Tema 5. SUBCONSULTAS.

Autora: Maria Teresa Miñana

¿Qué es una subconsulta?.

Una ***subconsulta*** en SQL consiste en utilizar los resultados de una consulta dentro de otra, que se considera la principal. Esta posibilidad fue la razón original para la palabra "estructurada" en el nombre Lenguaje de Consultas Estructuradas (Structured Query Language, SQL).

Anteriormente hemos utilizado la cláusula WHERE para seleccionar los datos que deseábamos comparando un valor de una columna con una constante, o un grupo de ellas. Si los valores de dichas constantes son desconocidos, normalmente por proceder de la aplicación de funciones a determinadas columnas de la tabla, tendremos que utilizar ***subconsultas***. Por ejemplo, queremos saber la lista de empleados cuyo salario supere el salario medio. En primer lugar, tendríamos que averiguar el importe del salario medio :

```
SQL> SELECT AVG(salario)"Salario Medio"
       FROM empleados;

Salario Medio
-------------
  256666,67
```

```
En Access:
           SELECT AVG(salario) AS Salario_Medio FROM empleados;
```

A continuación, anotarlo en un papel o recordarlo para la siguiente sentencia:

```
SQL> SELECT  dep_no "Nº Empleado",apellido,salario
       FROM empleados
       WHERE salario> 256666.67;

Nº Empleado APELLIDO   SALARIO
----------- -------- ---------
       30 GARRIDO     385000
       10 REY         600000
       20 GIL         335000
```

Sería mucho más eficiente utilizar una subconsulta:

```
En Acces:
        SELECT  dep_no as NºEmpleado,apellido,salario
        FROM empleados WHERE salario> 256666.67;
```

```
SQL> SELECT  dep_no "Nº Empleado",apellido,salario
       FROM empleados
       WHERE salario>(SELECT AVG(salario)
                      FROM empleados );
```

La subconsulta (comando SELECT entre paréntesis) se ejecuta primero y, posteriormente, el valor extraído es utilizado en la consulta principal.

```
En Acces:
        SELECT  dep_no as NºEmpleado,apellido,salario
        FROM empleados WHERE salario >
           (SELECT AVG(salario) AS Salario_Medio FROM empleados);
```

Formato de una subconsulta

```
??(SELECT???ALL  ???????*?????????????????????>
          ?DISTINCT?  ?columna_resultado?

  >??FROM lista_de_tablas ??????????????????????>

  >??????????????????????????????????????????????>
     ?WHERE condición_de_selección?

  >??????????????????????????????????????????????>
     ?GROUP BY lista_de_columnas_para_agrupar?

  >??????????????????????????????????????????????>)
     ?HAVING condición_de_selección?
```

donde el formato de la sentencia `SELECT` entre paréntesis tiene las siguientes diferencias con la sentencia `SELECT` de las consultas:

- No tiene sentido la cláusula `ORDER BY` ya que los resultados de una subconsulta se utilizan internamente y no son visibles al usuario.

- Los nombres de columna que aparecen en una subconsulta pueden referirse a columnas de la tabla de la consulta principal y se conocen como *referencias externas*.

Valores de retorno de las subconsultas y condiciones de selección.

Una subconsulta siempre forma parte de la condición de selección en las cláusulas `WHERE` o `HAVING.`

El resultado de una subconsulta puede ser un valor simple o más de un valor. Según el retorno de la subconsulta, el operador de comparación que se utilice en la condición de selección del `WHERE` o `HAVING` deberá ser del tipo apropiado según la tabla siguiente:

Operador comparativo	***Retorno de la subconsulta***
De tipo aritmético	Valor simple
De tipo lógico	Más de un valor

Condición de selección con operadores aritméticos de comparación.

Se utiliza cuando la subconsulta devuelve un único valor a comparar con una expresión, por lo general formada a partir de la fila obtenida en la consulta principal. Si la comparación resulta cierta (`TRUE`), la condición de selección también lo es. Si la subconsulta no devuelve ninguna fila (`NULL`), la comparación devuelve también el valor `NULL`.

Formato para la condición de selección con operadores aritméticos de comparación

```
?? expresión  operador_aritmético de comparación  subconsulta ??>

        Operadores_aritméticos de comparación: =,<>,<,>,<=,<=
```

Ejemplos.

1. Obtener todos los empleados que tienen el mismo oficio que 'Alonso'.

```
SQL> SELECT emp_no "Nº Empleado",apellido,oficio
       FROM empleados
       WHERE oficio=(SELECT oficio FROM empleados
                     WHERE UPPER(apellido)='ALONSO');
```

```
  Nº Empleado APELLIDO OFICIO
----------- -------- ----------
       7499 ALONSO   VENDEDOR
       7654 MARTIN   VENDEDOR
       7844 CALVO    VENDEDOR
```

```
En Access:
      SELECT emp_no AS NºEmpleado, apellido, oficio FROM empleados
            WHERE oficio=(SELECT oficio FROM empleados
                          WHERE UCASE(apellido)= 'ALONSO');
```

2. Obtener información de los empleados que ganan más que cualquier empleado del departamento 30.

```
SQL> SELECT emp_no "Nº Empleado",apellido,salario,dep_no
       "Nº Departamento"
       FROM empleados
       WHERE salario>(SELECT MAX(salario)
                      FROM empleados
                      WHERE dep_no=30);
```

```
Nº Empleado APELLIDO   SALARIO Nº Departamento
----------- -------- --------- ---------------
       7839 REY         600000              10
```

```
En Access:
SELECT emp_no As NºEmpleado, apellido, salario, dep_no AS NºDepartamento
  FROM empleados  WHERE salario>
          (SELECT MAX(salario)  FROM empleados  WHERE dep_no=30);
```

3. Visualizar el número de vendedores del departamento de ventas.

```
SQL> SELECT COUNT(*) "Total Empleados"
       FROM empleados
       WHERE dep_no=(SELECT dep_no
                     FROM departamentos
                     WHERE UPPER(dnombre)='VENTAS')
       AND UPPER(oficio)='VENDEDOR'
       GROUP BY oficio;
```

```
Total Empleados
---------------
```

```
En Access:
            SELECT COUNT(*) as Total_Empleados FROM empleados
               WHERE dep_no=(SELECT dep_no FROM departamentos
                                 WHERE UCASE(dnombre)='VENTAS')
               AND UCASE(oficio)='VENDEDOR';
```

4. Visualizar la suma de los salarios para cada oficio de los empleados del departamento de VENTAS.

```
SQL> SELECT oficio,sum(salario)"Suma salarios"
        FROM empleados WHERE dep_no=(SELECT dep_no
             FROM departamentos WHERE UPPER(dnombre)='VENTAS')
        GROUP BY oficio;

OFICIO     Suma salarios
---------- -------------
DIRECTOR          385000
VENDEDOR          470000
```

```
En Access:
        SELECT oficio,sum(salario) AS Suma_salarios FROM empleados
             WHERE dep_no= (SELECT dep_no FROM  Departamentos
                                  WHERE UCASE(dnombre)='VENTAS')
             GROUP BY oficio;
```

Condición de selección con operadores lógicos.

Se utiliza cuando la subconsulta devuelve más de una fila a comparar con la fila actual de la consulta principal. Los operadores lógicos permitidos por la mayoría de los gestores de bases de datos son: IN, EXISTS, ANY y ALL. El más utilizado es el operador IN.

Operador lógico IN.

Comprueba si valores de la fila actual de la consulta principal coincide con algunos de los devueltos por la subconsulta. Si el resultado es afirmativo la comparación resulta cierta (TRUE).

Formato para la condición de selección con el operador lógico IN

```
?? expresión  [NOT] IN  subconsulta ??>
```

Ejemplos.

1. Listar, en orden alfabético, aquellos empleados que no trabajen ni en Madrid ni en Barcelona.

```
SQL> SELECT emp_no "Nº Empleado",apellido,dep_no
       "NºDepartamento"
       FROM empleados
       WHERE dep_no IN (SELECT dep_no
                        FROM departamentos
                        WHERE UPPER(localidad) NOT IN
                        ('MADRID','BARCELONA'))
       ORDER BY apellido;
```

```
Nº Empleado APELLIDO NºDepartamento
----------- -------- --------------
       7876 GIL                  20
       7900 JIMENEZ              20
```

```
En Acces:
SELECT emp_no AS Nº_Empleado,apellido,dep_no  AS NºDepartamento
    FROM empleados  WHERE dep_no IN
      (SELECT dep_no  FROM departamentos WHERE UCASE(localidad) NOT IN
                ('MADRID','BARCELONA'))
    ORDER BY apellido;
```

La subconsulta selecciona todos los departamentos que no están en Madrid ni en Barcelona, y la consulta principal comprueba, empleado a empleado, si su departamento es uno de los seleccionados en la subconsulta, visualizando sus datos caso de ser cierto.

2. Listar los nombres de los departamentos que tengan algún empleado con fecha de alta anterior a 1982.

```
SQL> SELECT dep_no "NºDepartamento",dnombre Departamento
       FROM departamentos
       WHERE dep_no IN (SELECT dep_no
                        FROM empleados
                        WHERE fecha_alta<'01/01/82');

 NºDepartamento DEPARTAMENTO
-------------- --------------
            10 CONTABILIDAD
            30 VENTAS
```

```
En Acces:
       SELECT dep_no AS NºDepartamento, dnombre AS Departamento
           FROM departamentos
           WHERE dep_no IN (SELECT dep_no  FROM empleados
                            WHERE fecha_alta<#01/01/82#);
```

En este ejemplo, la subconsulta selecciona todos los empleados cuya fecha de alta sea anterior al año 1982, y la consulta principal compara, departamento a departamento, si coincide con el de alguno de los empleados seleccionados en la subconsulta.

3. Obtener los departamentos y sus nombres, siempre que haya más de un empleado trabajando en ellos.

```
SQL> SELECT dep_no "NºDepartamento",dnombre
         Departamento
         FROM departamentos
         WHERE dep_no IN (SELECT dep_no
                          FROM empleados
                          GROUP BY dep_no
                          HAVING COUNT(*)>1 );

NºDepartamento DEPARTAMENTO
-------------- --------------
            10 CONTABILIDAD
            20 INVESTIGACION
            30 VENTAS
```

```
En Acces:
       SELECT dep_no AS NºDepartamento,dnombre AS  Departamento
          FROM departamentos WHERE dep_no IN
              (SELECT dep_no FROM empleados GROUP BY dep_no
                                    HAVING COUNT(*)>1 );
```

Operador lógico EXISTS.

