1º CFGS Desarrollo De Aplicaciones Multiplataforma

UD 4. El lenguaje SQL - DML.

Módulo: Bases de Datos

Centro de Enseñanza
Gregorio Fernández

Macarena Cuenca Carbajo

1. Introducción

- **SQL** es el lenguaje fundamental de los SGBD relacionales.
- Es sin duda el lenguaje fundamental para manejar una base de datos relacional.
- Entre las principales características de SQL podemos destacar las siguientes:
  - Es un **lenguaje declarativo**, es decir, lo importante es definir qué se desea hacer, no cómo hacerlo.
  - Las instrucciones de SQL utilizan un **lenguaje natural**. De ahí que se le considere un lenguaje de cuarta generación.
  - Es un lenguaje para **todo tipo de usuarios**: administradores, desarrolladores y usuarios normales. Permite realizar consultas, actualizaciones, definición de datos y control.

1. Introducción

- El nacimiento del lenguaje SQL data de 1970 cuando Codd publica su libro: *"Un modelo de datos relacional para grandes bancos de datos compartidos"*.
- Apenas dos años después IBM (para quien trabajaba Codd) utiliza las directrices de Codd para crear el *Standard English Query Language,* al que se le llamó SEQUEL. Más adelante se le asignaron las siglas **SQL** (*Structured Query Language*, lenguaje estructurado de consulta).
- En 1979 **Oracle** presenta la primera implementación comercial del lenguaje. Poco después se convertía en un estándar en el mundo de las bases de datos avalado por los organismos ISO y ANSI.
- En el año 1986 se toma como lenguaje estándar por ANSI de los SGBD relacionales. Un año después lo adopta ISO, lo que convierte a SQL en estándar mundial como lenguaje de bases de datos relacionales.
- En 1989 aparece el estándar ISO (y ANSI) llamado **SQL89** o SQL1. En 1992 aparece la nueva versión estándar de SQL (a día de hoy sigue siendo la más conocida) llamada **SQL92**. En 1999 se aprueba un nuevo SQL estándar que incorpora mejoras que incluyen triggers, procedimientos, funciones,… y otras características de las bases de datos objeto-relacionales, dicho estándar se conoce como **SQL99**.
- A lo largo de los años se han ido añadiendo revisiones, que incorporan nuevas utilidades y mejoras.

2. Modos de ejecución de SQL

Ejecución directa. SQL interactivo

- Las instrucciones SQL se introducen a través de un cliente directamente conectado al servidor SQL. Las instrucciones se traducen sin intermediarios y los resultados se muestran en el cliente.

Ejecución incrustada o embebida

- Las instrucciones SQL se colocan como parte del código de otro lenguaje que se considera anfitrión (C, Java, Pascal, Visual Basic,...). Al compilar el código se utiliza un precompilador del propio SGBD para traducir el SQL y conectar la aplicación resultado con la base de datos a través de un software adaptador (driver) como JDBC u ODBC por ejemplo.

Ejecución a través de clientes gráficos

- Se trata de un software que permite conectar a la base de datos a través de un cliente. El software permite manejar de forma gráfica la base de datos y las acciones realizadas son traducidas a SQL y enviadas al servidor. Los resultados recibidos vuelven a ser traducidos de forma gráfica para un manejo más cómodo.

2. Modos de ejecución de SQL

Procesamiento de una instrucción SQL

1. Se analiza la instrucción, para comprobar su sintaxis.
2. Si es correcta se valora si los metadatos son correctos. Se comprueba esto con la información del diccionario de datos.
3. Si es correcta, se optimiza, a fin de consumir los mínimos recursos posibles.
4. Se ejecuta la sentencia y se muestra el resultado.

3. Elementos del lenguaje SQL: Sentencias SQL

- Son las distintas instrucciones que se pueden ejecutar utilizando SQL. El lenguaje SQL proporciona un gran repertorio de sentencias que permiten consultar datos, crear, actualizar y eliminar objetos de la base de datos, crear, actualizar y eliminar datos, controlar el acceso a la base de datos y a sus objetos.
- Dependiendo de las tareas, las sentencias SQL se pueden clasificar en los siguientes tipos:
  - **DML** (*Data Manipulation Language* - Lenguaje de Manipulación de Datos): INSERT, UPDATE, DELETE y SELECT.
  - **DDL** (*Data Definition Language* - Lenguaje de Definición de Datos). Sentencias que permiten crear y modificar la estructura de la base de datos. Instrucciones: CREATE, ALTER, DROP, RENAME y TRUNCATE.
  - **DCL** (*Data Control Language* - Lenguaje de Control de Datos). Sentencias que permiten administran los derechos y restricciones de los usuarios de la base de datos. Instrucciones GRANT y REVOKE.

3. Elementos del lenguaje SQL:
Normas de escritura

- En SQL no se distingue entre mayúsculas y minúsculas.
- Las instrucciones finalizan con el signo de punto y coma.
- Cualquier comando SQL (SELECT, INSERT,...) puede ser partido utilizando espacios o saltos de línea antes de finalizar la instrucción.
- Se pueden tabular líneas para facilitar la lectura si fuera necesario.
- Los comentarios en el código SQL comienzan por /* y terminan por */ (excepto en algunos SGBD).

3. Elementos del lenguaje SQL: Tipos de datos

Texto	
Tipo	Descripción
CHAR(n)	Texto de longitud fija
NCHAR(n)	Texto de longitud fija UNICODE
VARCHAR2(n)	Texto de longitud variable
NVARCHAR2(n)	Texto de longitud variable UNICODE
Números	
Tipo	Descripción
NUMBER	Enteros con precisión decimal
Fechas	
Tipo	Descripción
DATE	Fechas
TIMESTAMP	Fecha y hora
INTERVAL	Intervalos
Datos de gran tamaño	
Tipo	Descripción
CLOB	Texto de gran longitud
BLOB	Binario de gran longitud

4. Sentencia SELECT

- Para recuperar información o, lo que es lo mismo, realizar consultas a la base de datos, utilizaremos la sentencia **SELECT**.
- Sintaxis de la sentencia SELECT:

```
SELECT   [ALL | DISTINCT]
         [expr_columna1, expr_columna2, … , expr_columnaN | *]
FROM     tabla1 [, tabla2, … , tablaN]
[WHERE condición]
[ORDER BY expr_columna1 [DESC | ASC] [,expr_columna2 [DESC | ASC]…];
```

- Donde:
  - **expr_columna:** puede ser una columna de una tabla, una constante, una expresión aritmética, una función o varias funciones anidadas.
  - ***:** el asterisco significa que se seleccionan todas las columnas.

4.1. Sentencia SELECT: FROM

- Especifica la tabla o tablas de las que se recuperarán los datos.
- Ejemplo de consulta de los nombres de alumnos y su nota de la tabla ALUMNOS:

```
SELECT nom_alumno, nota FROM Alumnos;
```

- Si el usuario que hace la consulta no es el propietario de la tabla, es necesario especificar el nombre de usuario delante de ella: *nombre_usuario.nombre_tabla.* Por ejemplo, si el propietario de la tabla se llama PROFESOR pondremos:

```
SELECT nom_alumno, nota FROM Profesor.Alumnos;
```

- Es posible asociar un nuevo nombre a las tablas mediante *alias*. Por ejemplo podemos poner el alias A a la tabla de ALUMNOS de la siguiente forma:

```
SELECT A.nom_alumno, A.nota FROM Alumnos A;
```

4.2. Sentencia SELECT: WHERE

- Esta cláusula permite obtener las filas que cumplen la condición expresada.
- El formato de la condición es:

  expresión operador_comparación expresión

- Las expresiones pueden ser: una constante, una expresión aritmética, un valor nulo o un nombre de columna.
- Los operadores de comparación pueden ser los siguientes:

=, >, <, >=, <=, !=, <>
IN, NOT IN, BETWEEN, NOT BETWEEN
LIKE

- Se pueden construir condiciones múltiples utilizando los operadores lógicos AND, OR y NOT. Se pueden usar paréntesis para forzar el orden de evaluación.
- Veamos algunos ejemplos de condiciones en la cláusula WHERE:

```
WHERE nota=5;
WHERE (nota>=10) AND (curso=1);
WHERE (nota IS NULL) OR (UPPER(nom_alumno)='PEDRO')
```

4.3. Sentencia SELECT: ORDER BY

- El orden inicial de los registros obtenidos por una SELECT es el orden en el que fueron introducidos. Para ordenar en base a otros criterios, se utiliza esta claúsula.
- En esa cláusula se coloca una lista de campos que indica la forma de ordenar. Se ordena primero por el primer campo de la lista, si hay coincidencias por el segundo, si ahí también las hay por el tercero, y así sucesivamente.
- Se puede colocar las palabras ASC o DESC, que significan ascendente o descendente respectivamente(por defecto se toma ASC).
- Ejemplo:

```
SELECT * FROM Alumnos ORDER BY nom_alumno, curso DESC;
```

- Esta sentencia ordena por nombre de alumno ascendentemente y en caso de coincidencia por curso descendentemente.

4.4. Sentencia SELECT: ALL | DISTINCT

ALL

- Recupera todas las filas aunque algunas estén repetidas. Es la opción por defecto.

DISTINCT

- Sólo recupera las filas que son distintas. Por ejemplo, si consultamos los departamentos de la tabla EMPLEADO, vemos que aparecen números de departamento repetidos. Con DISTINCT se eliminan esas filas repetidas.

```
SELECT DISTINCT num_dpto FROM Empleados;
```

4.5. Sentencia SELECT: Alias de columnas

- Cuando se realizan consultas a la base de datos, los nombres de las columnas aparecen como cabeceras de presentación.
- Si el nombres es demasiado largo, corto ó críptico, existe la posibilidad de cambiarlo en la misma sentencia SELECT creando un *alias*.
- El *alias* se pone entre comillas dobles a continuación del nombre de la columna.
- Ejemplo:

```
SELECT nom_dpto “Departamento”, num_dpto “Nº departamento” FROM Departamentos;
```

5. Operadores aritméticos

- Los operadores aritméticos +, -, * y /, sirven para hacer cálculos en las consultas.
- Cuando se utilizan como expresión en una sentencia SELECT, no modifican los datos originales, sino que como resultado aparece una nueva columna.
- Ejemplo:

  SELECT nom_alumno, (nota1 + nota2 + nota3)/3 “Nota media” FROM Notas_Alumnos;

- La prioridad de los operadores aritméticos es la normal, tienen más prioridad la multiplicación y división, después la suma y la resta.
- En caso de igualdad de prioridad, se realiza primero la operación que esté más a la izquierda.
- Se puede modificar la prioridad usando paréntesis.

6. Concatenación de textos

- Todas las bases de datos incluyen algún operador para encadenar textos.
- En Oracle son los signos ||.
- Ejemplo:

```
SELECT tipo, modelo, tipo || '-' || modelo "Clave pieza" FROM Piezas;
```

7. Condiciones

- Se utiliza la cláusula **WHERE.**
- Permite colocar una condición que han de cumplir todas las filas.
- Las que no la cumplan no aparecen en el resultado.

7.1. Condiciones: Operadores de comparación

Operador	Significado
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
=	Igual
<>	Distinto
!=	Distinto

- Se pueden utilizar tanto para comparar números como para comparar textos y fechas. Ejemplo:

```
SELECT nom_alumno FROM Alumnos WHERE (nota1+nota2+nota3)/3 >6;
```

7.2. Condiciones: Operadores lógicos

Operador	Significado
AND	Devuelve TRUE si las dos condiciones son verdaderas.
OR	Devuelve TRUE cuando una de las dos condiciones es verdadera.
NOT	Invierte la lógica de la condición. Si era verdadera pasa a ser falsa, y viceversa.

- Ejemplo:

```
SELECT apellido, salario, num_dpto FROM Empleados
WHERE (salario>2000) AND (num_dpto=10 OR num_dpto=20);
```

7.3. Condiciones: BETWEEN

- Permite obtener datos que se encuentren dentro de un rango. Su formato es:

  <expresión> [**NOT**] **BETWEEN** valor_inicial **AND** valor_final

- Ejemplos:

```
/* Obtiene los empleados con salario entre 1500 y 2000 euros */
SELECT apellido, salario FROM Empleados WHERE salario BETWEEN 1500 AND 2000;

/* Obtiene los empleados cuyo salario no está entre 1500 y 2000 euros */
SELECT apellido, salario FROM Empleados WHERE salario NOT BETWEEN 1500 AND 2000;
```

7.4. Condiciones: IN

- Permite comprobar si una expresión pertenece o no a una lista de valores, haciendo posible la realización de comparaciones múltiples. