO #numeros VALUES (3)
ROLLBACK TRAN N2 --Deshago la transacción actual hasta N2

INSERT INTO #numeros VALUES (4)

COMMIT TRAN
```

Índices

miércoles, 7 de julio de 2021 16:33

Objeto de base de datos que **permite un acceso directo** a los **datos de las tablas**.

- Se pueden crear **distintos tipos** de índices **sobre uno o más campos**.
- Son **estructuras opcionales** asociadas a una tabla.
- Son **lógica y físicamente independientes** de los **datos en la tabla asociada**.
- Se puede **crear o borrar un índice en cualquier momento sin afectar** a las **tablas base o a otros índices**.

La función de los **índices** se puede analizar por:

- **Optimización**
 - Permite un acceso más rápido a los **datos de una tabla**.
- **Integridad de Entidad**
 - Permite chequear la integridad de los datos.
- **Unicidad**
 - Asegura la unicidad de los datos.

Tipos de Índices

miércoles, 7 de julio de 2021 16:45

- **Btree Index**

- Basado en Árbol-B
- Estructura de índice **estándar y más utilizada.**
- Tantas búsquedas como niveles.
- **Misma cantidad de lecturas para un dato que exista o no**, es importante porque nos **asegura previsibilidad**
- *Existe una estructura de índice llamada B Tree+ que permite búsqueda secuencial entre las hojas de un mismo nivel, esto agiliza mucho las búsquedas secuenciales.*

- **Btree Cluster Index**

- Al crearse sobre una tabla, **los datos de la tabla son ordenados físicamente por el mismo.**
 - (Informix / SQLServer / DB2)
- **Único por tabla.**
- **Cuando creamos una PK en una tabla le asigna un índice de este tipo, por eso se muestra ordenado en las queries cuando no se le aplica ningun ORDER BY.**

- **Bitmap Index (Oracle)**

- **Cada bit** en el Bitmap corresponde a una **fila en particular.**
- Son utilizados para **pocas claves con muchas repeticiones.**
- Si el **bit esta en on** significa que **la fila con el correspondiente row id tiene el valor de la clave**
 - Ej: índice sobre estado civil (5 estados para muchas personas)

- **Hash Index (MySql)**

- Están **implementados en tablas de hash** y se basan en otros índices Btree

existentes para una tabla.

- Utilizados para tablas que están íntegramente en memoria, es la opción mas rápida de acceso.

- **Functional Index / Function based Index**

- Son índices **cuya clave deriva del resultado de una función**.
- En general las **funciones deben ser funciones definidas por un usuario**.

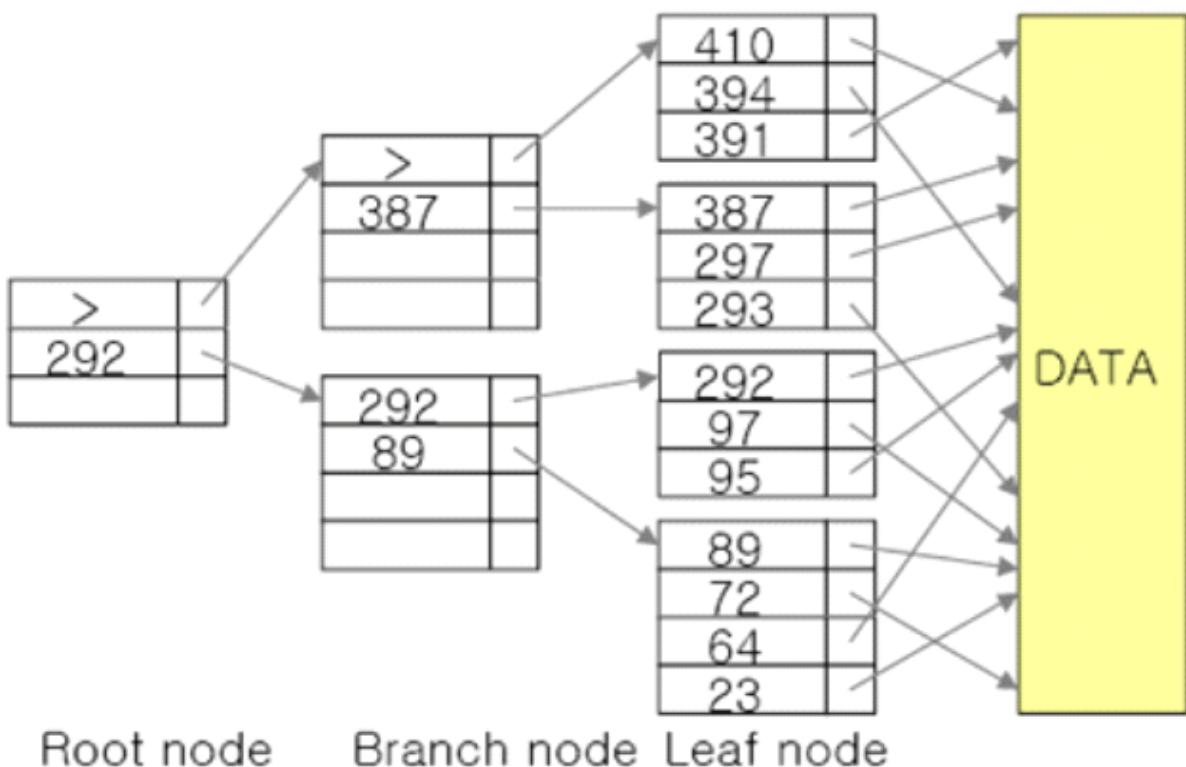
- **Reverse Key Index (Oracle)**

- **Invierte los bytes de la clave** a indexar.
- Esto sirve para los **índices cuyas claves son una serie constante** (ejemplo Crecimiento ascendente) para que las inserciones se distribuyan por todas las hojas del árbol de índice.

Ejemplos Índices Arbol-B

miércoles, 7 de julio de 2021 17:03

Indices B Tree

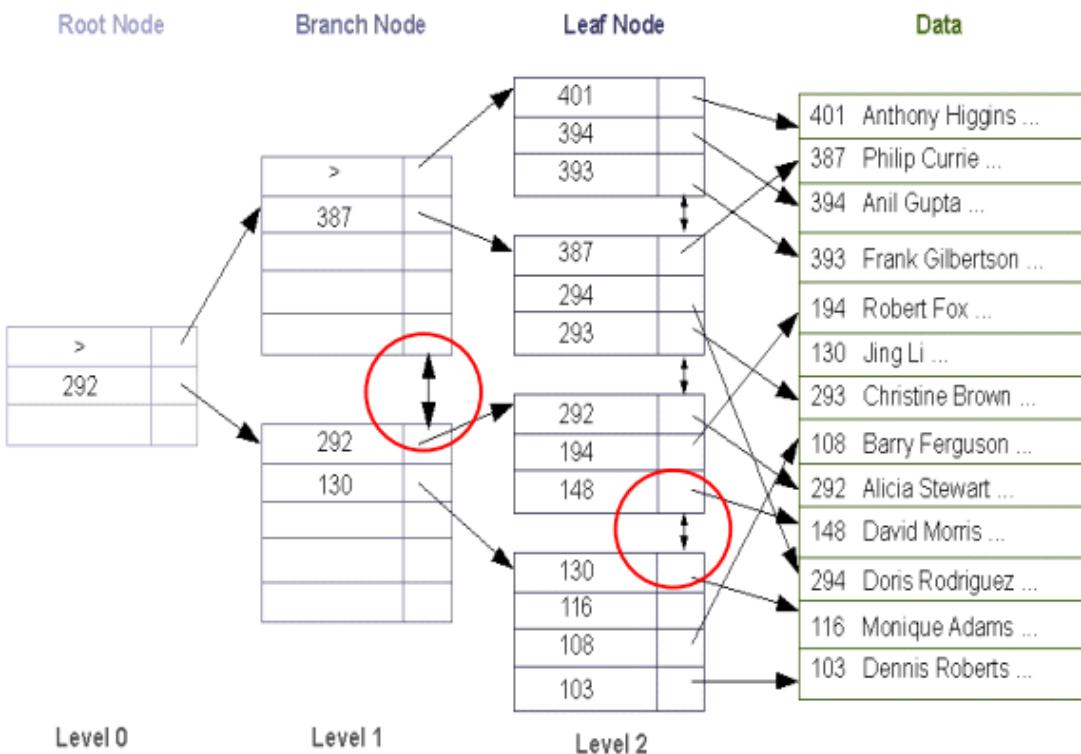


Root node

Branch node

Leaf node

Indices B Tree +



Level 0

Level 1

Level 2

Características Diferenciadoras para Índices

miércoles, 7 de julio de 2021 17:04

- **Unique**

- Índice de clave única.
- Sólo admite una fila por clave.**
- Casos de **PK** y Constraint **UNIQUE**
- Estos se usan para controlar la Integridad de Entidades**

- **Duplicado**

- Permite **múltiples filas para una misma clave.**
- Ejemplo
 - Crearle un índice sobre el campo `id_cliente` (FK) de la tabla Facturas, que referencia al campo `id_cliente` (PK) de la tabla Clientes.
 - `Id_cliente` estará repetido en varias filas de las tabla Facturas

Desde la composición del Índice:

- **Simple**

- La **clave** está **integrada** por **una sola columna**.
- Ej: `id_cliente` es el único campo de una PK en tabla Clientes

- **Compuesto**

- La **clave** está **integrada** de **varias columnas**.
- Ej: `nro_orden` y `nro_cliente` conforman la PK de la tabla Ordenes.
- Las **principales funciones** de un índice compuesto son:
 - **Facilitar múltiples joins entre columnas**
 - **Incrementar la unicidad del valor de los índices**

Sintaxis Indice

miércoles, 7 de julio de 2021 17:17

CREATE INDEX ix_sample

ON sample_table (a,b,c); -- Tengo que buscar por a primero, luego b, luego c. No puedo saltar.

Motor SQL SERVER

Sql Server utiliza una estructura de Arbol B+.

- Máxima cantidad de campos para la clave de un índice compuesto: 16.

Creación de un **índice único y simple**

CREATE UNIQUE INDEX ix1_ordenes ON ordenes (n_orden);

Creación de **Indice duplicado y compuesto**

CREATE INDEX ix2_ordenes ON ordenes (n_cliente, f_orden);

Creación de **Indice clustered**

CREATE CLUSTERED INDEX ix3_ordenes ON ordenes (N_orden);

Beneficios de usar Índices

miércoles, 7 de julio de 2021 17:17

- **Mejor performance en el acceso a datos**
 - No hace lecturas secuenciales, aprovecha la complejidad logarítmica.
 - Hace búsquedas n-arias (y no busquedas binarias) consiguiendo muchísima rapidez en los accesos a datos.
- **Mejor performance en el ordenamiento de filas**
 - Los indices ya proveen un ordenamiento previo evitandole al motor tener que realizar ese ordenamiento.
- **Asegura únicos valores para las filas almacenadas**
 - El índice creado sobre un campo con constraint UNIQUE es un ejemplo de esto.
- **Mejor performance en joins si puede utilizarlos para obtener los datos.**
 - En los casos de campos duplicados (FK) se optimiza el tiempo de búsqueda en los joins.
- **Asegura el cumplimiento de constraints y reglas de negocio**
 - Primary key
 - Foreign keys
 - Unique values.

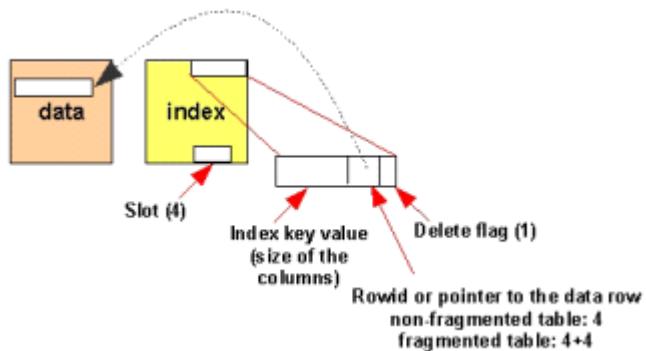
nota: por la velocidad puede cumplir con las búsquedas que aseguran la integridad referencial.

Costos de Utilización de Indices

miércoles, 7 de julio de 2021 17:21

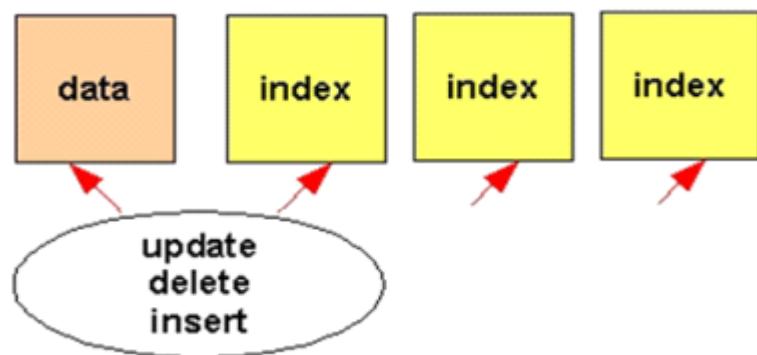
• COSTO DE ESPACIO DE DISCO

En la mayoría de los casos el tamaño del índice suele ser mayor al que ocupan los datos indexados por el índice.



• COSTO DE PROCESAMIENTO Y MANTENIMIENTO

Cada vez que una fila es insertada o modificada o borrada, el sistema deberá recorrer y actualizar los distintos índices.



Cuando se compara entre los costos y los beneficios de utilizar Índices estos últimos hacen mucho más la diferencia, salvo casos de manejos de plataformas de Big Data.

Cuando deberíamos Indexar

miércoles, 7 de julio de 2021 17:36

- Usar en columnas donde
 - uso **Joins**
 - frecuentemente se realizan **filtros**
 - frecuentemente se usan en **orders by**
 - hay **pocos valores diferentes**
 - Ej: Sexo, Estado Civil, Etc.
 - **quiero evitar duplicación de índices**, uso compuestos
 - **la longitud de atributo sea pequeña**
- Verificar que el **tamaño de índice debería ser pequeño comparado con la fila**.
- **No crear sobre tablas con poca cantidad de filas**
 - siempre se recupera de a páginas, evitamos que el sistema lea el árbol de índices
- **Limitar la cantidad** de índices en **tablas que son actualizadas frecuentemente**, sobre estas tablas se estarán ejecutando Selects extras
- Tratar de **usar índices compuestos para incrementar los valores únicos**.
- **Si una o más columnas intervienen en un índice compuesto el optimizador podría decidir acceder a través de ese índice**
 - nota: pasa aunque sea solo para la búsqueda de los datos de una columna, esto se denomina “partial key search”
- **Usando cluster index se agiliza la recuperación de filas**
 - nota: las filas serían almacenadas en bloques contiguos, facilitando el acceso y reduciría la cantidad de accesos, recuperando en menos páginas los mismos datos.

Construcción de Indices

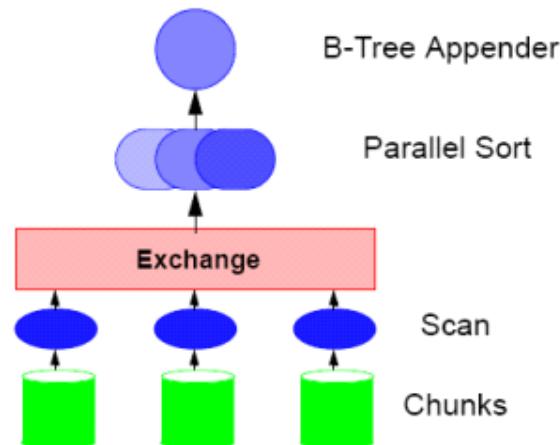
miércoles, 7 de julio de 2021 17:41

Los motores de BD **usan** en general **métodos de construcción de índices en paralelo**.

- Se **ordenan los datos, dividen en partes y múltiples threads crean el árbol**.

El arbol B+ es construido por 2 o más procesos paralelos. Para esto el motor realiza una muestra de la filas a Indexar (aproximadamente 1000) y luego decide como separar en grupos. Luego scanea las filas y las ordena usando el mecanismo de sort en paralelo.

Las claves ordenadas son colocadas en los grupos apropiados para luego ir armando en paralelo un subárbol por cada grupo. Al finalizar los subárboles se unen en un único Arbol B+.



FILL factor (o load factor)

Porcentaje de llenado de cada nodo del árbol al crearlo. Indica el espacio a ser dejado libre para datos adicionales.

Ejemplo

el FILLFACTOR=20, en la creación del índice se ocupará hasta el 80% de cada nodo.

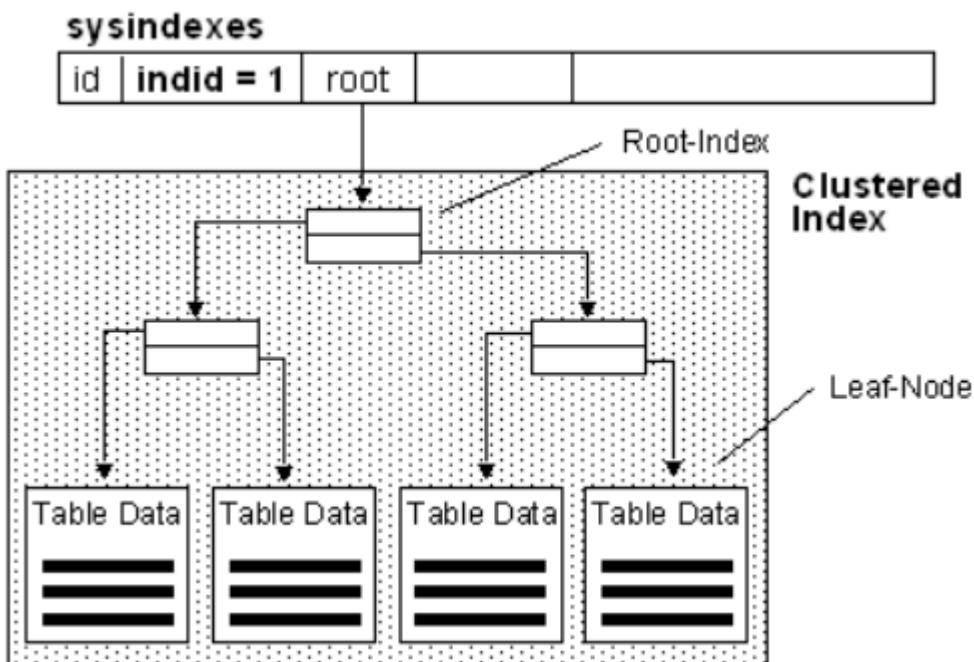
```
CREATE UNIQUE INDEX ix1_ordenes
```

