ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE MADRID

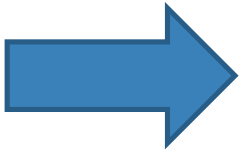
Bases de Datos

Isabel Riomoros

1 .- Introducción a las Bases de Datos

2.- Modelo Entidad-Relación

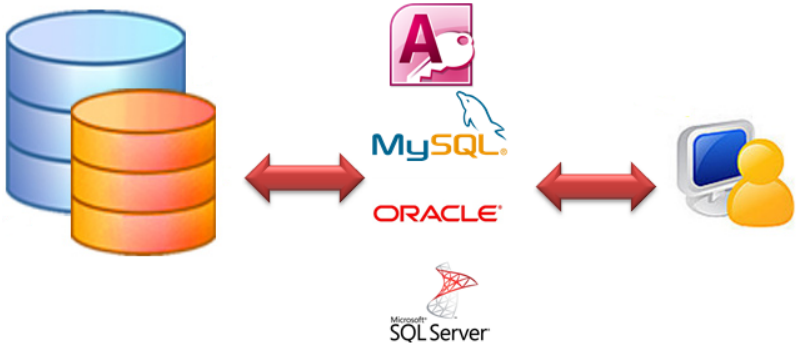
3.- Modelo Relacional



4.- SQL

4

SQL



SQL (Structured Query Language) es un lenguaje estándar e interactivo de acceso a bases de datos relacionales, permite realizar distintas operaciones en ellas, gracias a la utilización del álgebra y del cálculo relacional. SQL nos ofrece la posibilidad de realizar consultas con el fin de recuperar información de las bases de datos, realizando este proceso de forma sencilla. Las consultas permiten seleccionar, insertar, actualizar, averiguar la ubicación de los datos, …

SQL - (*Structured Query Language*).

SQL es un **lenguaje de consulta,** un lenguaje en el que el usuario solicita información de la base de datos, suelen ser de un nivel superior a los lenguajes de programación.

Además de consultas, con SQL, es posible definir la estructura de los datos, modificar los datos de la base de datos y especificar restricciones de seguridad.

SQL se basa en la Teoría Matemática del Álgebra Relacional. El lenguaje SQL consta de varios elementos:

❑**Lenguaje de definición de datos (DDL)**: proporciona órdenes para definir, modificar o eliminar los distintos objetos de la base de datos (tablas, vistas, índices…). **(CREATE, DROP, ALTER, ..)**

❑**Lenguaje de Manipulación de Datos (DML)**: proporciona órdenes para insertar, suprimir y modificar registros o filas de las tablas. También contempla la realización de consultas sobre la BD. **(SELECT, INSERT, UPDATE y DELETE)**

❑**Lenguaje de Control de Datos (DCL)**: permite establecer derechos de acceso de los usuarios sobre los distintos objetos de la base de datos. Lo forman las instrucciones **GRANT** y **REVOKE**.

SQL - (*Structured Query Language*).

codArticulo	denom	precio	unidades
0001	Ord. Sobremesa	600.00	12
0002	Ord. Portátil	1000.00	6
0003	Tarjeta Red	20.00	25
0004	Impresora Láser	200.00	4
0005	Ratón USB	7.00	50
0006	Monitor TFT	250.00	10
0007	Router inalámbrico	100.00	30
NULL	NULL	NULL	NULL

- Las distintas implementaciones de **SQL** pueden diferenciarse en detalles, o pueden admitir sólo un subconjunto del lenguaje completo.

- El resultado de ejecutar una instrucción **SQL** es una tabla (tabla resultado) con los registros que cumplen la instrucción ejecutada.

Algunas consideraciones

En SQL no se distingue entre mayúsculas y minúsculas. El final de una instrucción o sentencia lo marca el signo de punto y coma.

Las sentencias SQL (SELECT, INSERT, …) se pueden escribir en varias líneas siempre que las palabras no sean partidas.

Los comentarios en el código SQL pueden ser de 2 tipos:

```
/*
  Esto es un comentario
  de varias líneas.
   Fin.
*/
 -- Esto es un comentario de una línea
```

Algunas consideraciones

Las relaciones de cada base de datos debe especificarse en el sistema en términos de un lenguaje de definición de datos (LDD)

Además de las relaciones, se define la información relativa a ellas:

- Esquema de cada relación
- Dominio de valores asociados a cada atributo
- Restricciones de integridad
- Índices que se mantienen para cada relación
- Información de seguridad y autorización de cada relación
- Estructura de almacenamiento físico de cada relación en disco

Tipos de datos en SQL (dominios)

char(n). Cadena de caracteres de **longitud fija** n, especificada por el usuario

varchar(n). Cadena de caracteres de **longitud variable** con una longitud máxima n especificada por el usuario.

int. Integer, un subconjunto finito de los enteros depende de la máquina.

smallint. Small integer (un subconjunto dependiente de la máquina del tipo dominio entero).

real, double precision. Número en **coma flotante** y números en coma flotante de doble precisión, con precisión dependiente de la máquina.

numeric(p,d). Un número en **coma fija**, cuya precisión la especifica el usuario. El número está formado por p dígitos (más el signo) y de esos p dígitos, d pertenecen a la parte decimal.

float(n). Un número en **coma flotante** cuya precisión es de al menos **n dígitos**

date: Fechas, contiene un año (4 dígitos), mes y día
time: Hora del día, en horas, minutos y segundos.
timestamp: fecha y hora del día
interval: periodo de tiempo

Definición básica de esquemas SQL

Para crear una tabla utilizamos el comando **create table**:

CREATE TABLE r (
A_1 D_1, A_2 D_2, ..., A_n D_n,
(restricción de integridad $_1$),
...,
restricción de integridad$_k$)
);

r es el nombre de la relación

Cada A_i es un atributo del esquema de relación r
D_i es el tipo del dominio del atributo A_i

```
create table sucursal (
        nombre_sucursal    char(15) ,
        ciudad-sucursal    char(30),
        activos            numeric(16,2)
        Primary key nombre_sucursal)
```

```
CREATE TABLE cliente (
  codCliente  char(3),
  nombreC varchar(40) not null,
  direccion varchar(40) not null,
  telefono numeric(9,0),
  PRIMARY KEY(codCliente)
);
```

Restricciones de integridad

Las restricciones de integridad protegen contra problemas accidentales en la base de datos, asegurando que los cambios con autorización en la base de datos no generan pérdidas en la consistencia de los datos.

