01/Abril/2022 - Abel Rios

BBDD “World”

Ejercicio: En esta query buscamos la lista de nombres de ciudades de más de un millón de habitantes donde se hable español:

```sql
SELECT Name
FROM `city`, `countrylanguage`
WHERE Population>1000000
AND Language = "Spanish"
AND city.CountryCode=countrylanguage.CountryCode;
```

mezclamos las dos tablas y se hace el producto cartesiano, pero con la última línea deshacemos el cartesiano y nos quedamos con los únicos registros que queremos (cruzar tablas BIEN).

BBDD “classicmodels”

(esta base de datos es muy similar a la que tenemos que hacer para el carrito, seguro que podemos sacar ideas de aquí)

Ejercicio: Query que nos devuelva los clientes que hayan comprado el producto S24_1937:

```sql
SELECT DISTINCT customerName
FROM `customers`, `orders`, `orderdetails`
WHERE orderdetails.productCode="S24_1937"
AND orderdetails.orderNumber=orders.orderNumber
AND orders.customerNumber=customers.customerNumber;
```

Usamos el DISTINCT para eliminar los nombres duplicados y cruzamos las tres tablas, al igual que en el ejemplo anterior.

Ejercicio: Sacar todas las categorías de productos con el precio máximo de la categoría, precio mínimo de la categoría y precio medio de esa categoría de producto (product line)

```sql
SELECT productlines.productLine, MAX(products.buyPrice) AS maximo,
    MIN(products.buyPrice) AS minimo,
    AVG(products.buyPrice) AS media
FROM `products`, `productlines`
WHERE productlines.productLine = products.productLine
GROUP BY productLine;
```

Así asignamos en el SELECT un alias al máximo, al mínimo y al average. En el FROM informamos que mezclamos las dos tablas y con el where decimos cuáles son los datos a tomar, para evitar el producto cartesiano. Si no aplicamos el GROUP BY, nos da sólo un registro (no me preguntes porqué).

SENTENCIAS DE INSERCIÓN

Permiten insertar un nuevo registro en la tabla, pero pueden fallar si el registro YA EXISTE en la tabla.

Sintaxis (valores entre corchetes son opcionales):

INSERT [INTO] <TABLA>
[(campo1, campo2, campo3…)]
VALUES(val1, val2, val3…)

Los valores a insertar tienen que tener el mismo orden que los campos en la tabla. No tienen porqué ser todos los campos de la tabla. En este caso se les pondrá null, o un valor por defecto, dependiendo de cómo esté definida la tabla.

También tenemos el tipo INSERT SELECT:

INSERT [INTO] <TABLA>
[(campo1, campo2, campo3…)]
SELECT campo1, campo2, campo3…, campoN
FROM …….
WHERE …….

*** (De todo esto Antonio nos dará diapositivas la semana que viene)

Ejercicio: construir un insert para la tabla “city”

```
INSERT INTO `city`(`ID`, `Name`, `CountryCode`, `District`, `Population`)
VALUES ('4080','Alozaina','ESP','Andalusia','2000')
```

Otro INSERT pero ahora sin el ID, porque el atributo ID se auto incrementa:

```
INSERT INTO `city`( `Name`, `CountryCode`, `District`, `Population`)
VALUES ('Yunquera','ESP','Andalusia','2300')
```

SENTENCIA UPDATE (Actualizar)

```
UPDATE <TABLA>
   SET CAMPO1 = VALOR1
     , CAMPO2 = VALOR2
     ,...
     , CAMPON = VALORN
WHERE CONDICIONES
```

Estos cambios se efectúan en todas las filas en las que se cumpla la condición/es del WHERE.

Ejemplo (BBDD World):
Cambiar en la tabla countrylanguage todos los registros que tienen de idioma “spanish” a “Español”

```
UPDATE countrylanguage
SET LANGUAGE = 'Español'
where LANGUAGE = 'spanish';
```

Ejemplo2:
Usando un SELECT que nos devuelva un sólo valor y ese será el que insertamos. (en el ejemplo no terminamos de definir el SELECT, es sólo para comprender que también se puede usar así).

```
1 update countrylanguage
2 set Language = (select language from tablax where ...)
3 where Language = 'Español'
```

SENTENCIA DELETE

DELETE FROM <table_name> WHERE <condition>

****OJO****

Si nos fijamos, en la plantilla por defecto de las sentencias SQL siempre pone un WHERE 1, lo que significa que se aplica a todos los registros de la tabla que sea. Sin embargo, en la plantilla de la sentencia DELETE pone WHERE 0, para que en principio no afecte a ningún registro de la tabla.