```
ON ordenes(N_orden)
```

```
WITH FILLFACTOR = 20;
```

Indices Clustered

miércoles, 7 de julio de 2021 17:49



Mete la tabla dentro del índice y **guarda directamente las filas completas en las hojas de la tabla.**

nota: **es más rápido en SQL Server**, normalmente las hojas apuntan a la fila, la cual está en una tabla desorganizada, pero en este caso los datos de la tabla ya están organizados igual al índice.

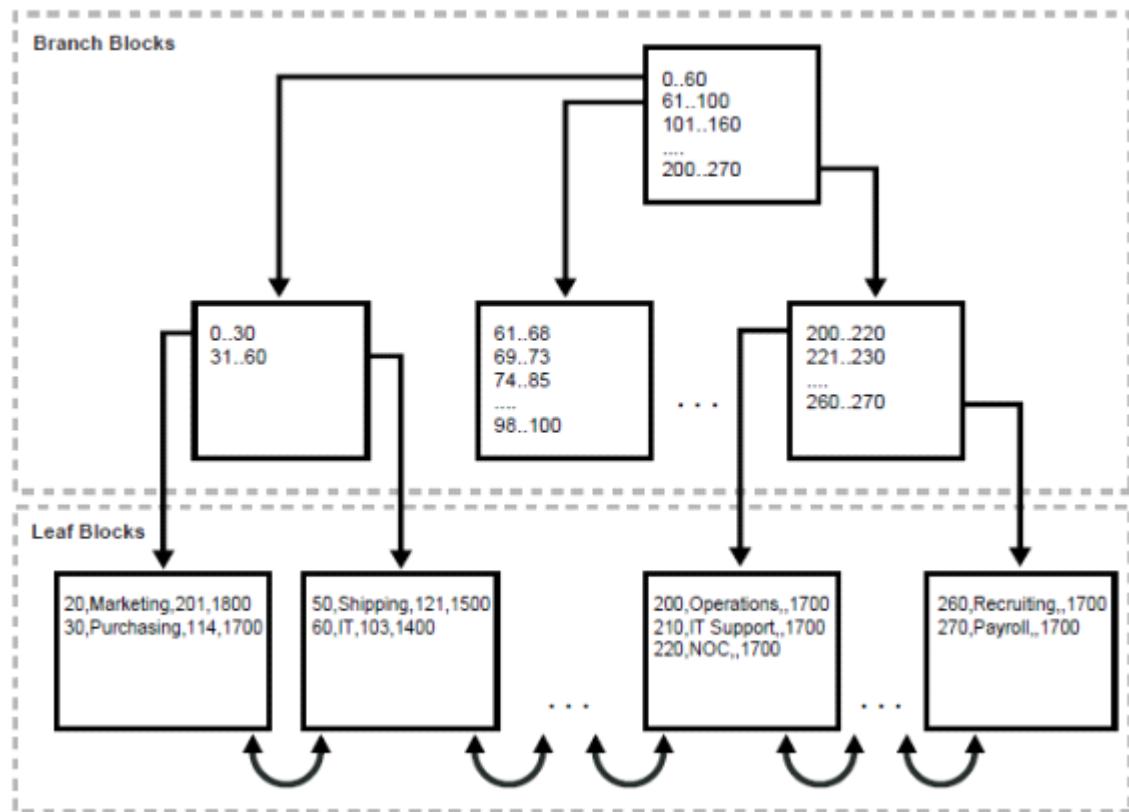
Tablas Organizadas por Índice

(IOT - Index-Organized Tables) (Ora)

Motor Oracle

La siguiente sentencia crea la tabla ordenes, como una tabla organizada por índice, indicando que la columna N_cliente divide el área del índice, de las columnas no clave.

Equivalente a los **clustered index** del SQL Server.



```
CREATE TABLE ordenes
```

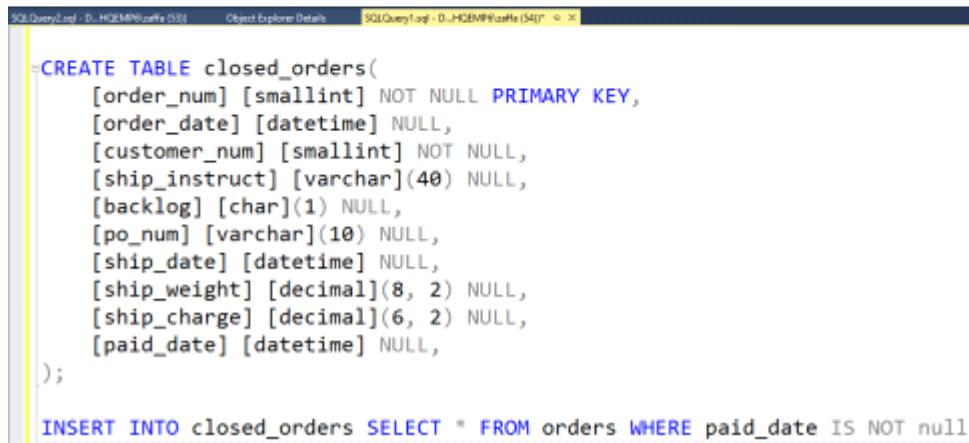
```
( N_orden NUMBER PRIMARY KEY,
  N_cliente NUMBER,
  F_orden DATE,
  C_estado NUMBER,
  F_alta_audit DATE,
  D_usuario VARCHAR2(20) )
ORGANIZATION INDEX INCLUDING N_CLIENTE OVERFLOW
```

SubQueries en INSERT

miércoles, 7 de julio de 2021 17:53

Se insertan las filas que devuelve el SELECT en una tabla previamente creada con la misma estructura.

```
INSERT INTO closed_orders
SELECT * FROM orders
WHERE paid_date IS NOT null
```



The screenshot shows a SQL query window in SSMS with two tabs: 'Object Explorer Details' and 'SQLQuery1.sql'. The code in the window is:

```
CREATE TABLE closed_orders(
    [order_num] [smallint] NOT NULL PRIMARY KEY,
    [order_date] [datetime] NULL,
    [customer_num] [smallint] NOT NULL,
    [ship_instruct] [varchar](40) NULL,
    [backlog] [char](1) NULL,
    [po_num] [varchar](10) NULL,
    [ship_date] [datetime] NULL,
    [ship_weight] [decimal](8, 2) NULL,
    [ship_charge] [decimal](6, 2) NULL,
    [paid_date] [datetime] NULL,
);
INSERT INTO closed_orders SELECT * FROM orders WHERE paid_date IS NOT null
```

SubQueries en DELETE

miércoles, 7 de julio de 2021 17:56

Utilizando tres subqueries como **condiciones** del **DELETE**, podríamos borrar todos los clientes de la Tabla customer que:

1. No posean ordenes de compra asociadas
 2. Que no hayan referenciado a otro cliente
 3. Que no tengan llamados telefónicos.

DELETE FROM customer

WHERE customer num NOT IN

(SELECT DISTINCT customer_num FROM cust_calls)

AND customer_num NOT IN

(SELECT DISTINCT customer_num FROM orders)

AND customer_num NOT IN

(SELECT DISTINCT customer_num_referredBy FROM customer c2

WHERE customer_num_referredBy IS NOT NULL)

SubQueries en UPDATE

miércoles, 7 de julio de 2021 18:00

UPDATE FROM customer

SET columna = -- <subquery>

WHERE columna (=/IN/NOT IN) -- <subquery>

(EXISTS / NOT EXISTS) -- <subquery>

- **En cláusula SET**

UPDATE FROM #clientesParaBorrar

SET columna = (SELECT state FROM state WHERE sname = 'Florida') -- subquery

WHERE customer_num = 101

Debe retornar una única columna y fila, un valor escalar

- **En cláusula WHERE**

UPDATE manufact **SET** lead_time = 15

WHERE manu_code IN (SELECT DISTINCT manu_code FROM items)

Puede retornar múltiples valores, pero debo usar IN .

SubQueries en SELECT

miércoles, 7 de julio de 2021 18:03

```
SELECT col, col, -- <subquery>
FROM table JOIN -- <subquery>
WHERE columna (=/<=/>=/IN/NOT IN) -- <subquery>
(EXISTS / NOT EXISTS) -- <subquery>
GROUP BY ...
HAVING fxAgregada (=/<=/>=/IN/NOT IN) -- <subquery>
(EXISTS / NOT EXISTS) -- <subquery>
ORDER BY -- <subquery>
```

- **En lista de columnas**

```
SELECT customer_num, count(order_num) cantOcCliente,
(SELECT COUNT (*) FROM orders) cantTotalOrdenes -- subquery
FROM orders
GROUP BY customer number
```

nota: devuelve la cantidad de ordenes como variable y las muestra en cada fila

- **En FROM**

```
SELECT lname apellido, fname nombre, cliente, cantidad
FROM customer c1 JOIN
(SELECT customer_num cliente, count(order_num) cantidad --
subquery
FROM orders
GROUP BY customer_num) c2
ON (c1.customer_num = c2.cliente)
WHERE cantidad > 3
```

- **En WHERE**

```
SELECT lname, '+fname, customer_num
FROM customer
```

```

WHERE customer_num IN
(SELECT customer_num FROM cust_calls
GROUP BY customer num HAVING COUNT (*)>1) -- subquery

```

nota: muchas veces estas se pueden resolver con joins

nota2: con = obtengo valor escalar

- Correlacionado

```
SELECT customer_num, lname, fname
```

```
FROM customer c
```

```
WHERE NOT EXISTS
```

```
(SELECT order_num FROM orders o where o.customer_name =
c.customer_num) -- subquery relacion
```

SQL – SubQuery en SELECT

listar las ordenes de los clientes con su total comprado pero sólo para los que el total de la orden sea mayor que el promedio comprado de dicho cliente.

The screenshot shows a SQL query being run in SQL Server Management Studio. The query uses a subquery in the FROM clause to calculate the average purchase amount per customer. It then filters the main query to only include customers whose total purchase amount is greater than their average purchase amount.

```

SQLQuery1.sql - D...HQEMP6\zaffa (53)*  ↗ X
SELECT o.order_num, o.customer_num, SUM(quantity*unit_price) totalOrden
    ,promedio
FROM orders o JOIN items i ON o.order_num=i.order_num
JOIN (SELECT customer_num, SUM(quantity*unit_price)
      /COUNT(DISTINCT o2.order_num) promedio
      FROM orders o2 JOIN items i2 ON (o2.order_num=i2.order_num)
      GROUP BY customer_num) tabSQ
      ON (o.customer_num=tabSQ.customer_num)
GROUP BY o.order_num, o.customer_num, tabSQ.promedio
HAVING SUM(i.quantity*i.unit_price) > tabSQ.promedio

```

	order_num	customer_num	totalOrden	promedio
1	1003	104	1355.00	570.950000
2	1008	110	1340.00	895.000000
3	1014	106	1440.00	1428.000000
4	1012	117	2840.00	2268.000000

Resolución con subquery en FROM

Operador UNION

miércoles, 7 de julio de 2021 18:13

Características

- Cantidad y posición de campos deben corresponderse
- ORDER BY debe ir al final del último SELECT.
- Solo se puede ordenar por posición de columnas.
- Si los valores son idénticos deja una sola de las dos filas.

```
SELECT stock_num,manu_code  
FROM products  
WHERE unit_price < 25  
UNION  
SELECT stock_num,manu_code  
FROM items  
WHERE stock_num=5  
ORDER BY 1,2
```

```
SELECT stock_num,manu_code  
FROM products  
WHERE unit_price < 25
```

```
SELECT stock_num,manu_code  
FROM items  
WHERE stock_num=5
```

	stock_num	manu_code
1	5	ANZ
2	9	ANZ
3	103	PRC
4	106	PRC
5	302	HRO
6	302	KAR

```
SELECT stock_num,manu_code  
FROM products  
WHERE unit_price < 25  
UNION  
SELECT stock_num,manu_code  
FROM items  
WHERE stock_num=5  
ORDER BY 1,2
```

Results

	stock_num	manu_code
1	5	ANZ
2	5	NRG
3	5	SMT
4	9	ANZ
5	103	PRC
6	106	PRC
7	302	HRO
8	302	KAR

	stock_num	manu_code
1	5	NRG
2	5	SMT
3	5	ANZ
4	5	ANZ
5	5	ANZ
6	5	NRG
7	5	ANZ
8	5	ANZ
9	5	SMT

```

SELECT stock_num PROD,manu_code FAB FROM products
WHERE unit_price < 25
UNION
SELECT stock_num,manu_code FROM items WHERE stock_num=5
ORDER BY 1,2

```

```

-SELECT stock_num PROD,manu_code FAB FROM products
WHERE unit_price < 25
UNION
SELECT stock_num,manu_code FROM items WHERE stock_num=5
ORDER BY 1,2

```

Results

PROD	FAB
5	ANZ
5	NRG
5	SMT
9	ANZ
103	PRC
106	PRC
302	HRO
302	KAR

La tabla de salida toma los nombres de columnas del primer SELECT.