- Obligatoriedad**, NOT NULL**
- Clave primaria**, PRIMARY KEY** (A_1, ..., A_n)- los atributos han de ser no nulos y únicos
- Clave ajena**, FOREIGN KEY** (A_1) **REFERENCES** r (A)
- Verificación de condiciones**, CHECK**
- **AUTO_INCREMENT**
- Valores por defecto**, DEFAULT**

- Una cuenta debe tener un saldo mayor que 10.000,00€
- El salario de un empleado del banco no puede ser menor de 4,00€ la hora.
- Un cliente debe tener un número de teléfono (no nulo)

```
CREATE TABLE  provincia(
   id SMALLINT AUTO_INCREMENT,
   nombre VARCHAR(30) NOT NULL,
   superficie INTEGER DEFAULT 0,
   habitantes INTEGER DEFAULT 0,
   idComunidad SMALLINT NOT NULL,
   PRIMARY KEY (id),
   FOREIGN KEY (idComunidad) REFERENCES comunidad(id)
   ON DELETE CASCADE
   ON UPDATE CASCADE
);
```

Claves

- Una **clave primaria** es una columna o conjunto de columnas que el diseñador ha elegido para identificar de manera única una fila de una tabla.
- Las claves proporcionan una forma rápida y eficiente de buscar datos en una tabla
- Una **clave ajena o externa** es una columna en una tabla que se corresponde con la clave primaria de otra tabla.
- Una **clave candidata** es un conjunto no vacío de atributos que identifican unívoca y mínimamente cada tupla de una relación.

Restricción not null

Declarar que el *nombre_sucursal* de una *sucursal* es **not null**
nombre_sucursal **char**(15) **not null**

Declarar que el dominio *Euros* sea **not null**
create domain *Euros* **numeric**(12,2) **not null**

Restricción unique

- La especificación unique indica que los atributos $A_1, A_2, \ldots A_m$ constituyan una clave candidata.
- Las claves candidatas pueden tener atributos nulos (al contrario que las claves primarias)

Cláusula check

- **check** (*P*), donde *P* es un predicado

```
CREATE TABLE persona (
    CodCliente  char(3),
    nombreC varchar(40) not null,
    fechaNac date,
    fechaBoda date,
    telefono  numeric(9,0),
    num int,
   PRIMARY KEY(CodCliente));
alter table persona add constraint check(num>0);
alter table persona add constraint check(fechaBoda>fechaNac);
```

```
CREATE TABLE sucursal
    (nombre_sucursal   CHAR(15),
     ciudad_sucursal    CHAR(30),
     activos              INTEGER,
     PRIMARY KEY (nombre_sucursal),
     CHECK (activos >= 0));
```

Ejemplo: Declarar *nombre_sucursal* como clave primaria para *sucursal* y asegurar que el valor de *activos* no sea negativo.

Integridad referencial

- Asegura que un valor que aparece en una relación para un conjunto de atributos determinado aparezca también en otra relación para un cierto conjunto de atributos.

Ejemplo: *Si "As Pontes" es un nombre de sucursal que aparece en una de las tuplas de la relación cuenta, entonces existirá una tupla en la relación sucursal para la sucursal "As Pontes".*

Si el código del artículo 'Ord. Sobremesa' aparece en la relación compra, ese código de artículo debe aparecer en la relación artículo.

Las claves primarias, candidatas y las claves externas o ajenas se pueden especificar como parte de la instrucción **create table** de SQL:

- La cláusula **primary key** incluye una lista de los atributos que comprende la clave primaria.
- La cláusula **unique key** incluye una lista de los atributos que comprende una clave candidata.
- La cláusula **foreign key** incluye una lista de los atributos que comprende la clave externa y el nombre de la relación a la que hace referencia mediante la clave externa. Por defecto, una clave externa hace referencia a los atributos de la clave primaria de la tabla referenciada.

La diferencia entre unique y primary key, una clave unique permite nulos, en cuanto una primary key no permite nulos es decir ya incluye la constraint de not null para cada atributo.

Restricción clave ajena

```
FOREIGN KEY(idCliente) REFERENCES cliente(CodCliente)
        ON DELETE cascade,
```

- Sirve para relacionar dos o más tablas, se necesita un campo en común, por ejemplo, idCliente y codCliente, existe en cliente y en compra.
- Si queremos eliminar algún cliente, las filas que se correspondan en compras con ese cliente serán eliminadas automáticamente.

Ejemplo creación de tablas

```
CREATE TABLE cliente (
  CodCliente  char(3),
  nombreC varchar(40) not null,
  direccion varchar(40) not null,
  telefono  numeric(9,0),
  PRIMARY KEY(CodCliente)
);
```

```
CREATE TABLE articulo (
  codArticulo  char(4),
  denom    varchar(40) not null,
  precio    numeric(6,2) not null,
  unidades    integer,
   descuento numeric(3,0),
  PRIMARY KEY(codArticulo)
);
```

```
CREATE TABLE compra (
  idCliente     char(3),
  idArticulo     char(4),
  fecCompra  date     not null,
  numUnidades   integer,
 CHECK(numUnidades)>0,
  PRIMARY KEY(idCliente, idArticulo),
  FOREIGN KEY(idCliente) REFERENCES cliente(CodCliente)
        ON DELETE cascade,
  FOREIGN KEY(idArticulo) REFERENCES articulo(codArticulo)
        ON DELETE cascade
);
```

Definición básica de esquemas SQL

Con **insert into** añadimos datos a la relación

INSERT INTO r **VALUES** (A_1,…,A_n)

Para borrar todas las tuplas de una relación, utilizamos **delete from**

DELETE FROM r

Con **drop table** eliminar una relación en una base de datos. Elimina el esquema de la relación.

DROP TABLE r

alter table este comando sirve para añadir atributos a una relación existente

ALTER TABLE *r* **ADD** *A D*

r es la relación, *A es el atributo a añadir,* *D es el dominio del atributo A*

A todas las tuplas de la relación se les asigna un valor null para el nuevo atributo.

El comando **alter table** también se puede utilizar para borrar atributos de una relación:

ALTER TABLE *r* **DROP** *A*

La sentencia **UPDATE** sirve para actualizar datos de las tablas de una BD.

```
UPDATE empleados
SET Direccion='Gran Vía 241', telefono='686567687'
WHERE Nombre='Ana García';
```

Si se omite WHERE, se actualizan todas las filas de la tabla destino.