Se utiliza cuando la condición de selección consiste exclusivamente en comprobar que la subconsulta devuelve alguna fila seleccionada según la condición incluida en la propia subconsulta. El operador EXISTS no necesita que la subconsulta devuelva alguna columna porque no utiliza ninguna expresión de comparación, justificando así la aceptación del * en el formato de la misma.

Formato para la condición de selección con el operador lógico EXISTS

```
?? [NOT] EXISTS subconsulta ??>
```

Una subconsulta expresada con el operador EXISTS también podrá expresarse con el operador IN.

Ejemplo.

Listar las localidades donde existan departamentos con empleados cuya comisión supere el 10% del salario.

```
SQL> SELECT localidad
          FROM departamentos d
          WHERE EXISTS (SELECT *
                              FROM empleados e
                              WHERE comision>10*salario/100
                              AND e.dep_no=d.dep_no);

LOCALIDAD
----------
MADRID
```

Las tablas de departamentos y de empleados necesitan llevar alias para poder realizar parte de la condición de selección en la subconsulta ya que en ambas existe una columna con el mismo nombre (dep_no).

La misma subconsulta podemos expresarla con el operador IN de la siguiente manera:

```
SQL> SELECT localidad
          FROM departamentos
          WHERE dep_no IN (SELECT dep_no
                              FROM empleados
                              WHERE comision>10*salario/100);
```

Operadores lógicos ANY y ALL.

Se utilizan junto a los operadores aritméticos de comparación para ampliar las posibles comprobaciones de valores obtenidos a partir de la fila seleccionada en la consulta principal con

valores obtenidos en la subconsulta. Su uso a menudo es sustituido por el del operador **IN**.

Formato para la condición de selección con operadores lógicos ANY y ALL

```
Expresión  operador_aritmético de comparación {ANY|ALL} subconsulta →

          Operadores_aritméticos de comparación: =,<>,<,>,<=,<=
```

El operador ANY con uno de los seis operadores aritméticos compara el valor de la expresión formada a partir de la consulta principal con valores producidos por la subconsulta. Si alguna de las comparaciones individuales produce un resultado verdadero (TRUE), el operador ANY devuelve un resultado verdadero(TRUE).

Ejemplos.

1. Comprobar que todos los empleados tengan asignado un código de departamento existente en la tabla de departamentos.

```
SQL> SELECT dep_no "Nº Departamento",dnombre Departamento
       FROM departamentos
       WHERE dep_no = ANY (SELECT dep_no FROM empleados);

Nº Departamento DEPARTAMENTO
--------------- --------------
             10 CONTABILIDAD
             20 INVESTIGACION
             30 VENTAS
```

```
En Access:
      SELECT dep_no AS NºDepartamento,dnombre AS Departamento
           FROM departamentos
           WHERE dep_no = ANY (SELECT dep_no FROM empleados);
```

Con el operador **IN**:

```
SQL> SELECT dep_no "Nº Departamento",dnombre Departamento
       FROM departamentos
       WHERE dep_no IN (SELECT dep_no
                        FROM empleados);
```

```
En Access:
        SELECT dep_no as NºDepartamento,dnombre AS Departamento
              FROM departamentos
              WHERE dep_no IN (SELECT dep_no FROM empleados);
```

2. Seleccionar aquellos departamentos en los que al menos exista un empleado con comisión.

```
SQL> SELECT dep_no "Nº Departamento",dnombre Departamento
       FROM departamentos
       WHERE dep_no = ANY (SELECT dep_no
                           FROM empleados
                           WHERE comision>0);
```

```
En Access:
 SELECT dep_no AS NºDepartamento, dnombre AS Departamento
    FROM departamentos
    WHERE dep_no = ANY (SELECT dep_no FROM empleados WHERE comision>0);
```

```
Nº Departamento DEPARTAMENTO
--------------- --------------
             30 VENTAS
```

El operador `ALL` también se utiliza con los operadores aritméticos para comparar un valor de la expresión formada a partir de la consulta principal con cada uno de los valores de datos producidos por la subconsulta. Si todos los resultados de las comparaciones son ciertos (`TRUE`), `ALL` devuelve un valor cierto (`TRUE`).

Ejemplo.

Listar aquellos departamentos en los que todos sus empleados carezcan de información sobre su comisión.

```
SQL> SELECT dep_no "Nº Departamento",dnombre Departamento
       FROM departamentos d
       WHERE dep_no = ALL (SELECT dep_no
                           FROM empleados e
                           WHERE comision=NULL
                           AND e.dep_no=d.dep_no);

ninguna fila seleccionada
```

```
En Acces para preguntar por valores nulos utilizaremos IS NULL.
En este ejemplo la función ALL devuelve todos los departamentos:
      SELECT dep_no AS NºDepartamento,dnombre AS Departamento
          FROM departamentos d
          WHERE dep_no  <> ALL (SELECT dep_no
                                FROM empleados
                                WHERE comision IS NOT NULL);
```

Subconsultas en la selección de grupos.

Aunque las subconsultas suelen encontrarse sobre todo en la cláusula `WHERE`, también pueden usarse en la **`HAVING`** formando parte de la selección del grupo de filas efectuada por dicha cláusula.

Ejemplos.

1. Visualizar el departamento con más empleados.

```
SQL> SELECT dep_no "Nº Departamento",COUNT(*) "Total Empleados"
       FROM empleados
       GROUP BY dep_no
       HAVING COUNT(*)=(SELECT MAX(COUNT(*))
                        FROM empleados
                        GROUP BY dep_no);

Nº Departamento Total Empleados
--------------- ---------------
             30               4
```

En este ejemplo la subconsulta agrupa los empleados por departamentos, cuenta el número de empleados que hay en cada uno y selecciona cuál es el mayor valor de todos ellos. La consulta principal también agrupa los empleados por departamentos, los cuenta y, para cada total, compara

con el máximo valor obtenido en la subconsulta y visualiza los datos del departamento para el que la comparación resulta ser cierta.

En Access:
En este ejemplo vemos que hay dos funciones de columna anidadas, **SELECT MAX(COUNT(*))** .... En Access esto no es posible, estas select con funciones anidadas las tenemos que hacer por separado, es decir crear una consulta almacenada que contenga el contador de empleados por departamento, y luego la consulta donde se preguntará por el máximo valor de la consulta almacenada. Es decir haremos lo siguiente:

1º - Creamos una consulta almacenada para obtener el departamento y los empleados por cada departamento:

```
SELECT Dep_no,COUNT(*) as numero FROM empleados
                          GROUP BY dep_no ;
```

Pulsamos  y guárdamos la consulta, la llamamos NUM_EMPLE_DEPAR.
Una vez guardada volvemos a la vista SQL.

2º- Escribimos la siguiente consulta, en esta preguntamos si el contador de empleados por departamento es el máximo de la consulta creada anteriormente NUM_EMPLE_DEPAR:

```
SELECT dep_no AS NºDepartamento,COUNT(*) AS Total_Empleados
       FROM empleados
       GROUP BY dep_no
       HAVING COUNT(*)=
         (SELECT MAX(NUMERO) FROM NUM_EMPLE_DEPAR);
```

Y esta nos devolverá el departamento que tiene más empleados.

2. Visualizar los departamentos en los que el salario medio de sus empleados sea mayor o igual que la media de todos los salarios.

```
SQL> SELECT dep_no "Nº Departamento",AVG(salario)"Salario Medio"
       FROM empleados GROUP BY dep_no
       HAVING AVG(salario)>=(SELECT AVG(salario)
                             FROM empleados);

  Nº Departamento Salario Medio
---------------- -------------
              10     326666,67
```

```
En Access:
SELECT dep_no AS NºDepartamento, AVG(salario) AS Salario_Medio
       FROM empleados GROUP BY dep_no
       HAVING AVG(salario)>=(SELECT AVG(salario)  FROM empleados);
```

3. Visualizar el departamento con más presupuesto asignado para pagar el salario y la comisión de sus empleados.

```
SQL> SELECT dep_no "Nº Departamento",
       SUM(salario+NVL(comision,0))"Mayor presupuesto"
       FROM empleados GROUP BY dep_no
       HAVING SUM(salario+NVL(comision,0))=
         (SELECT MAX(SUM(salario+NVL(comision,0)))
                         FROM empleados GROUP BY dep_no);

Nº Departamento Mayor presupuesto
```

```
-------------- -----------------
            30           1055000
```

En Access:
Esta consulta tiene dos funciones de columna anidadas, **SELECT MAX(SUM(*)) ....** Hacemos lo siguiente para resolverla:

1º - Creamos una consulta almacenada para obtener el departamento y la suma del salario de los empleados de cada departamento:

```
SELECT dep_no,SUM(salario+NZ(comision,0)) as PRESUPUESTO
FROM empleados GROUP BY dep_no;
```

Si estamos en una consulta que ya ha sido almacenada elegir la opción del menú *Archivo/Guardar como.* Si no Pulsar al botón  . Damos un nombre a la consulta DEPAR_PRESUPUESTO.
Una vez guardada volvemos a la vista SQL.

2º- En la siguente consulta tenemos que obtener el departamento con más presupuesto, es decir que la suma de los salarios de sus empleados sea = al mayor presupuesto de la consulta almacenada DEPAR_PRESUPUESTO.

```
SELECT dep_no AS NºDepartamento,
       SUM(salario+NZ(comision,0)) AS Mayor_presupuesto
       FROM empleados GROUP BY dep_no
       HAVING SUM(salario+NZ(comision,0))=
           (SELECT max(PRESUPUESTO) FROM DEPAR_PRESUPUESTO);
```

Y esta nos devolverá el departamento con más presupuesto.

4.- Visualizar el departamento con más personal asignado del oficio *'empleado'*.

```
SQL> SELECT dep_no "Nº Departamento",COUNT(*)"Total empleados"
       FROM empleados
       WHERE UPPER(oficio) LIKE'EMPLEADO'
       GROUP BY dep_no
       HAVING COUNT(*)=(SELECT MAX(COUNT(*))
                        FROM empleados
                        WHERE UPPER(oficio) LIKE 'EMPLEADO'
                        GROUP BY dep_no);

Nº Departamento Total empleados
--------------- ---------------
             10               1
             20               1
```

En Access:
Esta consulta tiene dos funciones de columna anidadas, **SELECT MAX(COUNT(*)) ....** Hacemos lo siguiente para resolverla:

1º - Creamos una consulta almacenada para obtener por cada departamento el número de empleados con el oficio EMPLEADO:

```
SELECT dep_no,COUNT(*) as NUMEMPLE FROM empleados
WHERE UCASE(oficio) LIKE 'EMPLEADO' GROUP BY dep_no;
```

Si estamos en una consulta que ya ha sido almacenada elegir la opción del menú *Archivo/Guardar como*. Si no Pulsar al botón  . Damos un nombre a la consulta NUM_EMPLEADOS_DEPAR.
Una vez guardada volvemos a la vista SQL.