```

SELECT 1 orden, customer_num, order_num, order_date
FROM orders WHERE customer_num=127
UNION
SELECT 3 orden, customer_num, order_num, order_date
FROM orders WHERE customer_num BETWEEN 111 AND 126
UNION
SELECT 2 orden, customer_num, order_num, order_date
FROM orders WHERE customer_num BETWEEN 1 AND 110
ORDER BY 1 ASC, 2 DESC

```

En este caso particular vemos que utilizamos un ordenamiento de filas determinado por una constante en cada select que dice que filas irán primero en la salida.

Operador UNION ALL

miércoles, 7 de julio de 2021 18:20

```
SELECT stock_num,manu_code FROM products WHERE unit_price < 25  
UNION ALL
```

```
SELECT stock_num,manu_code FROM items WHERE stock_num=5  
ORDER BY 1,2
```

SELECT stock_num,manu_code FROM products WHERE unit_price < 25																																
UNION ALL																																
SELECT stock_num,manu_code FROM items WHERE stock_num=5 ORDER BY 1,2																																
Results Messages																																
<table border="1"><thead><tr><th>stock_num</th><th>manu_code</th></tr></thead><tbody><tr><td>5</td><td>ANZ</td></tr><tr><td>5</td><td>ANZ</td></tr><tr><td>5</td><td>ANZ</td></tr><tr><td>5</td><td>ANZ</td></tr><tr><td>5</td><td>ANZ</td></tr><tr><td>5</td><td>ANZ</td></tr><tr><td>5</td><td>NRG</td></tr><tr><td>5</td><td>NRG</td></tr><tr><td>5</td><td>SMT</td></tr><tr><td>5</td><td>SMT</td></tr><tr><td>9</td><td>ANZ</td></tr><tr><td>103</td><td>PRC</td></tr><tr><td>106</td><td>PRC</td></tr><tr><td>302</td><td>HRO</td></tr><tr><td>302</td><td>KAR</td></tr></tbody></table>	stock_num	manu_code	5	ANZ	5	NRG	5	NRG	5	SMT	5	SMT	9	ANZ	103	PRC	106	PRC	302	HRO	302	KAR										
stock_num	manu_code																															
5	ANZ																															
5	ANZ																															
5	ANZ																															
5	ANZ																															
5	ANZ																															
5	ANZ																															
5	NRG																															
5	NRG																															
5	SMT																															
5	SMT																															
9	ANZ																															
103	PRC																															
106	PRC																															
302	HRO																															
302	KAR																															

El Operador UNION con la cláusula ALL, repite en la estructura de salida todas las filas de las tablas involucradas, sean o no iguales.

Operador INTERSECT

miércoles, 7 de julio de 2021 18:20

Devuelve **filas que están en ambas tablas**.

- **Mismas restricciones que UNION.**

```
SELECT stock_num,manu_code  
FROM products  
WHERE unit_price < 25  
INTERSECT  
SELECT stock_num,manu_code  
FROM items  
WHERE stock_num=5  
ORDER BY 1,2
```

Mismas restricciones del UNION.

Devuelve las filas que están en ambas consultas.

```
SELECT stock_num,manu_code  
FROM products  
WHERE unit_price < 25  
INTERSECT  
SELECT stock_num,manu_code  
FROM items  
WHERE stock_num=5  
ORDER BY 1,2
```



Results	
stock_num	manu_code
5	ANZ

Operador EXCEPT

miércoles, 7 de julio de 2021 18:21

Devuelve las **filas del Primer SELECT que no están en la intersección con el Segundo SELECT.**

- **Mismas restricciones del UNION.**

```
SELECT stock_num,manu_code
FROM products
WHERE unit_price < 25
EXCEPT
SELECT stock_num,manu_code
FROM items
WHERE stock_num=5
ORDER BY 1,2
```

The screenshot shows a SQL query results window. At the top, there are tabs for 'Results' and 'Messages'. The 'Results' tab is selected, displaying a table with two columns: 'stock_num' and 'manu_code'. The data rows are:

stock_num	manu_code
9	ANZ
103	PRC
106	PRC
302	HRO
302	KAR

Ejemplos Operadores

miércoles, 7 de julio de 2021 18:23

```
SELECT stock_num,manu_code  
FROM products  
WHERE unit_price < 25
```

	stock_num	manu_code
1	5	ANZ
2	9	ANZ
3	103	PRC
4	106	PRC
5	302	HRO
6	302	KAR

```
SELECT stock_num,manu_code  
FROM items  
WHERE stock_num=5
```

	stock_num	manu_code
1	5	NRG
2	5	SMT
3	5	ANZ
4	5	ANZ
5	5	ANZ
6	5	NRG
7	5	ANZ
8	5	ANZ
9	5	SMT

UNION

UNION ALL

INTERSECT

EXCEPT

stock_num	manu_code
5	ANZ
5	NRG
5	SMT
9	ANZ
103	PRC
106	PRC
302	HRO
302	KAR

stock_num	manu_code
5	ANZ
5	NRG
5	NRG
5	SMT
5	SMT
9	ANZ
103	PRC
106	PRC
302	HRO
302	KAR

stock_num	manu_code
5	ANZ

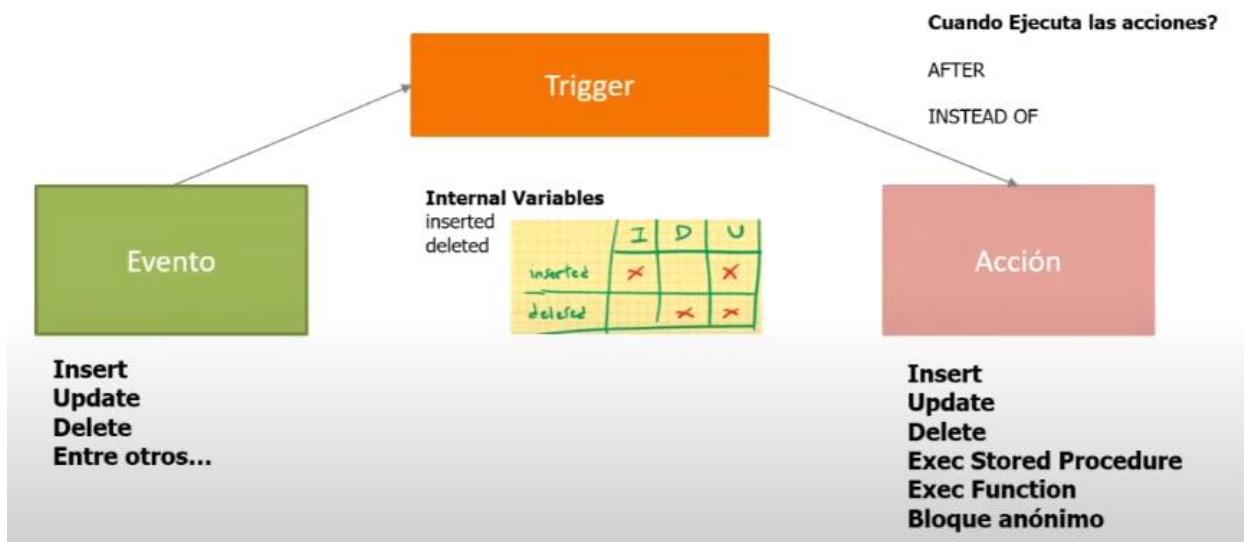
Gestión de Datos

Ing. Juan Zaffaroni

DML Subqueries – 51 v1

TRIGGER

miércoles, 7 de julio de 2021 07:10 p. m.



- Un **TRIGGER** es un mecanismo que **ejecuta** una o varias **sentencias SQL** automáticamente cuando cierto **evento** ocurre.
- Generalmente está **asociado a uno o varios eventos sobre** una determinada **tabla**.
- Las **acciones** del **TRIGGER** pueden **ejecutarse antes** del evento, **después** del evento o **reemplazar** completamente al **evento** que disparó al TRIGGER.
 - nota: no todos los motores tienen la opción de **AFTER, INSTEAD OF** y **BEFORE**.
- Internamente el TRIGGER tiene dos variables en el caso de SQL SERVER son de tipo Table y son llamadas "inserted" y "deleted". Con este gráfico representamos con que eventos se usan las variables internas del trigger:

	INSERT	DELETE	UPDATE
inserted	X		X
deleted		X	X

- Tabla **inserted** se utiliza cuando el evento es **INSERT** o **UPDATE**.
- Tabla **deleted** se utiliza cuando el evento es un **DELETE** o **UPDATE**.
 - **DELETE** almacena en **deleted** los datos que eliminaría el evento que disparó el trigger.
 - **INSERT** almacena en **inserted** los datos que insertaría el evento que disparó el trigger.
 - **UPDATE** almacena en **inserted** los datos con los que actualizaría los registros el evento que disparó el trigger y almacena en **deleted** los datos que serán actualizados.

- **El uso de estas variables es muy útil** por ejemplo para hacer un log en otra tabla con los datos que se reemplazaron y los datos que fueron reemplazados por cada evento por ejemplo. Ya que se poseen ambos valores de manera simultánea.

Usos Posibles

miércoles, 7 de julio de 2021 19:14

- **Aplicar las reglas de negocio**
 - Ej: Si al realizarse una venta el stock de un producto pasará a ser negativo se podría disparar un trigger que anote en otra tabla una advertencia por falta de stock; o capaz podría cancelar el evento de la venta.
- **Valores de columnas derivadas**
 - Ej: Si modifico la cantidad o precio de un ítem de una Orden de Compra, se actualiza el total de la cabecera automáticamente.
- **Ante tal acción se actualicen datos en otra base de datos**
- **Replicación automática de tablas**
 - Ej: cuándo inserto un campo en una tabla, que se inserte en la otra.
- **Implementación de Foreign Keys sobre Tablas de otras BD**
 - Ej: Los motores de BD Relacionales no permiten implementar FK contra PKs de tablas de otras BD. Un Trigger nos lo permitiría.
 - A través de un trigger se podría implementar la integridad relacional entre tablas de distintas bases de datos, por ejemplo ante un DELETE el trigger puede buscar si existe tal PK en otra base de datos antes de que se lleve a cabo el evento.
- **Logs de Auditoría**
 - Ej: esquema de auditoria de cambios en tablas de nuestro sistema en forma controlada por la misma aplicación. Voy guardando quién hace cada cambio, a qué hora, que se cambió, etc.
- **DELETE en Cascada**
 - nota: no recomendado, acción muy pesada.
 - Ej: Implementar un esquema de borrado en cascada, reemplazaría al ON DELETE CASCADE de las Foreign Keys (si borro la PK que se borren todas las FKs).
- **Autorización de Seguridad**
 - Ej: Ante el acceso a la tabla de Cierre de Caja, realizar un chequeo en la tabla de horarios por cajero para ver si el usuario que la realiza está habilitado en dicho horario para dicha operación.

Eventos Disparadores

miércoles, 7 de julio de 2021 08:36 p. m.

Eventos que pueden activar o disparar un TRIGGER.

DML

- **INSERT ON *tabla***
- **DELETE FROM *tabla***
- **UPDATE *tabla***
- **UPDATE of *columna* ON *tabla***
- **SELECT *tabla***
 - nota: cuando se realiza SELECT sobre cierta tabla.
 - nota: (Informix)
- **SELECT of *columna* ON *tabla***
 - nota: cuando se realiza un SELECT sobre cierta columna de determinada tabla.
 - nota: (Informix)

DDL

- Ante creación o modificación de estructuras u objetos de la BD.
 - **CREATE *objeto***
 - **ALTER *objeto***
 - **DROP *objeto***

Operaciones de Base de Datos

- **SERVERERROR**
 - Error del servidor.
- **LOGON**
 - Cuando un usuario se loguea en el servidor.
- **LOGOFF**
 - Cuando un usuario se loguea en el servidor.
- **STARTUP**
 - Cuando se inicia el servidor.

- **SHUTDOWN**

- Cuando se apaga el servidor.

Transacción - Evento - Acción

miércoles, 7 de julio de 2021 08:46 p. m.

Transaccionabilidad entre Evento y Acción de un TRIGGER.

- El **evento** y las **acciones** que ejecuta el trigger **conforman una transacción**, entonces si **alguno** de los dos **falla**, se **realiza** un **rollback automático**.
- Un **trigger** deja la Base de Datos en **estado consistente siempre**.

Ejemplo:

Transacciones- evento-accion implícitas en Triggers

Ejecución de Prueba

```
DROP TRIGGER stateUpdateAudit
CREATE TRIGGER stateUpdateAudit
ON state
AFTER update
AS
BEGIN
    INSERT INTO state_upd
    SELECT state,sname,'A',getdate() from deleted;

    INSERT INTO state_upd
    SELECT null,sname,'N',getdate() from inserted;
-- INSERTO UN NULO EN LA TABLA STATE_UPD FORZANDO ERROR

END
```

Forzaremos un error en el trigger queriendo insertar un nulo en la tabla de auditoria, no contemplado.

Transacciones- evento-accion implícitas en Triggers

Ejecución de Prueba

The screenshot shows two windows from SQL Server Management Studio. The top window is a command window with the following content:

```
update state SET sname='AZ...' WHERE state='AZ'
```

Below the command is the output window showing the error message:

```
(1 row(s) affected)
Msg 515, Level 16, State 2, Procedure stateUpdateAudit, Line 9 [Batch Start Line 44]
Cannot insert the value NULL into column 'state', table 'stores7ParaRomper.dbo.state_upd'; column does not allow nulls.
The statement has been terminated.
```

The bottom window is a results window showing the audit log table and a comparison table:

	id_auditoria	state	sname	accion	fechaYHora
1	1	AZ	Arizona	A	2020-05-06 09:04:43.503
2	2	AZ	AZ...	N	2020-05-06 09:04:43.507
3	3	AZ	AZ...	A	2020-05-06 09:06:29.453
4	4	AZ	Arizona	N	2020-05-06 09:06:29.453

A red box highlights the fourth row of the audit log table, and a red note to the right says: "Misma cantidad de filas ejemplo anterior."

	state	sname
1	AZ	Arizona

Luego de crear el trigger se observa que el error en el insert de state_upd hizo que nada se actualice. En definitiva hubo rollback de acción y evento.

Momentos de Ejecución

miércoles, 7 de julio de 2021 08:51 p. m.

Los **momentos de ejecución** de las **acciones** puede ser **ANTES**, **DURANTE** y **DESPUÉS** del **evento** asociado al Trigger.

- nota: no todos los motores tienen la opción de **FOR EACH ROW** y **BEFORE**.
- **BEFORE**
 - Antes de llevar a cabo el evento.
- **FOR EACH ROW**
 - *Por cada registro que afecta el evento disparador se realiza la acción.*
- **AFTER**
 - Despues de llevar a cabo el evento.
- **INSTEAD OF**
 - Existe una opción de **INSTEAD OF** que **reemplaza** el **evento** por la **acción** a ejecutar.
 - Se ejecutan las acciones en lugar del evento de trigger recibido. Reemplaza el evento que disparó el trigger por las acciones del trigger.

Tablas de Metadata

miércoles, 7 de julio de 2021 08:58 p. m.

Estas son algunas de las **tablas** que contienen **información** de la **Metadata** de todos los objetos creados en nuestra base de datos. En este caso puntual de los triggers.

Tablas de Catalogo / Tablas del Diccionario de Datos Asociadas a los Triggers

- **SYS.TRIGGERS**
 - Tabla **utilizada puntualmente** para el registro de **Triggers** en la BD.
- **SYS.OBJECTS**
 - **Registro por objeto** con el nombre, el esquema al cual pertenece, el tipo, la descripción y la fecha de creación y modificación, almacena los **Triggers** entre los objetos de BD.
- **SYS.ALL_SQL_MODULES**
 - Contiene el **id** y el **código fuente** de muchos objetos de BD entre ellos los **Triggers**, si es recompilado y otros datos más.

The image displays three separate SQL queries and their results in SSMS:

- Query 1:** `SELECT * FROM SYS.TRIGGERS`
Results:

	name	object_id	parent_class	parent_class_desc	parent_id	type	type_desc	create_date	modify_date
1	upd_items_ordenes	1525580473	1	OBJECT_OR_COLUMN	437576597	TR	SQL_TRIGGER	2020-05-06 08:36:41.930	2020-05-06 08:36:41.
2	stateUpdateAudit	1637580872	1	OBJECT_OR_COLUMN	565577053	TR	SQL_TRIGGER	2020-05-06 09:04:19.223	2020-05-06 09:04:19.
- Query 2:** `select * from sys.objects where type='TR'`
Results:

	name	object_id	principal_id	schema_id	parent_object_id	type	type_desc	create_date	modify_date	is_ms_shipped
1	upd_items_ordenes	1525580473	NULL	1	437576597	TR	SQL_TRIGGER	2020-05-06 08:36:41.930	2020-05-06 08:36:41.930	0
2	stateUpdateAudit	1637580872	NULL	1	565577053	TR	SQL_TRIGGER	2020-05-06 09:04:19.223	2020-05-06 09:04:19.223	0
- Query 3:** `SELECT * FROM SYS.all_sql_modules WHERE object_id=1525580473`
Results:

	object_id	definition	uses_ansi_nulls	uses_quoted_identifier	is_schema_bound	uses_database_collation	is_ms_shipped
1	1525580473	CREATE TRIGGER upd_items_ordenes ON items AFTE...	1	1	0	0	0

Trigger Auditoría

miércoles, 7 de julio de 2021 09:32 p. m.

Trigger de Auditoria sobre Tabla

```
CREATE TABLE state_upd(
    id_auditoria INT IDENTITY(1,1),
    code char(2) NOT NULL,
    sname varchar(15) NULL,
    accion char,
    fechaYHora datetime
)
```

Previamente Creamos tabla de auditoria para tabla state.