Ejemplos

```
INSERT INTO cliente VALUES ('015', 'Pedro Glez.', 'Gerona 14',      917845308);
INSERT INTO articulo VALUES ('0001', 'Ord. Sobremesa',     600, 12);
INSERT INTO compra VALUES('015', '0007', '2015/11/06', 2);
 INSERT INTO cuenta VALUES ('A-9732', 'Navacerrada',1200);
INSERT INTO cuenta VALUES ('A-777','Navacerrada', null)
```

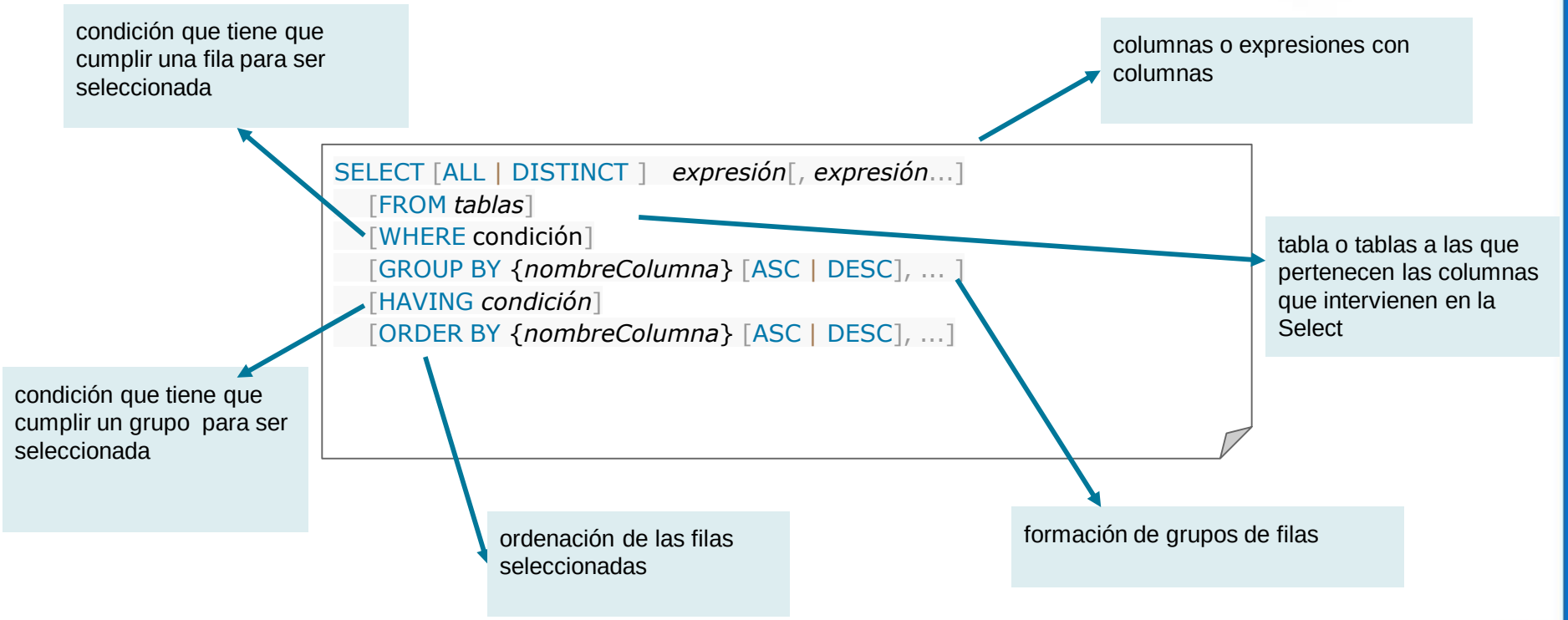
```
DROP DATABASE IF EXISTS tiendaInformatica;
DROP DATABASE tiendaInformatica;

CREATE DATABASE tiendaInformatica;

USE tiendaInformatica;

DROP TABLE IF EXISTS compra;
DROP TABLE IF EXISTS cliente;
DROP TABLE IF EXISTS articulo;
```

Cargar datos de un archivo

```
/*SET GLOBAL local_infile='ON';*/
LOAD DATA  INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/PERSONA.txt'
 INTO TABLE nombreTabla
FIELDS TERMINATED BY ';'
 LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

Los ficheros a cargar es necesario guardarles en el directorio que nos dé la siguiente select:
SELECT @@GLOBAL.secure_file_priv;
La carpeta dependiendo de la versión de workbench, es muy parecida a:
'C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/persona.csv'
Ojo / y no \

La claúsula SELECT



La claúsula SELECT

Una consulta característica de SQL tiene la forma:

SELECT $A_1, A_2, ..., A_n$
FROM $r_1, r_2, ..., r_m$
WHERE P

A_i representan los atributos
r_i representan las relaciones
P es un predicado

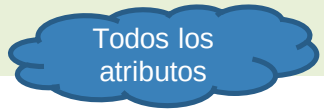
El resultado de una consulta de SQL es una relación.
La cláusula **SELECT** se utiliza para dar la relación de los atributos deseados en el resultado de una consulta

Ejemplo: *obtener los nombres de todas las sucursales en la relación prestamo:*

SELECT *nombre_sucursal*
FROM *prestamo*

Ejemplo: *obtener el listado de todos los datos de los artículos de la tienda de productos informáticos*

SELECT *
FROM *articulo*



La cláusula SELECT

Para forzar la eliminación de duplicados, insertar la clave **distinct** después de SELECT.

Obtener los nombres de todas las sucursales en las relaciones prestamos, y anular los duplicados

SELECT DISTINCT *nombre_sucursal*
FROM *préstamo*

La clave **all** especifica que los duplicados no se han anulado.

SELECT ALL *nombre_sucursal*
FROM *prestamo*

La cláusula **SELECT** puede contener expresiones aritméticas que involucran la operación, +, –, * y /, y que funcionan en las constantes o en los atributos de las tuplas.

La siguiente relación es la misma que préstamo, excepto que el atributo importe se multiplica por 100. pero no modifica la Base de Datos.

SELECT *número_prestamo, nombre_sucursal, importe * 100*
FROM *prestamo*

La cláusula WHERE

La cláusula **WHERE** especifica las condiciones que debe satisfacer el resultado

La búsqueda de todos los números de crédito de los prestamos ha dado como resultado la sucursal Navacerrada con las cantidades de prestamos mayores a 1200 €.

SELECT *número_prestamo*
FROM *prestamo*
WHERE *nombre_sucursal* = ‘ Navacerrada’ **AND** *importe* > 1200

Escribir los artículos que se han acabado

SELECT *codArticulo, denom*
FROM *articulo*
WHERE *unidades =0*

Los resultados de la comparación se pueden combinar utilizando las conectivas lógicas **and, or** y **not.** Las comparaciones se pueden aplicar a los resultados de las expresiones aritméticas.
SQL incluye un operador de comparación **BETWEEN**

Ejemplo: *Obtener el número de préstamode aquellos con cantidades de crédito entre 90,000€ y 100,000€*

SELECT *número_prestamo*
FROM *prestamo*
WHERE *importe* **between** 90000 **AND** 100000

La cláusula FROM

En la cláusula **FROM** se especifica una lista de las relaciones que se van a explorar en la evaluación de la expresión. Corresponde a la operación del producto cartesiano del álgebra relacional.

Buscar el producto cartesiano prestatario X prestamo

