

Tema 5. Las subconsultas (I)

Definiciones

Una **subconsulta** es una sentencia **SELECT** que aparece dentro de otra sentencia **SELECT** que llamaremos **consulta principal**.

Se puede encontrar **en la lista de selección, en la cláusula WHERE o en la cláusula HAVING** de la consulta principal.

Una subconsulta tiene la misma sintaxis que una sentencia **SELECT** normal exceptuando que aparece **encerrada entre paréntesis**, no puede contener la cláusula **ORDER BY**, ni puede ser la **UNION** de varias sentencias **SELECT**, además tiene algunas restricciones en cuanto a número de columnas según el lugar donde aparece en la consulta principal. Estas restricciones las iremos describiendo en cada caso.

Cuando se ejecuta una consulta que contiene una subconsulta, **la subconsulta se ejecuta por cada fila de la consulta principal**.

Se aconseja no utilizar campos calculados en las subconsultas, ralentizan la consulta.

Las consultas que utilizan subconsultas suelen ser **más fáciles de interpretar por el usuario**.

Referencias externas

A menudo, es necesario, dentro del cuerpo de una subconsulta, hacer referencia al valor de una columna en la fila actual de la consulta principal, ese nombre de columna se denomina referencia externa.

Una **referencia externa** es un nombre de columna que estando en la subconsulta, no se refiere a ninguna columna de las tablas designadas en la **FROM** de la subconsulta sino a una **columna de las tablas designadas en la FROM de la consulta principal**. Como la subconsulta se ejecuta por cada fila de la consulta principal, el valor de la referencia externa irá cambiando.

Ejemplo:

```
SELECT numemp, nombre, (SELECT MIN(fechapedido) FROM pedidos WHERE rep = numemp)  
FROM empleados;
```

En este ejemplo la consulta principal es **SELECT... FROM empleados**.
La subconsulta es **(SELECT MIN(fechapedido) FROM pedidos WHERE rep = numemp)**.
En esta subconsulta tenemos una referencia externa (*numemp*) es un campo de la tabla empleados (origen de la consulta principal).

¿Qué pasa cuando se ejecuta la consulta principal?

- se coge el primer empleado y se calcula la subconsulta sustituyendo numemp por el valor que tiene en el primer empleado. La subconsulta obtiene la fecha más antigua en los pedidos del rep = 101,
 - se coge el segundo empleado y se calcula la subconsulta con numemp = 102 (numemp del segundo empleado)... y así sucesivamente hasta llegar al último empleado.
- Al final obtenemos una lista con el número, nombre y fecha del primer pedido de cada empleado.

Si quitamos la cláusula **WHERE** de la subconsulta obtenemos la fecha del primer pedido de todos los pedidos no del empleado correspondiente.

Anidar subconsultas

Las subconsultas pueden **anidarse** de forma que **una subconsulta aparezca en la cláusula WHERE** (por ejemplo) **de otra subconsulta** que a su vez forma parte de otra consulta principal. En la práctica, una consulta consume mucho más tiempo y memoria cuando se incrementa el número de niveles de anidamiento. La consulta resulta también más difícil de leer, comprender y mantener cuando contiene más de uno o dos niveles de subconsultas.

Ejemplo:

```
SELECT numemp, nombre  
FROM empleados  
WHERE numemp = (SELECT rep FROM pedidos WHERE clie = (SELECT numclie FROM
```

clientes WHERE nombre = 'Julia Antequera'))

En este ejemplo, por cada línea de pedido se calcula la subconsulta de clientes, y esto se repite por cada empleado, en el caso de tener 10 filas de empleados y 200 filas de pedidos (tablas realmente pequeñas), la subconsulta más interna se ejecutaría 2000 veces (10 x 200).

Subconsulta en la lista de selección

Cuando la subconsulta aparece **en la lista de selección** de la consulta principal, en este caso la subconsulta, **no puede devolver varias filas ni varias columnas**, de lo contrario se da un mensaje de error.

Muchos SQLs no permiten que una subconsulta aparezca en la lista de selección de la consulta principal pero eso no es ningún problema ya que normalmente se puede obtener lo mismo utilizando como origen de datos las dos tablas. El ejemplo anterior se puede obtener de la siguiente forma:

```
SELECT numemp, nombre, MIN(fechapedido)  
FROM empleados LEFT JOIN pedidos ON empleados.numemp = pedidos.rep  
GROUP BY numemp, nombre
```

En la cláusula FROM

En la cláusula FROM se puede encontrar una sentencia SELECT encerrada entre paréntesis pero **más que subconsulta sería una consulta** ya que no se ejecuta para cada fila de la tabla origen sino que se ejecuta una sola vez al principio, su resultado se combina con las filas de la otra tabla para formar las filas origen de la SELECT primera y no admite referencias externas.

En la cláusula FROM vimos que se podía poner un nombre de tabla o un nombre de consulta, pues en vez de poner un nombre de consulta se puede poner directamente la sentencia SELECT correspondiente a esa consulta encerrada entre paréntesis.

Subconsulta en las cláusulas WHERE y HAVING

Se suele utilizar subconsultas en las cláusulas WHERE o HAVING cuando los datos que queremos visualizar están en una tabla pero para seleccionar las filas de esa tabla necesitamos un dato que está en otra tabla.