```
CREATE TRIGGER stateUpdateAudit
ON state
AFTER update
AS
BEGIN
    INSERT INTO state_upd
    SELECT code,sname,'A',getdate() from deleted;

    INSERT INTO state_upd
    SELECT code,sname,'N',getdate() from inserted;
END
```

Explicación del Trigger de auditoría

- Se carga en una tabla **state_upd** de auditoría todos los cambios realizados sobre los registros de la tabla **state** de nuestra Base de Datos.
- Se insertan 2 registros de auditoria por cada **UPDATE** a la tabla **state**
 - Un registro indica el **estado del registro antes de ser modificado**, la fecha y la hora de modificación. En este se completa el atributo de **accion** con 'A' para indicar que así estaba el registro "**Antes**" del cambio.
 - El otro registro indica el **estado del registro antes de modificarse**, la fecha y hora de modificación. En este se completa el atributo de **accion** con 'N' para indicar que es el "**Nuevo**" estado del registro después de la actualización.

Trigger INSTEAD OF en view

miércoles, 7 de julio de 2021 09:50 p. m.

nota: inserta la información que corresponde en las tablas apropiadas, en este caso lo manejamos con un Trigger ya que es una vista con JOIN y por lo tanto no se podría insertar directamente.

Trigger de INSTEAD OF sobre Vista

```
CREATE VIEW ordenes_por_cliente
    (cod_cliente, nombre, apellido, nro_orden, fecha_orden)
AS
SELECT c.customer_num, fname, lname, order_num, order_date
FROM customer c JOIN orders o
    ON (c.customer_num = o.customer_num)

CREATE TRIGGER InsertaClienteyOrdenEnVista
ON ordenes_por_cliente
INSTEAD OF INSERT
AS
BEGIN

    INSERT INTO customer (customer_num, fname, lname)
    SELECT cod_cliente, nombre, apellido FROM inserted;

    INSERT INTO orders (order_num, order_date, customer_num)
    SELECT nro_orden, fecha_orden, cod_cliente FROM inserted;
END
```

Trigger de INSTEAD OF sobre Vista

Ejecución de Prueba

The screenshot shows two SQL queries being run in SSMS. The first query attempts to insert into the view 'ordenes_por_cliente' with the following code:

```
INSERT INTO ordenes_por_cliente
VALUES (1666, 'Filomeno', 'Mas', 2666, '2020-04-23');
```

The second query selects from the 'customer' and 'orders' tables to verify the data:

```
SELECT customer_num, fname, lname
FROM customer WHERE customer_num=1666;
SELECT order_num, customer_num, order_date
FROM orders WHERE order_num=2666;
```

The 'Messages' tab shows an error message:

Msg 4405, Level 16, State 1, Line 23
View or function 'ordenes_por_cliente' is not updatable because the modification affects multiple base tables.

The 'Results' tab shows the output of the SELECT statements. The first query returns the inserted data:

customer_num	fname	lname
1666	Filomeno	Mas

The second query returns the data from the 'orders' table:

order_num	customer_num	order_date
2666	1666	2020-04-23 00:00:00.000

Luego de crear el trigger se observa que el insert funcionó insertando los datos correctos en cada tabla.

Trigger de AFTER UPDATE

miércoles, 7 de julio de 2021 09:56 p. m.

Utilizamos un bloque de código anónimo escrito en T-SQL para modificar un campo cuando se actualiza la tabla items a través del Trigger.

Trigger de AFTER Update con ejecución de Bloque anónimo Modificación automática de columnas derivadas

```
CREATE TRIGGER upd_items_ordenes
ON items
AFTER UPDATE
AS
BEGIN
    DECLARE @i_prec_del dec(8,2), @n_orden int, @i_prec_ins dec(8,2) ,
        @quantity_del int, @quantity_ins int;

    SELECT @i_prec_del=unit_price, @quantity_del=quantity
    FROM deleted;

    SELECT @n_orden= order_num, @i_prec_ins=unit_price, @quantity_ins=quantity
    FROM inserted;

    IF UPDATE (unit_price) OR UPDATE(quantity)
    BEGIN
        UPDATE orders
        SET total= total
            -(@quantity_del*@i_prec_del)
            +(@quantity_ins*@i_prec_ins)
        WHERE order_num = @n_orden;
    END
END
```

Para implementar este ejemplo previamente creamos un total en la tabla Orders y actualizamos dicho campo.

```
ALTER TABLE orders ADD total decimal(12,2)

UPDATE orders SET total=
(SELECT SUM(unit_price*quantity) FROM items i
 WHERE i.order_num= orders.order_num)

CREATE TRIGGER...
```

Ejecución de Prueba

```
select total from orders where order_num=1001
select * from items where order_num=1001
```

item_num	order_num	stock_num	menu_code	quantity	unit_price
1	1001	1	HRO	1	250.00

Luego de actualizar la cantidad a 100, se observa aplicado el cambio en el campo total de la tabla orders.

```
update items set quantity=100 where order_num=1001
select total,order_num from orders where order_num=1001
select * from items where order_num=1001
```

item_num	order_num	stock_num	menu_code	quantity	unit_price
1	1001	1	HRO	100	250.00

Stored Procedures

miércoles, 7 de julio de 2021 19:09

Es un **procedimiento programado** en un lenguaje permitido (SQL ; T-SQL ; PL/SQL ; SPL) que es **almacenado en la Base de Datos como un objeto**.

- Algunos motores **también los permiten en Java**

El mismo luego de creado, **puede ser ejecutado por usuarios que posean los permisos respectivos.**

Son **parseadas y optimizadas antes de ser almacenadas** por lo que el **código corre más rápido al ser ejecutado**, ya que se va chequeando la forma óptima de ejecutar el SP en cuestión.

Ventajas de usar SPs

miércoles, 7 de julio de 2021 19:31

- Pueden **reducir la complejidad en la programación**
 - Las funciones más usadas podemos ejecutarlas a través de Stored Procedures
- Permite **chequeo de permisos**, garantizando un **nivel de seguridad extra**
- Pueden definirse **reglas de negocio independientemente de las aplicaciones.**
- **Desde distintas aplicaciones/tecnologías** se puede acceder al mismo **código ya compilado y optimizado**, y ejecutarlo **evitando repetición de lógica.**
- Se **gana mucha performance** utilizando los Stored Procedures.
- En proyectos donde el código puede ser **ejecutado desde diferentes interfaces**, se mantiene **un solo tipo de código.**
- **Menor tráfico en el PIPE / SOCKET**, no en la cantidad de bytes que viajan sino en los ciclos que debo ejecutar una instrucción.
 - Todo calculo necesario se realiza en el SP (en el servidor de la Base de Datos) y luego es retornado a lo que lo necesite.

Estructura de SP

miércoles, 7 de julio de 2021 19:41

CREATE PROCEDURE [esquema].[nombre_proc] (parámetros de entrada o de salida)

AS

sentencias SPL y/o SQL

GO

PARA RETORNAR UNA VARIABLE NO HACE FALTA

RETURN variable

La retorna sola

“[]” Cláusulas opcionales en cada definición se ponen entre corchetes.

- Si el procedure tiene todos los parámetros como IN se puede ejecutar:

EXECUTE suma 15,13

- Si tiene parámetro de OUT se tiene que poner una variable.

EXECUTE suma2 15, 13, @variable OUT

Ejemplo SP

miércoles, 7 de julio de 2021 19:46

```
CREATE PROCEDURE [suma] (@var1 INTEGER, @var2 INTEGER)
```

```
AS
```

```
DECLARE @var3 INTEGER
```

```
SET @var3 = @var1 + @var2
```

```
RETURN @var3
```

```
GO
```

Ejecución SP

miércoles, 7 de julio de 2021 19:50

EXECUTE nombre_proc param1, param2

EXEC nombre_proc param1, param2

Ejemplo:

EXECUTE suma 15, 13

EXEC suma 15, 13

Alteración y Borrado de SP

miércoles, 7 de julio de 2021 20:17

```
ALTER PROCEDURE nombre_proc
```

```
DROP PROCEDURE nombre_proc
```

```
ALTER PROCEDURE suma (@var1 int, @var2 int)
```

```
AS
```

```
DECLARE @var3 INTEGER
```

```
DECLARE@var4 int
```

```
SET @var4 = 3
```

```
SET @var3 = @var1 + @var2 + @var4
```

```
RETURN @var3
```

```
GO
```

```
DROP PROCEDURE suma
```

Invocación SP

miércoles, 7 de julio de 2021 20:20

CREATE PROCEDURE [otorgar_descuento]

@p_customer_num **NUMERIC**

AS

EXECUTE busca_mayor_orden @p_customer_num

.....

END PROCEDURE;

Se invoca de la misma forma que si se estuviera llamando externamente.

Funciones de Usuario

miércoles, 7 de julio de 2021 19:43

Objeto de la base de datos **programado en un lenguaje válido** por el motor de base de datos.

Funciones **User Defined**, definidas por el usuario.

Puede **recibir uno o más parámetros de input y devolver SOLO un parámetro de output**.

SP vs Funciones

miércoles, 7 de julio de 2021 19:51

Funciones

- Puede ser invocada por una consulta u operaciones DML (Insert, update, etc)
- No puede contener ni ejecutar instrucciones de escritura, solo puede tener SELECT.
 - nota: no puede escribir, eliminar, o modificar datos.

Stored Procedures

- No pueden ser invocados en consultas u operaciones DML (Insert, update, etc).
- Pueden ejecutar cualquier instrucción DML.

Estructura Función

miércoles, 7 de julio de 2021 19:55

**CREATE FUNCTION <nombreFuncion> (nombreParametro
TIPOPARAMETRO, ...)**

RETURN TIPOVARIABLE;

DECLARE (Opcional, para declarar variables)

...

...

BEGIN

...

...

...

...

RETURN <nombreVariable>;

Ejemplo de Función

miércoles, 7 de julio de 2021 19:58

```
CREATE FUNCTION nombre_dpto (p_c_empleado NUMBER)
RETURN departamentos.d_dept%TYPE;
DECLARE
v_nombre_dpto departamentos.d_dept%TYPE;
v_error VARCHAR2(70);
BEGIN

SELECT d_dpto INTO v_nombre_dpto
FROM departamentos d, empleados e
WHERE d.c_dpto = e.c_dpto
AND e.c_empleado = p_c_empleado;

RETURN v_nombre_dpto;

EXCEPTION
WHEN NO_DATA_FOUND THEN

v_error := 'Error: empleado ' || p_c_empleado || ' no se ha encontrado';
raise_application_error(-20101, v_error);

END nombre_dpto;
```

Invocación de Función

miércoles, 7 de julio de 2021 20:00

- **Invocación de función en un SELECT en la sección WHERE**

SELECT *

FROM departamentos d

WHERE d.d_dpto = **nombre_dpto(6010);**

- **Invocación de función en un SELECT en la sección Lista de Columnas**

SELECT e.c_empleado,

 e.d_nombre,

 e.d_apellido,

nombre_dpto(e.c_empleado) depto_nombre

FROM empleados e

WHERE e.c_empleado = 6010;

Funciones Built-In

miércoles, 7 de julio de 2021 20:03

Estos objetos son **funciones ya desarrolladas con el Motor de BD (Propias del Motor / Built-In)**, las cuales pueden clasificarse de la siguiente manera:

• Funciones agregadas

Se aplican a un conjunto de valores derivados de una expresión, actúan específicamente en agrupamientos.

- **SUM(columna)**

- **COUNT(*)**
nota: cuenta todas las filas de la tabla

- **COUNT(columna)**
- nota: cuenta filas con dicha columna No Nula

- **COUNT(DISTINCT columna)**
- nota: cuenta solo una vez cada valor.

- **MIN(columna)**
- **MAX(columna)**
- **SUM(columna)**
- **AVG(columna)**

○ Funciones Escalares

Se aplican a un dato específico de cada fila de una consulta, se pueden utilizar cuando se realizan comparaciones con el operador **WHERE**

- Funciones Algebraicas/ Trigonométricas
- Funciones Estadística
- Funciones de Fecha
- Funciones de Strings
- Funciones de Conversión
- Funciones de Manejo de Null

- Funciones de Entorno
- Funciones XML (Ora)
- Funciones de Minería de Datos (Ora)

Funciones de tablas

Devuelven el equivalente a una tabla y pueden ser usadas solamente en la sección FROM.

Diccionario de Datos Tablas Involucradas

- **SYS.OBJECTS**

Crea un registro con el nombre, el esquema al cual pertenece el objeto, el tipo de objeto que es, la descripción, y la fecha de creación y modificación.

- **SYS.PROCEDURES**

Contiene datos parecidos al SYS.OBJECTS, agregado a detalles de las funcionalidades propias de los procedimientos; tales como si se auto ejecuta, si es de ejecución pública, si es replica constraints, etc.

- Vista sobre SYS.OBJECTS (o sea cierta parte de SYS.OBJECTS)

- **SYS.ALL_SQL_MODULES**

Contiene el ID y el código fuente del procedure, si es recompilado y otros datos más.

- **SYS.PARAMETERS**

Lista de parámetros para los procedimientos, con nombre, tipo, longitud, número de orden para la llamada al procedure, etc.

Definición de variables

miércoles, 7 de julio de 2021 20:26

```
CREATE procedure [nombre_proc]
AS
DECLARE @nombre_var DATATYPE
GO
```

Asignación de valores a variables

miércoles, 7 de julio de 2021 20:27

SELECT @local_variable = valor

ó

SET @local_variable = valor

- @local_variable

Es una variable declarada a la que se va a asignar un valor.

= Asigna el valor de la derecha a la variable de la izquierda.

{= | += | -= | *= | /= | %= | &= | ^= | |= }

Operador de asignación compuesta:

- += Sumar y asignar
- -= Restar y asignar
- *= Multiplicar y asignar
- /= Dividir y asignar
- %= Módulo y asignar
- &= AND bit a bit y asignar
- ^= XOR bit a bit y asignar
- |= OR bit a bit y asignar

Ejemplo

SET @var1 = 'Jorge';

SET @var2 = 'Jorge'+' contatenado'

```
SELECT @var1 = 'Jorge';
```

```
SELECT @var1 = nombre (este SELECT solo debe tener un solo nombre como  
resultado)
```

```
FROM clientes
```

```
WHERE clienteID = 1000 ;
```

Sentencias de Manejo de Bloques

miércoles, 7 de julio de 2021 20:30

BEGIN Inicia Bloque

END Finaliza Bloque

Ejemplos