```
SELECT *
FROM prestatario, prestamo
```

- *Buscar el nombre, el número de préstamo y la cantidad del préstamo de todos los clientes que tengan un crédito en la sucursal Navacerrada.*

```
SELECT nombre_cliente, prestatario.numero_prestamo, importe
FROM prestatario, prestamo
WHERE prestatario.número_prestamo= prestamo.numero_prestamo
      AND   nombre_sucursal = 'Navacerrada'
```

nombreTabla.nombreAtributo

- *Nombre de los clientes que han comprado más de 3 unidades*

```
SELECT  distinct nombre
FROM cliente, compra
WHERE cliente.codCliente=compra.IdCliente AND compra.numUnidades>3
```

La operación de renombramiento

SQL permite renombrar las relaciones y atributos utilizando la cláusula **as**:

nombre_antiguo **as** *nombre_Nuevo*

Obtener el nombre, el número de préstamo y la cantidad del préstamo de todos los clientes; renombrar el nombre de la columna número_préstamo como identificador_prestamo.

```
SELECT nombre_cliente, prestatario.numero_préestamo AS identificador_prestamo, importe
FROM prestatario, prestamo
WHERE prestatario.numero_prestamo= prestamo.numero_prestamo
```

```
SELECT nombre_cliente, prestatario.numero_prestamo AS identificador_prestamo, importe
FROM prestatario AS p, prestamo AS pr
WHERE p.numero_prestamo= pr.numero_prestamo
```

```
SELECT cliente.nombreC, cliente.telefono
FROM cliente as cli, compra as co
WHERE cli.idCliente = co.idClienteAND compra.idArticulo = '0006';
```

La operación de renombramiento

Las variables tupla se definen en la cláusula **FROM** mediante el uso de la cláusula **as**.

- *Obtener los nombres , números de préstamo e importe de todos los clientes que tengan un préstamo en alguna sucursal.*

```
SELECT nombre_cliente, T.número_prestamo, S.importe
FROM prestatario as T, préstamoas S
WHERE  T.número_préstamo= S.número_prestamo
```

- *Obtener los nombres de todas las sucursales que tengan activos mayores que las sucursales situadas en Barcelona.*

```
SELECT distinct T.nombre_sucursal
FROM sucursal as T, sucursal as S
 WHERE T.activos> S.activos AND S.ciudad_sucural = ‘ Barcelona’
```

- *Obtener el nombre del cliente y el teléfono de todos los clientes que han comprado el articulo ‘0006’.*

```
SELECT nombreC, telefono
FROM cliente as cli, compra as co
WHERE cli.idCliente = co.idClienteAND co.idArticulo = '0006';
```

Operaciones con cadenas

SQL incluye un operador de coincidencia de cadenas para comparaciones de cadenas de caracteres.
*el operador “LIKE” utiliza patrones que son descritos por los caracteres especiales:

tanto por ciento**(%).** El carácter **%** encaja con cualquier **subcadena**.
guión bajo **(_)**. El carácter _ encaja con cualquier **carácter**.

Obtener los nombres de todos los clientes cuyas calles incluyan la subcadena “Mayor”.
SELECT *nombre_cliente*
FROM *cliente*
WHERE *calle_cliente* **LIKE** ‘%Mayor%’
Coincide el nombre “Mayor%” (para que puedan contener los caracteres especiales, se pone la palabra clave escape.

LIKE ‘Mayor\%’ **escape** ‘\’

SQL soporta una variable de operaciones con cadenas como concatenación (que utiliza “||”) conversión de mayúscula a minúsculas(y viceversa) upper() lower()

Búsqueda de la longitud de la cadena, extracción de subcadena, etc.

Orden en la presentación de las tuplas

Lista en orden alfabético los nombres de todos los clientes que tengan un crédito en la sucursal Navacerrada

SELECT distinct *nombre_cliente*
FROM *prestatario, prestamo*
WHERE *prestatario.número_préstamo=prestamo.número_préstamo* **AND**
sucural_nombre = 'Navacerrada'
order by *nombre_cliente*

Listar los clientes en orden descendente

SELECT * FROM cliente ORDER BY nombreC DESC;

Se puede especificar la cláusula **desc** para orden descendente o **asc** para orden ascendente, de cada atributo; ***el orden ascendente es el orden por defecto***.

Ejemplo: **order by** *nombre_cliente* **desc**

Operaciones con conjuntos

Las operaciones de conjuntos **union, intersect,** y **except** operan sobre relaciones y corresponden a las operaciones de álgebra relacional ∪, ∩, –.
Cada una de las operaciones antes citadas elimina duplicados automáticamente; para retener todos los duplicados se utilizan las versiones de multiconjunto correspondientes **union all, intersect all** y **except all.**

Obtener todos los clientes que tengan un préstamo, una cuenta o ambos:

(SELECT *nombre_cliente* **FROM** *impositor*)
union
(SELECT *nombre_cliente* **FROM** *prestatario)*

Obtener todos los clientes que tengan un préstamo y una cuenta.

(**SELECT** *nombre_cliente* **FROM** *impositor*)
intersect
(SELECT *nombre_cliente* **FROM** *prestatario)*

Obtener todos los clientes que tengan una cuenta pero no un préstamo.

(**SELECT** *nombre_cliente* **FROM** *impositor*)
except
(SELECT *nombre_cliente* **FROM** *prestatario)*

Funciones de agregación

Estas funciones operan en el multiconjunto de valores de una columna de una relación, y devuelven un valor

avg: valor medio
min: valor mínimo
max: valor máximo
sum: suma de valores
count: número de valores

Obtener el saldo medio de las cuentas de la sucursal Navacerrada.

SELECT avg *(saldo)*
FROM *cuenta*
WHERE *nombre_sucursal* = 'Navacerrada'

Obtener el número de tuplas de la relación cliente

SELECT count (*)
FROM *cliente*

Obtener el número de impositores en el banco

SELECT count (distinct *nombre_clientes)*
FROM *impositor*

Funciones de agregación – Group By

Obtener el número de impositores de cada sucursal.

SELECT *nombre_sucursal,* **count (distinct** *nombre_cliente)*
FROM *impositor, cuenta*
WHERE *impositor.número_cuenta = cuenta.número_cuenta*
group by *nombre_sucursal*

Los atributos de la cláusula SELECT fuera de las funciones de agregación deben aparecer en la lista group by