Ejemplo:

```
SELECT numemp, nombre  
FROM empleados  
WHERE contrato = (SELECT MIN(fechapedido) FROM pedidos)
```

En este ejemplo listamos el número y nombre de los empleados cuya fecha de contrato sea igual a la primera fecha de todos los pedidos de la empresa.

En una cláusula **WHERE / HAVING** tenemos siempre una condición y la subconsulta actúa de operando dentro de esa condición.

En el ejemplo anterior se compara contrato con el resultado de la subconsulta. Hasta ahora las condiciones estudiadas tenían como operandos valores simples (el valor contenido en una columna de una fila de la tabla, el resultado de una operación aritmética...) ahora la subconsulta puede devolver una columna entera por lo que es necesario definir otro tipo de **condiciones especiales** para cuando se utilizan con subconsultas.

Condiciones de selección con subconsultas

Las **condiciones de selección** son las condiciones que pueden aparecer en la cláusula **WHERE** o **HAVING**. La mayoría se han visto en el tema 2 pero ahora incluiremos las condiciones que utilizan una subconsulta como operando.

En SQL tenemos cuatro nuevas condiciones:

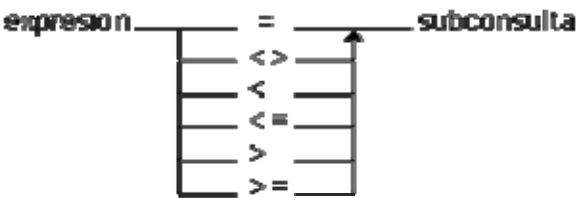
- el **test de comparación con subconsulta**
- el **test de comparación cuantificada**
- el **test de pertenencia a un conjunto**
- el **test de existencia**

En todos los tests estudiados a continuación *expresión* puede ser cualquier nombre de columna de la consulta principal o una expresión válida como ya vimos en el tema 2.

● El **test de comparación con subconsulta**.

Es el **equivalente al test de comparación simple**. Se utiliza para comparar un valor de la fila que se está examinado con un único valor producido por la subconsulta. La subconsulta debe devolver una **única columna**, sino se produce un error.
Si la subconsulta no produce **ninguna fila** o devuelve el valor **nulo**, el test devuelve el **valor nulo**, si la subconsulta produce **varias filas**, SQL devuelve una **condición de error**.

La sintaxis es la siguiente:



```
SELECT oficina, ciudad
FROM oficinas
WHERE objetivo > (SELECT
SUM(ventas) FROM empleados
WHERE empleados.oficina =
oficinas.oficina)
```

Lista las oficinas cuyo objetivo sea superior a la suma de las ventas de sus empleados.
En este caso la subconsulta devuelve una única columna y una única fila (es una consulta de resumen sin **GROUP BY**)

● El **test de comparación cuantificada**.

Este test es una extensión del test de comparación y del test de conjunto. **Compara el valor** de la expresión **con cada uno de los valores** producidos por la subconsulta. La subconsulta debe devolver una **única columna** sino se produce un error.
Tenemos el test **ANY** (*algún, alguno* en inglés) y el test **ALL** (*todos* en inglés).

La sintaxis es la siguiente:



● El **test ANY**.

La subconsulta debe devolver una única columna sino se produce un error.
Se evalúa la comparación con cada valor devuelto por la subconsulta.
Si **alguna de las comparaciones individuales produce el resultado verdadero, el test ANY devuelve el resultado verdadero**.
Si la subconsulta no devuelve **ningún valor**, el test **ANY** devuelve **falso**.
Si el test de comparación es **falso** para **todos los valores** de la columna, **ANY** devuelve **falso**.
Si el test de comparación **no es verdadero** para ningún valor de la columna, **y es nulo** para al menos alguno de los valores, **ANY** devuelve **nulo**.

```
SELECT oficina, ciudad
FROM oficinas
WHERE objetivo > ANY (SELECT
SUM(cuota) FROM empleados GROUP
BY oficina)
```

En este caso la subconsulta devuelve una única columna con las sumas de las cuotas de los empleados de cada oficina.
Lista las oficinas cuyo objetivo sea superior a alguna de las sumas obtenidas.

● El **test ALL**.

La subconsulta debe devolver una única columna sino se produce un error.
Se evalúa la comparación con cada valor devuelto por la subconsulta.
Si todas las comparaciones individuales, producen un resultado **verdadero**, el test devuelve el valor

verdadero.

Si la subconsulta no devuelve **ningún valor** el test **ALL** devuelve el valor **verdadero**. (¡Ojo con esto!)

Si el test de comparación es **falso** para algún valor de la columna, el resultado es **falso**.

Si el test de comparación **no es falso** para ningún valor de la columna, pero es **nulo** para alguno de esos valores, el test **ALL** devuelve valor **nulo**.

```
SELECT oficina, ciudad
FROM oficinas
WHERE objetivo > ALL (SELECT
SUM(cuota) FROM empleados GROUP
BY oficina)
```

En este caso se listan las oficinas cuyo objetivo sea superior a todas las sumas.

● **Test de pertenencia a conjunto (IN).**

Examina si el **valor** de la expresión es uno de los valores **incluidos en la lista de valores producida por la subconsulta**.

La subconsulta debe generar una única columna y las filas que sean.

Si la subconsulta no produce ninguna fila, el test da falso.