```
CREATE PROCEDURE proc1 ()
```

```
AS
```

```
-- Bloque implícito
```

```
DECLARE @var1 INTEGER;
```

```
@var1 = 10
```

```
BEGIN -- Bloque explícito
```

```
sentencias.....
```

```
END -- Fin Bloque explícito
```

Cuando se **crea un procedure existe al menos un bloque con un BEGIN y END implícitos.**

Sentencias Condicionales

miércoles, 7 de julio de 2021 20:32

IF condición1 **THEN**

 Sentencia1

ELSE

 Sentencia2

Ejemplo

IF (@var1 > 5)

BEGIN

PRINT 'valor mayor a 5';

END

ELSE

BEGIN

PRINT @var1;

END

- EXPRESIONES en UNA SENTENCIA IF

IF EXISTS(SELECT CUSTOMER_NUM FROM CUSTOMER

WHERE CUSTOMER_NUM = @customer_num)

BEGIN

PRINT 'Existe el cliente';

END

ELSE

BEGIN

PRINT 'Cliente Inexistente';

END

Sentencia CASE

miércoles, 7 de julio de 2021 20:33

Esta sentencia puede utilizarse en las siguientes condiciones:

- Dentro de la cláusula SELECT de la instrucción SELECT
- Dentro de la cláusula ORDER BY de una instrucción SELECT
- Dentro de una instrucción UPDATE
- En una instrucción SET
- En una cláusula HAVING de una instrucción SELECT

Ejemplo en un SELECT de un CASE

SELECT Fabricante = **CASE** manu_code

WHEN 'ANZ' **THEN** 'ANZA'

WHEN 'HRO' **THEN** 'HERO'

WHEN 'HSK' **THEN** 'HUSKY'

ELSE 'RESTO'

END,

stock_num,order_num,item_num,quantity,total_price **AS** Precio

FROM items

WHERE total_price **IS NOT NULL**

ORDER BY manu_code, Precio

Ejemplo de un SELECT con un CASE de Búsqueda

SELECT manu_code, stock_num, unit_price, 'Rango de Precios' =

CASE

WHEN unit_price=0 **THEN** '0 - Item no negociable'

WHEN unit_price<100 **THEN** '1 - Precio Menor \$100'

WHEN unit_price>=50 **AND** unit_price<250

THEN '2 - Precio Menor a \$250'

WHEN unit_price>=250 and unit_price<500

THEN '3 - Precio Menor a \$500'

```
ELSE '4 - Precio mayor a $500'  
END  
FROM stock  
ORDER BY "Rango de Precios",manu_code,stock_num
```

Ejemplo de un CASE en un ORDER BY

```
SELECT manu_code, stock_num, order_num, item_num  
FROM items  
WHERE manu_code IN ('ANZ','HRO')  
ORDER BY CASE WHEN manu_code='HRO' THEN order_num END,  
CASE manu_code WHEN 'ANZ' THEN stock_num END;
```

Ejemplo de un CASE en un UPDATE

```
UPDATE stock  
SET unit_price =  
( CASE WHEN (unit_price <= 250) THEN unit_price * 1.05  
  
ELSE (unit_price * 1.10)  
END  
)  
WHERE manu_code = 'ANZ';
```

Ejemplo de un SELECT en una instrucción SET

```
SET @ContactType =  
CASE  
  
WHEN EXISTS(SELECT * FROM HumanResources.Employee AS e  
WHERE e.BusinessEntityID = @BusinessEntityID)  
  
THEN 'Employee'  
  
WHEN EXISTS(SELECT * FROM Person.BusinessEntityContact AS bec  
WHERE bec.BusinessEntityID = @BusinessEntityID)  
  
THEN 'Vendor'
```

WHEN EXISTS(SELECT * FROM Purchasing.Vendor AS v

WHERE v.BusinessEntityID = @BusinessEntityID)

THEN 'Store Contact'

WHEN EXISTS(SELECT * FROM Sales.Customer AS c

WHERE c.PersonID = @BusinessEntityID)

THEN 'Consumer'

END;

Sentencias Ciclicas

miércoles, 7 de julio de 2021 20:39

WHILE condición

BEGIN

.....

BREAK -- Abandona el Bloque del While accediendo a la próxima instrucción fuera del ciclo.

.....

CONTINUE -- No ejecuta próximas instrucciones y continúa con la próxima iteración del WHILE

.....

END

.....

BREAK – Cuando esta cláusula es ejecutada dentro de un While el programa abandona el mismo y ejecuta la próxima instrucción siguiente al End del bloque donde se ejecuta.

CONTINUE – Cuando esta cláusula es ejecutada dentro de un While el programa vuelve al principio del While para evaluar la condición, descartando todo lo que le seguía de código (**END** termina un ciclo).

Ejemplo

WHILE (SELECT AVG(unit_price) **FROM** stock

WHERE manu_code='ANZ') < 300

-- Mientras que el promedio sea menor que 300 va a

-- continuar iterando

BEGIN

UPDATE stock

SET unit_price = unit_price * 1.10

WHERE manu_code= 'ANZ'

```
IF (SELECT MAX(unit_price) FROM stock  
WHERE manu_code='ANZ')>1500  
BREAK -- Si se llega a un producto con precio  
-- mayor a 1500 TAMBIÉN se finaliza  
-- la actualización
```

ELSE

```
PRINT 'Continuamos actualizando los precios'
```

END

```
PRINT 'Finalizamos la Actualización de Productos'
```

GO

Ejecución de comandos del Sistema Operativo

miércoles, 7 de julio de 2021 20:44

Existe el procedimiento xp_cmdshell el cuál recibe como parámetro el comando del sistema operativo que uno quiere ejecutar.

Si el mismo no se encuentra habilitado se deberá ejecutar estas sentencias para habilitar la ejecución en el Motor SqlServer.

```
EXEC sp_configure 'show advanced options', 1
```

```
RECONFIGURE;
```

```
EXEC sp_configure 'xp_cmdshell', 1;
```

```
RECONFIGURE;
```

Ejemplo

```
IF (@var1 > 5)
```

```
BEGIN
```

```
EXEC xp_cmdshell 'ipconfig'
```

```
END
```

El resultado será:

Configuracion IP de Windows

Adaptador Ethernet Conexion de Area local :

Sufijo de conexion especifica DNS :

Direccion IP.....: 192.168.131.65

Mascara de subred: 255.255.255.0

Puerta de enlace predeterminada : 192.168.131.254

Manejo de Cursos.

miércoles, 7 de julio de 2021 20:46

En SQLSERVER un cursor se define con la declaración, luego se abre con una sentencia OPEN, y se asignan los valores con la operación FETCH – INTO. Una vez finalizado se cierra con la sentencia CLOSE y se libera la memoria con DEALLOCATE.

```
DECLARE nombre_cursor CURSOR
[ LOCAL | GLOBAL ]
[ FORWARD_ONLY | SCROLL ]
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
[ TYPE_WARNING ]
FOR sentencia_sql
```

```
OPEN <nombre_cursor>
```

```
FETCH nombre_cursor INTO lista_variables
```

WHILE (@@FETCH_STATUS = 0) -- Está encontrando resultados en donde está apuntando

```
BEGIN
...
FETCH nombre_cursor INTO lista_variables
END
```

```
CLOSE nombre_cursor
DEALLOCATE nombre_cursor
```

Ejemplo

```
/*Creamos un procedimiento que a partir de un número de almacén nos inserta en una
tabla auxiliar todos los ítems que están en él */
```

```
CREATE PROCEDURE guardar_items_tabla
```

```
@almacen INTEGER
AS
DECLARE items_en_almacen CURSOR FOR
SELECT id_item FROM item
WHERE id_almacen = @almacen
DECLARE @item_del_cursor INTEGER
OPEN items_en_almacen
FETCH items_en_almacen INTO @item_del_cursor
WHILE (@@FETCH_STATUS = 0)

BEGIN

INSERT INTO ITEMS_AUX VALUES (@item_del_cursor)
FETCH items_en_almacen INTO @item_del_cursor

END
CLOSE items_en_almacen
DEALLOCATE items_en_almacen
END PROCEDURE;
```

Procedimientos Recursivos

miércoles, 7 de julio de 2021 20:50

Es un procedimiento que se llama asimismo.

No hay límite de numero de call procedures anidados.

No hay límite de cursos abiertos.

Un nuevo cursor puede ser declarado para cada invocación de cada evento.

Un ejemplo típico es el de Cálculo del Factorial.

Ejemplo

```
CREATE PROCEDURE dbo.sp_calcfactorial  
    @base_number decimal(38,0),  
    @factorial decimal(38,0) OUT  
AS  
SET NOCOUNT ON  
DECLARE @previous_number DECIMAL(38,0)  
IF ((@base_number>26) and (@@MAX_PRECISION<38))  
OR (@base_number>32)  
BEGIN  
RAISERROR('Computing this factorial would exceed the servers max. numeric  
precision of %d or the max. procedure nesting level of 32',16,10,  
@@MAX_PRECISION)  
RETURN(-1)  
  
END  
IF (@base_number<0)  
BEGIN  
    RAISERROR('Can''t calculate negative factorials',16,10)  
RETURN(-1)  
END
```

```
IF (@base_number<2)
SET @factorial=1 -- Factorial of 0 or 1=1
ELSE
BEGIN
SET @previous_number=@base_number-1
EXEC dbo.sp_calcfactorial @previous_number, @factorial OUT
-- Recursive call
IF (@factorial=-1) RETURN(-1) -- Got an error, return
SET @factorial=@factorial*@base_number
IF (@@ERROR<>0) RETURN(-1) -- Got an error, return
END
RETURN(0)
GO
DECLARE @factorial decimal(38,0)
EXEC dbo.sp_calcfactorial 32, @factorial OUT
SELECT @factorial
```

Obtención del valor asignado a un campo Identity

miércoles, 7 de julio de 2021 20:54