2º- En la siguente consulta tenemos que obtener el departamento con más Número de empleados en el oficio EMPLEADO, es decir que el contador de empleados sea = al mayor NUMEMPLE de la consulta almacenada NUM_EMPLEADOS_DEPAR.

```
SELECT dep_no AS NºDepartamento,COUNT(*) AS Total_empleados
   FROM empleados WHERE UCASE(oficio) LIKE  'EMPLEADO'
   GROUP BY dep_no  HAVING COUNT(*)=
        (SELECT MAX(NUMEMPLE) from NUM_EMPLEADOS_DEPAR);
```

Y esta nos devolverá lo que se pide.

Subconsultas anidadas.

Cuando una subconsulta forma parte de una condición de selección en una cláusula `WHERE` o `HAVING` de otra subconsulta se dice que es una ***subconsulta anidada***.

Ejemplos.

1.- Visualizar el número y el nombre del departamento con más personal de oficio *"empleado"*, usando las tablas de *empleados* y de *departamentos*.

```
SQL> SELECT dep_no "Nº Departamento",dnombre Departamento
       FROM departamentos
       WHERE dep_no=(SELECT dep_no FROM empleados
                     WHERE UPPER(oficio) LIKE 'EMPLEADO'
                     GROUP BY dep_no
                     HAVING COUNT(*)=
                     (SELECT MAX(COUNT(*)) FROM empleados
                        WHERE UPPER(oficio) LIKE 'EMPLEADO'
                        GROUP BY dep_no));
```

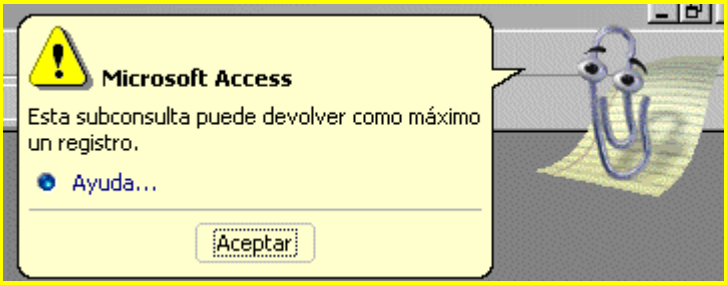
ORA-01427: la subconsulta de una sola fila devuelve más de una fila

En Access. Utilizamos la consumta almacenada NUM_EMPLEADOS_DEPAR.

```
    SELECT dep_no AS NºDepartamento, dnombre  AS Departamento
         FROM departamentos
         WHERE dep_no=(SELECT dep_no FROM empleados
                       WHERE UCASE(oficio) LIKE 'EMPLEADO'
                       GROUP BY dep_no
                       HAVING COUNT(*)=
                    (SELECT MAX(NUMEMPLE) from NUM_EMPLEADOS_DEPAR));
```

Al ejecutarla aparece el mismo error de la siguiente manera:

Figura 9. Error la subconsulta devuelve varias filas.



La subconsulta anidada `SELECT MAX(...)....`obtiene cuál es el mayor número de empleados (de oficio) por departamento.

La subconsulta `SELECT ...HAVING COUNT(*)`obtiene el número de departamento en el que se ha producido ese máximo.

La consulta principal visualiza el código y nombre de dicho departamento.

El mensaje anterior, obtenido como salida del ejercicio, se debe a la selección de dos departamentos con el máximo número de empleados: el 10 y el 20 tienen un empleado que cumple la condición. Los demás departamentos no tienen ninguno.

Si en el ejemplo anterior cambiamos el oficio por “vendedor”:

```
SQL> SELECT dep_no “Nº Departamento”,dnombre Departamento
          FROM departamentos
          WHERE dep_no=(SELECT dep_no FROM empleados
                        WHERE UPPER(oficio) LIKE ‘VENDEDOR’
                        GROUP BY dep_no
                        HAVING COUNT(*)=
                        (SELECT MAX(COUNT(*)) FROM empleados
                           WHERE UPPER(oficio) LIKE ‘VENDEDOR’
                           GROUP BY dep_no));

Nº Departamento DEPARTAMENTO
--------------- --------------
             30 VENTAS
```

Cuando una subconsulta devuelve varias filas en lugar de poner **=** ponemos la palabra **IN** , es decir que el valor que queremos comparar esté dentro de los valores que devuelve la subconsulta:

```
SELECT dep_no AS NºDepartamento, dnombre  AS Departamento
  FROM departamentos
    WHERE dep_no  IN (SELECT dep_no FROM empleados
                       WHERE UCASE(oficio) LIKE 'EMPLEADO'
                       GROUP BY dep_no
                       HAVING COUNT(*) =
                         (SELECT MAX(NUMEMPLE) from NUM_EMPLEADOS_DEPAR));
```

Subconsultas correlacionadas.

Cuando los nombres de columnas que aparecen en una subconsulta son nombres de columnas de la consulta principal o de otra subconsulta más externa, caso de las anidadas, se dice que son ***referencias externas*** y la ***subconsulta*** que es ***correlacionada***.

En las subconsultas correlacionadas, cada vez que se selecciona una nueva fila, en la consulta principal o en otra subconsulta más externa, que contenga la columna referenciada, se repetirá el proceso de selección.

Si en una subconsulta correlacionada coincide el nombre de una referencia externa con el nombre de alguna columna de la tabla que está siendo seleccionada en la subconsulta, se deberá asignar un alias a cada tabla y se utilizará para identificar cada columna.

Ejemplos.

1. Visualizar el número de departamento, el oficio y el salario de los oficios con mayor salario de cada departamento.

```
SQL> SELECT dep_no "Nº Departamento",oficio,salario
       FROM empleados e1
       WHERE salario=(SELECT MAX(salario)
                        FROM empleados e2
                        WHERE e1.dep_no=e2.dep_no);

 Nº Departamento OFICIO       SALARIO
--------------- ---------- ---------
             30 DIRECTOR      385000
             10 PRESIDENTE    600000
             20 ANALISTA      335000
```

```
En Access:
       SELECT dep_no AS NºDepartamento,oficio,salario
       FROM empleados e1
       WHERE salario=(SELECT MAX(salario)
                       FROM empleados e2 WHERE e1.dep_no=e2.dep_no);
```

La ***referencia externa*** sería **dep_no** y, aunque sea a la misma tabla, la subconsulta la trata como otra tabla, de la que la suya es copia. Asignamos alias para distinguir la referencia dep_no a una u otra tabla.

Tema 6. CONSULTAS MULTITABLA.

Autora: María Teresa Miñana

Multiplicaciones de tablas.

Hasta ahora, las órdenes SQL que hemos utilizado están basadas en una única tabla, pero a menudo es necesario consultar datos procedentes de dos o más tablas de la base de datos.

Para poder acceder a dos o más tablas de una base de datos, SQL genera internamente una tabla en la que cada fila de una tabla se combina con todas y cada una de las filas de las demás tablas. Esta operación es el **producto cartesiano** de las tablas que se están accediendo y la tabla resultante contiene todas las columnas de todas las tablas que se han multiplicado.

La multiplicación de tablas es también conocida como **composición** o **combinación** de tablas, según los diferentes gestores de bases de datos.

Ejemplo. Obtener todos los empleados con su nombre de departamento y su localidad.

```
SQL> SELECT emp_no "Nº empleado",apellido,dnombre Departamento,
       localidad
       FROM empleados,departamentos;

Nº empleado APELLIDO DEPARTAMENTO   LOCALIDAD
----------- -------- -------------- ----------
       7499 ALONSO   CONTABILIDAD   BARCELONA
       7521 LOPEZ    CONTABILIDAD   BARCELONA
       7654 MARTIN   CONTABILIDAD   BARCELONA
       7698 GARRIDO  CONTABILIDAD   BARCELONA
       7782 MARTINEZ CONTABILIDAD   BARCELONA
       7839 REY      CONTABILIDAD   BARCELONA
       7844 CALVO    CONTABILIDAD   BARCELONA
       7876 GIL      CONTABILIDAD   BARCELONA
       7900 JIMENEZ  CONTABILIDAD   BARCELONA
       7499 ALONSO   INVESTIGACION  VALENCIA
       7521 LOPEZ    INVESTIGACION  VALENCIA
       7654 MARTIN   INVESTIGACION  VALENCIA
       7698 GARRIDO  INVESTIGACION  VALENCIA
       7782 MARTINEZ INVESTIGACION  VALENCIA
       7839 REY      INVESTIGACION  VALENCIA
       7844 CALVO    INVESTIGACION  VALENCIA
       7876 GIL      INVESTIGACION  VALENCIA
       7900 JIMENEZ  INVESTIGACION  VALENCIA
       7499 ALONSO   VENTAS         MADRID
       7521 LOPEZ    VENTAS         MADRID
       7654 MARTIN   VENTAS         MADRID
       7698 GARRIDO  VENTAS         MADRID
       7782 MARTINEZ VENTAS         MADRID
       7839 REY      VENTAS         MADRID
       7844 CALVO    VENTAS         MADRID
       7876 GIL      VENTAS         MADRID
       7900 JIMENEZ  VENTAS         MADRID
       7499 ALONSO   PRODUCCION     SEVILLA
       7521 LOPEZ    PRODUCCION     SEVILLA
       7654 MARTIN   PRODUCCION     SEVILLA
       7698 GARRIDO  PRODUCCION     SEVILLA
       7782 MARTINEZ PRODUCCION     SEVILLA
       7839 REY      PRODUCCION     SEVILLA
       7844 CALVO    PRODUCCION     SEVILLA
       7876 GIL      PRODUCCION     SEVILLA
       7900 JIMENEZ  PRODUCCION     SEVILLA

36 filas seleccionadas.
```

```
Ebn Access:
   SELECT emp_no AS Nºempleado,apellido,dnombre AS Departamento, localidad
         FROM empleados,departamentos;
```

Cada empleado de la tabla de *empleados* aparece tantas veces como departamentos hay en la tabla de *departamentos.*

Composiciones o combinaciones simples.