Tiene la siguiente sintaxis:

expresion — IN — subconsulta

```
SELECT numemp, nombre, oficina
FROM empleados
WHERE oficina IN (SELECT oficina
FROM oficinas WHERE región = 'este')
```

Con la subconsulta se obtiene la lista de los números de oficina del **este** y la consulta principal obtiene los empleados cuyo número de oficina sea uno de los números de oficina del **este**. Por lo tanto lista los empleados de las oficinas del **este**.

● **El test de existencia EXISTS.**

Examina si la subconsulta produce alguna fila de resultados.

Si la subconsulta contiene filas, el test adopta el valor **verdadero**, si la subconsulta no contiene ninguna fila, el test toma el valor falso, nunca puede tomar el valor nulo.

Con este test la subconsulta **puede tener varias columnas**, no importa ya que el test se fija no en los valores devueltos sino en si hay o no fila en la tabla resultado de la subconsulta.

Cuando se utiliza el test de existencia en la mayoría de los casos habrá que utilizar una referencia externa. Si no se utiliza una referencia externa la subconsulta devuelta siempre será la misma para todas las filas de la consulta principal y en este caso se seleccionan todas las filas de la consulta principal (si la subconsulta genera filas) o ninguna (si la subconsulta no devuelve ninguna fila)

La sintaxis es la siguiente:

EXISTS — subconsulta
NOT ↑

```
SELECT numemp, nombre, oficina
FROM empleados
WHERE EXISTS (SELECT * FROM
oficinas WHERE región = 'este' AND
empleados.oficina = oficinas.oficina)
```

Este ejemplo obtiene lo mismo que el ejemplo del test **IN**.

Observa que delante de **EXISTS** no va ningún nombre de columna.

En la subconsulta se pueden poner las columnas que queramos en la lista de selección (hemos utilizado el *). Hemos añadido una condición adicional al **WHERE**, la de la referencia externa para que la oficina que se compare sea la oficina del empleado.

NOTA. Cuando se trabaja con tablas muy voluminosas el test **EXISTS** suele dar mejor rendimiento que el test **IN**.

Resumen del tema

● Una subconsulta es una sentencia **SELECT** que aparece en la lista de selección, o en las cláusulas **WHERE** o **HAVING** de otra sentencia **SELECT**.

● La subconsulta se ejecuta por cada fila de la consulta principal.

● Dentro de una consulta se puede utilizar una columna del origen de la consulta principal, una referencia externa.

● Aunque se puedan anidar subconsultas no es aconsejado más de un nivel de anidamiento.

● La subconsulta sufre una serie de restricciones según el lugar donde se encuentre.

● Las condiciones asociadas a las subconsultas son las siguientes:

el test de comparación con subconsulta

el test **ANY**, el test **ALL**, el test **IN**, el test **EXISTS**

Ejercicios tema 5. Las subconsultas Los ejercicios que te proponemos a continuación se pueden resolver de varias maneras, intenta resolverlos utilizando subconsultas ya que de eso trata el tema, además un mismo ejercicio lo puedes intentar resolver de diferentes maneras utilizando distintos tipos de condiciones, así un ejercicio se puede convertir en dos o tres ejercicios.

1 Listar los nombres de los clientes que tienen asignado el representante Alvaro Jaumes (suponiendo que no pueden haber representantes con el mismo nombre).

2 Listar los vendedores (numemp, nombre, y nº de oficina) que trabajan en oficinas "buenas" (las que tienen ventas superiores a su objetivo).

3 Listar los vendedores que no trabajan en oficinas dirigidas por el empleado 108.

4 Listar los productos (idfab, idproducto y descripción) para los cuales no se ha recibido ningún pedido de 25000 o más.

5 Listar los clientes asignados a Ana Bustamante que no han remitido un pedido superior a 3000 Pts..

6 Listar las oficinas en donde haya un vendedor cuyas ventas representen más del 55% del objetivo de su oficina.

7 Listar las oficinas en donde todos los vendedores tienen ventas que superan al 50% del objetivo de la oficina.

8 Listar las oficinas que tengan un objetivo mayor que la suma de las cuotas de sus vendedores.

● Cuando indicamos nombres de columnas, estos corresponden a nombres de **columna de la tabla**, pero no tienen por qué estar en el **orden** en que aparecen en la ventana diseño de la tabla, también **se pueden omitir algunas columnas**, la columnas que no se nombran tendrán **por defecto el valor NULL o el valor predeterminado** indicado en la ventana de diseño de tabla.

El ejemplo anterior se podría escribir de la siguiente forma:

INSERT INTO empleados (numemp,oficina, nombre, titulo,cuota, contrato, ventas)
VALUES (200, 30, 'Juan López', 'rep ventas',350000, #06/23/01#,0)

Observar que ahora hemos variado el orden de los valores y los nombres de columna no siguen el mismo orden que en la tabla origen, no importa, lo importante es poner los valores en el mismo orden que las columnas que enunciamos. Como no enunciamos las columnas oficina y director se rellenarán con el valor nulo (porque es el valor que tienen esas columnas como valor predeterminado).