```
CREATE PROCEDURE identity_insert  
DEFINE ser INT;  
INSERT INTO orders (order_date,customer_num)  
VALUES ("04/01/93",102)  
SET @orderId = @@IDENTITY  
ó  
SELECT @orderId = SCOPE_IDENTITY()  
GO
```

- @@IDENTITY y SCOPE_IDENTITY devuelven el último valor de identidad generado en una tabla en la sesión actual.
- No obstante, SCOPE_IDENTITY solo devuelve el valor en el ámbito actual; @@IDENTITY no se limita a un ámbito específico.

Manejo de Transacciones

miércoles, 7 de julio de 2021 20:55

En las transacciones de SQLSERVER se debe especificar si la transacción tiene una finalización correcta o incorrecta, y así saber si existe una confirmación de los datos o rollback de los mismos.

BEGIN TRANSACTION

Bloque de Sentencias SQL

[COMMIT | ROLLBACK] TRANSACTION

Ejemplo

CREATE PROCEDURE borra_desde_fecha @fecha smalldate

AS

BEGIN TRANSACTION

INSERT INTO...

UPDATE....

DELETE FROM...

IF getdate() > @fecha THEN

COMMIT TRANSACTION

ELSE

ROLLBACK TRANSACTION

Manejo de Errores

sábado, 10 de julio de 2021 08:23 p. m.

Acceso a variables

- **ERROR_NUMBER()** : siempre mayor a 50.000
- **ERROR_MESSAGE()** : mensaje de error
- **ERROR_STATE()** : valor de referencia

Lanzamiento

Raiserror

- No corta el flujo de ejecución.

Sintaxis:

- **RAISERROR('Mensaje de error', <severidad>, <state>)**
 - Nota: severidad es el grado de la excepción

THROW

- Corta el flujo de ejecución

Sintaxis:

- **THROW <codigo>, 'Mensaje de error', <state>**
 - Nota: **THROW** siempre tiene severidad 16
 - Nota: normalmente conviene el **THROW** para que corte la ejecución y salte directamente al bloque **CATCH**

Manejo de Excepciones

miércoles, 7 de julio de 2021 20:56

Funciones TRY-CATCH:

Dentro del bloque TRY, las funciones que levanten algún tipo de error permiten manejar las distintas excepciones en el bloque de CATCH; previendo errores y perdidas de procesamiento.

Es muy común utilizar un bloque de BEGIN TRAN, COMMIT TRAN dentro del bloque del TRY, y colocar la sentencia ROLLBACK TRAN en el catch.

BEGIN TRY

 Sentencias SQL

END TRY

BEGIN CATCH

 Sentencias SQL

END CATCH

Ejemplo:

--Error por clave primaria duplicada

BEGIN TRY

BEGIN TRAN

INSERT INTO numeros_enteros VALUES (1)

INSERT INTO numeros_enteros VALUES (2)

INSERT INTO numeros_enteros VALUES (1)

COMMIT TRAN

END TRY

BEGIN CATCH

PRINT 'ERROR EN CLAVE DUPLICADA'

ROLLBACK TRAN

END CATCH

En este caso la tabla numeros_enteros quedaría sin valores, porque se vació en un comienzo de un TRY, y luego la transacción comenzó, arrojó un error al querer duplicar la clave de número 1 y en el catch realizó el rollback.

Control de Conurrencia

jueves, 8 de julio de 2021 11:09 a. m.

- Los motores permiten **controlar el acceso concurrente a sus recursos** a través de **bloqueos** y de la definición de **niveles de aislamiento** (*isolation levels*).
- Puede ocurrir que dos usuarios o sesiones distintas intenten actualizar, eliminar o modificar de alguna manera un recurso de la BD en el mismo instante.
 - Por ejemplo, si una transacción se encuentra trabajando sobre una tabla y otra transacción al mismo tiempo intentara modificar dicha tabla.
- Los **bloqueos** son parte esencial del requisito de **aislamiento** y se utilizan para bloquear los objetos afectados dentro de una operación.
 - En el momento en que los objetos están bloqueados el DBMS no permitirá que otras operaciones lleguen a realizarse haciendo cambios en los datos que se guardan en los objetos afectados por el bloqueo que se realiza.
 - Una vez liberado el bloqueo, al confirmar los cambios o al revertir los cambios al estado inicial, se podrán realizar otras operaciones que permitirán hacer los cambios de datos que se requieran.
- Significa que **cuando se realiza una operación, se antepone el bloqueo en un objeto**, el resto de las operaciones que se requieren para dar el acceso al objeto serán **forzadas a esperar hasta la liberación del bloqueo**.
 - **Tipos de Bloqueos**
 - **Compartido (Shared)**
 - Este tipo de bloqueo se lleva a cabo cuando se impone y reserva un objeto para que esté disponible solamente en caso de lectura, lo cual significa que cualquier otra operación no podrá modificar el registro bloqueado mientras permanezca el bloqueo.
 - Tampoco se podrían realizar otros bloqueos sobre el mismo objeto cuando ya existe un bloqueo de tipo compartido.
 - Al conectarse un usuario a una BD se asigna automáticamente un lock de tipo shared por lo que nadie puede bloquear la BD de forma exclusiva.
 - ◆ Siempre que haya usuarios conectados a la base de datos existirá un bloqueo de tipo compartido sobre los recursos de la base de datos, por lo tanto un DBA no podría realizar un bloqueo exclusivo sobre toda la BD si lo quisiera.

- **Exclusivo (Exclusive)**

- Este tipo de bloqueo asegurará que objeto se reserve exclusivamente para la operación que impuso el bloqueo exclusivo, siempre y cuando la operación mantenga el bloqueo.
 - Se puede imponer un bloqueo exclusivo a una página o una fila solamente cuando no hay otro bloqueo compartido o exclusivo impuesto en el destino del objeto.

- **Promovible (Promutable-Update)**

- Inicia como compartido y se promueve a exclusivo.

- **Granularidad de Bloqueos**

- **Nivel de Base de Datos**

- Se bloquea completamente la base de datos.

- **Nivel de Tabla**

- Se bloquea una tabla individualmente.

- **Nivel de Página**

- Página: Conjunto de muchos registros de una tabla.
 - No todos los DBMS manejan este tipo de bloqueos.

- **Nivel de Fila o Registro**

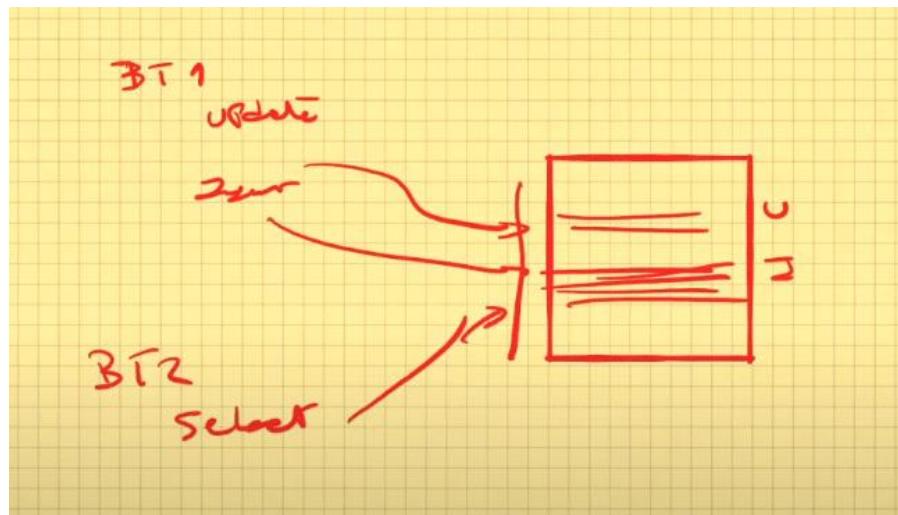
- Se bloquea un registro específicamente.

- **Nivel de Clave de Índice**

Aislamiento de Transacciones

jueves, 8 de julio de 2021 12:58 p. m.

- El **aislamiento de transacciones** se produce cuando dos transacciones utilizan el mismo recurso de manera simultánea.
- Una transacción **T2** podría consultar los datos de una tabla **mientras** la otra transacción **T1** se encuentra abierta (no realizó COMMIT) modificando datos de la misma tabla, lo cual podría generar una lectura errónea o no deseada, esto se evita configurando el **nivel de aislamiento** de cada transacción.



Niveles de Aislamiento de Transacciones (*Isolation Levels*)

- El **tipo de lecturas** que realice una transacción sobre los registros depende del **nivel de aislamiento** con el que se la configure al momento de su creación.
- Cuanto **más restrictivo** sea el **nivel de aislamiento** de la transacción, **más impacto** tiene en el **funcionamiento** de un **sistema multiusuario**, porque las transacciones con mayor nivel de aislamiento tendrán que esperar que finalicen las otras transacciones concurrentes.

	<i>Dirty Reads</i>	<i>Phantom Records</i>	<i>Repeatable Reads</i>
<i>Read Uncommitted</i>	SI	SI	NO
<i>Read Committed</i>	NO	SI	NO
<i>Repeatable Read</i>	NO	SI	SI
<i>Serializable Read</i>	NO	NO	SI

Tipos de lecturas posibles

- **Dirty Read / Lectura sucia**
 - *Datos actualizados en una transacción concurrente que luego se deshacen por un rollback.*

- Si una transacción **T1** se encuentra actualizando datos de una tabla y otra transacción **T2** lee estos datos actualizados antes de que se confirme **T1**, entonces **T1** podría realizar un rollback y la **T2** habría leído datos "sucios", por eso se lo llama **dirty read**.

- **Phantom Read / Phantom Records / Lectura fantasma**

- *Registros o filas que aparecen durante la transacción que fueron insertadas en otra transacción concurrente.*
- Si una transacción **T1** se encuentra insertando registros en una tabla y otra transacción **T2** lee estos registros insertados antes de que se confirme **T1**, entonces **T1** podría realizar un rollback y la **T2** habría leído registros "fantasmas", por eso se lo llama **phantom read o phantom records**.

- **Repeatable Read / Lectura repetible**

- *En la misma transacción poder hacer dos veces o más la misma consulta en distintos momentos, asegurando siempre el mismo resultado de la consulta.*
- Si se realiza una misma consulta SELECT en dos momentos diferentes de la transacción debe arrojar como resultado los mismos registros exactamente en cantidad y estado.

Read Uncommitted

jueves, 8 de julio de 2021 01:29 p. m.

- Es el **nivel de aislamiento más permisivo**, por lo tanto **menor impacto** tendrá en el **funcionamiento** de un sistema **multiusuario**.
- El nivel de aislamiento **Read Uncommitted** no chequea bloqueos exclusivos en la tablas a consultar, lo que mejora el rendimiento porque no espera que finalicen otras transacciones concurrentes pero afecta la integridad en cuanto a que:
 - Pueden existir **lecturas sucias** (*dirty read*)
 - Pueden existir **lecturas fantasma**s (*phantom records*)
 - **NO** asegura **lecturas repetibles** (*repeatable reads*)

Ejemplo:

	USER 1	USER 2
1	CREATE TABLE ##nums (valor INT)	
2	INSERT INTO ##nums VALUES (1)	
3		SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
4	BEGIN TRANSACTION T1	
5		BEGIN TRANSACTION T2
6	INSERT INTO ##nums VALUES (2)	
7	INSERT INTO ##nums VALUES (3)	
8	INSERT INTO ##nums VALUES (4)	
9		SELECT * FROM ##NUMS
10	ROLLBACK TRANSACTION T1	
11		SELECT * FROM ##NUMS
12		
13		COMMIT TRANSACTION T2

- La transacción **T1** se **rollbackea**, por lo que la **inserción de valores 2, 3 y 4 no se realiza**.
- Pero la **lectura** de **T2** a la tabla de **##nums** devuelve **lecturas sucias** y **registros fantasma**s en el instante 9, ya que no se establecen bloqueos porque el **nivel de aislamiento** de **T2** es **READ UNCOMMITTED**.
- Además se observa que en el momento 11 se realiza la misma consulta dando valores distintos, observando que **no se asegura** tampoco una **lectura repetible**.

Read Committed

jueves, 8 de julio de 2021 01:29 p. m.

- Es el **nivel de aislamiento** configurado por **defecto** en los distintos **DBMS**, si no se especifica el nivel de aislamiento de una transacción en su creación entonces el DBMS la configura como **Read Committed por defecto**.
- El nivel de aislamiento **Read Committed** a diferencia del Read Uncommitted ante una lectura de datos **chequea la existencia de bloqueos exclusivos** en la tabla a consultar. Ante un bloqueo exclusivo la transacción espera hasta que se confirma la otra transacción concurrente.
- El nivel de aislamiento **Read Committed** asegura que sólo leerá datos confirmados o commiteados por otra transacción, entonces:
 - **NO** realizará **lecturas sucias**.
- Pero si bien el **Read Committed** ante una lectura de datos **chequea la existencia de bloqueos exclusivos** en la tabla a consultar, una vez que leyó los datos estos podrían ser **bloqueados o modificados** por otra transacción concurrente, entonces:
 - Pueden existir **lecturas fantasma**s.
 - **NO** asegura **lecturas repetibles**.

Ejemplo:

	USER 1	USER 2
1	CREATE TABLE ##nums (valor INT)	
2	INSERT INTO ##nums VALUES (1)	
3		SET TRANSACTION ISOLATION LEVEL READ COMMITTED
4	BEGIN TRANSACTION T1	
5		BEGIN TRANSACTION T2
6	INSERT INTO ##nums VALUES (2)	
7	INSERT INTO ##nums VALUES (3)	
8	ROLLBACK TRANSACTION T1	SELECT * FROM ##NUMS
9	BEGIN TRANSACTION T3	
10	INSERT INTO ##nums VALUES (4)	
11		SELECT * FROM ##NUMS
12	COMMIT TRANSACTION T3	COMMIT TRANSACTION T2
13		

- El hilo de la transacción **T2** queda bloqueado en el instante 8 ya que **intenta poner un bloqueo compartido** para leer los datos insertados por **T1** y **queda a la espera** debido a que la **tabla ##nums se encuentra bloqueada de forma exclusiva**.
- En el instante 8 cuando la transacción **T1** hace el rollback, se desbloquea **T2 y devuelve únicamente los datos confirmados**, en este caso sólo el valor 1, por lo tanto no hay **lecturas sucias (dirty read)**.
- Luego en el instante 11 el hilo de la transacción **T2** queda bloqueado esperando la confirmación de la transacción **T3** que sucede en el instante 9. Por lo que en el instante 12 la transacción **T2** devuelve los valores 1 y 4 commiteados.
 - Podemos observar las **lecturas fantasma**s ya que se agregan registros durante la ejecución de la transacción T2.
 - Podemos observar que **NO** se aseguran las **lecturas repetibles** ya que en el instante 12 la consulta arroja un resultado distinto al de la misma consulta realizada en el instante 8.

Repeatable Read

jueves, 8 de julio de 2021 01:29 p. m.

- El nivel de aislamiento **Repeatable Read**:
 - Asegura que **NO existan lecturas sucias**.
 - **NO evita las lecturas fantasma**s.
 - iii Asegura que las **lecturas sean repetibles** solo para **dirty read**, pero **NO** para **phantom record !!!**
 - **Asegura lecturas repetibles pero solo ante UPDATES, no ante INSERTS.**
- El nivel de aislamiento **Repeatable Read** establece **bloqueo exclusivo sobre los registros INVOLUCRADOS** en el **SELECT** de **consulta** que realice la transacción, **NO** a la **tabla completa**.
 - Cuando otra transacción intenta modificar o borrar algún registro que esté bloqueado por este SELECT quedará en espera en estado de "**Executing Query**".
- Pero las **lecturas fantasma**s podrán aparecer, ya que nivel de aislamiento **Repeatable Read** establece **bloqueo exclusivo sobre los registros INVOLUCRADOS** en el **SELECT** de **consulta** que realice la transacción y **NO** sobre la **tabla completa**.
 - La **inserción de nuevos registros** por otra transacción concurrente a la misma tabla generará **registros fantasma**s los cuales se arrojarán como resultado cuando la transacción original realice la consulta SELECT.
- Su **uso es muy particular**, se tiende a utilizar este tipo de nivel de aislamiento para conservar dentro de una misma transacción dos resultados iguales en un SELECT a un cierto rango de números.

Ejemplo asegurando Repeatable Read para dirty read

	USER 1	USER 2
1	CREATE TABLE ##nums (valor INT)	
2	INSERT INTO ##nums VALUES (1)	
3	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ	
4	BEGIN TRANSACTION T1	
5		BEGIN TRANSACTION T2
6	SELECT * FROM ##NUMS	
7		UPDATE ##NUMS SET valor=2 WHERE valor=1
8	SELECT * FROM ##NUMS	
9		
10	COMMIT TRANSACTION T1	
11		COMMIT TRANSACTION T2

- La lectura del instante 6 por parte de **T1** realiza un **SELECT** a toda la tabla estableciendo **bloqueo exclusivo a todos los registros involucrados** en el **SELECT**, en este caso son todos los registros pero podrían ser menos.
- En el instante 7 existe una instrucción de **UPDATE** por parte de la **T2**, gracias al nivel de aislamiento **Repeatable Read** de **T1** se bloqueará el hilo de ejecución **T2** ya que quiere modificar un registro de los que fueron **bloqueados de forma exclusiva** por **T1**.
 - La transacción **T2** permanece en un estado de "**Executing Query**" hasta que se confirme **T1**.
- En el instante 8 la transacción **T1** realiza la **misma consulta SELECT** sobre los valores de prueba y arrojarán mismo resultado que en el instante 6.

- Podemos observar que asegura que las **lecturas sean repetibles para dirty read**.
- Una vez confirmada la transacción **T1** en el instante 10, el hilo de ejecución de la transacción **T2** se liberará y podrá actualizar los datos.

Ejemplos NO asegurando Repeatable Read para phantom records

	USER 1	USER 2
1	CREATE TABLE ##nums (valor INT)	
2	INSERT INTO ##nums VALUES (1)	
3	SET TRANSACTION LEVEL REPEATABLE READ	
4	BEGIN TRANSACTION T1	
5		BEGIN TRANSACTION T2
6	SELECT * FROM ##nums	
7		INSERT INTO ##nums VALUES (5)
8		COMMIT TRANSACTION T2
9	SELECT * FROM ##nums	
10	COMMIT TRANSACTION T1	

- En el **instante 6** la transacción **T1** realiza la **consulta SELECT * FROM ##nums** y devuelve solo como resultado el valor 1, pero en el **instante 8** la **misma consulta** devuelve como valores **1 y 5** ya que la **T2** ha sido **confirmada**.
 - Podemos observar entonces que este nivel de aislamiento **NO** asegura que las **lecturas sean repetibles para phantom records**.

Otro Ejemplo con INSERT pero sin COMMIT de T2 para phantom records

	USER 1	USER 2
1	CREATE TABLE ##nums (valor INT)	
2	INSERT INTO ##nums VALUES (1)	
3	SET TRANSACTION LEVEL REPEATABLE READ	
4	BEGIN TRANSACTION T1	
5		BEGIN TRANSACTION T2
6	SELECT * FROM ##nums	
7		INSERT INTO ##nums VALUES (5)
8	SELECT * FROM ##nums	
9		COMMIT TRANSACTION T2
10	COMMIT TRANSACTION T1	

- En el **instante 6** la transacción **T1** realiza la **consulta SELECT * FROM ##nums** y devuelve solo como resultado el valor 1, pero en el **instante 8** al realizar la **misma consulta** no devuelve ningún valor porque la transacción **T2** no ha sido **confirmada**. Por lo tanto **T1** permanece "**Executing Query**" porque la instrucción **INSERT INTO** de la **T2** **bloqueó de manera exclusiva** la tabla **##nums**.

Serializable Read

jueves, 8 de julio de 2021 01:29 p. m.

- Es el **nivel de aislamiento menos permisivo**, por lo tanto es el que **mayor impacto** tendrá en el **funcionamiento** de un **sistema multiusuario**.
- El nivel de aislamiento **Serializable Read** es el único que:
 - Asegura que **NO** existan **lecturas sucias**.
 - Asegura que **NO** existan **lecturas fantasma**s.
 - Asegura que las **lecturas sean repetibles**.
- El nivel de aislamiento **Serializable Read** establece **bloqueo exclusivo** de un **rango de índice** según la condición que se coloque en la cláusula **WHERE** de la consulta **SELECT**.
 - **Bloquea de forma exclusiva** las **claves adyacentes de índices** para que **NO** se puedan **insertar registros** en el **rango de registros** que arroja como **resultado la consulta**.
- Si no es posible bloquear los índices o no existieran entonces realiza un **bloqueo exclusivo** sobre la **tabla COMPLETA** sobre la cual se realiza el **SELECT de consulta** en la transacción.
- Esa es la **gran diferencia** entre los niveles de aislamiento **Serializable Read y Repeatable Read** ya que no se establece **bloqueo exclusivo solo sobre los registros INVOLUCRADOS** en el **SELECT de consulta** que realice la transacción.

Ejemplo:

	USER 1	USER 2
1	CREATE TABLE ##nums (valor INT)	
2	INSERT INTO ##nums VALUES (1)	
3	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE	
4	BEGIN TRANSACTION T1	
5		BEGIN TRANSACTION T2
6	SELECT * FROM ##NUMS	
7		INSERT INTO ##NUMS VALUES (2)
8		COMMIT TRANSACTION T2
9	SELECT * FROM ##NUMS	
10	COMMIT TRANSACTION T1	

- La lectura del **instante 6** se realiza un **SELECT** a toda la tabla estableciendo un **bloqueo exclusivo** en la **tabla completa o el índice correspondiente**.
- En el **instante 7** existe una instrucción de **INSERT** la cual bloqueará el hilo de ejecución de la transacción **T2** ya que quiere insertar un registro y la tabla se encuentra bloqueada exclusivamente por **T1**.
- El **instante 9** realizará la **misma consulta SELECT** sobre los valores de prueba y arrojarán **mismo resultado**.
- Una vez **confirmada la transacción T1** en el **instante 10**, el hilo de ejecución **T2 se liberará** y podrá actualizar los datos.
 - Entonces podemos observar que **este nivel de aislamiento nos asegurará lecturas no sucias, NO permite registros fantasma y asegura que todas las lecturas sean repetibles**.

Deadlock

jueves, 8 de julio de 2021 06:07 p. m.

- Un **Deadlock** se produce cuando **dos transacciones o procesos compiten** por el **acceso exclusivo** a un **recurso**, pero **no son capaces de poder obtener acceso exclusivo** a él **porque el otro proceso lo impide**.
- Esto da como resultado un **enfrentamiento donde ninguno de los procesos puede continuar**. La única forma de **salir del estancamiento** es que **uno de los procesos sea concluido**.
- Cuando se produce un **DeadLock** los **DBMS** normalmente **matan la transacción más nueva**, permitiendo que **finalice correctamente la transacción más vieja**.

Ejemplo:

- Para el siguiente caso, tenemos un **DeadLock** porque cada transacción quiere leer de datos no confirmados de otra.

