TEMA 4. LENGUAJE DE CONSULTA ESTRUCTURADO (SQL).

1. QUÉ ES SQL
2. CÓMO SE USA SQL
3. PARA QUÉ SIRVE SQL
4. TIPOS DE SENTENCIAS
    - 4.1. LENGUAJE DE MANIPULACIÓN DE DATOS (LMD)
    - 4.2. LENGUAJE DE DEFINICIÓN DE DATOS (LDD)

1. QUÉ ES SQL.

- SQL (Structured Query Language) es un lenguaje que permite expresar operaciones diversas sobre datos almacenados en bases de datos relacionales.
- SQL es un lenguaje de cuarta generación.
- Fue desarrollado por IBM y la versión original se denominaba SEQUEL.
- ANSI-SQL es el nombre que damos a una estandarización de las diversas implementaciones que de la evolución del lenguaje original inventado por IBM se han ido desarrollando posteriormente.

2. CÓMO SE USA.

Las peticiones sobre los datos se expresan en SQL mediante *sentencias*, que deben escribirse de acuerdo con las reglas sintácticas y semánticas de este lenguaje.

- **SQL INTERACTIVO**: Las sentencias pueden escribirse directamente en la pantalla de un terminal interactivo, en el cual también se recibe el resultado.
- **SQL INCORPORADO Y DINÁMICO:** Esto quiere decir que se pueden incorporar instrucciones de SQL en lenguajes de programación como: C++, C, Java, Cobol, Pascal y Fortran.

3. PARA QUÉ SIRVE

Permite:

1. Principalmente a los Programadores y a los Usuarios:
    - Insertar, modificar, borrar y consultar los datos contenidos en las tablas de la Base de Datos. (Lenguaje de Manipulación de Datos: **LMD**).
2. Realizar tareas propias del Administrador de la Base de Datos, como son:
    - Definición, modificación y destrucción de objetos de la Base de Datos, gestión de las autorizaciones de acceso, etc. (Lenguaje de Definición de Datos: **LDD** y Lenguaje de Control de Datos: **LCD**).

4. TIPOS DE SENTENCIAS.

4.1 LENGUAJE DE MANIPULACIÓN DE DATOS (LMD)

Las sentencias de esta parte de SQL permiten realizar consultas y mantenimiento de datos. Son las siguientes:

SELECT, INSERT, UPDATE y DELETE

Veamos la sintaxis de estas sentencias:
(Ver "Convenciones de sintaxis" en enlace Manual de Referencia de Transact-SQL)

SENTENCIA SELECT

Permite realizar consultas sobre una o varias tablas de la base de datos. La sintaxis completa de la instrucción SELECT es compleja, aunque las cláusulas principales se pueden resumir del modo siguiente:

SELECT [ALL | DISTINCT]

[TOP (número | porcentaje de las filas) [PERCENT]] **expresión** [,expresiones,…]

[FROM tabla1 [, tabla2,...] [JOIN…]]

[WHERE **predicados**]

[GROUP BY columna/s]

[HAVING condición]

[ORDER BY columna/s] [DESC];

Si tenemos una sentencia SELECT formada por las cláusulas anteriores, el resultado de ejecutarla se puede obtener mediante los pasos siguientes:

1. Ejecutar la cláusula FROM. Lo obtenido de momento es la tabla resultante de la sentencia.
2. Ejecutar la cláusula WHERE. Se eliminan todas las filas que no satisfagan el predicado.
3. Ejecutar la cláusula GROUP BY. Formar grupos con las filas de la tabla resultante en el paso anterior que tengan iguales valores en las columnas de agrupamiento.
4. Ejecutar la cláusula HAVING. Descartar los grupos que no satisfagan la condición especificada.
5. Ejecutar la cláusula SELECT. Esto implica evaluar sus expresiones para cada grupo, produciendo por cada uno de ellos una fila de la tabla resultante final, con tantos valores como expresiones. Si se ha especificado DISTINCT se eliminan las filas repetidas.
6. Ejecutar ORDER BY. Es decir presentar la tabla resultante clasificada por las columnas especificadas. Por defecto ascendentemente.
7. Si se ha limitado el número de filas, ejecutar TOP.

Consideraciones:

1. Una **expresión** es una combinación de operandos, operadores aritméticos y paréntesis (no todos ellos obligatorios) que cuando el SGBD la ejecuta produce un único resultado. Los operandos pueden ser nombres de columnas, constantes, *funciones* u otras expresiones. Como expresión también podemos usar *.

2. Las funciones pueden ser colectivas o escalares.

- Son ***funciones colectivas o de agregado***: SUM, MAX, MIN, AVG, COUNT. El resultado se obtiene a partir de una colección de valores. Representan un único valor que se obtiene aplicando una *operación a un conjunto de valores.* Un resultado por tabla o por grupo.
- *Son* ***funciones*** **escalares**: LEN, SUBSTRING, UPPER, LOWER. El resultado se obtiene a partir de un único valor (un resultado por fila). Al menos debéis saber localizar las de cadenas, matemáticas y las de fecha-hora.

(Ver "Operadores" en enlace Manual de Referencia de Transact-SQL: Elementos de lenguaje/Operadores/Visión general)

(Ver "Funciones" en el enlace Manual SQL w3schools: SQL Server Functions, y en enlace Manual de Referencia de Transact-SQL: Funciones)