● El utilizar la opción de **poner una lista de columnas** podría parecer peor ya que se tiene que escribir más pero realmente **tiene ventajas** sobre todo **cuando la sentencia la vamos a almacenar y reutilizar**:

● la sentencia queda **más fácil de interpretar** leyéndola vemos qué valor asignamos a qué columna,

● de paso nos **aseguramos** que el valor lo asignamos a la columna que queremos,

● **si** por lo que sea **cambia el orden de las columnas en la tabla** en el diseño, no pasaría nada mientras que de la otra forma intentaría asignar los valores a otra columna, esto produciría errores de *'tipo no corresponde'* y lo que es peor podría asignar valores erróneos sin que nos demos cuenta,

● otra ventaja es que **si se añade una nueva columna a la tabla** en el diseño, la primera sentencia INSERT daría error ya que el número de valores no corresponde con el número de columnas de la tabla, mientras que la segunda **INSERT** no daría error y en la nueva columna se insertaría el valor predeterminado.

● **Errores que se pueden producir** cuando se ejecuta la sentencia **INSERT INTO**:

● **Si la tabla de destino tiene clave principal** y en ese campo intentamos no asignar valor, asignar el valor nulo o un valor que ya existe en la tabla, el motor de base de datos Microsoft Jet no añade la fila y da un mensaje de error de *'infracciones de clave'*.

● **Si tenemos** definido **un índice único** (sin duplicados) e intentamos asignar un valor que ya existe en la tabla también devuelve el mismo error.

● **Si la tabla está relacionada con otra**, se seguirán las **reglas de integridad referencial**.

Insertar varias filas INSERT INTO...SELECT

Podemos **insertar en una tabla varias filas** con una sola sentencia **SELECT INTO** si los valores a insertar se pueden obtener como resultado de una consulta, en este caso sustituimos la cláusula **VALUES lista de valores** por una sentencia **SELECT** como las que hemos visto hasta ahora. **Cada fila resultado** de la **SELECT forma una lista de valores** que son los que se insertan en una nueva fila de la tabla destino. Es como si tuviésemos una **INSERT...VALUES** por cada fila resultado de la sentencia **SELECT**.

La sintaxis es la siguiente:



● El **origen** de la **SELECT** puede ser el **nombre de una consulta guardada, un nombre de tabla o una composición de varias tablas** (mediante **INNER JOIN, LEFT JOIN, RIGHT JOIN** o producto cartesiano).

● Cada fila devuelta por la **SELECT** actúa como la lista de valores que vimos con la **INSERT...VALUES** por lo que tiene las mismas restricciones en cuanto a tipo de dato, etc. La asignación de valores se realiza por posición por lo que la **SELECT** debe devolver el mismo número de columnas que las de la tabla destino y en el mismo orden, o el mismo número de columnas que indicamos en la lista de columnas después de destino.

● Las columnas de la **SELECT** no tienen porque llamarse igual que en la tabla destino ya que el sistema sólo se fija en los valores devueltos por la **SELECT**.

● Si no queremos asignar valores a todas las columnas entonces tenemos que indicar entre paréntesis la lista de columnas a rellenar después del nombre del destino.

● El estándar ANSI/ISO especifica varias restricciones sobre la consulta que aparece dentro de la sentencia **INSERT**:

● la consulta no puede tener una cláusula **ORDER BY**,

● la tabla destino de la sentencia **INSERT** no puede aparecer en la cláusula **FROM** de la consulta o de ninguna subconsulta que ésta tenga. Esto prohíbe insertar parte de una tabla en sí misma,

● la consulta no puede ser la **UNION** de varias sentencias **SELECT** diferentes,

● el resultado de la consulta debe contener el mismo número de columnas que las indicadas para insertar y los tipos de datos deben ser compatibles columna a columna.

● Sin embargo en **SQL de Microsoft Jet**,

● se puede incluir la cláusula **ORDER BY** aunque no tiene mucho sentido.

● se puede poner en la cláusula **FROM** de la consulta, la tabla en la que vamos a insertar,

● pero no podemos utilizar una **UNION**.

Ejemplo: Supongamos que tenemos una tabla llamada *repres* con la misma estructura que la tabla *empleados*, y queremos insertar en esa tabla los empleados que tengan como título *rep ventas*

INSERT INTO repres SELECT * FROM empleados WHERE titulo = 'rep ventas'

Con la **SELECT** obtenemos las filas correspondientes a los empleados con título *rep ventas*, y las insertamos en la tabla *repres*. Como las tablas tienen la misma estructura no hace falta poner la lista de columnas y podemos emplear * en la lista de selección de la **SELECT**.

Ejemplo: Supongamos ahora que la tabla *repres* tuviese las siguientes columnas *numemp*, *oficinarep*, *nombrerep*. En este caso no podríamos utilizar el asterisco, tendríamos que poner:

INSERT INTO repres SELECT numemp, oficina, nombre FROM empleados WHERE titulo = 'rep ventas'

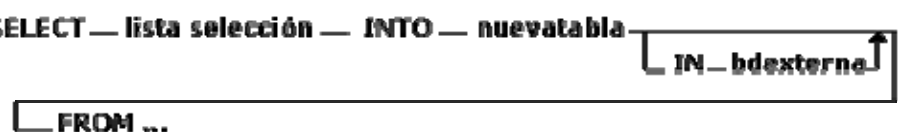
O bien:

INSERT INTO repres (numemp, oficinarep, nombrerep) SELECT numemp, oficina, nombre FROM empleados WHERE titulo = 'rep ventas'

Insertar filas en una nueva tabla **SELECT ... INTO**

Esta sentencia **inserta filas creando** en ese momento **la tabla donde se insertan** las filas. Se suele utilizar **para guardar en una tabla el resultado de una SELECT**.