```
CREATE TABLE ##suben (valor INT)
CREATE TABLE ##bajan (valor INT)
INSERT INTO ##suben VALUES (1)
INSERT INTO ##bajan VALUES (9)
```

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

1	BEGIN TRAN T1	
2		BEGIN TRAN T2
3	INSERT INTO ##bajan values (8)	
4		INSERT INTO ##suben VALUES (2)
5	SELECT * FROM ##suben	
6		SELECT * FROM ##bajan
7	COMMITTRAN T1	
8		COMMITTRAN T2

Base de Datos

jueves, 8 de julio de 2021 12:01

Conjunto de datos persistentes e interrelacionados

que es utilizado por los sistemas de aplicación de una empresa,

Estos **datos** son almacenados en un **conjunto independiente y sin redundancias**.

(DBMS – Data Base Manager System)

- Sistema de Administración de Base de Datos
- Es lo que nosotros llamamos **Motor de Base de datos**

Programa que permite **administrar la base de datos** y se encarga de todas las **funcionalidades de la misma**.

El DBMS ofrece a los usuarios:

- **Percepción de la base de datos**
 - Al estar por encima del nivel del hardware, **entiende** como están lógicamente almacenados los datos, permitiéndole al usuario poder **observar el contenido de la base a través de consultas SQL**.
- **Manejar las operaciones del usuario** expresadas en el nivel más alto de percepción.

El DBMS también interpreta y ejecuta todas los comandos SQL.

Ejemplos:

- Oracle
- MS SqlServer
- DB2
- MySql

- PostgreSQL
- Informix
- Sybase

Componentes de una DB

jueves, 8 de julio de 2021 12:06

• DATOS

- Son **integrados** (Integridad de Entidades), provocando que se **minimice las redundancias de los mismos**.
- Son **compartidos** ya que **múltiples usuarios pueden acceder a los mismos datos** en forma concurrente (**al mismo tiempo**).

• TECNOLOGIA

- **Hardware** utilizado para almacenar y ejecutar todo lo necesario.
- **Sistema Operativo** con el cual se comunica el Motor.

• PROGRAMAS / PROCESOS

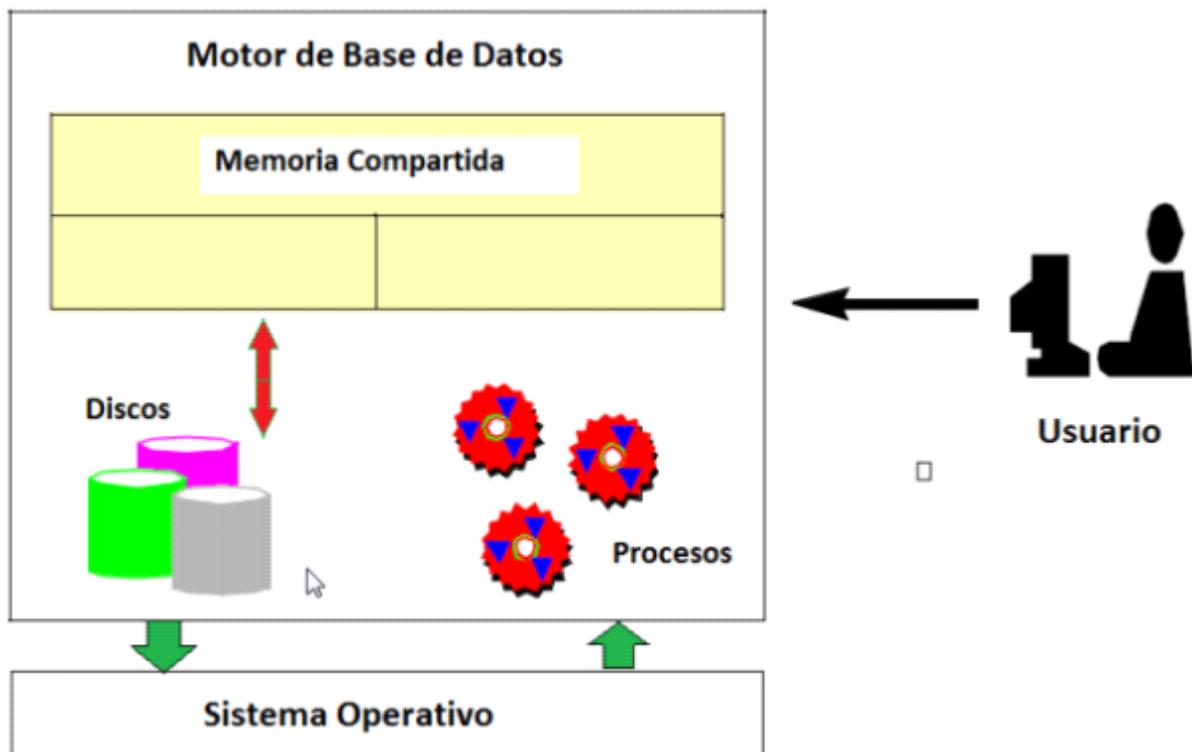
- **Componentes del DBMS** que están **corriendo constantemente para poder hacer que el Motor funcione correctamente**.
 - Estos controlan todas las actividades del DBMS y abstraen a los usuarios de los detalles tecnológicos.

• USUARIOS

- Programador de Aplicaciones
- DBA – DataBase Administrator
- Usuarios Finales
- Otros perfiles funcionales.

Comunicación

jueves, 8 de julio de 2021 12:52



Motor interactúa con el SO

Motor tiene área de memoria compartida, estructuras de disco y procesos corriendo.

Usuario se conecta al motor y puede trabajar en él.

Componentes principales

• Procesos

- Controlan el motor de Base de Datos.
- Proveen **funcionalidades específicas** asociadas a **seguridad, integridad, operación, organización interna entre otros**, el usuario no tiene que ponerse a crear funcionalidades para esto.

nota: procesos que corren on demand o están corriendo (Ej: Optimizer).

• Memoria Compartida

- Cachear datos del disco (queries hechas recientemente) para tener un acceso más rápido.
- Mantener y controlar los recursos necesarios para los procesos.
 - Por ejemplo los loqueos los mantiene en memoria.

- Proveer **mecanismos de comunicación** para los procesos clientes o propios del motor para **conversar con otras aplicaciones y coordinar actividades**.
 - En algunos casos (Ej: Informix) evita que se use comunicación por red (Port-TCP), provocando que sea **mucho mas rápida la conexión a la base de datos**.
- **Unidades de Discos**
 - Colección de una o más unidades de espacio de disco asignadas al DBMS.
 - Esta colección **almacena** toda la **información de las bases de datos**, más la **información propia del sistema** necesarias para **mantener el DBMS**.
 - Datos almacenados físicamente y de manera duradera.
 - **Nos asegura la Durabilidad** de los datos en la base.

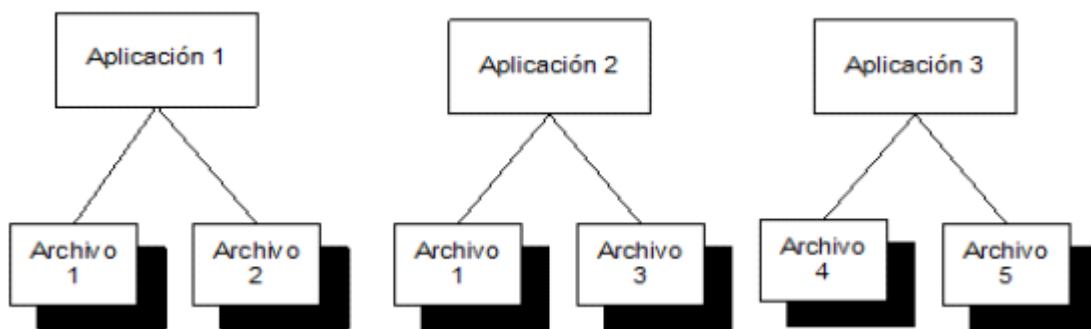
Enfoque orientado a la aplicación

jueves, 8 de julio de 2021 13:12

Aplicación tradicional, ya ahora todo lo realiza el Motor de base de datos.

Todos los **controles** que actualmente los hace el Motor **antes se realizaba a nivel aplicación**, o sea **externo al Motor**.

- Esto derivaba en que la **aplicación** tenía **muchas dependencias** de los **datos a acceder**, **aplicación totalmente acoplada a la estructura del archivo**.



En el ejemplo, supongamos un sistema simple que consta de tres programas o aplicaciones a la vista del usuario.

Cada programa manipula a nivel físico los archivos, peticionando al S.O. las operaciones de lectura/escritura sobre los mismos.

El desarrollador debe tener en cuenta durante el diseño y construcción factores relativos tales como:

conurrencia entre la aplicación 1 y 2, ya que ambas utilizan el Archivo 1.

Enfoque de Bases de Datos

jueves, 8 de julio de 2021 13:16

Las **aplicaciones** se tienen que conectar al Motor de base de datos para acceder a los datos.

Las **aplicaciones** son **independientes de los datos**, desacoplando la aplicación de los datos a acceder.

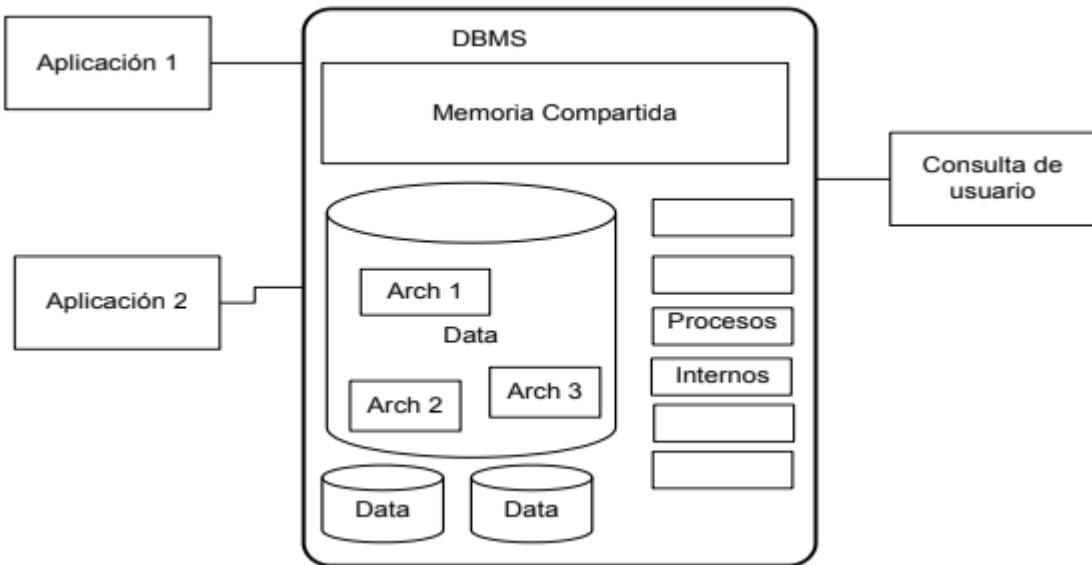
- **EVITAR USAR EL SELECT * FROM tabla**, esto **SI** te acopla a la estructura de la tabla y empeora el tráfico de datos.
- **USAR SELECT campo1, campo2, ... FROM tabla**

El motor debe **manejar internamente** los controles de **seguridad, concurrencia, performance**, etc.

Ventajas

- **Seguridad**
 - user y password para acceder al motor, y ese user tiene acceso a determinados recursos del motor.
- **Estándares de documentación y normalización de nomenclaturas.**
- **Redundancia mínima**
- **Consistencia de datos**
- **Transacciones**
- **Concurrencia en acceso a datos** (distintos niveles de aislamiento)
- **Integridad de los datos**
- **Restricciones, Constraints**
- **Objetos de BD**
 - Ej: triggers
- **Criterios y normativas para organización de datos**
- **Independencia de los programas vs las apps.**

Ejemplo



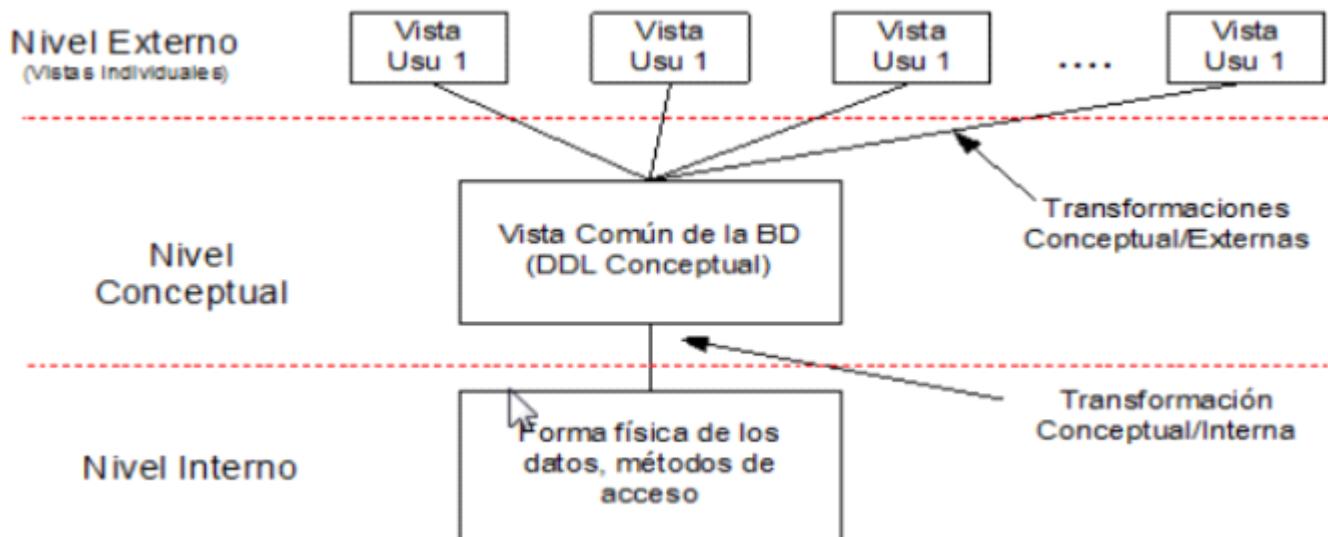
En este ejemplo los archivos de datos (tablas) residen en la Base de Datos, y los programas o consultas de usuarios son enviadas mediante algún mecanismo de comunicación estándar al DBMS, que es quién realiza realmente la consulta/actualización y devuelve el resultado al programa.

El DBMS se encarga entre otras cosas de la seguridad, concurrencia, etc. Las peticiones no se hacen al S.O. sino al DBMS mediante instrucciones SQL.

Arquitectura de Sistema de BD (ANSI / SPARC)

jueves, 8 de julio de 2021 13:26

Arquitectura que representa como es una Base de Datos por dentro, esta presenta 3 capas/niveles:



• Nivel Externo o Vistas

- **Vistas individuales a cada usuario, es una vista externa ya que es como ve el contenido de la base algún usuario en particular.**
 - Ej: Un programa en Java que muestra la tabla Clientes, o hacer una query de SELECT dentro del Manager de la Base de Datos (SSMS), o el resultado de una View.
- **La percepción de cada usuario de la base de datos es una vista externa.**
- El nivel más cercano al usuario.

• Nivel Conceptual o Lógico

- Vista conceptual la base de datos, definida mediante un esquema conceptual creado a partir de DDL.
 - Definiciones del contenido de la base
 - tipos de datos.
 - Restricciones.
 - Reglas de integridad.
 - Ej: indices, not nulls, unique, filas, columnas, etc
- Normalmente lo usan los DB admins, desarrolladores o analistas.

• Nivel Interno o Físico

- **Vista de cómo se almacenan los datos físicamente.**

- Métodos de acceso.
 - Tamaños de pagina.
 - Estructuras internas de disco y memoria.
 - Secuencia física en la que se encuentran los registros.
- **Representación de bajo nivel** de toda la base de datos.
- Es el nivel más cercano al **Sistema Operativo**
- Entre medio de los niveles hay distintas transformaciones**
- Transformación Externa / Conceptual
 - Dada una determinada vista externa, esta transformación define la correspondencia entre dicha vista externa y la vista conceptual.
 - Transformación Conceptual / Interna.
 - Define la correspondencia entre la vista conceptual y la base de datos almacenada, y especifica la representación en el nivel interno de las filas y columnas del modelo conceptual.

Funcionalidades de un DBMS

jueves, 8 de julio de 2021 16:44

el Motor tiene distintas funcionalidades para mantener en correcto funcionamiento y estado la DB.

Administrar el Diccionario de Datos

jueves, 8 de julio de 2021 18:24

- **Diccionario:** Conjunto de tablas de la base de datos del sistema, que **define** cada uno de los **objetos de la base** dentro del mismo (metadatos).
 - SYS.OBJECTS
 - SYS.PROCEDURES
 - SYS.ALL_SQL_MODULES
 - SYS.PARAMETERS
- **Estas tablas no pueden ser alteradas por ningún usuario** de la base de datos, pero al realizar un CREATE por ejemplo se modifican indirectamente.

○

Por ejemplo de una Tabla, tiene almacenado todos los parámetros propios de la definición de la tabla, sus campos y sus restricciones (constraints), así como las relaciones entre las tablas.

Sentencias relacionadas

- **CREATE** – Creación de objetos de BD (Tablas, indices, vistas, etc.)
- **ALTER** – Modificación de objetos de BD
- **DROP** – Eliminación de objetos de BD

Control de Seguridad

jueves, 8 de julio de 2021 18:24

La base tiene programas y procesos corriendo por detrás para asegurar:

- **Autenticación de usuarios**
 - El usuario existe y su clave es correcta permitiendo el acceso a la BD.
- **Autorización para que los usuarios realicen determinadas acciones**
 - Determinar los permisos que tiene el usuario para ejecutar acciones sobre diferentes objetos de la BD.

Para ello usa un **catálogo** formado por **entidades e interrelaciones** (en un **esquema relacional** van a ser **tablas**).

Entidades a tener en cuenta para administrar la seguridad, almacenados en el catálogo:

- **Usuarios**
 - Internos (Creados en la DB)
 - Externos (Propios del SO, EJ: Usuario de Windows Server)
 - Mixtos
- **Roles o Perfiles**
 - Built-in roles (Creados por la DB)
 - SysAdmin
 - User defined roles (Creados por quien administra la Base)
 - EJ: Rol de Ventas para poder realizar acciones sobre objetos que estén en el esquema Ventas solamente.
 - En caso de irle otorgando permisos a un usuarios estos se suman, o sea va quedando las acciones más permisivas (no restringe para menos).
 - Si alguien ya tenía permisos para leer/escribir TODO en la DB y se le otorga un rol Ventas para que solo lea la tabla Ventas, resultará que el usuario va a seguir pudiendo leer/escribir TODO en la DB.
- **Acciones posibles a realizar sobre los objetos**

- Realizar cierta acción sobre cierto objeto
 - Hacer un SELECT sobre una tabla
- **Objetos de la BD sobre los que se otorgarán permisos**

Sentencias SQL relacionadas

GRANT – Para otorgar permisos.

REVOKE – Para revocar permisos **previamente otorgados**

Otros objetos relacionados con la seguridad

- **Vistas**
 - Muestra los datos que nosotros elegimos que muestre
 - Con ciertos permisos podemos restringir que no se modifiquen datos desde estas mismas.
- **Triggers**
 - Chequear permisos de horarios para utilizar objetos
- **Stored Procedures**
 - Ejecutar controles de permisos
- **Sinónimos**
 - Esconder donde se encuentra ciertos objetos
- **Funciones**
 - Ocultando funcionalidad de ciertos procesos

Garantizar Integridad de Datos

jueves, 8 de julio de 2021 18:26

El DBMS cuenta con **distintos mecanismos** para poder controlar la **integridad de los datos** que se contienen en la/s Base/s de Datos.

Cuenta con **distintas restricciones y objetos** que pueden ser creados, y a partir de ello el DBMS cuenta con **procesos** que se encarga de controlar que se cumplan dichas restricciones asegurando la integridad de los datos.

CONSISTENCIA E INTEGRIDAD NO SON SINONIMOS

- **Datos Íntegros cumplen las reglas de Integridad**
 - Implementadas con **Constraints**, estas aseguran la **Integridad de los Datos**.
 - UNIQUE, NOT NULL, CHECK, DEFAULT
 - PRIMARY KEY
 - FOREIGN KEY

Objetos relacionados con la Integridad

- **CONSTRAINTS**
- **Triggers**. eg: chequear como una FK contra un elemento de otra DB.
- **Indices (únicos)**
 - al crear PK o UNIQUE el motor crea un índice único.
- **Views (con check option)**
 - Check Option otorga cierta integridad sobre la vista no permitiendo cargar datos no mostrables por esta.
 - Ej: no puedo insertar un campo con una columna sobre la que la view no hace select.
- **Stored Procedures y Funciones**
 - Chequeo con SP revisando la integridad entre 2 tablas de una db en las que no tengo PK o FK.

- nota: transacciones NO ayudan en la integridad, sino en la consistencia.

Garantizar Consistencia de Datos

jueves, 8 de julio de 2021 18:26

El DBMS cuenta con **distintos mecanismos** para poder asegurar la consistencia de los datos que existentes en la/s Base/s de Datos.

Consistencia NO depende de las reglas de integridad.

- Está más relacionado a **reglas y criterios de negocio** que a reglas de integridad.

Objetos relacionados con la Consistencia

- **Transacciones**
 - **Garantizar** que la **operación completa se ejecute o falle, no permitiendo** que queden **datos inconsistentes**.
 - **Ejecutadas en una unidad lógica de trabajo (Atomicidad)**
 - **Llevan datos de estado correcto a otro estado correcto.**
 - Sentencias (INSERT, UPDATE o DELETE) son singleton transaction.
- **Logs Transaccionales**
 - **Registro** de las **transacciones realizadas** a medida que se van haciendo, incluso se graba antes la información de la transacción en este log antes de realizarla.
 - **Permite deshacerlas** (por eso está asociado a la consistencia).
- **Recovery**
 - **Método de recuperación ante caídas.**
 - Al prender el motor revisa si fue apagado de forma correcta en el log de transacciones.
 - Busca el checkpoint (último momento con datos consistentes entre memoria y disco, o sea que grabó en disco)
 - A partir del checkpoint el **Motor revisa el log transaccional** verificando que **todo** lo que indican **está en disco, rollbackeando las que no están**.

Propiedades de DB en función de las transacciones:

○ Atomicidad

- La transacción no puede ser dividida, o se ejecuta toda o no se ejecuta

○ Consistencia

- La transacción parte de un punto en que la Base está consistente y debe dejarla en otro punto consistente, la consistencia la determinan las reglas de negocio que se definan para esa transacción.

○ Isolation (Aislamiento)

- Dos transacciones concurrentes no se molestan entre si.

○ Durabilidad

- Un dato cuando es grabado es durable hasta que uno determine borrarlo.

Resguardo y restauración (BACKUP and RESTORE)

jueves, 8 de julio de 2021 18:27

Motores de DB cuentan con **distintas herramientas** que permiten **realizar estas acciones**.

Backup es la copia total o parcial de la información de una base de datos la cual debe ser guardada en algún otro sistema de almacenamiento masivo.

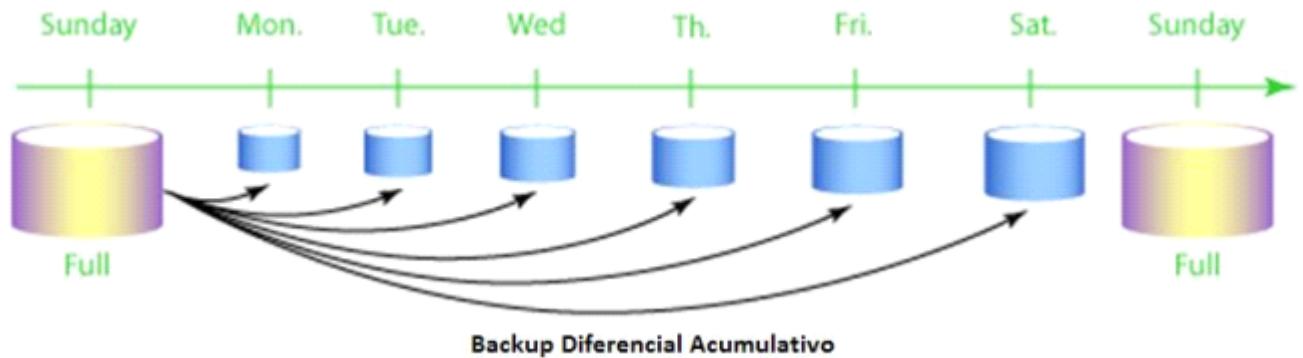
- Se pueden guardar TODOS los datos en otra unidad de almacenamiento ó guardar los logs transaccionales a medida que se van ejecutando transacciones.

Restore es la acción de tomar un back up ya realizado y restaurar la estructura y datos sobre una base dada.

Tipos de Backup

- **Backup diferencial acumulativo / incremental**
 - Solo se guardan las modificaciones a partir de cierta fecha.
- **Backup completo**
 - Se guardan todos los datos.
- **Backup en caliente**
 - Se realiza mientras el aplicativo está en uso, usuarios trabajando con la base.
- **Backup en frío**
 - Se realiza con el aplicativo bajo, la base de datos solo para el backup.
- **Backup de Logs Transaccionales**
 - Se realizan sobre los logs de transacciones a medida que se van generando transacciones.

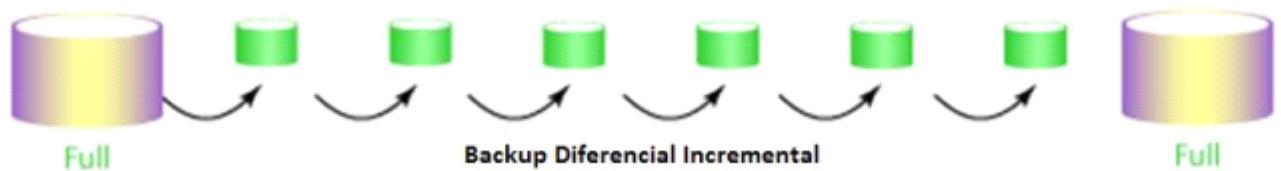
Backup Diferencial Acumulativo



Acá se va **guardando cada día lo que se hizo comparándolo con el backup full** (en el ejemplo el domingo).

Ejemplo: En caso de que falle el miércoles debe restaurarse el full + el del martes.

Backup Diferencial Incremental



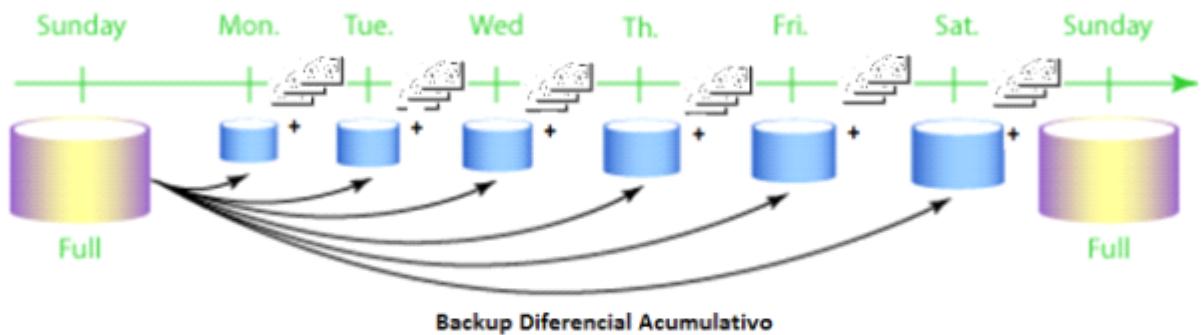
Acá se va **guardando las diferencias entre días anteriores**, el lunes se guarda la diferencia entre ese día y el full, el martes se guarda lo que se hizo en el día y el lunes, y así consecutivamente.

Ejemplo: En caso de que falle el miércoles debe restaurarse el full + el del lunes + el del martes.

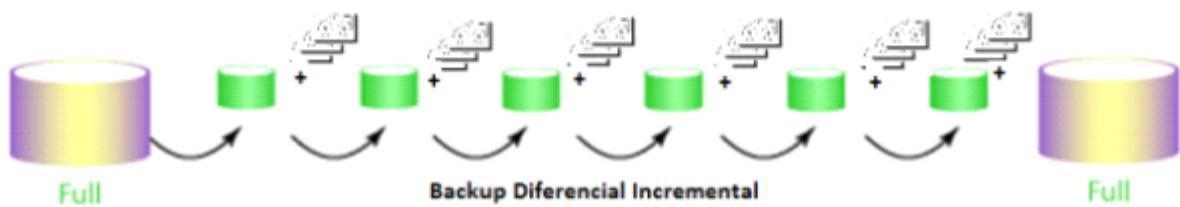
Los dos modos de backup **están resguardados con el backup de los logs transaccionales**, ya que no hay forma de backuppear los datos si es que ocurre algo en el medio de un día.

Se levanta el backup hasta ese cierto día y se ejecutan las transacciones hasta el momento del accidente.

Backup de DATA + Logs Transaccionales



Backup Diferencial Acumulativo



Backup Diferencial Incremental

Control de Concurrencia

jueves, 8 de julio de 2021 18:27

Los motores permiten controlar el acceso concurrente a sus recursos a través de loqueos y de la definición de niveles de aislamiento (isolation levels)

Loqueos

- Granularidad de Loqueos
- Nivel de Base de Datos.
- Nivel de Tabla
- Nivel de Página
- Nivel de Fila
- Nivel de Clave de Índice

Tipos de Loqueos

- Compartido (Shared)
- Exclusivo (Exclusive)
- Promovible (Promutable - Update).

Niveles de Aislamiento

- Read uncommitted (El más permisivo)
- Read committed
- Repeatable read
- Serializable Read (El menos permisivo)

ISOLATION LEVELS – Resumen

	Dirty Read	Repeatable Read	Phantom Read
Read Uncommitted	SI	NO	SI
Read Committed	NO	NO	SI
Repeatable Read	NO	SI	SI
Serializable Read	NO	SI	NO

Dirty Read/ Lectura Sucia

Datos actualizados en una transacción concurrente que luego se deshacen por un rollback.

Phantom Reads/Lecturas Fantasmas

Filas que aparecen durante la transacción que fueron insertadas en otra transacción concurrente.

Repeatable Read (la columna de la tabla de acá arriba)

En la misma transacción poder hacer dos veces la misma consulta asegurando siempre el mismo resultado.

Manejo de DeadLock

Un **Deadlock** se produce cuando **dos transacciones o procesos compiten** por el **acceso exclusivo** a un **recurso**, pero **no son capaces de poder obtener acceso exclusivo** a él porque el otro proceso lo impide.

- Cuando se produce un **DeadLock** los **DBMS** normalmente **matan la transacción más nueva**, permitiendo que **finalice correctamente la transacción más vieja**.

Manejo de Lockeos - Deadlock

Para nuestro siguiente caso, tenemos un deadlock. Cada transacción quiere leer de datos no confirmados de otra.