```
SELECT nombre_sucursal, sum(saldo), count(*), avg(saldo), min(saldo), max(saldo)
FROM cuenta
group by nombre_sucursal;
```

nombreSucur...	sum(saldo)	count(*)	avg(saldo)	min(saldo)	max(saldo)
Becerril	1000	3	333.3333	100	700
Centro	900	1	900.0000	900	900
Collado Mediano	42350	3	14116.6667	350	30000
Galapagar	235650	3	78550.0000	750	234000

Funciones de agregación –Cláusula Having

Clausula **Having** búsqueda por grupos

Obtener los nombres de todas las sucursales en las que el saldo medio de las cuentas es mayor de 1.200€.

```
SELECT nombre_sucursal, avg (saldo)
FROM cuenta
group by nombre_sucursal
having avg (saldo) > 1200
```

Los predicados de la cláusula** having **se aplican después de la formación de grupos mientras que los permitidos en la cláusula** WHERE **se aplican antes de la formación de grupos

Clausula WHERE y Having

La cláusula WHERE se aplica primero a las filas individuales de las tablas. Solo se agrupan las filas que cumplen las condiciones de la cláusula WHERE.

La cláusula HAVING se aplica a continuación a las filas del conjunto de resultados. Solo aparecen en el resultado de la consulta los grupos que cumplen las condiciones HAVING. Solo puede aplicar una cláusula HAVING a las columnas que también aparecen en la cláusula GROUP BY o en una función de agregado.

```
SELECT editorial, count(*)
FROM libros
 WHERE editorial<>'Planeta'
GROUP BY editorial;

SELECT editorial, count(*)
FROM libros
GROUP BY editorial
HAVING editorial<>'Planeta';
```

Ambas devuelven el mismo resultado, pero son diferentes.
La primera, selecciona todos los registros rechazando los de editorial "Planeta" y luego los agrupa para contarlos.

La segunda, selecciona todos los registros, los agrupa para contarlos y finalmente rechaza fila con la cuenta correspondiente a la editorial "Planeta".

Clausula WHERE y Having

```
SELECT  Cod_Depto, COUNT(*)
FROM empleados
GROUP BY Cod_Depto
HAVING COUNT(*) >= 2
ORDER BY COUNT(*) DESC;
```

Obtiene el número de empleados por departamento siempre que haya al menos 2 empleados en el departamento. Además se ordena la salida por el número de empleados por departamento en orden descendente.

- **WHERE**: Selecciona las filas
- **GROUP BY**: Agrupa estas filas
- **HAVING**: Filtra los grupos. Selecciona y elimina los grupos
- **ORDER BY**: Clasifica la salida. Ordena los grupos.

Valores nulos

Es posible que las tuplas tengan un valor nulo, indicado por medio de ***null*****,** en alguno de sus atributos

null significa un valor desconocido o un valor que no existe.

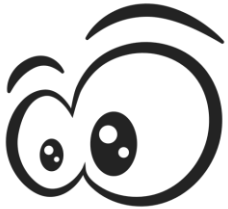
El predicado **is null** se puede utilizar para comprobar los valores nulos.

Ejemplo: obtener todos los números de prestamos que aparecen en la relación préstamo con valores nulos para importe
SELECT *número_prestamo*
FROM prestamo
WHERE importe **is null**

El resultado de la expresión aritmética que involucra a *null* es nulo

Ejemplo: 5 + *null* devuelve nulo

Todas las operaciones agregadas excepto count(*) ignoran las tuplas con valores nulos de los atributos agregados



Valores nulos y lógica de tres valores

Cualquier comparación con *null* se convierte en *desconocido*

Ejemplo*: 5 < null o null <> null o null = null*

Lógica de tres valores que utiliza el valor real desconocido:

OR: (*desconocido* **or** *cierto*) = *true*, (*desconocido* **or** *falso*) = *desconocido,* (*desconocido* **or** *desconocido) = desconocido*

AND: *(cierto* **AND** *desconocido) = desconocido, (falso* **AND** *desconocido) = falso,*
(desconocido **AND** *desconocido) = desconocido*

NOT*: (***not** *desconocido) = desconocido*

"*P* **is desconocido"** se evalúa a *cierto* si el predicado *P* se evalúa a *desconocido*

El resultado del predicado de la cláusula **WHERE** se toma como *falso* si se evalúa en *desconocido*

El total de todas las cantidades de prestamos

SELECT sum (*importe*)
FROM *prestamo*

La instrucción anterior ignora las cantidades nulas. El resultado es *null* si todas las cantidades son nulas

.

Subconsultas anidadas

SQL proporciona un mecanismo para las subconsultas anidadas.

Una subconsulta es una expresión **SELECT-FROM-WHERE** que se anida dentro de otra consulta.

Obtener todos los clientes que tengan una cuenta y un préstamo en el banco (intersect).

SELECT distinct *nombre_cliente*
FROM *prestatario*
WHERE *nombre_cliente* **in** (**SELECT** *nombre_cliente*
FROM *impositor*)

Obtener todos los clientes que tengan un préstamo en el banco pero que no tengan una cuenta en dicho banco (except-minus)

SELECT distinct *nombre_cliente*
FROM *prestatario*
WHERE *nombre_cliente* **not in** (**SELECT** *nombre_cliente*
FROM *impositor*)

Ejemplo de consulta

Obtener todos los clientes que tengan tanto una cuenta como un préstamo en la sucursal Navacerrada

```
SELECT distinct nombre_cliente
FROM prestatario, prestamo
WHERE prestatario.número_préstamo= prestamo.número_préstamo AND
                nombre_sucursal = 'Navacerrada' AND
          (nombre_sucursal, nombre_cliente) in
                          (SELECT nombre_sucursal, nombre_cliente
                           FROM impositor, cuenta
                           WHERE impositor.número_cuenta = cuenta.número_cuenta )
```

- **Se puede escribir la consulta anterior de forma mucho más simple. Se ha escrito así para ilustrar las características de SQL**

Comparación de conjuntos

Obtener los nombres de todas las sucursales que tengan activos mayores que al menos una sucursal situada en Barcelona.