La **sintaxis** es la siguiente:

SELECT — lista selección — **INTO** — nuevatabla — **IN — bexterna** 

Las columnas de la nueva tabla tendrán el mismo tipo y tamaño que las columnas origen, y se llamarán con el nombre de alias de la columna origen o en su defecto con el nombre de la columna origen, pero no se transfiere ninguna otra propiedad del campo o de la tabla como por ejemplo las claves e índices.

La sentencia **SELECT** puede ser cualquier sentencia **SELECT** sin ninguna restricción, puede ser una consulta multitabla, una consulta de resumen, una **UNION** ...

Ejemplo:

SELECT * INTO t2 FROM t1

Esta sentencia genera una nueva tabla t2 con todas las filas de la tabla t1. Las columnas se llamarán igual que en t1 pero t2 no será una copia exacta de t1 ya no tendrá clave principal ni relaciones con las otras tablas, ni índices si los tuviese t1 etc...

Si en la base de datos hay ya una tabla del mismo nombre, el sistema nos avisa y nos pregunta si la queremos borrar. Si le contestamos que no, la **SELECT** no se ejecuta.

Para formar una sentencia **SELECT INTO** lo mejor es escribir la **SELECT** que permite generar los datos que queremos guardar en la nueva tabla, y después añadir delante de la cláusula **FROM** la cláusula **INTO nuevatabla**.

La sentencia **SELECT INTO** se suele utilizar para crear tablas de trabajo, o tablas intermedias, las creamos para una determinada tarea y cuando hemos terminado esa tarea las borramos. También puede ser útil para sacar datos en una tabla para enviarlos a alguien.

Por ejemplo: Queremos enviarle a un representante una tabla con todos los datos personales de sus clientes para que les pueda enviar cartas etc...

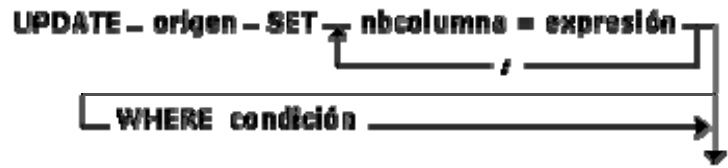
SELECT numclie AS codigo, nombre, direccion, telefono INTO susclientes FROM clientes WHERE repclie = '103';

Vamos a suponer que hemos añadido a nuestra tabla de *clientes* los campos *direccion* y *telefono*. En el ejemplo anterior la nueva tabla tendrá cuatro columnas llamadas *codigo*, *nombre*, *direccion*, *telefono* y contendrá las filas correspondientes a los clientes del representante 103.

Modificar el contenido de las filas (UPDATE)

La sentencia **UPDATE** modifica los valores de una o más columnas en las filas seleccionadas de una o varias tablas.

La sintaxis es la siguiente:



Origen puede ser un nombre de tabla, un nombre de consulta o una composición de tablas, también puede incluir la cláusula **IN** si la tabla a modificar se encuentra en una base de datos externa.

La cláusula **SET** especifica qué columnas van a modificarse y qué valores asignar a esas columnas.

nbcolumna, es el nombre de la columna a la cual queremos asignar un nuevo valor por lo tanto debe ser una columna de la tabla origen. El SQL estándar exige nombres sin cualificar pero algunas implementaciones (como por ejemplo el SQL de Microsoft Jet que estamos estudiando) sí lo permiten.

La **expresión** en cada asignación debe generar un valor del tipo de dato apropiado para la columna indicada. La expresión debe ser calculable a partir de los valores de la fila que se está

actualizando. *Expresión no puede ser una subconsulta.*

Ejemplo:

```
UPDATE oficinas INNER JOIN empleados
ON oficinas.oficina = empleados.oficina
SET cuota=objetivo*0.01;
```

En este ejemplo queremos actualizar las cuotas de nuestros empleados de tal forma que la cuota de un empleado sea el 1% del objetivo de su oficina. La columna a actualizar es la cuota del empleado y el valor a asignar es el 1% del objetivo de la oficina del empleado, luego la cláusula **SET** será **SET cuota = objetivo*0.01** o **SET cuota = objetivo/100**. El origen debe contener la cuota del empleado y el objetivo de su oficina, luego el origen será el **INNER JOIN** de empleados con oficinas.

🟡 La cláusula **WHERE** indica **qué filas van a ser modificadas**. Si se omite la cláusula **WHERE** se actualizan todas las filas.

🟡 En la condición del **WHERE** se puede incluir una subconsulta. En SQL standard la tabla que aparece en la **FROM** de la subconsulta no puede ser la misma que la tabla que aparece como origen, pero en el SQL de Microsoft Jet sí se puede.

Ejemplo: Queremos poner a cero las ventas de los empleados de la oficina 12

```
UPDATE empleados SET ventas = 0 WHERE oficina = 12;
```

Ejemplo: Queremos poner a cero el limite de credito de los clientes asignados a empleados de la oficina 12.

```
UPDATE clientes SET limitecredito = 0
WHERE repclie IN (SELECT numemp FROM empleados WHERE oficina = 12);
```

🟡 Si para el cálculo de *expresión* se utiliza una columna que también se modifica, el valor que se utiliza es el antes de la modificación, lo mismo para la condición de búsqueda.