```
CREATE TABLE ##suben (valor INT)
CREATE TABLE ##bajan (valor INT)
INSERT INTO ##suben VALUES (1)
INSERT INTO ##bajan VALUES (9)
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

1	BEGIN TRAN T1	
2		BEGIN TRAN T2
3	INSERT INTO ##bajan values (8)	
4		INSERT INTO ##suben VALUES (2)
5	SELECT * FROM ##suben	
6		SELECT * FROM ##bajan
7	COMMIT TRAN T1	
8		COMMIT TRAN T2

Otras funcionalidades

jueves, 8 de julio de 2021 18:37

Facilidades de auditoría

A los DBMS se les puede activar la opción de guardar en un log un registro del uso de los recursos de la base para auditar posteriormente.

Logs del sistema

Mediante este tipo de logs, el DBA puede llegar a determinar cuál fue, por ejemplo, el problema que produjo una caída del sistema.

Acomodarse a los cambios, crecimientos, modificaciones del esquema.

El motor permite realizar cambios a las tablas constantemente, casi en el mismo momento en que la tabla está siendo consultada.

Optimizador de Queries

Los DBMS tienen un módulo que cuando se ejecuta una query que evalúa cual es la mejor estrategia más optima de ejecutarla a partir de las estadísticas almacenadas por el motor.

Mirroring de Discos

Grabar en más de un disco y mientras el motor lee los datos del primario o del slave si el primero falla.

Data Replication

Correr el motor en un server principal y varios slave para que alguno de estos tome el control ante una caída.

Data Encryption

Monitoring Features

Dashboard y gráficos con información del sistema.

Event Alarms

Ejemplo: Alarma de x porcentaje de disco lleno

Particionamiento de Tablas e Índices

Particionar tablas y guardar particiones en diferentes discos
acelerando la búsqueda.