```
SELECT distinct  T.nombre_sucursal
FROM sucursal as T, sucursal as S
WHERE  T.activo > S.activo AND
                    S.ciudad_sucursal = ' Barcelona '
```

La misma consulta utilizando la clausula > some

```
SELECT nombre_sucursal
FROM sucursal
WHERE activo > some
                    (SELECT activo
                     FROM sucursal
                     WHERE ciudad_sucursal = 'Barcelona')
```

Obtener los nombres de todas las sucursales que tienen activos mayores que todas las sucursales situadas en Barcelona.

```
SELECT nombre_sucusal
FROM sucursal
WHERE activo > all
                    (SELECT activo
                    FROM sucursal
                    WHERE ciudad_sucursal = 'Barcelona')
```

Consulta ejemplo

Obtener todos los clientes que tengan una cuenta en todas las sucursales situadas en Barcelona.

```
SELECT distinct S.nombre_cliente
        FROM impositor as S
        WHERE not exists (
            (SELECT nombre_sucursal
            FROM sucursal
            WHERE ciudad_sucursal = 'Barcelona')
            except
            (SELECT R.nombre_sucursal
            FROM impositor as T, cuenta as R
            WHERE T.número_cuenta = R.número_cuenta AND
                    S.nombre_cliente = T.nombre_cliente)
```

Los nombres de clientes dónde no existe

Nombre de la sucursales de Barcelona

Sucursales de Barcelona dónde no tiene cuenta el cliente

Nombre sucursales dónde tienen cuenta los clientes

- $X - Y = \emptyset \Leftrightarrow X \subseteq Y$
- No se puede escribir una consulta utilizando = **all** y sus variantes

Comprobación de ausencia de tuplas duplicadas

La construcción **unique** comprueba si una subconsulta tiene alguna tupla duplicada en sus resultados.

Obtener todos los clientes que sólo tengan una cuenta en la sucursal Navacerrada.

```
SELECT T.nombre_cliente
FROM impositor as T
WHERE unique (
        SELECT R.nombre_cliente
        FROM cuenta, impositor as R
        WHERE T.nombre_cliente = R.nombre_cliente AND
              R.número_cuenta= cuenta.número_cuenta AND
              cuenta.nombre_sucursal = ‘ Navacerrada’ )
```

Obtener todos los clientes que tengan al menos dos cuentas en la sucursal Navacerrada.

```
SELECT distinct T.nombre_cliente
FROM impositor as T
WHERE not unique (
    SELECT R.nombre_cliente
    FROM cuenta, impositor as R
    WHERE T.nombre_cliente = R.nombre_cliente AND
          R.número_cuenta = cuenta.número_cuenta  AND
          cuenta.nombre_sucursal  = ‘Navacerrada’)
```

Relaciones derivadas

SQL permite utilizar expresiones de subconsulta en la cláusula **FROM**

Obtener el saldo promedio de las cuentas en las que dicho saldo sea mayor de 1200€.

```
SELECT nombre_sucursal, saldo_medio
FROM (SELECT nombre_sucursal, avg (saldo)
        FROM cuenta
        group by nombre_sucursal)
          as media_sucursal(nombre_sucursal, saldo_medio)
        WHERE saldo_medio > 1200
```

No es necesario utilizar la cláusula **having**, puesto que se calcula la relación temporal (vista) resultado en la cláusula **FROM**, y los atributos de media_sucursal se pueden utilizar directamente en la cláusula **WHERE.**

Vistas

En algunos casos, no es deseable para todos los usuarios ver el modelo lógico completo (es decir, todas las relaciones actuales almacenadas en la base de datos).

Se utilizan para tres fines:

1. Prohibir el acceso a datos confidenciales
2. Simplificar la formulación de consultas complejas o repetitivas
3. Aumentar la independencia de los programas respecto a los datos

Una persona que necesita conocer un número de préstamo de un cliente pero no tiene necesidad de conocer el importe del préstamo. Esta persona debería ver una relación descrita en SQL como

(**SELECT** *nombre_cliente, número_prestamo*
FROM *prestatario, prestamo*
WHERE *prestatario.número_préstamo = prestamo.número_prestamo*)

Una **vista** proporciona un mecanismo para ocultar ciertos datos de la vista de ciertos usuarios.

Cualquier relación que no es del modelo conceptual pero se hace visible para el usuario como una “relación virtual” se denomina una **view**.

Definición de vista

Una vista se define utilizando la instrucción **create view** que tiene la forma

create view *v* **as** <expresión de consulta>

donde <expresión de consulta> es cualquier expresión de consulta legal de SQL. El nombre de la vista se representa por *v.*

Una vez definida la vista, su nombre puede utilizarse para referirse a la relación virtual que la vista genera.

La definición de vista no es lo mismo que la creación de una nueva relación mediante la evaluación de la expresión de consulta.

Una definición de vista permite el ahorro de una expresión para ser sustituida por consultas que utilizan esa vista.

Consultas de ejemplo

Una vista de las sucursales y sus clientes.

```
create view todos_los_clientes as
   (SELECT nombre_sucursal, nombre_cliente
    FROM impositor, cuenta
    WHERE impositor.número_cuenta =
          cuenta. número_cuenta )
    union
    (SELECT nombre_sucursal, nombre_cliente
    FROM prestatario, prestamo
    WHERE prestatario.número_cuenta = prestamo.número_cuenta )
```

Averiguar todos los clientes de la sucursal de Navacerrada

```
SELECT nombre_cliente
FROM todos_los_clientes
WHERE nombre_sucursal = 'Navacerrada'
```

Modificación de la base de datos– Borrado

Borrar todos los registros de cuentas de la sucursal Navacerrada

```
DELETE FROM cuenta
WHERE nombre_sucursal = 'Navacerrada'
```

Borrar todas las cuentas de cada sucursal situada en la ciudad de Navacerrada.

```
DELETE FROM cuenta
WHERE nombre_sucursal in (SELECT nombre_sucursal
                          FROM sucursal
                          WHERE ciudad_sucursal = 'Navacerrada')
```

Borrar el registro de todas las cuentas con saldos inferiores a la media del banco.