Los siguientes ejemplos de tipos de expresiones se encuentran en "Ejemplos de tipos de expresiones.SQL". El archivo pertenece a EJEMPLOS_LMD.

```
--EJEMPLOS DE TIPOS DE EXPRESIONES QUE NOS PODEMOS ENCONTRAR:

USE empresa;

--SELECT expresión. Donde expresión es un operando (función).
SELECT GETDATE();

/*SELECT expresión. Donde expresión es un operando (función),
que tiene como parámetro
otra función */
SELECT FORMAT(GETDATE(),'d');

--SELECT * y usando la cláusula FROM.
SELECT *
FROM temple;

/*SELECT expresión1, expresión2 con cláusula FROM y cláusula
ORDER BY.
Donde expresión1 y expresión2 son operandos (nombres de
columnas)*/
SELECT  nomem, salar
FROM temple
ORDER BY nomem ASC;

--Igual si no ponemos ASC (valor por defecto).
SELECT  nomem, salar
FROM temple
ORDER BY nomem;

--Igual, pero ordenando descendentemente.
SELECT  nomem, salar
FROM temple
ORDER BY nomem DESC;

/*SELECT expresión1, expresión2 con cláusula FROM y cláusula TOP
para limitar el número de filas mostradas*/
--Del resultado de la consulta muestra las cinco primeras filas.
SELECT TOP(5) nomem, salar
FROM temple;
```

```sql
/*Dos ejemplos de SELECT con TOP y ORDER BY*/
--Del resultado de la consulta muestra las cinco primeras filas.
/*ORDER BY 1: 1 indica el lugar que ocupa la expresión del
SELECT por la que ordeno.*/
SELECT TOP(5) nomem, salar
FROM temple
ORDER BY 1;

--Del resultado de la consulta muestra el 50%.
SELECT TOP(50) PERCENT nomem, salar
FROM temple
ORDER BY 1;

--SELECT con la cláusula DISTINCT y ORDER BY
/*Así mostramos los salarios con las comisiones de los 14
empleados.*/
--ALL es el valor por defecto. Podemos no ponerlo.
SELECT ALL salar, comis
FROM temple
ORDER BY 1;
--Si solo queremos conocer los distintos salarios con sus
comisiones, me interesa que no se repitan las filas.
SELECT DISTINCT salar,comis
FROM temple
ORDER BY 1;

/*SELECT expresión1, expresión2, expresión3 con cláusula FROM.
Donde expresión1 y expresión2 son operandos (nombres de
columnas) y expresión3 está formada por operando (nombre de
columna), operador aritmético (producto) y operando (constante
numérica 2).*/
/*También usamos un alias de columna para darle nombre a una
columna.*/
SELECT nomem, salar, (salar * 2) AS 'DOBLE DEL SALARIO'
FROM temple;
```

```
/*SELECT expresión1, expresión2, expresión3 con cláusula FROM.
Donde expresión1 y expresión2 son operandos (nombres de
columnas) y expresión3 está formada por operando (nombre de
columna), operador aritmético (producto) y otra expresión
(10/100). A su vez, esta última expresión está formada
por operando (constante numérica 10), operador aritmético
(división) y operando (constante numérica 100).*/
/*También usamos un alias de columna para darle nombre a una
columna.*/
SELECT nomem, salar, (salar * 10/100) AS '10% DEL SALARIO'
FROM temple;

--FUNCIONES ESCALARES

/*SELECT expresión. Donde expresión es un operando (función
escalar).*/
/*También usamos un alias de columna para darle nombre a una
columna.*/
SELECT nomem, LEN(nomem) AS 'LONGITUD'
FROM temple;

SELECT fecna, YEAR(GETDATE()) - YEAR(fecna) AS 'AÑOS'
FROM temple;

--FUNCIONES COLECTIVAS O DE AGREGADO

/*SELECT expresión. Donde expresión es un operando (función de
agregado).*/
/*También usamos un alias de columna para darle nombre a una
columna.*/
SELECT AVG(salar) AS 'SALARIO MEDIO'
FROM temple;

--FUNCIÓN DE AGREGADO COUNT
/*
COUNT(nombre_columna) cuenta el número de veces que
el atributo tiene valor distinto de null en la tabla

COUNT(*) cuenta el número de filas de la tabla
*/
SELECT COUNT(extel)
FROM temple;

SELECT COUNT(DISTINCT extel)
FROM temple;
```

```sql
SELECT COUNT(*)
FROM temple;

SELECT COUNT(comis)
FROM temple;

SELECT COUNT(DISTINCT comis)
FROM temple;

SELECT COUNT(*)
FROM temple;
```

3. Cuando en la **cláusula FROM** se especifica una sola tabla se buscan los valores en ella. Cuando se pone más de una tabla separadas por coma se realiza el producto cartesiano de estas y sobre ese resultado se buscan los datos. También puede aparecer la **cláusula JOIN**, en ese caso se realiza una concatenación de las tablas sobre el campo que se le indique.

Los siguientes ejemplos de la cláusula FROM se encuentran en “Ejemplos FROM1.SQL”. El archivo pertenece a EJEMPLOS_LMD.

```
/*En estos ejemplos vemos que en la cláusula FROM puede aparecer:
- Una sola tabla.
- Varias tablas separadas por "coma". Entonces, tendremos el producto
cartesiano de dichas tablas.
- Varias tablas usando la cláusula JOIN. Entonces, tendremos la concatenación
  de dichas tablas mediante el campo indicado.
*/
USE ejemploempresa;

SELECT * FROM departamento;
SELECT * FROM empleado;

--PRODUCTO CARTESIANO.
SELECT *
FROM departamento, empleado;
SELECT *
FROM empleado, departamento;

--PRODUCTO CARTESIANO Y SELECCIÓN DE FILAS.
--Uso de alias de tablas.
SELECT *
FROM departamento d, empleado e
WHERE d.numde=e.numde;

SELECT e.nomem,d.nomde
FROM departamento d, empleado e
WHERE d.numde=e.numde;

SELECT *
FROM empleado e, departamento d
WHERE e.numde=d.numde;

SELECT e.nomem,d.nomde
FROM empleado e, departamento d
WHERE e.numde=d.numde;

--CONCATENACIÓN: USO DE LA CLÁUSULA JOIN.
SELECT *
FROM departamento d JOIN empleado e ON (d.numde=e.numde);

SELECT *
FROM empleado e JOIN departamento d ON (e.numde=d.numde);
```

```sql
SELECT e.nomem,d.nomde
FROM departamento d JOIN empleado e ON (d.numde=e.numde);

SELECT e.nomem,d.nomde
FROM empleado e JOIN departamento d ON (e.numde=d.numde);
```

4. Un **predicado** expresa una condición entre valores, y según estos pueden resultar Verdadero o Falso. Los predicados pueden ser simples o compuestos.

PREDICADOS SIMPLES

Predicados Básicos

Expresan condiciones de comparación entre dos valores. Tienen esta forma:

"ELEMENTO DE COMPARACIÓN_1" Operador_Relacional "ELEMENTO DE COMPARCIÓN_2"

Operador_Relacional (operadores de comparación) puede ser: =, <>, >, <, >=, <=

Los "ELEMENTOS DE COMPACIÓN" pueden ser expresiones. El "ELEMENTO DE COMPARCIÓN_2" puede ser a su vez el resultado de una sentencia SELECT, que **debe ir entre paréntesis** y **producir un único valor**. A las sentencias SELECT incluidas dentro de otra sentencia SELECT se le llaman ***sentencias subordinadas o subselects***.

Los siguientes ejemplos de predicados básicos se encuentran en "Ejemplos predicados básicos.SQL". El archivo pertenece a EJEMPLOS_LMD.

```
USE ejemploempresa;

/*Ejemplo de predicado básico donde los dos elementos de
comparación son expresiones.
numde es operando (nombre de columna) y 2 es operando (constante
numérica)*/
SELECT nomem
FROM empleado
WHERE numde > 2;

/*Ejemplo de predicado básico donde el segundo elemento de
comparación es un SELECT: Queremos saber el nombre de los
empleados que están en un departamento con un código mayor que
el código del departamento de Dep2*/
SELECT * FROM empleado;
SELECT numde FROM departamento WHERE nomde='Dep2';

--SOLUCIÓN:
SELECT nomem
FROM empleado
WHERE numde > (SELECT numde FROM departamento WHERE
nomde='Dep2');
```

```sql
--También se podría haber hecho con producto cartesiano.
SELECT *
FROM empleado e,departamento d;

--SOLUCIÓN:
SELECT nomem
FROM empleado e,departamento d
WHERE d.nomde='Dep2' AND e.numde>d.numde;

--También se podría haber hecho con concatenación o JOIN.
SELECT *
FROM empleado e JOIN departamento d ON (e.numde>d.numde);

--SOLUCIÓN:
SELECT nomem
FROM empleado e JOIN departamento d ON (e.numde>d.numde)
WHERE d.nomde='Dep2';
```

Predicado NULL

Su formato es: nombre_columna IS [NOT] NULL

Los siguientes ejemplos de predicados NULL se encuentran en “Ejemplos predicado NULL.SQL”. El archivo pertenece a EJEMPLOS_LMD.

```
USE empresa;

SELECT * FROM temple;

/*Obtiene los nombres de los empleados cuya comisión es nula.
El campo comis tiene el valor NULL.
*/
SELECT nomem
FROM temple
WHERE comis IS NULL;

/*Obtiene los nombres de los empleados cuya comisión no es nula.
El campo comis tiene un valor distinto de NULL.
*/
SELECT nomem
FROM temple
WHERE comis IS NOT NULL;
```

Predicados Cuantificados

Cuando se usa una sentencia subordinada en un predicado de comparación, su resultado debe ser un valor único, sin embargo se admite que el resultado tenga varios valores **(una única columna, pero varias filas)** si la sentencia subordinada va precedida de algunas de las palabras cuantificadoras: ANY o ALL.

Si se usa ANY . El predicado cuantificado es verdadero si la comparación es verdadera para alguno de los valores resultantes en la sentencia subordinada.

Los siguientes ejemplos de predicados cuantificados (ANY) se encuentran en "predicado ANY.SQL". El archivo pertenece a EJEMPLOS_LMD.

```
USE EMPRESA

/*Obtener el número de los empleados (su identificador único)
que ganen más que alguno de los empleados del departamento 110*/

--Vemos todos los empleados.
SELECT * FROM temple;

--Vemos los datos que nos interesa de los empleados del
departamento 110.
SELECT numem, salar, numde
FROM temple
WHERE numde=110;

--Estos son los empleados cuyo salario es mayor que 1500 o mayor
que 1800.
SELECT *
FROM temple
WHERE salar > ANY(SELECT salar
                  FROM temple
                  WHERE numde=110);
```

```sql
/*Si no queremos que salgan los empleados del departamento 110
vamos a poner un predicado compuesto utilizando el operador
lógico AND*/
SELECT *
FROM temple
WHERE (salar > ANY(SELECT salar
                   FROM temple
                   WHERE numde=110) ) AND numde <> 110;

--SOLUCIÓN:
SELECT numem
FROM temple
WHERE (salar > ANY(SELECT salar
                   FROM temple
                   WHERE numde=110) ) AND numde <> 110;

--OTRAS FORMAS DE HACER EL EJEMPLO:

/*Si el salario es mayor que el más pequeño, entonces lo será
para los dos*/
SELECT numem
FROM temple
WHERE (salar > (SELECT MIN (salar) FROM temple WHERE numde=110)
) AND numde <> 110;

/*SELF JOIN: Se llama SELF JOIN cuando hacemos un producto
cartesiano o una concatenación de una tabla consigo misma.*/

--Con producto cartesiano.
SELECT E1.numem ,E1.numde,E1.salar,E2.numem,E2.numde,E2.salar
FROM temple E1, temple E2
WHERE (E1.salar>E2.salar) AND E2.numde=110 AND E1.numde<>110;

/*Tenemos que poner DISTINCT para que no salgan dos veces los
empleados
como el 150 cuyo salario es 2200 y cumple que es mayor que
1500 y mayor que 1800*/
SELECT DISTINCT E1.numem
FROM temple E1, temple E2
WHERE (E1.salar>E2.salar) AND E2.numde=110 AND E1.numde<>110;

--Con JOIN
SELECT DISTINCT E1.numem
FROM temple E1 JOIN temple E2 ON (E1.salar>E2.salar)
WHERE  E2.numde=110 AND E1.numde<>110;
```

Si se usa ALL . El predicado cuantificado es verdadero si la comparación es verdadera para todos y cada uno de los valores resultantes en la sentencia subordinada.

Los siguientes ejemplos de predicados cuantificados (ALL) se encuentran en "predicado ALL.SQL". El archivo pertenece a EJEMPLOS_LMD.

```
USE EMPRESA

/*Obtener el número de los empleados (su identificador único)
que ganen más que cualquiera de los empleados del departamento
110*/

--Vemos todos los empleados.
SELECT * FROM temple;

--Vemos los datos que nos interesa de los empleados del
departamento 110.
SELECT numem, salar, numde
FROM temple
WHERE numde=110;

/*Estos son los empleados cuyo salario es mayor que 1500 y mayor
que 1800. */
SELECT *
FROM temple
WHERE salar > ALL (SELECT salar
                   FROM temple
                    WHERE numde=110);

/*En este caso no haría falta eliminar los empleados del
departamento 110, puesto que es un mayor estricto.
Sí habría que eliminarlo si fuera mayor o igual. */
--SOLUCIÓN:
SELECT numem
FROM temple
WHERE (salar > ALL (SELECT salar
                          FROM temple
                       WHERE numde=110) );
```

```sql
--OTRAS FORMAS DE HACER EL EJEMPLO:

/*Si es mayor que el más grande, entonces lo será para todos*/
SELECT numem
FROM temple
WHERE salar > (SELECT MAX(salar) FROM temple WHERE numde=110);

/*SELF JOIN:*/
--Producto cartesiano.
SELECT E1.numem,E1.salar,E2.numem,E2.salar, E2.numde
FROM temple E1, temple E2
WHERE  E2.numde=110 AND (E1.salar> (SELECT MAX(salar) FROM
temple WHERE numde=110) );

/*Eliminamos las repeticiones*/
SELECT DISTINCT E1.numem
FROM temple E1, temple E2
WHERE  E2.numde=110 AND (E1.salar> (SELECT MAX(salar) FROM
temple WHERE numde=110) );
```

Predicado BETWEEN

Su formato es: Expresion1 [NOT] BETWEEN Expresion2 AND Expresion3

Sirve para determinar si un valor está comprendido entre otros dos.

Los siguientes ejemplos de predicados BETWEEN se encuentran en “predicado BETWEEN.SQL”. El archivo pertenece a EJEMPLOS_LMD.

```
USE EMPRESA

/*Obtener los empleados cuyo salario estén comprendido entre
1000 y 1600 ambos inclusive.*/
--Lo que nos pide es que: salario >= 1000 AND salario <= 1600.

--Vemos todos los empleados.
SELECT * FROM temple;

SELECT *
FROM temple
WHERE salar BETWEEN 1000 AND 1600;

--Si no usamos el predicado BETWEEN, se podría hacer así.
SELECT *
FROM temple
WHERE salar >= 1000 AND salar<=1600;

/*Obtener los empleados cuyo salario sea menor que 1000 o bien
mayor que 1600. */
--Lo que nos pide es que: salario < 1000 OR salario > 1600.
/*Al ser lo contrario de BETWEEN, podemos usar el predicado NOT
BETWEEN.*/

SELECT *
FROM temple
WHERE salar NOT BETWEEN 1000 AND 1600;

--Si no usamos el predicado BETWEEN, se podría hacer así.
SELECT *
FROM temple
WHERE salar < 1000 OR salar> 1600;
```

Predicado LIKE

Su formato es: Nombre_columna [NOT] LIKE constante alfanumérica.

Sirve para buscar combinaciones de caracteres, se pueden usar comodines.

(Ver SQL Like en el enlace Manual SQL w3schools, y ver "LIKE" en enlace Manual de Referencia de Transact-SQL)

Carácter comodín	Descripción	Ejemplo
`%`	Cualquier cadena de cero o más caracteres.	`WHERE title LIKE '%computer%'` busca todos los títulos de libros que tengan la palabra `computer` en cualquier parte del título.
`_` (subrayado)	Cualquier carácter individual.	`WHERE au_fname LIKE '_ean'` busca todos los nombres de cuatro letras que finalicen con `ean` (`Dean`, `Sean` y así sucesivamente).
`[ ]`	Cualquier carácter individual que se encuentre en el intervalo `[a-f]` o el conjunto `[abcdef]` que se haya especificado.	`WHERE au_lname LIKE '[C-P]arsen'` busca apellidos de autores que terminen por `arsen` y que empiecen por cualquier carácter individual entre `C` y `P`, por ejemplo, `Carsen`, `Larsen`, `Karsen` y así sucesivamente. En las búsquedas de intervalos, los caracteres incluidos en el intervalo pueden variar, dependiendo de las reglas de ordenación de la intercalación.
`[^]`	Cualquier carácter individual que no se encuentre en el intervalo `[^a-f]` o el conjunto `[^abcdef]` que se haya especificado.	`WHERE au_lname LIKE 'de[^l]%'` busca todos los apellidos de autores que empiecen por `de` y en los que la letra siguiente no sea `l`.

Los siguientes ejemplos de predicados LIKE se encuentran en “predicado LIKE.SQL”. El archivo pertenece a EJEMPLOS_LMD.

```
USE EMPRESA

/*Obtener el número del departamento (su identificador único)
Sector Servicios*/
--Vemos todos los empleados.
SELECT * FROM tdepto;

--SOLUCIÓN: Usando un predicado básico.
SELECT numde
FROM tdepto
WHERE nomde = 'Sector Servicios';

/*Obtener el número de los departamentos (su identificador
único) cuyo nombre de departamento empiece por la palabra
Sector*/
SELECT numde
FROM tdepto
WHERE nomde LIKE 'Sector %';
```

Predicado IN

Su formato es:

Expresión [NOT] IN (constante1, constante2, ...)

Sirve para preguntar si el resultado de una expresión está incluido en esa lista. En vez de una lista podemos poner una sentencia SELECT subordinada. Tendrá esta forma:

Expresión [NOT] IN (Subselect) equivale a Expresión = ANY (Subselect)

Los siguientes ejemplos de predicados IN se encuentran en "predicado IN.SQL". El archivo pertenece a EJEMPLOS_LMD.

```
USE EMPRESA

/*Obtener los datos de los empleados que ganen igual que
alguno de los empleados del departamento 111*/

--Vemos todos los empleados.
SELECT * FROM temple;

/*Vemos los datos que nos interesa de los empleados del
departamento 111.*/
SELECT numem,numde,salar
FROM temple
WHERE numde=111
ORDER BY 2;

--SOLUCIÓN:
--Si conocemos la información de la lista, la podemos poner.
SELECT *
FROM temple
WHERE salar IN (2200,1800) AND numde <> 111;

--SOLUCIÓN:
/*Si no conocemos lo que ganan los empleados del departamento
111, tendremos que usar IN con un subselect*/
SELECT *
FROM temple
WHERE (salar IN (SELECT salar
                 FROM temple
                 WHERE numde=111) ) AND numde <> 111;
```

```
/*El resultado es equivalente a utilizar un predicado ANY
con el operador relacional = .*/
SELECT *
FROM temple
WHERE (salar = ANY(SELECT salar
                   FROM temple
                   WHERE numde=111) ) AND numde <> 111;

--OTRAS FORMAS DE HACER EL EJEMPLO:

/*SELF JOIN:*/
--Producto cartesiano.
SELECT E1.*
FROM temple E1, temple E2
WHERE E2.numde=111 AND E1.numde<>111 AND E2.salar=E1.salar;

--JOIN.
SELECT E1.*
FROM temple E1 JOIN temple E2 ON (E2.salar=E1.salar)
WHERE E2.numde=111 AND E1.numde<>111;
```

Predicado EXISTS

Su formato es: [NOT] EXISTS (Subselect)

Es verdadero si el resultado de la sentencia subordinada contiene una o más filas.

Los siguientes ejemplos de predicados EXISTS se encuentran en “predicado EXISTS.SQL”. El archivo pertenece a EJEMPLOS_LMD.

```
USE empresa;
/*
   EXISTS     = TRUE  si el subselect devuelve 1 o más filas
   EXISTS     = FALSE si el subselect devuelve 0 filas

   NOT EXISTS = TRUE  si el subselect devuelve 0 filas
   NOT EXISTS = FALSE si el subselect devuelve 1 o más filas
*/

/*Si existe algún departamento, entonces decir cuántos empleados
tiene la empresa.*/
SELECT COUNT(*)
FROM temple
WHERE EXISTS (SELECT * FROM tdepto);

/*Si todos los directores son en propiedad, entonces decir
cuántos empleados
tiene la empresa*/
--Con EXISTS
SELECT COUNT(*)
FROM temple
WHERE EXISTS (SELECT *
                  FROM tdepto
                  WHERE 'P' = ALL (SELECT TIDIR FROM tdepto)
                  );

--Con NOT EXISTS
/*Si todos los directores son en propiedad, entonces decir
cuántos empleados tiene la empresa, es lo mismo que decir: que
si no existe ningún director en funciones, entonces decir
cuántos empleados tiene la empresa*/
SELECT COUNT(*)
FROM temple
WHERE NOT EXISTS (SELECT * FROM tdepto WHERE tidir='F');
```

```
/*Si queremos que cuando la respuesta sea 0 salga un mensaje,
lo podríamos hacer así:*/
SELECT IIF(COUNT(*)= 0,'No todos los directores son en
propiedad',CONVERT(VARCHAR(4),COUNT(*)))
FROM temple
WHERE NOT EXISTS (SELECT * FROM tdepto WHERE tidir='F');

--OTRA FORMA DE HACER EL EJEMPLO:
--Usamos la cláusula CASE.
SELECT CASE WHEN (SELECT COUNT(*) FROM tdepto) =
                 (SELECT COUNT(*) FROM tdepto WHERE tidir='P')
                THEN CONVERT(VARCHAR(4),COUNT(*))
            ELSE 'No todos los directores son en propiedad'
            END
FROM temple;

/*Obtener el nombre de los centros que tienen al menos un
departamentos con presupuestos inferior a 3000 euros*/
--Vemos todos los departamentos y centros.
SELECT * FROM tcentr;
SELECT * FROM tdepto ORDER BY numce;

SELECT *
FROM tcentr c
WHERE EXISTS (SELECT *
              FROM tdepto d
              WHERE d.numce=c.numce AND d.presu<3000
             );

--SOLUCIÓN:
SELECT nomce
FROM tcentr c
WHERE EXISTS (SELECT *
              FROM tdepto d
              WHERE d.numce=c.numce AND d.presu<3000
             );

--OTRA FORMA DE HACER EL EJEMPLO:
--Con el predicado ANY.
SELECT *
FROM tcentr c
WHERE numce = ANY (SELECT numce
                   FROM tdepto
                   WHERE  presu<3000
                  );
```

```sql
--Con el predicado IN.
SELECT nomce
FROM tcentr C
WHERE numce IN (SELECT numce
                FROM tdepto
                WHERE presu<3000
               );

--Con JOIN.
/*En este caso hay que poner la cláusula DISTINCT, porque en el
caso de que hubiera más de un departamento en el mismo centro
que cumpliera el WHERE, el nombre del centro se repetiría.
Prueba con presu<=3000
*/
SELECT DISTINCT nomce
FROM tcentr c JOIN tdepto D ON (C.numce=d.numce)
WHERE presu<3000;

/* Obtener el nombre de los departamentos que no tienen
empleados con sueldos inferiores a 1500 euros*/
--Vemos todos los departamentos y empleados.
SELECT * FROM tdepto;
SELECT * FROM temple ORDER BY numde;

SELECT *
FROM tdepto d
WHERE NOT EXISTS(SELECT *
                 FROM temple e
                 WHERE e.numde=d.numde AND salar < 1500
                );

--SOLUCIÓN:
SELECT nomde
FROM tdepto d
WHERE NOT EXISTS(SELECT *
                 FROM temple e
                 WHERE e.numde=d.numde AND salar < 1500
                 );
```

```
--OTRAS FORMAS DE HACER EL EJEMPLO:
--Con predicado ALL
SELECT nomde
FROM tdepto
WHERE numde <> ALL(SELECT numde
                   FROM temple
                   WHERE salar<1500);

--Con Predicado IN
SELECT nomde
FROM tdepto
WHERE numde NOT IN (SELECT numde
                    FROM temple
                    WHERE salar<1500);
```

PREDICADOS COMPUESTOS

Son combinaciones de otros predicados simples o compuestos, con los operadores lógicos AND, OR o NOT.

5. La **cláusula GROUP BY** indica que se han de agrupar filas de la tabla, de modo que todas las que tengan iguales valores en las columnas de agrupamiento formen un grupo.

6. La **cláusula HAVING** sirve para descartar grupos de filas.

Los siguientes ejemplos de la cláusula GROUP BY se encuentran en Ejemplos GROUP BY y HAVING.SQL". El archivo pertenece a EJEMPLOS_LMD.

```
USE empresa;

/*CUANDO USAMOS LA CLÁUSULA GROUP BY, LA EXPRESIÓN QUE VA JUNTO
AL SELECT SOLO ADMITE COMO EXPRESIÓN EL/LOS NOMBRE/S
DE COLUMNA/S UTILIZADA/S PARA AGRUPAR Y FUNCIONES
DE AGREGADO.*/

/*EN LA CLÁUSULA HAVING NOS ENCONTRAMOS NORMALMENTE:
FUNCIÓN DE AGREGADO OP_RELACIONAL EXPRESIÓN.

RECUERDA QUE EXPRESIÓN PUEDE SER UN SUBSELECT. Y SI EL SUBSELECT
PRODUCE MÁS DE UNA FILA, PUEDES PONER LAS CLÁUSULAS ALL O ANY.
TAMBIÉN PODEMOS TENER UN PREDICADO PREDICADO IN*/
```

```sql
/*Listar para cada departamento, su número (numde) y el máximo
salario que se gana dentro de él.*/
--Vemos los datos que nos interesa de los empleados.
SELECT numem,numde,salar
FROM temple
ORDER BY 2;

--SOLUCIÓN:
SELECT numde, MAX(salar) AS 'Máximo salario del departamento'
FROM temple
GROUP BY numde;

/*Igual, pero indicar también cuántos empleados hay en cada
departamento. La consulta es solo para los departamentos con
código superior a 100*/
SELECT numde, COUNT(*) AS 'Número de empleados del dpto.'
,MAX(salar)as 'Máximo salario del dpto.'
FROM temple
WHERE numde > 100
GROUP BY numde;

--Listar para cada salario cuántos empledos los ganan.
SELECT salar, COUNT(*) AS 'Número de empleados con el mismo
salario'
FROM temple
GROUP BY salar;

/*Listar para los departamentos con más de dos empleados,
su número (numde) y el máximo salario.*/
SELECT numde, MAX(salar) AS 'Máximo salario del departamento'
FROM temple
GROUP BY numde
HAVING COUNT(*)>2
ORDER BY 1;
```

```sql
--USAMOS TODAS LAS CLÁUSULAS:

/*Listar para cada departamento distinto del 112, su número
(numde) y el máximo salario que se gana dentro de él.
Teniendo en cuenta que el departamento debe tener más
de dos empleados. */
SELECT numde, MAX(salar)
FROM temple
WHERE numde <> 112
GROUP BY numde
HAVING COUNT(*)>2
ORDER BY 1;

--EJEMPLO DE HAVING CON SUBSELECT:

/*Igual que el anterior, pero teniendo en cuenta que el
departamento debe tener más empleados que los que
tiene el departamento 110 */
SELECT numde, MAX(salar),COUNT(*)
FROM temple
WHERE numde <> 112
GROUP BY numde
HAVING COUNT(*)> (SELECT COUNT(*) FROM temple WHERE numde=110)
ORDER BY 1;

--TAMBIÉN PODEMOS AGRUPAR POR MÁS DE UNA COLUMNA.

/*Obtener en cada departamento cuántas personas
hay con el mismo número de hijos*/
--Vemos los datos que nos interesa de los empleados.
SELECT numem, numde,numhi
FROM temple
ORDER BY numde,numhi;

/*Aquí estamos agrupando por número de departamento y dentro de
cada grupo (cada departamento), por número de hijos
(igual número de hijos) */
SELECT numde,numhi,COUNT(*) AS 'número de elementos del grupo'
FROM temple
GROUP BY numde,numhi
ORDER BY 1;
```

7. La **cláusula ORDER BY** presenta la tabla resultante final clasificada por las columnas indicadas. Pero además, puede indicar cuántas filas se deben visualizar. Esto último ya lo hacía la cláusula TOP, pero en los siguientes ejemplos veremos que con ORDER BY podemos especificar más detalladamente las filas que queremos mostrar.

Los siguientes ejemplos de la cláusula ORDER BY se encuentran en Ejemplos ORDER BY.SQL". El archivo pertenece a EJEMPLOS_LMD.

```
USE empresa;

--Vemos la tabla de empleado.
SELECT *
FROM temple
ORDER BY nomem;

--Con TOP puedo:
--Obtener el nombre de los primeros cinco empleados.
SELECT TOP(5) nomem
FROM temple
ORDER BY nomem;

--Obtener el nombre de los últimos cinco empleados.
SELECT TOP(5) nomem
FROM temple
ORDER BY nomem DESC;

/*Pero NO puedo obtener por ejemplo, el nombre de los últimos
cinco empleados ordenados ascendentemente.*/

/*La cláusula ORDER BY, cuya sintaxis se especifica a
continuación, sí nos va a permitir obtener
la consulta anterior*/

/*
 ORDER BY order_by_expression
    [ ASC | DESC ] [ , ...n ]
    [ OFFSET { integer_constant | offset_row_count_expression }
          { ROW | ROWS }
    [ FETCH { FIRST | NEXT } {integer_constant |
          fetch_row_count_expression } { ROW | ROWS } ONLY]
      ]
OFFSET: especifica el número de filas que se deben omitir antes
de que
```

```
comience a devolver filas la expresión de consulta. El valor
puede ser
una constante entera o una expresión mayor o igual a cero.

FETCH: especifica el número de filas que se devolverán después
de que se
haya procesado la cláusula OFFSET. El valor puede ser una
constante entera
o una expresión mayor o igual a uno.
*/

/*Practicamos unos cuantos ejemplos para entender la cláusula
y después resolveremos el problema anterior.*/

--Ejemplos:
/*Por orden alfabético, obtener el nombre y salario de cuatro
empleados a partir de tercero. */
SELECT nomem, salar
FROM temple
ORDER BY nomem
OFFSET 3 ROWS
FETCH NEXT 4 ROWS ONLY;

/*Obtener el nombre y salario del undécimo empleado (teniendo en
cuenta el orden alfabético).*/
SELECT nomem,salar
FROM temple
ORDER BY nomem
OFFSET 10 ROW
FETCH FIRST 1 ROW ONLY;

/*Por orden alfabético obtener el nombre y salario de la segunda
mitad de los empleados.*/
SELECT nomem,salar
FROM temple
ORDER BY nomem
OFFSET (SELECT COUNT(*)/2 FROM temple) ROWS
FETCH NEXT (SELECT COUNT(*)/2 FROM temple) ROWS ONLY;

--Nota la diferencia con el resultado de las siguientes dos
consultas:
SELECT TOP(SELECT COUNT(*)/2 FROM TEMPLE) nomem,salar
FROM temple
ORDER BY nomem DESC;
```

```
--O bien, esta:
SELECT TOP(50) PERCENT nomem,salar
FROM temple
ORDER BY nomem DESC;

--Para finalizar, vamos a resolver la consulta que no pudimos
resolver con TOP:
--Obtener el nombre de los últimos cinco empleados ordenados
ascendentemente.
SELECT nomem
FROM temple
ORDER BY nomem ASC
OFFSET ((SELECT COUNT(*) FROM temple)- 5) ROWS
FETCH NEXT 5  ROWS ONLY;

/*Puesto que FETCH no es obligatorio, si no lo ponemos nos
muestra todas las filas desde el desplazamiento hasta el final
del resultado de la consulta*/
SELECT nomem
FROM temple
ORDER BY nomem ASC
OFFSET ((SELECT COUNT(*) FROM temple)- 5) ROWS;
```

8. Tipos de **JOIN: LEFT JOIN, RIGHT JOIN Y FULL JOIN**

Para entender cuándo debemos utilizar estos tipos de JOIN, veamos los siguientes ejemplos que se encuentran en FROM2.SQL". El archivo pertenece a EJEMPLOS_LMD.

```
USE ejemploempresa;

--Veamos las tablas de departamento y empleados.
SELECT * FROM departamento;
SELECT * FROM empleado;

/*Preparamos las tablas poniendo los departamentos
de los empleados 4 y 5 a NULL. Así esos dos empleados
no tienen departamento y a su vez el departamento 2 no tiene
empleados*/
```

```
/*Ejecutemos las tres siguientes consultas, observando que me
dan el mismo resultado.*/
SELECT *
FROM departamento D, empleado E
WHERE D.numde=E.numde;

/*JOIN Y INNER JOIN SON EQUIVALENTES*/
SELECT *
FROM departamento D JOIN empleado E ON (D.numde=E.numde);

SELECT *
FROM departamento D INNER JOIN empleado E ON (D.numde=E.numde);

/*Observemos que al juntar las dos tablas, ya sea con producto
cartesiano o con JOIN, el departamento Dep2 no aparece, porque
su numde no aparece nunca en la tabla de empleado. A su vez, los
empleados 4 y 5 tampoco aparecen porque su numde están a NULL.
Por tanto:

-Si queremos obtener TODOS los departamentos con sus empleados,
pero queremos que aparezcan también los departamentos que no
tienen empleados, debemos utilizar LEFT JOIN o RIGHT JOIN.

-Si queremos obtener TODOS los empleados con el departamento al
que pertenecen, pero queremos que aparezcan también los
empleados que de momento no tienen asignado departamento,
debemos utilizar LEFT JOIN o RIGHT JOIN.

-Si queremos obtener TODOS los departamentos con sus empleados,
pero queremos que aparezcan también los departamentos que no
tienen empleados, y además queremos obtener TODOS los empleados
con el departamento al que pertenecen, pero queremos que
aparezcan también los empleados que de momento no tienen
asignado departamento, debemos utilizar FULL JOIN.
*/
--Veamos ejemplos:

/*Queremos sacar el nombre de los departamentos, junto con el
nombre de los empleados que tienen. Si el departamento aún no
tiene empleados, este debe salir igualmente.
LEFT JOIN Y LEFT OUTER JOIN SON EQUIVALENTES
 */
SELECT nomde,nomem
FROM departamento D LEFT OUTER JOIN  empleado E ON
(D.numde=E.numde);
```

```sql
SELECT nomde,nomem
FROM departamento D LEFT OUTER JOIN  empleado E ON
(D.numde=E.numde);

/*PARA OBTENER ESTE MISMO RESULTADO, TAMBIÉN PODEMOS
 INTERCAMBIAR LAS TABLAS Y PONER RIGHT EN VEZ DE LEFT*/
SELECT nomde,nomem
FROM  empleado E RIGHT OUTER JOIN  departamento D ON
(D.numde=E.numde);

/* Queremos sacar el nombre de los empleados, junto con el
nombre del departamento en el que está. Si el empleado aún no
tiene asignado departamento, este debe salir igualmente.
RIGHT JOIN Y RIGHT OUTER JOIN SON EQUIVALENTES*/
SELECT nomem,nomde
FROM departamento D RIGHT JOIN  empleado E ON (D.numde=E.numde);

SELECT nomem,nomde
FROM departamento D RIGHT OUTER JOIN  empleado E ON
(D.numde=E.numde);

/*PARA OBTENER ESTE MISMO RESULTADO, TAMBIÉN PODEMOS
 INTERCAMBIAR LAS TABLAS Y PONER LEFT EN VEZ DE RIGTH*/
SELECT nomem,nomde
FROM empleado E LEFT OUTER JOIN  departamento D ON
(D.numde=E.numde);

/* Queremos sacar todos los departamentos aunque no tengan
empleados y además queremos sacar a todos los empleados aunque
no tengan asignado departamento.
FULL JOIN Y FULL OUTER JOIN SON EQUIVALENTES
*/
SELECT nomde,nomem
FROM departamento D FULL JOIN  empleado E ON (D.numde=E.numde);

SELECT nomde,nomem
FROM departamento D FULL OUTER JOIN  empleado E ON
(D.numde=E.numde);
```

SENTENCIA INSERT INTO

Permite añadir una o más filas completas a una tabla.

INSERT [INTO] tabla [(col1,col2, ...)]
VALUES (valor1, valor2,...);

SENTENCIA DELETE FROM

Permite borrar una o más filas completas de una tabla. Borra las filas que cumplan el predicado.

DELETE [FROM] tabla
[WHERE predicado];

SENTENCIA UPDATE

Permite modificar o actualizar varias filas de la tabla. En concreto, permite actualizar o modificar algunas columnas de las filas de una tabla. Modifica las filas que cumplan el predicado.

UPDATE tabla
SET col1 = expresion1 [,col2 = expresion2]
[WHERE predicado];

Veamos ejemplos simples de las sentencias INSERT INTO, DELETE FROM y UPDATE que se encuentran en Ejemplos INSERT-DELETE-UPDATE.SQL". El archivo pertenece a EJEMPLOS_LMD.

```
USE empresa;

--EJEMPLOS DE INSERT.

--Vemos la tabla de centros.
SELECT * FROM tcentr;

--Añadimos una sola fila.
INSERT INTO tcentr (numce,nomce,señas)
VALUES (30,'ALMACEN1','C.ALM1');

/*Si introducimos valores en todos los campos
y en el mismo orden, no es necesario poner los nombres
de las columnas.*/
INSERT INTO tcentr
VALUES (40,'ALMACEN1',NULL);

--La siguiente sentencia daría error.
INSERT INTO tcentr
VALUES (50,'ALMACEN1');
```

```
--Añadimos dos filas en una sola sentencia INSERT INTO.
INSERT INTO tcentr
VALUES (50,'ALMACEN1','C.ALM1'),
       (60,'ALMACEN1','C.ALM1');

INSERT  tcentr (numce,nomce,señas)
VALUES (70,'ALMACEN2','C.ALM2'),
       (80,'ALMACEN3','C.ALM3');

/*Esta sentencia es correcta porque 90 se almacena en numce y
'ALMACEN4' en nomce. El campo señas al admitir nulos, se rellena
con NULL.*/
INSERT INTO tcentr (numce, nomce)
VALUES (90,'ALMACEN4');

--EJEMPLOS DE DELETE.

--Vemos la tabla de centros.
SELECT * FROM tcentr;

/*Borrar el centro 60. Observa que FROM es opcional, pero se
recomienda ponerlo porque es el estándar*/
DELETE tcentr
WHERE numce=60;

--Borrar los centros con número mayor que 20 y menor que 100
DELETE FROM tcentr
WHERE numce>20 AND numce<100

--EJEMPLOS DE UPDATE.

--Vemos la tabla de centros.
SELECT * FROM tcentr;


--Modificar la dirección y el nombre del centro número 10.
UPDATE tcentr
SET señas='C.ALCALÁ ,821, MADRID',nomce='NUEVA SEDE CENTRAL'
WHERE numce=10;
```

Veamos ejemplos de las sentencias DELETE FROM y UPDATE con subconsultas y con JOIN. Además en estos ejemplos, se introduce el concepto de TRANSACCIÓN. El archivo Ejemplos DELETE y UPDATE con subconsultas y JOIN.SQL pertenece a EJEMPLOS_LMD.

Una transacción es un conjunto de sentencias que, o bien se ejecutan todas (tendremos que confirmarlo con COMMIT), o bien se deshacen todas (haremos un ROLLBACK). Una sola sentencia es por sí misma una transacción.

```
USE empresa;

/*Preparamos las tablas, para ello añadimos un departamento
nuevo y un empleado que pertenezca al mismo.*/
INSERT INTO tdepto(numde,numce,direc,tidir,presu,depde,nomde)
VALUES (200,10,130,'F',1,121,'NUEVO');

INSERT INTO
temple(numem,numde,extel,fecna,fecin,salar,comis,numhi,nomem)
VALUES
(400,200,350,'12/03/1980','8/1/2019',1500,100,0,'VEGA,SANDRA');

--Vemos las tablas de departamentos y empleados.
SELECT * FROM tdepto;
SELECT * FROM temple;

--EJEMPLO DE DELETE FROM CON SUBCONSULTA:

--Borrar todos los empleados del departamento llamado "Nuevo".
DELETE FROM temple
WHERE numde = (SELECT numde FROM tdepto WHERE nomde LIKE
'NUEVO');

--EJEMPLO UPDATE CON SUBCONSULTA EN EL WHERE:

/*A todos los empleados del departamento de personal subir el
sueldo en un 2%.*/
--Vemos cuáles son los empleados del departamento de personal.
SELECT * FROM tdepto;
SELECT * FROM temple WHERE numde=121;
```

```
/*Vamos a meter esta sentencia dentro de una transacción
para asegurarnos que la modificación se hace correctamente.
Cuando estemos seguros de que está bien, ya podremos confirmarla
con COMMIT*/
BEGIN TRANSACTION;

UPDATE temple
SET salar=salar*1.02
WHERE numde = (SELECT numde FROM tdepto WHERE nomde LIKE
              'PERSONAL');

SELECT * FROM temple WHERE numde=121;

ROLLBACK TRANSACTION;
--COMMIT WORK;

--EJEMPLO UPDATE CON SUBCONSULTA EN LA EXPRESIÓN DEL SET:

/*Al empleado 400 asígnale un sueldo igual al doble
del salario más pequeño de su departamento.*/

/*Añadimos de nuevo al empleado 400, esta vez pertenece al
departamento 112, de esta manera tendrá más compañeros en el
departamento.*/
INSERT INTO
temple(numem,numde,extel,fecna,fecin,salar,comis,numhi,nomem)
VALUES
(400,112,350,'12/03/1980','8/1/2019',1500,100,0,'VEGA,SANDRA');

/*Vemos cuál es el mínimo salario de los empleados del
departamento al que pertenece el empleado 400*/
SELECT MIN(salar) FROM temple WHERE numde=112;

/*Vamos a meter esta sentencia dentro de una transacción
para asegurarnos que la modificación se hace correctamente.
Cuando estemos seguros de que está bien, ya podremos confirmarla
con COMMIT.*/
```

```sql
BEGIN TRANSACTION;

UPDATE temple
SET salar= (SELECT MIN(E.salar)*2 FROM temple E WHERE
             E.numde=temple.numde)
WHERE numem=400;

SELECT * FROM temple WHERE numem=400;

ROLLBACK TRANSACTION;
--COMMIT WORK

/*EJEMPLO UPDATE CON SUBCONSULTA EN EL WHERE Y
EN LA EXPRESIÓN DEL SET:*/

/*Asignar a los empleados del departamento de personal un sueldo
igual al doble del salario más pequeño de su departamento.*/

/*Vamos a meter esta sentencia dentro de una transacción
para asegurarnos que la modificación se hace correctamente.
Cuando estemos seguros de que está bien, ya podremos confirmarla
con COMMIT.*/
BEGIN TRANSACTION;

UPDATE temple
SET salar = (SELECT MIN(E.salar)*2 FROM temple E WHERE
            numde=temple.numde)
WHERE numde = (SELECT numde FROM tdepto WHERE nomde LIKE
              'PERSONAL');

-- O bien,
UPDATE temple
SET salar= (SELECT MIN(SALAR)*2
            FROM temple
            WHERE numde= (SELECT numde
                          FROM tdepto
                          WHERE nomde LIKE 'PERSONAL'))
WHERE numde = (SELECT numde
               FROM tdepto
               WHERE nomde LIKE 'PERSONAL');

SELECT * FROM temple WHERE numde=121;

ROLLBACK TRANSACTION;
--COMMIT WORK;
```

```
-- DELETE CON JOIN:

--Borrar todos los empleados del departamento llamado "Nuevo".

--Añadimos dos nuevos empleados en el departamento "Nuevo".
INSERT INTO
temple (numem,numde,extel,fecna,fecin,salar,comis,numhi,nomem)
VALUES (401,200,350,'12/03/2000','8/1/2025',1800,100,0,'BERNAL,
        ROSA'),
       (402,200,350,'14/04/2002','10/1/2025',1800,100,0,'RUIZ,
        ANTONIO');

BEGIN TRANSACTION;

DELETE FROM temple
FROM temple E JOIN tdepto D ON (E.numde=D.numde)
WHERE nomde LIKE 'NUEVO';

SELECT * FROM temple WHERE numde=200;

ROLLBACK TRANSACTION;
--COMMIT WORK;

--UPDATE CON JOIN:

/*Asignar a los empleados del departamento de personal un sueldo
igual al doble del salario más pequeño de su departamento.*/
BEGIN TRANSACTION;

UPDATE temple
SET salar= (SELECT MIN(E1.salar)*2
            FROM temple E1
            WHERE E1.numde=E.numde)
FROM temple E JOIN tdepto D ON (E.numde=D.numde)
WHERE  nomde LIKE 'PERSONAL';

SELECT * FROM temple WHERE numde=121;

ROLLBACK TRANSACTION;
--COMMIT WORK;
```

SENTENCIAS SELECT INTO, INSERT INTO SELECT y TRUNCATE

Estas tres sentencias también forman parte del LMD de SQL. La sintaxis son las siguientes:

SELECT expresión

INTO newtable

FROM oldtable

WHERE condition;

INSERT [INTO] tabla [(col1,col2, ...)]
Subselect;

TRUNCATE TABLE nombre_tabla;

Para ver su funcionamiento veamos los ejemplos que se encuentran en Ejemplos SELECT INTO - INSERT INTO SELECT y TRUNCATE.SQL". El archivo pertenece a EJEMPLOS_LMD.

```
USE empresa;
/*SQL SELECT INTO*/
/*1. Se ejecuta el SELECT
  2. Se crea la tabla con la estructura del resultado del SELECT
  3. Se carga la tabla con el resultado de la consulta
  Nota: La tabla que se crea no tiene restricciones
*/
--Ver w3school SQL Select Into

--En el siguiente ejemplo el sistema crea la tabla1. No se crean
claves.
SELECT *
INTO tabla1
FROM temple;

SELECT * FROM tabla1;

/*Almacenar en la tabla tabla2 el número, nombre y extensiones
telefónicas de los empleados
con salario inferior a 1800.*/
SELECT numem, nomem, extel
INTO tabla2
FROM temple
WHERE salar<1800;

SELECT * FROM tabla2;
```

```sql
/* SQL INSERT INTO SELECT */
/*1. Se ejecuta el SELECT
  2. Se carga la tabla con el resultado de la consulta
  Nota: La tabla debe existir y tener la misma estructura que
  el resultado de la consulta.
*/
--Ver w3school SQL Insert Into Select

--En el siguiente ejemplo la tabla3 debe existir.
/*Una forma de crearla vacía sería la siguiente,
pero ten en cuenta que no se crea clave primaria:*/
SELECT *
INTO tabla3
FROM temple
WHERE 0=1;

SELECT * FROM tabla3;

/*Una vez creada la tabla ya podemos ejecutar la sentencia
INSERT INTO SELECT.*/
INSERT INTO tabla3
SELECT * FROM temple;

SELECT * FROM tabla3;

/*TRUNCATE TABLE.
Es similar a la instrucción DELETE FROM sin una cláusula WHERE.
Quita todas las filas de una tabla sin registrar las
eliminaciones individuales de filas. ;*/

TRUNCATE TABLE tabla3;

SELECT * FROM tabla3;
```