🟡 Cuando se ejecuta una sentencia **UPDATE** primero se genera el origen y se seleccionan las filas según la cláusula **WHERE**. A continuación se coge una fila de la selección y se le aplica la cláusula **SET**, se actualizan todas las columnas incluidas en la cláusula **SET a la vez** por lo que los nombres de columna pueden especificarse en cualquier orden. Después se coge la siguiente fila de la selección y se le aplica del mismo modo la cláusula **SET**, así sucesivamente con todas las filas de la selección.

Ejemplo:

```
UPDATE oficinas SET ventas=0, objetivo=ventas;
```

O bien:

```
UPDATE oficinas SET objetivo=ventas, ventas=0;
```

Los dos ejemplos anteriores son equivalentes ya que el valor de *ventas* que se asigna a *objetivo* es el valor antes de la actualización, se deja como *objetivo* las ventas que ha tenido la oficina hasta el momento y se pone a cero la columna *ventas*.

🟡 Si actualizamos una columna definida como clave foránea, esta columna se podrá actualizar o no siguiendo las reglas de integridad referencial. El valor que se le asigna debe existir en la tabla de referencia.

🟡 Si actualizamos una columna definida como columna principal de una relación entre dos tablas, esta columna se podrá actualizar o no siguiendo las reglas de integridad referencial.

Borrar filas (DELETE)

La sentencia **DELETE** elimina filas de una tabla.

La sintaxis es la siguiente:



● **Origen** es el nombre de la **tabla de donde vamos a borrar**, podemos indicar un nombre de tabla, incluir la cláusula **IN** si la tabla se encuentra en una base de datos externa, también podemos escribir una composición de tablas.

● La opción **tabla.*** se utiliza cuando el **origen está basado en varias tablas**, y sirve para indicar en **qué tabla vamos a borrar**.

● La opción ***** es opcional y es la que se asume **por defecto** y se puede poner únicamente cuando el **origen es una sola tabla**.

● La cláusula **WHERE** sirve para especificar **qué filas queremos borrar**. Se eliminarán de la tabla todas las filas que cumplan la condición. Si **no se indica la cláusula WHERE**, **se borran TODAS las filas** de la tabla.

● **En la condición de búsqueda** de la sentencia **DELETE**, **se puede utilizar una subconsulta**. En SQL standard la tabla que aparece en la **FROM** de la subconsulta no puede ser la misma que la tabla que aparece en la **FROM** de la **DELETE** pero en el SQL de Microsoft Jet sí se puede hacer.

● Una vez borrados, **los registros no se pueden recuperar**.

● **Si la tabla donde borramos está relacionada con otras tablas** se podrán borrar o no los registros **siguiendo las reglas de integridad referencial** definidas en las relaciones.

Ejemplo:

```
DELETE * FROM pedidos WHERE clie IN (SELECT numclie FROM clientes WHERE nombre = 'Julian Lopez');
```

O bien:

```
DELETE pedidos.* FROM pedidos INNER JOIN clientes ON pedidos.clie = clientes.numclie WHERE nombre = 'Julian López';
```

Las dos sentencias borran los pedidos del cliente *Julian López*. En la segunda estamos obligados a poner **pedidos.*** porque el origen está basado en varias tablas.

DELETE * FROM pedidos; o **DELETE FROM pedidos;** Borra todas las filas de pedidos.

Resumen del tema

● Si queremos **añadir** en una tabla **una fila con valores conocidos** utilizamos la sentencia **INSERT INTO tabla VALUES (lista de valores)**.

● **Si los valores a insertar se encuentran en una o varias tablas** utilizamos **INSERT INTO tabla SELECT ...**

● Para **crear una nueva tabla con el resultado de una consulta** con la sentencia **SELECT...INTO tabla FROM...**

● Para **cambiar los datos contenidos en una tabla**, tenemos que actualizar las filas de dicha tabla con la sentencia **UPDATE tabla SET asignación de nuevos valores**.

● Para **eliminar filas de una tabla** se utiliza la sentencia **DELETE FROM tabla**.

● Con la cláusula **WHERE** podemos indicar **a qué filas afecta la actualización o el borrado**.

Conceptos básicos de integridad referencial.

Introducción

La **integridad referencial** es un sistema de **reglas** que utilizan la mayoría de las bases de datos relacionales para **asegurarse que los registros de tablas relacionadas son válidos** y que no se borren o cambien datos relacionados de forma accidental produciendo errores de integridad.

Primero repasemos un poco los tipos de relaciones.

Tipos de relaciones.

Entre dos tablas de cualquier base de datos relacional pueden haber dos tipos de relaciones, relaciones uno a uno y relaciones uno a muchos:

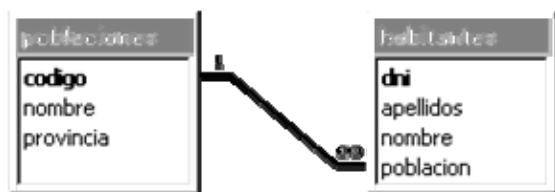
● **Relación Uno a Uno:** Cuando **un registro de una tabla sólo puede estar relacionado con un único registro de la otra tabla y viceversa**.

Por ejemplo: tenemos dos tablas una de profesores y otra de departamentos y queremos saber qué profesor es jefe de qué departamento, tenemos una relación uno a uno entre las dos tablas ya que un departamento tiene un solo jefe y un profesor puede ser jefe de un solo departamento.

● **Relación Uno a Varios:** Cuando **un registro de una tabla** (tabla secundaria) **sólo puede estar relacionado con un único registro de la otra tabla** (tabla principal) **y un registro de la tabla principal puede tener más de un registro relacionado en la tabla secundaria**, en este caso se suele hacer referencia a la tabla principal como tabla 'padre' y a la tabla secundaria como tabla 'hijo', entonces la regla se convierte en 'un padre puede tener varios hijos pero un hijo solo tiene un padre (regla más fácil de recordar).

Por ejemplo: tenemos dos tablas una con los datos de diferentes poblaciones y otra con los habitantes, una población puede tener más de un habitante, pero un habitante pertenecerá (estará empadronado) en una única población. En este caso la tabla principal será la de poblaciones y la tabla secundaria será la de habitantes. Una población puede tener varios habitantes pero un habitante pertenece a una sola población. Esta relación se representa incluyendo en la tabla 'hijo' una columna que se corresponde con la clave principal de la tabla 'padre', esta columna es lo denominamos clave foránea (o clave ajena o clave externa).

Una clave foránea es pues un campo de una tabla que contiene una referencia a un registro de otra tabla. Siguiendo nuestro ejemplo en la tabla habitantes tenemos una columna población que contiene el código de la población en la que está empadronado el habitante, esta columna es clave ajena de la tabla habitantes, y en la tabla poblaciones tenemos una columna código de población clave principal de la tabla.



● **Relación Varios a Varios:** Cuando **un registro de una tabla puede estar relacionado con más de un registro de la otra tabla y viceversa**. En este caso las dos tablas no pueden estar relacionadas directamente, se tiene que añadir una tabla entre las dos que incluya los pares de valores relacionados entre sí.

Por ejemplo: tenemos dos tablas una con los datos de *clientes* y otra con los *artículos* que se venden en la empresa, un cliente podrá realizar un pedido con varios artículos, y un artículo podrá ser vendido a más de un cliente.

No se puede definir entre *clientes* y *artículos*, hace falta otra tabla (por ejemplo una tabla de pedidos) relacionada con clientes y con artículos. La tabla pedidos estará relacionada con cliente por una relación uno a muchos y también estará relacionada con artículos por un relación uno a muchos.



Integridad referencial

Cuando se define una columna como clave foránea, las filas de la tabla pueden contener en esa columna o bien el valor nulo (ningún valor), o bien un valor que existe en la otra tabla, un error sería asignar a un habitante una población que no está en la tabla de poblaciones. Eso es lo que se denomina **integridad referencial** y consiste en que **los datos que referencian otros (claves foráneas) deben ser correctos**. La **integridad referencial** hace que el sistema gestor de la base de datos se asegure de que no hayan en las claves foráneas valores que no estén en la tabla principal.

La integridad referencial se activa en cuanto creamos una clave foránea y a partir de ese momento se comprueba cada vez que se modifiquen datos que puedan alterarla.

¿ Cuándo se pueden producir errores en los datos?

- **Cuando insertamos una nueva fila en la tabla secundaria y el valor de la clave foránea no existe en la tabla principal.** insertamos un nuevo habitante y en la columna *población* escribimos un código de población que no está en la tabla de poblaciones (una población que no existe).

- **Cuando modificamos el valor de la clave principal de un registro que tiene 'hijos'**, modificamos el código de Valencia, sustituimos el valor que tenía (1) por un nuevo valor (10), si Valencia tenía habitantes asignados, qué pasa con esos habitantes, no pueden seguir teniendo el código de población 1 porque la población 1 ya no existe, en este caso hay dos alternativas, no dejar cambiar el código de Valencia o bien cambiar el código de población de todos los habitantes de Valencia y asignarles el código 10.

- **Cuando modificamos el valor de la clave foránea, el nuevo valor debe existir en la tabla principal.** Por ejemplo cambiamos la población de un habitante, tenía asignada la población 1 (porque estaba empadronado en valencia) y ahora se le asigna la población 2 porque cambia de lugar de residencia. La población 2 debe existir en la tabla de poblaciones.

- **Cuando queremos borrar una fila de la tabla principal y ese registro tiene 'hijos'**, por ejemplo queremos borrar la población 1 (Valencia) si existen habitantes asignados a la población 1, estos no se pueden quedar con el valor 1 en la columna población porque tendrían asignada una población que no existe. En este caso tenemos dos alternativas, no dejar borrar la población 1 de la tabla de poblaciones, o bien borrarla y poner a valor nulo el campo población de todos sus 'hijos'.

Asociada a la integridad referencial están los conceptos de actualizar los registros en cascada y eliminar registros en cascada.

Actualización y borrado en cascada

El actualizar y/o eliminar registros en cascada, son opciones que se definen cuando definimos la clave foránea y que le indican al sistema gestor qué hacer en los casos comentados en el punto anterior

● Actualizar registros en cascada:

Esta opción le indica al sistema gestor de la base de datos que **cuando se cambie un valor del campo clave de la tabla principal**, automáticamente **cambiará el valor de la clave foránea de los registros relacionados en la tabla secundaria**.

Por ejemplo, si cambiamos en la tabla de poblaciones (la tabla principal) el valor 1 por el valor 10 en el campo código (la clave principal), automáticamente se actualizan todos los habitantes (en la tabla secundaria) que tienen el valor 1 en el campo población (en la clave ajena) dejando 10 en vez de 1.