Acabamos de ver, en el ejemplo anterior, que la salida producida por la multiplicación de las tablas de *empleados* y de *departamentos* no tiene sentido ni aplicación. La solución correcta del ejemplo sería:

```
SQL> SELECT emp_no "Nº empleado",apellido,dnombre Departamento,
localidad
          FROM empleados e,departamentos d
          WHERE e.dep_no=d.dep_no;
```

que produciría la siguiente salida:

```
Nº empleado APELLIDO DEPARTAMENTO   LOCALIDAD
----------- -------- -------------- ----------
       7499 ALONSO   VENTAS         MADRID
       7521 LOPEZ    CONTABILIDAD   BARCELONA
       7654 MARTIN   VENTAS         MADRID
       7698 GARRIDO  VENTAS         MADRID
       7782 MARTINEZ CONTABILIDAD   BARCELONA
       7839 REY      CONTABILIDAD   BARCELONA
       7844 CALVO    VENTAS         MADRID
       7876 GIL      INVESTIGACION  VALENCIA
       7900 JIMENEZ  INVESTIGACION  VALENCIA

9 filas seleccionadas.
```

```
Ebn Access:
   SELECT emp_no AS Nºempleado,apellido,dnombre AS Departamento, localidad
        FROM empleados e,departamentos d WHERE e.dep_no=d.dep_no;
```

SQL resuelve el anterior ejercicio con el siguiente proceso:

- La cláusula FROM genera todas las combinaciones posibles de filas de la tabla de *empleados* (9 filas) por las de la tabla de *departamentos* (4 filas), resultando una tabla producto de 4x9=36 filas.

- La cláusula WHERE selecciona únicamente aquellas filas de la tabla producto donde coinciden los números de departamento, que necesitan del alias por tener el mismo nombre en ambas tablas. En total se han seleccionado 9 filas y las 27 restantes se eliminan.

- La sentencia SELECT visualiza las columnas especificadas de la tablas producto para las filas seleccionadas.

Una composición o combinación (join) consiste en aplicar una condición de selección a las filas obtenidas de la multiplicación de las tablas sobre las que se está realizando una consulta.

La composición más sencilla es aquella en que la condición de selección se establece con el operador de igualdad entre las columnas que deban coincidir exactamente en tablas diferentes. Esta

composición es una ***equicomposición*** (**EQUAL JOIN**).

Formato de una equicomposición

```
SELECT columnas seleccionadas de las tablas de la FROM
 FROM tabla1 alias1,tabla2 alias2,...
WHERE {tabla1.columna=tabla2.columna|alias1.columna=alias2.columna};
```

Reglas de composición

- Pueden unirse tantas tablas como se desee (aunque la experiencia aconseja que no sean más de 8 por optimización).
- El criterio de emparejamiento para las tablas se denomina *predicado o criterio de composición.*
- Puede usarse más de una pareja de columnas para especificar la composición entre dos tablas cualesquiera.
- SQL no exige que las columnas de emparejamiento estén relacionadas como clave primaria y clave ajena. Pueden servir cualquier par de columnas de dos tablas, siempre que tengan tipos de datos comparables.
- En la SELECT pueden seleccionarse columnas de ambas tablas.
- Si hay columnas con el mismo nombre en las distintas tablas de la FROM, deben identificarse como *nombre de tabla.nombre de columna o* utilizar una alias para cada tabla.

Estas reglas se aplican a todo tipo de composición o combinación.

Ejemplos.

1. Obtener los distintos departamentos existentes en la tabla de *empleados.*

```
SQL> SELECT DISTINCT d.dep_no "Nº Departamento",d.dnombre
       Departamento
       FROM empleados e,departamentos d
       WHERE e.dep_no=d.dep_no;

Nº Departamento DEPARTAMENTO
--------------- --------------
             10 CONTABILIDAD
             20 INVESTIGACION
             30 VENTAS
```

```
En Access:
SELECT DISTINCT d.dep_no AS NºDepartamento,d.dnombre  AS  Departamento
FROM empleados e,departamentos d   WHERE e.dep_no=d.dep_no;
```

2. Mostrar los siguientes datos relativos a empleados: apellido, número, nombre de departamento y localidad**.**

```
SQL> SELECT emp_no "Nº Empleado",apellido,dnombre,localidad
```

```
       FROM empleados e,departamentos d
       WHERE e.dep_no=d.dep_no;

Nº Empleado APELLIDO DNOMBRE        LOCALIDAD
----------- -------- -------------- ----------
       7499 ALONSO   VENTAS         MADRID
       7521 LOPEZ    CONTABILIDAD   BARCELONA
       7654 MARTIN   VENTAS         MADRID
       7698 GARRIDO  VENTAS         MADRID
       7782 MARTINEZ CONTABILIDAD   BARCELONA
       7839 REY      CONTABILIDAD   BARCELONA
       7844 CALVO    VENTAS         MADRID
       7876 GIL      INVESTIGACION  VALENCIA
       7900 JIMENEZ  INVESTIGACION  VALENCIA
```

```
En Access:
 SELECT emp_no AS NºEmpleado,apellido,dnombre,localidad
       FROM empleados e,departamentos d
       WHERE e.dep_no=d.dep_no;
```

3. Seleccionar el número, apellido y oficio de los empleados que pertenezcan al departamento de VENTAS.

```
SQL> SELECT emp_no "Nº Empleado",apellido,oficio
       FROM empleados e,departamentos d
       WHERE e.dep_no=d.dep_no AND UPPER(dnombre)='VENTAS';

Nº Empleado APELLIDO OFICIO
----------- -------- ---------
       7499 ALONSO   VENDEDOR
       7654 MARTIN   VENDEDOR
       7698 GARRIDO  DIRECTOR
       7844 CALVO    VENDEDOR
```

```
En Access:
      SELECT emp_no AS NºEmpleado,apellido,oficio
      FROM empleados e,departamentos d
      WHERE e.dep_no=d.dep_no AND UCASE(dnombre)='VENTAS';;
```

3. Visualizar los apellidos y su longitud, de los empleados cuyo departamento contenga las letras 'ON' en la 2ª y 3ª posición.

```
   SQL> SELECT apellido,LENGTH(apellido)Medida,dnombre
          Departamento
          FROM empleados e,departamentos d
          WHERE e.dep_no=d.dep_no AND SUBSTR(dnombre,2,2)='ON';

APELLIDO    MEDIDA DEPARTAMENTO
-------- --------- --------------
LOPEZ             5 CONTABILIDAD
MARTINEZ          8 CONTABILIDAD
REY               3 CONTABILIDAD
```

```
En Access:
 SELECT apellido,LEN(apellido)AS Medida,dnombre AS Departamento
   FROM empleados e,departamentos d
   WHERE e.dep_no=d.dep_no AND MID(dnombre,2,2)='ON';
```

Composiciones o combinaciones de una tabla consigo misma.

Si desde una fila de una tabla podemos acceder a otra fila de la misma tabla estamos realizando una **composición o combinación de una tabla consigo misma.**

E**jemplo.**

1. Obtener la lista de los empleados con los nombres de sus directores.

```
SQL> SELECT e1.emp_no "NºEmpleado",e1.apellido,e1.director
     "NºDirector",e2.apellido "Nombre Director"
        FROM empleados e1,empleados e2
        WHERE e1.director=e2.emp_no;

 NºEmpleado APELLIDO NºDirector Nombre D
 ---------- -------- ---------- --------
       7499 ALONSO         7698 GARRIDO
       7654 MARTIN         7698 GARRIDO
       7844 CALVO          7698 GARRIDO
       7521 LOPEZ          7782 MARTINEZ
       7876 GIL            7782 MARTINEZ
       7900 JIMENEZ        7782 MARTINEZ
       7698 GARRIDO        7839 REY
       7782 MARTINEZ       7839 REY
```

```
En Access:
SELECT e1.emp_no AS NºEmpleado,e1.apellido,e1.director As NºDirector,
     e2.apellido AS Nombre_Director
      FROM empleados e1,empleados e2 WHERE e1.director=e2.emp_no;
```

Cada empleado de la tabla tiene una columna para su número de empleado (*emp_no*) y otra para el número de empleado de su director (*director*).

A partir del dato de la columna *director* de un empleado se puede acceder a otro empleado que contenga el mismo dato en su columna *emp_no*. Las dos filas de la tabla se están relacionando a través de las columnas *director* y *emp_no*.

El uso de alias es obligado por tratarse de la misma tabla donde todas las filas tienen las mismas columnas**.**

2. Obtener los jefes de los empleados cuyo oficio sea el de 'VENDEDOR'.

```
SQL> SELECT e1.emp_no "NºEmpleado",e1.apellido,e1.director
       "NºDirector",e1.oficio,e2.apellido "Nombre Director"
        FROM empleados e1,empleados e2
        WHERE e1.director=e2.emp_no AND e1.oficio='VENDEDOR';

 NºEmpleado APELLIDO NºDirector OFICIO     Nombre D
 ---------- -------- ---------- ---------- --------
       7499 ALONSO         7698 VENDEDOR   GARRIDO
       7654 MARTIN         7698 VENDEDOR   GARRIDO
       7844 CALVO          7698 VENDEDOR   GARRIDO
```

```
En Access:
SELECT e1.emp_no AS NºEmpleado,e1.apellido,e1.director AS NºDirector,
        e1.oficio,e2.apellido AS Nombre_Director
       FROM empleados e1,empleados e2
            WHERE e1.director=e2.emp_no AND e1.oficio ='VENDEDOR';
```

Las composiciones de una tabla consigo misma también reciben el nombre de **autocomposiciones** (**SELF JOIN**).

Formato de una autocomposición

```
SELECT columnas seleccionadas de las tablas de la FROM
 FROM tabla alias1,tabla alias2,...
 WHERE {alias1.columna=alias2.columna};
```

Composiciones o combinaciones externas.

Cuando se realiza una composición o combinación de tablas estableciendo una determinada relación entre sus columnas, puede ocurrir que no se emparejen todas las filas que debieran por faltar algunas de ellas. Es aconsejable que la salida, obtenida por una consulta en la que se pudiera presentar esta posibilidad, muestre todas las filas, aunque algunas con falta de información. Para conseguir este resultado se utiliza la **composición o combinación externa (OUTER JOIN)**.

La **representación gráfica** de una composición o combinación externa depende del SQL utilizado.

En ORACLE y en SQLBASE se coloca **(+) detrás de la columna en cuya tabla puede faltar información:**

WHERE *tabla1.columna1***(+)**=*tabla2.columna2* o

WHERE *tabla1.columna1*=*tabla2.columna2***(+)**

En SQLServer la representación gráfica es * :

WHERE *tabla1.columna1**=*tabla2.columna2*,

WHERE *tabla1.columna1*=**tabla2.columna2*,

WHERE *tabla1.columna1**=**tabla2.columna2* (composición externa completa, no permitida por ORACLE ni SQLBASE), o

WHERE *tabla1.columna1**>=*tabla2.columna2*.

En Acces se utiliza: RIGHT JOIN y LEFT JOIN

- LEFT JOIN – Combinación Izquierda obtendremos todos los registros de la tabla de la izquierda auque no exista combinación en la tabla de la derecha:

```
SELECT EMP_NO, APELLIDO, DNOMBRE, LOCALIDAD
        FROM DEPARTAMENTOS LEFT JOIN EMPLEADOS
        ON DEPARTAMENTOS.DEP_NO = EMPLEADOS.DEP_NO;
```

Esta SELECT obtiene los empleados y los departamentos, si un departamento no tiene empleados se visualiza también.

- RIGHT JOIN – Combinación derecha obtendremos todos los registros de la tabla de la derecha auque no exista combinación en la tabla de la izquierda:

```
SELECT EMP_NO, APELLIDO, DNOMBRE, LOCALIDAD
         FROM DEPARTAMENTOS RIGHT JOIN EMPLEADOS
         ON DEPARTAMENTOS.DEP_NO = EMPLEADOS.DEP_NO;
```

Esta SELECT obtiene los empleados y los departamentos, si un empleado no tuviera departamento también se visualizará.

Ejemplos.

1. Obtener datos de los empleados y de sus departamentos, incluyendo los que no tienen todavía empleados.(Ejemplo en composiciones simples)

```
SQL> SELECT emp_no "Nº empleado",apellido,dnombre Departamento,
       localidad
       FROM empleados e,departamentos d
       WHERE e.dep_no(+)=d.dep_no;

Nº empleado APELLIDO DEPARTAMENTO   LOCALIDAD
----------- -------- -------------- ----------
       7521 LOPEZ    CONTABILIDAD   BARCELONA
       7782 MARTINEZ CONTABILIDAD   BARCELONA
       7839 REY      CONTABILIDAD   BARCELONA
       7876 GIL      INVESTIGACION  VALENCIA
       7900 JIMENEZ  INVESTIGACION  VALENCIA
       7499 ALONSO   VENTAS         MADRID
       7844 CALVO    VENTAS         MADRID
       7654 MARTIN   VENTAS         MADRID
       7698 GARRIDO  VENTAS         MADRID
                     PRODUCCION     SEVILLA
```

El departamento de PRODUCCIÓN no tiene ningún empleado asignado.

```
En Acces:
     SELECT EMP_NO, APELLIDO, DNOMBRE, LOCALIDAD
           FROM DEPARTAMENTOS LEFT JOIN EMPLEADOS
           ON DEPARTAMENTOS.DEP_NO = EMPLEADOS.DEP_NO;
```

2. Obtener la lista de empleados con los nombres de sus directores, incluyendo al PRESIDENTE. (Ejemplo en autocomposiciones).

```
SQL> SELECT e1.emp_no "NºEmpleado",e1.apellido,e1.director
       "NºDirector",e2.apellido "Nombre Director"
       FROM empleados e1,empleados e2
       WHERE e1.director=e2.emp_no(+);

NºEmpleado APELLIDO NºDirector Nombre D
```

```
---------- -------- ---------- --------
      7499 ALONSO         7698 GARRIDO
      7654 MARTIN         7698 GARRIDO
      7844 CALVO          7698 GARRIDO
      7521 LOPEZ          7782 MARTINEZ
      7876 GIL            7782 MARTINEZ
      7900 JIMENEZ        7782 MARTINEZ
      7698 GARRIDO        7839 REY
      7782 MARTINEZ       7839 REY
       7839 REY
```

El PRESIDENTE no tiene ningún director.

```
En Acces:
     SELECT E1.EMP_NO AS NºEmpleado, E1.APELLIDO, E1.DIRECTOR AS NºDirector,
     E2.APELLIDO AS NOMBRE_DIRECTOR
     FROM EMPLEADOS AS E1 LEFT JOIN EMPLEADOS AS E2 ON E1.EMP_NO = E2.EMP_NO;
```

Composiciones o combinaciones basadas en desigualdad.

La condición de selección que establezcamos para componer o combinar tablas no tiene por qué ser siempre mediante el operador aritmético de igualdad, aunque su uso sea el más frecuente. SQL permite utilizar cualquier operador aritmético de comparación.

Ejemplos.

1. Listar los empleados de los departamentos diferentes al de VENTAS.

```
SQL> SELECT e.emp_no "NºEmpleado",e.apellido
       FROM empleados e,departamentos d
       WHERE d.dnombre='VENTAS' AND e.dep_no!=d.dep_no;

NºEmpleado APELLIDO
---------- --------
      7521 LOPEZ
      7782 MARTINEZ
      7839 REY
      7876 GIL
      7900 JIMENEZ
```

```
En Access:
SELECT e.emp_no AS NºEmpleado,e.apellido
          FROM empleados e,departamentos d
          WHERE d.dnombre='VENTAS' AND e.dep_no<>d.dep_no;
```

2. Listar los empleados de departamentos con códigos mayores que el código del departamento de contabilidad.

```
SQL> SELECT e.emp_no "NºEmpleado",e.apellido
       FROM empleados e,departamentos d
       WHERE d.dnombre='CONTABILIDAD' AND e.dep_no>d.dep_no;

NºEmpleado APELLIDO
---------- --------
      7499 ALONSO
      7654 MARTIN
      7698 GARRIDO
      7844 CALVO
      7876 GIL
      7900 JIMENEZ
```

```
En Access
SELECT e.emp_no AS NºEmpleado,e.apellido
               FROM empleados e,departamentos d
               WHERE d.dnombre='CONTABILIDAD' AND e.dep_no>d.dep_no;
```

3. Listar los empleados de departamentos con códigos menores que el código del departamento de producción.

```
SQL> SELECT e.emp_no "NºEmpleado",e.apellido
       FROM empleados e,departamentos d
       WHERE d.dnombre='PRODUCCION' AND e.dep_no<d.dep_no;

NºEmpleado APELLIDO
---------- --------
      7499 ALONSO
      7521 LOPEZ
      7654 MARTIN
      7698 GARRIDO
      7782 MARTINEZ
      7839 REY
      7844 CALVO
      7876 GIL
      7900 JIMENEZ
```

```
En Access
SELECT e.emp_no AS NºEmpleado,e.apellido
               FROM empleados e,departamentos d
               WHERE d.dnombre='PRODUCCION' AND e.dep_no<d.dep_no;
```

Composiciones y subconsultas.

Hay ocasiones en que una consulta puede resolverse con una composición o combinación (**join**) de tablas, o con una subconsulta. En este caso sería preferible hacerlo con una subconsulta.

En general, si no se necesita visualizar columnas de más de una tabla, se utiliza una subconsulta.

Si se necesita visualizar columnas de más de una tabla, se usará una composición o combinación.

Ejemplo.

Obtener apellido y oficio de los empleados que tienen el mismo oficio y mismo número de departamento que el de `INVESTIGACIÓN`.

Con subconsulta:

```
SQL> SELECT apellido,oficio FROM empleados
       WHERE oficio IN (SELECT oficio FROM empleados WHERE dep_no IN
       (SELECT dep_no FROM departamentos
                  WHERE dnombre='INVESTIGACION'));


APELLIDO OFICIO
-------- ---------
GIL      ANALISTA
LOPEZ    EMPLEADO
JIMENEZ  EMPLEADO
```

```
En Access
SELECT apellido,oficio
   FROM empleados WHERE oficio IN
    (SELECT oficio FROM empleados WHERE dep_no IN
     (SELECT dep_no FROM departamentos WHERE dnombre='INVESTIGACION'));
```

Con composición de tablas:

```
SQL> SELECT apellido,oficio
       FROM empleados
       WHERE oficio IN (SELECT e.oficio
                        FROM empleados e,departamentos d
                        WHERE e.dep_no=d.dep_no AND
                        d.dnombre='INVESTIGACION');

APELLIDO OFICIO
-------- ---------
GIL      ANALISTA
LOPEZ    EMPLEADO
JIMENEZ  EMPLEADO
```

```
En Access
SELECT apellido,oficio FROM empleados WHERE oficio IN
 (SELECT e.oficio FROM empleados e,departamentos d
       WHERE e.dep_no=d.dep_no AND d.dnombre='INVESTIGACION');
```

Tema 7. ACTUALIZACIONES

Autor: Fernando Montero

Introducción.

Como ya hemos explicado el lenguaje SQL incluye un conjunto de sentencias que permiten modificar, borrar e insertar nuevas filas en una tabla. Este subconjunto de comandos del lenguaje SQL recibe la denominación de *Lenguaje de Manipulación de Datos* (DML)

Inserción de nuevas filas en la base de datos.

Para añadir nuevas filas a una tabla utilizaremos la sentencia INSERT. Su formato típico es:

```
INSERT INTO nombretabla (listacolumnas) VALUES (listavalores);
```

Donde:

- *nombretabla*: es el nombre de la tabla en la que queremos insertar una fila.
- *listacolumnas*: incluye las columnas cuyos valores queremos insertar separadas por comas.
- *listavalores*: indica los valores que se insertarán separados por comas.

Por ejemplo, para insertar una nueva fila en la tabla *departamentos* introduciremos la siguiente instrucción:

```
SQL> INSERT INTO departamentos
     (dep_no, dnombre, localidad)
     VALUES (50, 'MARKETING','BILBAO');
```

Este formato de inserción tiene, además, las siguientes características:

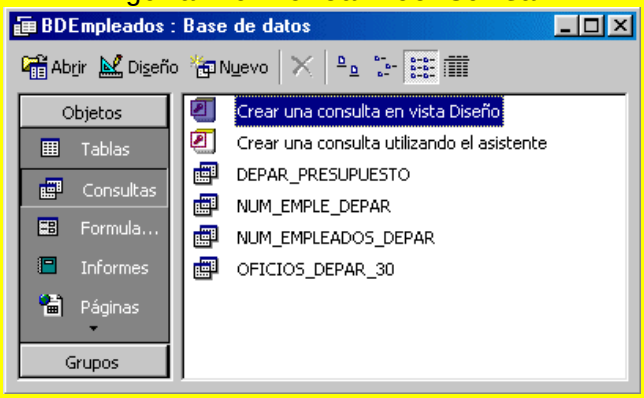
- En la lista de columnas se pueden indicar todas o algunas de las columnas de la tabla. En este último caso, aquellas columnas que no se incluyan en la lista quedarán sin ningún valor en la fila insertada, es decir, se asumirá el valor *NULL* para las columnas que no figuren en la lista.
- Los valores incluidos en la lista de valores deberán **corresponderse posicionalmente** con las columnas indicadas en la lista de columnas, de forma que el primer valor de la lista de valores se incluirá en la columna indicada en primer lugar, el segundo en la segunda columna, y así sucesivamente.
- Se puede utilizar **cualquier expresión** para indicar un valor siempre que el resultado de la expresión sea **compatible con el tipo de dato de la columna** correspondiente.

En Access para insertar filas hay que crear *Consultas de datos anexados* almacenarlas y luego ejecutarlas desde la *Ventana de la base de datos*.

Pasos:

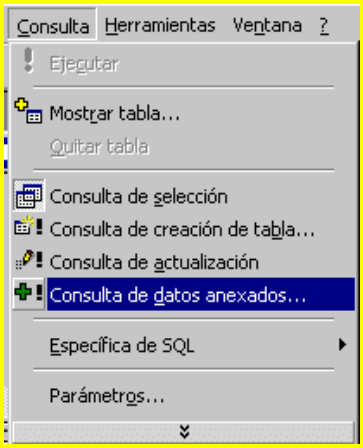
1º- Desde la *Ventana de la base de datos* elegimos el objeto *Consultas* y doble clic en *Crear consulta en vista de diseño.* Ver Figura 10.

Figura 10. Crear Consulta.



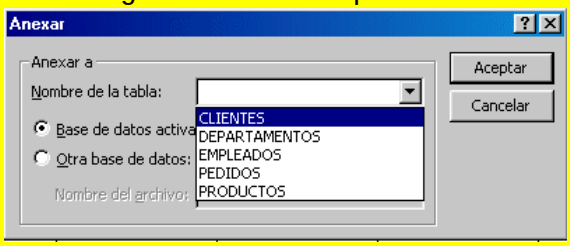
2º-Cerramos la ventana *Mostrar tabla*, a continuación abrimos el menú *Consulta* y elegimos la opción *Consulta de datos anexados* ver Figura 10.

Figura 10. Crear consulta de datos anexados.



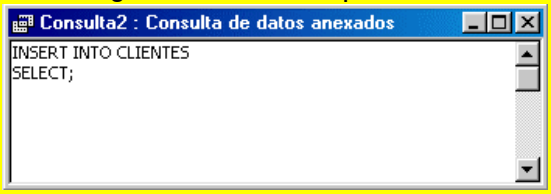
En la ventana que aparece elegimos la tabla donde vamos a insertar registros Ver Figura 11, y pulsamos *Aceptar*.

Figura 11. Elegir la tabla para insertar datos.



3º- A continuación abrimos la vista SQL, y vemos que aparece una orden INSERT
en la tabla CLIENTES y acompañada de una SELECT, Ver Figura 12

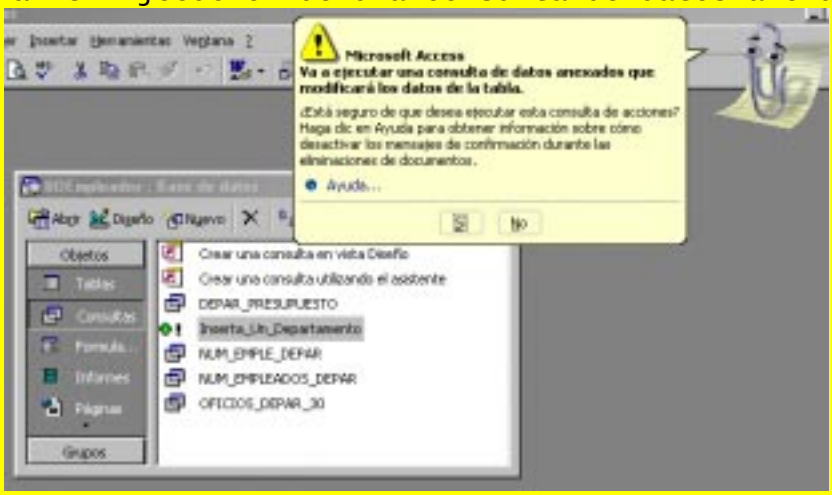
Figura 12. Elegir la tabla para insertar datos.



En esta ventana escribiremos la orden INSERT.

4º- Por último para ejecutar la orden INSERT y que los datos se inserten en la tabla es necesario guardar la consulta, darla un nombre, cerrar la vista SQL y desde la ventana de la base de datos seleccionarla y ejecutarla haciendo doble clic. Antes de insertar Acces visualizará mensajes avisando que se va a insertar un nuevo registro. Ver Figura 13.

Figura 13. Ejecución de una consulta de datos anexados.



Cada vez que escribamos una orden INSERT hay que guardar la consulta y luego ejecutarla desde la ventana de la base de datos.

Algunos ejemplos de inserciones válidas son:

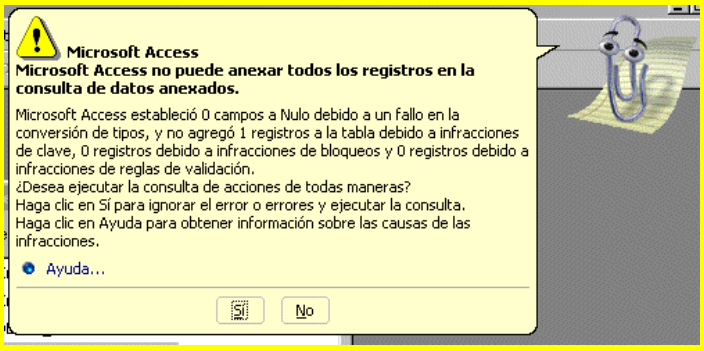
```
SQL> INSERT INTO departamentos
     (dnombre, localidad, dep_no)
     VALUES ( 'MARKETING','BILBAO', 50);

SQL> INSERT INTO departamentos
     (dep_no, dnombre)
     VALUES (50,'MARKETING');

SQL> INSERT INTO departamentos
     (dnombre, dep_no)
     VALUES ('MARKETING',50);

SQL> INSERT INTO departamentos
     (dnombre, localidad, dep_no)
     VALUES ( 'MARKETING',NULL, 50);
```

Nota: Cualquiera de las inserciones anteriores es válida, no obstante, el gestor de base de datos no permitirá insertar varias filas con el mismo valor en la columna *dep_no* ya que está definida como clave primaria con valores únicos. En el caso de que intentemos realizar todas las inserciones anteriores recibiremos un error como el siguiente:

```
ERROR en línea 1:
ORA-00001: restricción única (CURSOSQL.PK_DEPARTAMENTOS_DEP_NO) violada
```

En Access si intentamos insertar un registro con clave duplicada aparece un mensaje avisando que no se puede insertar, ver Figura 14.

Figura 14. Inserción errónea, clave duplicada.



Algunos intentos de inserciones erróneas:

```
SQL> INSERT INTO departamentos
     (dep_no, dnombre, localidad) VALUES (50,'MARKETING');

SQL> INSERT INTO departamentos
     (dep_no, dnombre) VALUES (50,'MARKETING','BILBAO');

SQL> INSERT INTO departamentos
     (dep_no, dnombre, localidad)
     VALUES ('MARKETING', 50, 'BILBAO');
```

```
SQL> INSERT INTO departamentos
     (dep_no, dnombre, localidad)
     VALUES (50,’BILBAO’,‘MARKETING’);
```

Access permite insertar valores numéricos para columnas definidas como VARCHAR o Texto y viceversa. Por ejemplo estas inserciones son válidas:

```
INSERT INTO departamentos ( dep_no, dnombre, localidad )
       VALUES (55,989801, 9898);

INSERT INTO departamentos ( dep_no, dnombre, localidad )
       VALUES ('556', 989801, 9898);
```

Esta No es válida:

```
INSERT INTO departamentos ( dep_no, dnombre, localidad )
       VALUES ('Nueva', 989801, 9898);
```

Para insertar valores tipo fecha, access obedece el patrón #mm/dd/año#, es decir primero teclear el mes, luego el día y luego el año.

Si ponemos:

```
INSERT INTO emple ( EMP_NO, APELLIDO, OFICIO, DIRECTOR,
FECHA_ALTA, SALARIO, COMISION, DEP_NO )
VALUES(6699,'Nuevo1','VENDEDOR',7698,#2/11/1999#,180000,Null, 30);
```

Access inserta la fecha correctamente y a la hora de ver los datos visualiza: 11/02/99 (patrón dd/mm/año).

Si ponemos:

```
INSERT INTO emple ( EMP_NO, APELLIDO, OFICIO, DIRECTOR,
FECHA_ALTA, SALARIO, COMISION, DEP_NO )
VALUES(6799,'Nuevo2','VENDEDOR',7698,#20/11/1999#,180000,Null,30);
```

Access inserta bien la fecha y visualiza: 20/11/99 (patrón dd/mm/año).
Si tecleamos un mes mayor que 12 y un día menor que 12, access, no hace caso a su patrón de inserción mm/dd/año y toma por defecto el mes como día y el día como mes.

Si ponemos:

```
INSERT INTO emple ( EMP_NO, APELLIDO, OFICIO, DIRECTOR,
FECHA_ALTA, SALARIO, COMISION, DEP_NO )
VALUES(6899,'Nuevo3','VENDEDOR',7698,#20/30/1999#,180000,Null,30);
```

Access emite un ERROR de sintaxis en la fecha #20/30/1999#.

Fechas correctas serían:
#9/11/1999# Access insertará 11/09/99 (dd/mm/aa)
#5/6/1999# Access insertará 06/05/99 (dd/mm/aa)
#3/16/1999# Access insertará 16/03/99 (dd/mm/aa)
#4/23/1999# Access insertará 23/04/99 (dd/mm/aa)

Fechas erróneas:
#20/13/1999#, #14/15/1999#, #13/13/1999#

También se pueden **insertar filas en una tabla sin especificar la lista de columnas** pero en este caso la **lista de valores deberá coincidir en número y posición** con las columnas de la tabla. El orden establecido para las columnas será el indicado al crear la tabla (el mismo que aparece al dar una instrucción *SELECT * FROM*...).

El formato genérico cuando no se especifica la lista de columnas es:

```
INSERT INTO nombretabla VALUES (listavalores);
```

A continuación se muestra un ejemplo de una instrucción INSERT sin lista de columnas:

```
SQL> INSERT INTO departamentos
     VALUES (50,'MARKETING','BILBAO');
```

Cuando no se especifica lista de columnas, se deberán especificar valores para todas las columnas. Si en algún caso no queremos incluir ningún valor para una columna deberemos indicar explícitamente el valor NULL para dicha columna.

Los siguientes son ejemplos válidos:

```
SQL> INSERT INTO departamentos
      VALUES (50,'MARKETING', NULL);

SQL> INSERT INTO departamentos
      VALUES (50, NULL, 'BILBAO');
```

Los siguientes ejemplos son erróneos:

```
SQL> INSERT INTO departamentos
     VALUES (50, 'MARKETING');

SQL> INSERT INTO departamentos
     VALUES (50, 'BILBAO', NULL);
```

Modificación de filas.

En ocasiones necesitaremos modificar alguno de los datos de las filas existentes de una tabla. Por ejemplo cambiar el salario o el departamento de uno o varios empleados, etcétera. En estos casos utilizaremos la sentencia UPDATE cuyo formato genérico es el siguiente:

```
UPDATE nombretabla
SET nobrecolumna = valor [,nombrecolumna = valor...]
[WHERE condición];
```

Donde:

- *nombretabla*: indica la tabla destino donde se encuentran las filas que queremos borrar.

- la cláusula *SET* va seguida de una o más asignaciones todas con el mismo formato: *nombrecolumna = valor*
- *nombrecolumna* : indica la columna cuyo valor queremos cambiar.
- *valor* : es una expresión que indica el valor nuevo para la columna. Se pueden incluir literales, variables del sistema, nombres de columna (en este caso tomará el valor que tiene a la columna de la fila que se va a modificar antes de que se produzca la modificación), etcétera. No se pueden incluir funciones de grupo ni subconsultas.
- La cláusula WHERE permite especificar una condición de selección de las filas. En el caso de que no se utilice, la actualización afectará a toda la tabla.

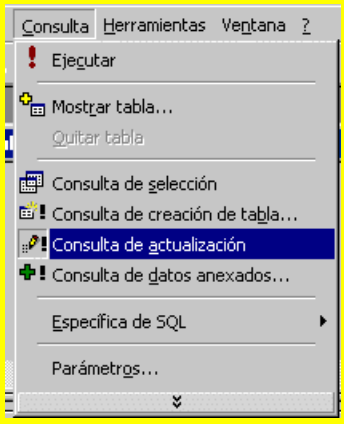
En Access para actualizar filas hay que crear *Consultas de actualización* almacenarlas y luego ejecutarlas desde la *Ventana de la base de datos*.

Pasos:

1º- Desde la *Ventana de la base de datos* elegimos el objeto *Consultas* y doble clic en *Crear consulta en vista de diseño*. Ver Figura 10.

2º-Cerramos la ventana *Mostrar tabla*, a continuación abrimos el menú *Consulta* y elegimos la opción *Consulta de actualización* ver Figura 14.

Figura 14. Crear consulta de actualización.



3º- A continuación abrimos la vista SQL, y vemos que aparece una orden `UPDATE SET ;`

En esta ventana escribiremos la orden `UPDATE`.

4º- Por último para ejecutar la orden y que los datos se actualicen en la tabla es necesario guardar la consulta, darla un nombre, cerrar la vista SQL y desde la ventana de la base de datos seleccionarla y ejecutarla haciendo doble clic. Access visualizará un mensaje para avisar que se va a actualizar un registro. Si ocurre algún error Access visualizará los mensajes correspondientes.

El siguiente ejemplo modificará el OFICIO del empleado cuyo número es el 7900 cambiando el oficio que tuviese por el de ANALISTA:

```
SQL> UPDATE empleados
```

```
      SET oficio = 'ANALISTA'
      WHERE emp_no = 7900;
```

Ahora supongamos que se desea aumentar 30.000 Ptas. el salario del mismo empleado:

```
SQL> UPDATE empleados
      SET salario = salario + 30000
      WHERE emp_no = 7900;
```

También podíamos haber modificado las dos columnas con una sola sentencia:

```
SQL> UPDATE empleados
      SET oficio = 'ANALISTA', salario = salario + 30000
      WHERE emp_no = 7900;
```

Las actualizaciones anteriores afectan a una única fila pero podemos escribir comandos de actualización que afecten a varias filas:

```
SQL> UPDATE empleados
      SET comision = comision + 10000
      WHERE oficio = 'VENDEDOR';
3 filas actualizadas.
```

Si no se incluye la cláusula WHERE la actualización afectará a todas las filas. El siguiente ejemplo modifica el salario de todos los empleados incrementándolo en un 3%

```
SQL> UPDATE empleados
      SET salario = salario * 1.03;
9 filas actualizadas.
```

En Access estas sentencias se escriben igual, pero recuerda que para que se ejecuten hay que guardar la consulta, cerrar la ventana SQL y ejecutar la consulta desde la ventana de la base de datos.

Eliminación de filas.

Para eliminar o suprimir filas de una tabla utilizaremos la sentencia DELETE cuyo formato genérico es el siguiente:

```
DELETE FROM nombretabla [WHERE condicion];
```

Donde:

- *nombretabla*: indica la tabla destino donde se encuentran las filas que queremos borrar.
- *condicion*: permite especificar la condición que deben cumplir las filas que serán eliminadas. En el caso de que **no se especifique ninguna condición** el SGDBR **asumirá que se desean eliminar todas las filas** de la tabla; en este caso no se eliminará la tabla pero quedará completamente vacía.

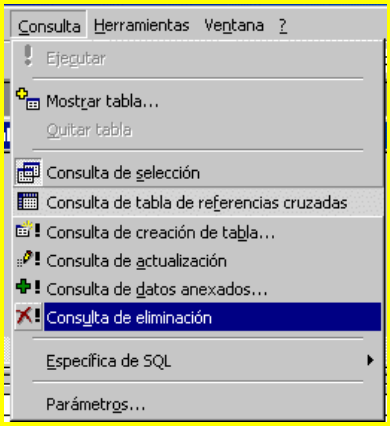
En Access para borrar filas hay que crear *Consultas de eliminación* almacenarlas y luego ejecutarlas desde la *Ventana de la base de datos.*

Pasos:

1º- Desde la *Ventana de la base de datos* elegimos el objeto *Consultas* y doble clic en *Crear consulta en vista de diseño.* Ver Figura 10.

2º-Cerramos la ventana *Mostrar tabla*, a continuación abrimos el menú *Consulta* y elegimos la opción *Consulta de eliminación* ver Figura 15.

Figura 15. Crear consulta de eliminación.



3º- A continuación abrimos la vista SQL, y vemos que aparece una orden `DELETE *;`

En esta ventana escribiremos la orden `DELETE`.

4º- Por último se hace como en los casos anteriores.

A continuación se muestran algunos ejemplos de instrucciones DELETE:

```
SQL> DELETE FROM departamentos
     WHERE dnombre = 'CONTABILIDAD';

SQL> DELETE FROM departamentos
     WHERE localidad= 'MADRID';

SQL> DELETE FROM departamentos
     WHERE LENGTH(dnombre)> 8;
```

Control de transacciones: COMMIT y ROLLBACK.

Los gestores de bases de datos disponen de dos comandos que permiten confirmar o deshacer los cambios realizados en la base de datos:

- COMMIT : confirma los cambios realizados haciéndolos permanentes y visibles para los demás usuarios.

- ROLLBACK : deshace los cambios realizados.

Para nuestro caso, el comando ROLLBACK nos permite deshacer todos los cambios realizados en las tablas como resultado de los ejemplos.

```
SQL> ROLLBACK;

Rollback terminado.
```

En Access no existe el control de transacciones `COMMIT y ROLLBACK.`

Actualizaciones con subconsultas.

Inserciones con subconsultas.

Podemos insertar en una tabla el resultado de una consulta sobre otra tabla. En este caso normalmente se insertarán varias filas con una sola sentencia. Utilizaremos el siguiente formato:

```
INSERT INTO nombretabla [(listacolumnas)] consulta;
```

En el formato anterior podemos destacar:

- La tabla que sigue a la cláusula INTO es, igual que en las ocasiones anteriores, **la tabla destino** en la que queremos insertar filas.
- La **lista de columnas es opcional** pero deberá especificarse cuando las columnas que devuelve la consulta no coinciden en número o en orden con las columnas de la tabla destino.
- La **consulta puede ser cualquier comando de selección** válido (salvo las restricciones indicadas abajo) siempre que exista una correspondencia entre las columnas devueltas y las columnas de la tabla destino, o la lista de columnas.

Supongamos que disponemos de la tabla *ndepart* cuyas columnas son *numdept*, *departamento* y *localidad*; y queremos insertar en ella las filas correspondientes a los departamentos de la tabla *departamentos* cuyo nombre tiene más de 8 letras.

En este caso existe correspondencia en número y en orden entre las columnas de la tabla destino y las columnas de la selección; por tanto, no hace falta especificar la lista de columnas y el comando requerido será:

```
SQL> INSERT INTO ndepart
     SELECT * FROM departamentos
     WHERE LENGTH(dnombre)> 8;
```

```
En Access es necesario especificar las columnas origen y destino:

     INSERT INTO ndepart ( NUMDEPT, DEPARTAMENTO, LOCALIDAD)
     SELECT DEP_NO, DNOMBRE, LOCALIDAD
     FROM departamentos WHERE LEN(dnombre)>8;
```

Si la tabla destino tuviese una estructura diferente deberemos forzar la correspondencia, bien al

especificar la lista de selección, bien especificando la lista de columnas, o bien utilizando ambos recursos.

Por ejemplo, supongamos que la tabla destino es *n2dep* cuyas columnas son *departamento* y *numero*. En este caso procederemos:

```
SQL> INSERT INTO n2dep
     SELECT dnombre, dep_no FROM departamentos
     WHERE LENGTH(dnombre)> 8;
```

O bien:

```
SQL> INSERT INTO n2dep (numero, departamento) SELECT dep_no, dnombre
FROM departamentos WHERE LENGTH(dnombre)> 8;
```

```
En Acces, hay que poner las columnas de las tablas:

INSERT INTO n2dep (departamento, numero )
      SELECT dnombre, dep_no FROM departamentos
             WHERE LEN(dnombre)> 8;

INSERT INTO n2dep (numero, departamento)
      SELECT dep_no, dnombre FROM departamentos
             WHERE LEN(dnombre)> 8;
```

Restricciones: Además de la necesidad de coincidencia en número y compatibilidad en tipo de las columnas, el estándar establece algunas restricciones para la cláusula INSERT multifila:

- La tabla destino no puede aparecer en la consulta.
- La consulta no puede ser una UNION.
- No se puede incluir una cláusula ORDER BY en la consulta.

Por ejemplo la siguiente orden transgrede la primera restricción:

```
SQL> INSERT INTO departamentos
     SELECT * FROM departamentos
     WHERE LENGTH(dnombre)> 8;
```

En la práctica la mayoría de los productos comerciales permiten saltar estas restricciones.

Modificación y eliminación de filas con subconsultas en la cláusula WHERE.

En ocasiones la condición que deben cumplir las filas que deseamos eliminar o modificar implica realizar una subconsulta a otras tablas. En estos casos se incluirá la subconsulta en la condición y los operadores necesarios tal como estudiamos en el apartado correspondiente del tema SUBCONSULTAS.

Por ejemplo, supongamos que se desea elevar en 5000 Ptas. el salario de todos los empleados cuyo departamento no esté en MADRID.

```
SQL> UPDATE empleados SET salario = salario + 5000
     WHERE dep_no NOT IN
     (SELECT dep_no FROM departamentos
                 WHERE localidad = 'MADRID');
```

```
En Access:
     UPDATE empleados SET salario = salario + 5000
         WHERE dep_no NOT IN (SELECT dep_no FROM departamentos
                  WHERE localidad ='MADRID');
```

Ahora supongamos que queremos eliminar de la tabla *departamentos* aquellos que no tienen ningún empleado.

```
SQL> DELETE FROM departamentos WHERE dep_no NOT IN
         (SELECT DISTINCT dep_no FROM empleados);
```

```
En Access:
  DELETE * FROM departamentos
        WHERE dep_no NOT IN (SELECT DISTINCT dep_no FROM
empleados);
```

El siguiente ejemplo eliminará los departamentos que tienen menos de tres empleados.

```
SQL> DELETE FROM departamentos WHERE dep_no IN
     (SELECT dep_no FROM empleados
     GROUP BY dep_no HAVING count(*) < 3);
```

```
En Access:
DELETE * FROM departamentos WHERE dep_no IN
    (SELECT dep_no FROM empleados
         GROUP BY dep_no HAVING count (*) < 3);
```

Nota.- El ejemplo anterior borrará los departamentos que tengan menos de tres empleados pero, para que entre en la lista de selección, el departamento deberá tener al menos un empleado. Por tanto, los empleados que no tengan ningún empleado no se borrarán. Para evitar esto podemos cambiar la condición indicando que se borren aquellos departamentos que no están en la lista de departamentos con tres o más empleados.

```
SQL> DELETE FROM departamentos WHERE dep_no NOT IN
     (SELECT dep_no FROM empleados
     GROUP BY  dep_no HAVING count (*) >= 3);
```

```
En Access:
DELETE * FROM departamentos WHERE dep_no NOT IN
        (SELECT dep_no FROM empleados
             GROUP BY  dep_no HAVING count (*) >= 3);
```

Esta última orden borrará todos los departamentos que tienen: ninguno, uno o dos empleados.

Se pueden utilizar subconsultas anidadas a varios niveles, pero respetando la siguiente **restricción: la tabla destino no puede aparecer en la cláusula from de ninguna de las subconsultas que intervienen en la selección. Si se permiten referencias externas,** como en el siguiente ejemplo:

```
SQL> DELETE FROM departamentos WHERE NOT EXISTS
     (SELECT DISTINCT dep_no FROM empleados
     WHERE empleados.dep_no = departamentos.dep_no);
```

```
En Access:
   DELETE * FROM departamentos WHERE NOT EXISTS
       (SELECT DISTINCT dep_no FROM empleados
            WHERE empleados.dep_no = departamentos.dep_no);
```

En cualquier caso muchos productos comerciales permiten saltar esta restricción. En estos casos la subconsulta con la referencia externa realiza la selección sobre la tabla destino antes de que se elimine ninguna fila.

Restricciones de integridad y actualizaciones.

Los gestores de bases de datos relacionales permiten especificar ciertas condiciones que deban cumplir los datos contenidos en las tablas, como por ejemplo:

- Conjunto de valores que puede o debe tomar una columna.
- Columnas que tienen que tener necesariamente un valor no nulo.
- Columnas que no pueden tener valores duplicados.
- Columnas que hacen referencia a otras de la misma o de otras tablas.
- Etcétera.

En nuestras tablas de ejemplo están definidas las siguientes restricciones:

- **Claves primarias** (PRIMARY KEY). Sirven para referirse a una fila de manera inequívoca. No se permiten valores repetidos.

```
TABLA DEPARTAMENTOS: PRIMARY KEY (DEP_NO)
TABLA EMPLEADOS: PRIMARY KEY (EMP_NO)
TABLA CLIENTES: PRIMARY KEY(CLIENTE_NO)
TABLA PRODUCTOS: PRIMARY KEY (PRODUCTO_NO)
TABLA PEDIDOS: PRIMARY KEY (PEDIDO_NO)
```

- **Claves ajenas** (FOREIGN KEY): Se utilizan para hacer referencia a columnas de otras tablas. Cualquier valor que se inserte en esas columnas tendrá su equivalente en la tabla referida. Opcionalmente se pueden indicar las acciones a realizar en caso de borrado o cambio de las columnas a las que hace referencia .

```
TABLA EMPLEADOS: FOREIGN KEY (DEP_NO)
     REFERENCES DEPARTAMENTOS(DEP_NO)

TABLA CLIENTES: FOREIGN KEY (VENDEDOR_NO)
     REFERENCES EMPLEADOS(EMP_NO)
```

```
TABLA PEDIDOS: FOREIGN KEY (PRODUCTO_NO)
     REFERENCES PRODUCTOS(PRODUCTO_NO)

TABLA PEDIDOS: FOREIGN KEY (CLIENTE_NO)
     REFERENCES CLIENTES(CLIENTE_NO)
```

Estas condiciones se determinan al diseñar la base de datos y se especifican e implementan mediante el lenguaje de definición de datos (DDL) que estudiaremos más adelante. No obstante, **todos los comandos de manipulación de datos deberán respetar estas restricciones de integridad** ya que en caso contrario el comando fallará y el sistema devolverá un error.

Cuando en Access se realiza una operación y no se respetan las restricciones de integridad de datos, access visualiza distintos mensajes de error dependiendo de la operación. En la Figura 16, 17 y 18 se muestran errores en operaciones de inserción de registros, de borrado y de actualización.

Figura 16. Mensaje de error en Inserción.

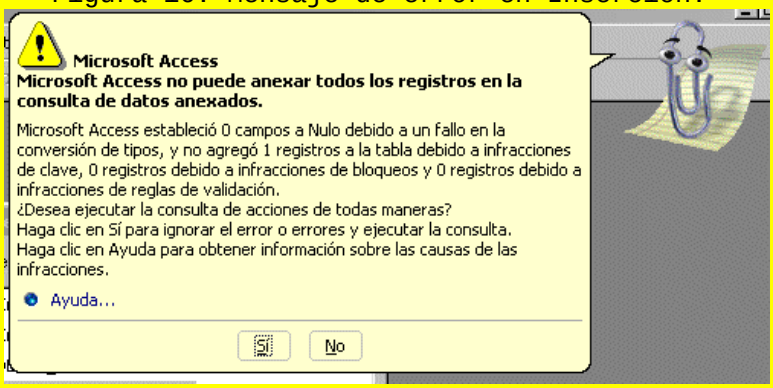


Figura 17. Mensaje de error en Eliminación de registros.

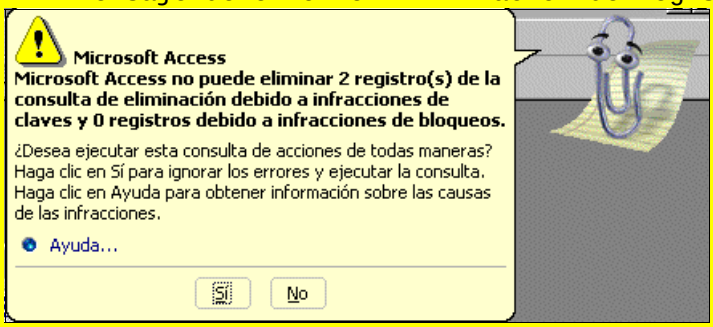
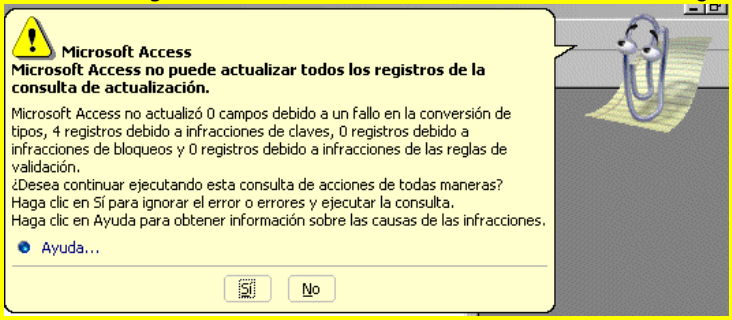


Figura 17. Mensaje de error en Actualización de registros.



Ejemplos:

```
SQL> INSERT INTO departamentos
        VALUES (30, 'PUBLICACIONES', 'SANTANDER');

     ERROR en línea 1:
     ORA-00001: restricción única
     (CURSOSQL.PK_DEPARTAMENTOS_DEP_NO) violada


SQL> INSERT INTO empleados
     VALUES (8674,'MARTINEZ', 'VENDEDOR', 7782, '28/01/00', 250000,
     130000, 36);

     ERROR en línea 1:
     ORA-02291: restricción de integridad
     (CURSOSQL.FK_EMP_DEP_NO) violada
      - clave padre no encontrada


SQL> INSERT INTO empleados
     VALUES (8674,'MARTINEZ', 'VENDEDOR', 7909, '28/01/00', 250000,
     130000, 30);

     ERROR en línea 1:
     ORA-02291: restricción de integridad (CURSOSQL.FK_EMP_DIRECTOR)
     violada
     - clave padre no encontrada
```