```
DELETE FROM cuenta
 WHERE saldo < (SELECT avg (saldo)
                FROM cuenta )
```

- **Problema**: al borrar tuplas, el saldo medio cambia
- ***Solución utilizada en SQL:***

  1. Primero, calcular el saldo medio **avg** (saldo) de todas las tuplas que se van a borrar
  2. Después, borrar todas las tuplas encontradas antes (sin recalcular avg (saldo) o recomprobando las tuplas)

Modificación de la base de datos– Inserción

Se proporciona como regalo a todos los clientes que tengan un préstamo en la sucursal Navacerrada, una cuenta de ahorro de 200€. Hacer que el número de préstamo sirva como número de cuenta de la nueva cuenta de ahorro

INSERT INTO *cuenta*
SELECT *número_prestamo, nombre_sucursal,* 200
FROM *prestamo*
WHERE *nombre_sucursal* = 'Navacerrada'

INSERT INTO *impositor*
SELECT *nombre_cliente, número_prestamo*
FROM *prestamo, prestatario*
WHERE nombre_sucursal = 'Navacerrada' **AND** *prestamo.número_cuenta*=
prestatario.número_cuenta

La sentencia **SELECT FROM WHERE** se evalúa completamente antes de que ninguno de sus resultados se inserte en la relación (de otra forma las consultas como

insert into *tabla*1 **SELECT** * **FROM** *tabla*1

generarían problemas)

Modificación de la base de datos– Actualizaciones

Aumentar todas las cuentas con saldos por encima de 10.000€ con el 6%, todas las demás cuentas reciben un 5%.

Escribir dos instrucciones **update**:

update *cuenta*
set *saldo* = *saldo* ∗ 1,06
WHERE *saldo* > 10000

El orden es importante

update *cuenta*
set *saldo* = *saldo* ∗ 1,05
WHERE *saldo* ≤ 10000

Aumentar el precio de los artículos con precios inferiores a 50 euros en un 10% y los demás artículos aumentar el precio en un 5%

update *articulo*
set *precio* =*precio*∗ 1,05
WHERE *precio* > 50

update *articulo*
set *precio*=*precio*∗ 1,10
WHERE *precio* ≤ 50

Actualización de una vista

Crear una vista de todos los datos de prestamos en la relación préstamo, ocultando el atributo importe

create view *sucursal_préstamo***as**
SELECT *número_prestamo, nombre_sucursal,*
FROM *prestamo*

Añadir una tupla nueva a sucursal_prestamo

insert into *sucursal_préstamo* **values** (‘P-37’,‘Navacerrada’)

Esta inserción se debe representar mediante la inserción de la tupla
(‘P-37’, ‘Navacerrada’, *null*)
dentro de la relación *prestamo*

Actualización de una vista

Algunas actualizaciones de vistas son difíciles o imposibles de traducir en relaciones de la base de datos

create view *v* **as**

SELECT *nombre_sucursal* **FROM** *cuenta*

insert into *v* **values (**‘ *Navacerrada*’**)**

Otras no se pueden traducir de forma única

insert into *todos_los_clientes* **values** *(*‘Navacerrada’*,* ‘Juan’**)**

¡Hay que elegir préstamo cuenta y crear un nuevo número de prestamo/cuenta!

La mayor parte de las implementaciones de SQL permiten actualizar sólo vistas simples (sin agregados) definidas sobre una sola relación.

Reunión de relaciones

Las **operaciones de reunión** toman dos relaciones y las devuelven como resultado otra relación.

Estas operaciones adicionales se utilizan generalmente como expresiones de subconsulta de la cláusula **FROM**

Condición de reunión – define qué tuplas de las dos relaciones coinciden, y qué atributos están presentes en el resultado de la reunión.
Tipo de reunión – define cómo se tratan las tuplas de cada relación que no coincide con ninguna tupla de la otra relación (basada en la condición de reunión).

Tipos de reunión
inner join
left outer join
right outer join
full outer join

Condiciones de reunión
natural
on <predicado>
using (A_1, A_2, ..., A_n)

Reunión de relaciones – Ejemplos

Relación prestamo

número-prestamo	*nombre-sucursal*	*importe*
P-170	Centro	3000
P-230	Moralzarzal	4000
P-260	Navacerrada	1700

- ***Relación prestatario***

nombre-cliente	*número-préstamo*
Santos	P-170
Gómez	P-230
López	P-155

- *no se tiene la información del prestatario para P-260 ni la información de préstamopara P-155*

Reunión de relaciones – Ejemplos

Préstamo **inner join** *prestatario* **on** *prestamo.número_préstamo= prestatario.número_prestamo*

número-préstamo	*nombre-sucursal*	*importe*	*nombre-cliente*	*número-prestamo*
P-170 P-230	Centro Moralzarzal	3000 4000	Santos Gómez	P-170 P-230

Préstamo **left inner join** *prestatario* **on** *prestamo.número_préstamo= prestatario.número_prestamo*

número-prestamo	*nombre-sucursal*	*importe*	*nombre-cliente*	*número-prestamo*
P-170 P-230 P-260	Centro Moralzarzal Navacerrada	3000 4000 1700	Santos Gómez *null*	P-170 P-230 *null*

Reunión de relaciones – Ejemplos

prestamo natural inner join prestatario

número-préstamo	*nombre-sucursal*	*importe*	*nombre-cliente*
P-170	Centro	3000	Santos
P-230	Moralzarzal	4000	Gómez

prestamo natural right outer join prestatario

número-prestamo	*nombre-sucursal*	*importe*	*nombre-cliente*
P-170	Centro	3000	Santos
P-230	Moralzarzal	4000	Gómez
P-155	null	null	López

Reunión de relaciones – Ejemplos

prestamo full outer join prestatario using (número_prestamo)

número-prestamo	*nombre-sucursal*	*importe*	*nombre-cliente*
P-170	Centro	3000	Santos
P-230	Moralzarzal	4000	Gómez
P-260	Navacerrada	1700	*null*
P-155	null	null	López

Obtener todos los clientes que tengan una cuenta o un préstamo(pero no ambos) en el banco.

SELECT *nombre_cliente*
FROM (*impositor* **natural full outer join** *prestatario*)
WHERE *número_cuenta* **is** *null* **or** *número_préstamo* **is** *null*

Relaciones prestamo, prestatario, cuenta

prestamo

número-prestamo	*nombre-sucursal*	*importe*
L-200	Madrid	3000
L-230	Rascafria	4000
L-260	Navacerrada	1700

prestatario

nombre-cliente	*número-prestamo*
Gonzalez	L-200
Pérez	L-230
López	L-155

cuenta

nombre-sucursal	*número-cuenta*	*saldo*
Navacerrada	A-102	400
Navacerrada	A-201	900
Lozoya	A-217	750
Lozoya	A-215	750
Rascafria	A-222	700

Cuenta agrupada por nombre de sucursal

nombre-sucursal	*saldo*
Navacerrada	1300
Barcelona	1500
Reus	700

Asertos

Un *aserto* es un predicado que expresa una condición que se desea que la base de datos satisfaga siempre**.**

Un aserto en SQL tiene la forma

create assertion <nombre-aserto > **check** <predicado>

Cuando se crea un aserto, el sistema comprueba su validez, y la comprueba de nuevo en cada actualización que puede violar el aserto

Esta prueba puede introducir una cantidad considerable de sobrecarga; por lo tanto **se deben utilizar los asertos con mucha cautela**.

El aserto para todo *X, P(X)* se consigue en un modo indirecto utilizando no existe *X* tal que *no P(X)*

Ejemplo de aserto

Cada préstamo tiene al menos un prestatario que mantiene una cuenta con un saldo mínimo o 1.000,00€

```
create assertion restricción-saldo check
(not exists (
    SELECT * FROM prestamo
        WHERE not exists (
        SELECT *
            FROM prestatario, impositor, cuenta
            WHERE prestamo.número-préstamo= prestatario.número-prestamo
                AND prestatario.nombre-prestatario =
                    impositor.nombre-cliente
                AND impositor.número-cuenta =
                    cuenta.número-cuenta
                AND cuenta.saldo >= 1000)))
```

Ejemplo de aserto

La suma de todas las cantidades de préstamo de cada sucursal debe ser menores que la suma de todos los saldos de las cuentas de la sucursal.

create assertion *restricción-suma* **check**
(not exists (SELECT * FROM *sucursal*
WHERE (SELECT sum*(importe)* **FROM** *prestamo*
WHERE *prestamo.nombre-sucursal = sucursal.nombre-sucursal)*
>= **(SELECT sum***(importe)* **FROM** *cuenta*
WHERE *prestamo.nombre-sucursal = sucursal.nombre-sucursal)))*

Trigger

Un trigger o disparador en una Base de Datos , es un procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación. se activa cuando ocurre un evento en particular para esa tabla. El disparador queda asociado a la tabla nombre_tabla. Esta debe ser una tabla permanente, no puede ser una vista.

Dependiendo de la base de datos, los triggers pueden ser de inserción (INSERT), actualización (UPDATE) o borrado (DELETE). Algunas bases de datos pueden ejecutar triggers al crear, borrar o editar usuarios, tablas, bases de datos u otros objetos.

```
CREATE TRIGGER {BEFORE|AFTER} {INSERT|UPDATE|DELETE}
ON FOR EACH ROW
BEGIN


END;
```

Ejemplo

```
drop trigger if exists productosAI;
create trigger productosAI  after insert  --  convenio A after I insert
on producto for each row
  insert into registroProducto(idenProducto, nombreProducto,precioProd,insertado)
        values(new.idProducto, new.nomProducto,new.precio,now());
```

Este trigger es para que después de insertar un producto en la tabla productos , guardemos esos datos en una tabla registroProducto que tiene como campos el idProducto, el nomProducto, el precio y el momento en el que se ha insertado.

CREATE TRIGGER nombreDisp momentoDisp eventoDisp

ON nombreTabla FOR EACH ROW sentenciaDisp

momentoDisp es el momento en que el disparador entra en acción. Puede ser BEFORE (antes) o AFTER

eventoDisp indica la clase de sentencia que activa al disparador. Puede ser INSERT, UPDATE, o DELETE.

Por ejemplo, un disparador BEFORE para sentencias INSERT podría utilizarse para validar los valores a insertar.

No puede haber dos disparadores en una misma tabla que correspondan al mismo momento y sentencia.

NO se pueden tener dos disparadores BEFORE UPDATE.

SI es posible tener los disparadores BEFORE UPDATE y BEFORE INSERT o BEFORE UPDATE y AFTER UPDATE.

sentenciaDisp es la sentencia que se ejecuta cuando se activa el disparador.

Si se desean ejecutar múltiples sentencias, deben colocarse entre BEGIN ... END, el constructor de sentencias compuestas. Esto además posibilita emplear las mismas sentencias permitidas en rutinas almacenadas.

Autorización

Las formas de autorización para diferentes partes de la base de datos:
lectura – permite la lectura, pero no la modificación de datos.
inserción - permite la inserción de datos nuevos, pero no la modificación de los existentes.
actualización - permite la modificación, pero no el borrado de los datos.
borrado - permite el borrado de los datos

Las formas de autorización para modificar el esquema de la base de datos:
índices - permite la creación y borrado de índices.
recursos - permite la creación de relaciones nuevas.
alteración - permite el añadido o el borrado de atributos de las relaciones.
eliminación - permite el borrado de relaciones.

Especificación de autorización en SQL

La instrucción de concesión **grant** se utiliza para conferir autorización

grant <lista de privilegio>
on <nombre de relación o nombre de vistas> a <lista de usuario>

<lista de usuario> es: una identificación de usuario
public, que permite a todos los usuarios válidos el privilegio concedido

Privilegios en SQL

SELECT: permite el acceso de lectura a la relación, o la capacidad para hacer consultas utilizando la vista
Ejemplo: autorizan a los usuarios U_1, U_2, y U_3 **SELECT** autorización en la relación *sucursal*:

grant SELECT on *sucursal* **to** U_1, U_2, U_3

insert: la capacidad para insertar tuplas
update: la capacidad para actualizar utilizando la instrucción actualización de SQL
delete: la capacidad para borrar tuplas.
all privileges: utilizado como una forma abreviada de todos los privilegios permisibles

Retirada de autorización en SQL

La instrucción **revoke** se utiliza para retirar la autorización.
revoke<lista de privilegios>
on <nombre de relación o de vista> **FROM** <lista de usuarios>

Ejemplo:
revoke SELECT on *sucursal* **FROM** U_1, U_2, U_3

La <listadeprivilegios> puede ser **all to** para retirar todos los privilegios que la retirada puede mantener.

Si la <listaderetirada> incluye **public** todos los usuarios pierden el privilegio excepto aquellos que lo autorizan explícitamente.

Si el mismo privilegio se concede dos veces al mismo usuario por diferentes concesiones, el usuario puede retener el privilegio después de la retirada.

Todos los privilegios que dependen del privilegio que se ha retirado se retiran también.

ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE MADRID