BEGIN / COMMIT / ROLLBACK

En un sistema de base de datos, todo lo que hacemos (aunque no lo veamos) son tipos de transacciones. Escondido en el código SQL siempre hay un BEGIN al principio y un COMMIT / ROLLBACK al final (esto normalmente lo tenemos en el menú variables como AUTOCOMMIT).

Si desactivamos la opción de AUTOCOMMIT (a la izquierda en el menú, nueva -> variables -> Autocommit OFF), todos los cambios que hagamos en la práctica se van a acumular y cuando hagamos un COMMIT entonces se aplicarán esos cambios.

Si alguien intenta acceder a los registros que aún no han sido commiteados, se le deja en espera por un tiempo prudencial, y el registro se bloquea pues alguien está usándolo / modificándolo.

Si los registros están siendo leídos/modificados por lo tanto el sistema va bloqueando esos registros para que nadie más pueda entrar en ellos. Si el mismo usuario está accediendo a una gran cantidad de registros de la misma tabla (y por ende éstos están siendo bloqueados por el sistema) en algún momento el sistema escala el bloqueo y bloquea la tabla entera para que sólo ese quién está usando la tabla pueda acceder a ella y nadie más.

ROLLBACK deshace lo hecho. Por ejemplo, vas haciendo SELECTs o INSERTs o cualquier cosa, y en algún momento encuentras algo que no debería estar así, o te das cuenta que la has liado parda, haces un ROLLBACK y todo lo que habías hecho ¡PUF! se convierte en chocapick y desaparece, aquí no ha pasado nada de nada.

VIEW (VISTA)

Interpretar una query como si fuese una tabla. Básicamente es como crear una tabla nueva con lo que nos devuelva una query (SELECT por ejemplo).

Ejemplo:

```
CREATE VIEW CUSTOMER_DISTINCT AS
select DISTINCT customers.customerName
from customers, orders, orderdetails
where customers.customerNumber = orders.customerNumber
and orders.orderNumber = orderdetails.orderNumber;
```

Y ahora podemos usar esa VIEW (que nos aparecerá también a la izquierda en el menú de la base de datos) como si fuera una tabla nueva y hacerle nuevas querys.

OJO
La vista que creamos tendrá los campos del select con el que la hayamos creado, por lo que a nuestra vista que acabamos de crear no podemos hacerle un select como el de a continuación, dado que nuestra vista sólo tiene customerName.

```
SELECT * FROM customer_distinct
WHERE orderdetails.productCode = 'S24_1937
```

ESTO DA ERROR

Pues como de BBDD no avanzadas ya estamos, volvemos a practicar.

Ejercicio: (BBDD classicmodels) de la tabla de employees hacer una query que muestre el nombre y apellidos del empleado y el nombre y apellidos del jefe.

Yo he creado una vista con el Id del empleado, el nombre y el apellido, (nombresynumerosempleados) y luego he relacionado la tabla con mi lista:

y mi query se queda así:

```
SELECT employees.firstName, employees.lastName, nombresynumerosempleados.firstName,
nombresynumerosempleados.lastName
FROM `employees`, nombresynumerosempleados
WHERE nombresynumerosempleados.employeeNumber=employees.reportsTo;
```

Pero Antonio lo ha hecho usando la misma tabla (sin crear una vista) y dándole dos alias distintos:

```sql
SELECT a.firstName, a.lastName, b.firstName, b.lastName
FROM employees as A, employees as B
WHERE a.reportsTo = b.employeeNumber;
```

y de esta manera concatenamos nombres y apellidos, y le damos un alias a la concatenación:

```sql
SELECT  concat (a.firstName,a.lastName) as empleado, concat(b.firstName,b.lastName) as
jefe
FROM employees A, employees B
where a.reportsTo = b.employeeNumber
```

Ejercicio: sacar las ventas por mes de cada empleado.

ejemplo:

Nombre del empleado / Mes / Suma de sus ventas

John Murphy / 2022-01 / 45.000€

Creamos una vista, que nos va a ayudar en el proceso, en la que almacenaremos cliente, fecha y suma de las ventas de esa fecha)

```sql
CREATE VIEW clientefechatotal AS
SELECT customers.customerNumber, year(orders.orderDate)*100+month(orders.orderDate),
       SUM(orderdetails.quantityOrdered*orderdetails.priceEach)
FROM customers, orders, orderdetails
WHERE orders.customerNumber=customers.customerNumber
AND orderdetails.orderNumber=orders.orderNumber
GROUP BY 1,2
```

Le decimos GROUP BY 1,2 para que agrupe por Cliente y por Año/Mes

Nos falta terminar de mezclar con la tabla de empleados