Si no se tiene definida esta opción, no se puede cambiar los valores de la clave principal de la tabla principal. En este caso, si intentamos cambiar el valor 1 del código de la tabla de poblaciones, no se produce el cambio y el sistema nos devuelve un error o un mensaje que los registros no se han podido modificar por infracciones de clave.

● Eliminar registros en cascada:

Esta opción le indica al sistema gestor de la base de datos que **cuando se elimina un registro de la tabla principal** automáticamente **se borran también los registros relacionados en la tabla secundaria**.

Por ejemplo: Si borramos la población Onteniente en la tabla de poblaciones, automáticamente todos los habitantes de Onteniente se borrarán de la tabla de habitantes.

Si no se tiene definida esta opción, no se pueden borrar registros de la tabla principal si estos tienen registros relacionados en la tabla secundaria. En este caso, si intentamos borrar la población Onteniente, no se produce el borrado y el sistema nos devuelve un error o un mensaje que los registros no se han podido eliminar por infracciones de clave.

Conceptos básicos sobre índices.

Definición

Un índice en informática es como el índice de un libro donde tenemos los capítulos del libro y la página donde empieza cada capítulo. No vamos a entrar ahora en cómo se implementan los índices internamente ya que no entra en los objetivos del curso pero sí daremos unas breves nociones de cómo se definen, para qué sirven y cuándo hay que utilizarlos y cuando no.

Un índice es una estructura de datos que **permite recuperar las filas de una tabla de forma más rápida** además de **proporcionar una ordenación** distinta a la natural de la tabla. **Un índice se define sobre una columna o sobre un grupo de columnas**, y las filas se ordenarán según los valores contenidos en esas columnas. Por ejemplo, si definimos un índice sobre la columna *población* de la tabla de *clientes*, el índice permitirá recuperar los clientes ordenados por orden alfabético de población.

Si el índice se define **sobre varias columnas**, los registros se ordenarán **por la primera columna, dentro de un mismo valor de la primera columna se ordenarán por la segunda columna**, y así sucesivamente. Por ejemplo si definimos un índice sobre las columnas *provincia* y *población* se ordenarán los clientes por provincia y dentro de la misma provincia por población, aparecerían los de ALICANTE Denia, ALICANTE Xixona, VALENCIA Benetússer, VALENCIA Oliva.

El orden de las columnas dentro de un índice es **importante**, si retomamos el ejemplo anterior y definimos el índice sobre *población* y *provincia*, aparecerían los de VALENCIA Benetússer, ALICANTE Denia, VALENCIA Oliva, ALICANTE Xixona. Ahora se ordenan por población y los clientes de la misma población se ordenarían por el campo provincia.

Ventajas e inconvenientes

🟡 Ventajas:

Si una tabla tiene definido un índice sobre una columna **se puede localizar mucho más rápidamente una fila** que tenga un determinado valor en esa columna.

Recuperar las filas de una tabla **de forma ordenada** por la columna en cuestión también será mucho **más rápido**.

🟡 Inconvenientes:

Al ser el índice una estructura de datos adicional a la tabla, **ocupa** un poco **más de espacio** en disco.

Cuando se añaden, modifican o se borran filas de la tabla, el sistema debe actualizar los índices afectados por esos cambios lo que supone un **tiempo de proceso mayor**.

Por estas razones **no** es aconsejable **definir índices de forma indiscriminada**.

Los inconvenientes comentados en este punto no son nada comparados con las ventajas si la columna sobre la cual se define el índice es una columna que se va a utilizar a menudo para buscar u ordenar las filas de la tabla. Por eso una regla bastante acertada es **definir índices sobre columnas** que se vayan a utilizar **a menudo** para **recuperar u ordenar** las filas de una tabla.

El Access de hecho crea automáticamente índices sobre las columnas claves principales y sobre las claves foráneas ya que se supone que se utilizan a menudo para recuperar filas concretas.

Ejercicios tema 6. Actualización de datos

Como en estos ejercicios vamos a modificar los valores almacenados en la base de datos, es conveniente guardar antes una copia de las tablas, en los cuatro primeros ejercicios crearemos una copia de los datos almacenados para luego poder recuperar los valores originales.

- 1 Crear una tabla (llamarla *nuevaempleados*) que contenga las filas de la tabla *empleados*.
- 2 Crear una tabla (llamarla *nuevaoficinas*) que contenga las filas de la tabla *oficinas*.
- 3 Crear una tabla (llamarla *nuevaproductos*) que contenga las filas de la tabla *productos*.
- 4 Crear una tabla (llamarla *nuevapedidos*) que contenga las filas de la tabla *pedidos*.
- 5 Subir un 5% el precio de todos los productos del fabricante *ACI*.
- 6 Añadir una nueva oficina para la ciudad de *Madrid*, con el número de oficina *30*, con un objetivo de *100000* y región *Centro*.
- 7 Cambiar los empleados de la oficina *21* a la oficina *30*.
- 8 Eliminar los pedidos del empleado *105*.
- 9 Eliminar las oficinas que no tengan empleados.
- 10 Recuperar los precios originales de los productos a partir de la tabla *nuevosproductos*.
- 11 Recuperar las oficinas borradas a partir de la tabla *nuevaoficinas*.
- 12 Recuperar los pedidos borrados en el ejercicio 8 a partir de la tabla *nuevapedidos*.
- 13 A los empleados de la oficina *30* asignarles la oficina *21*.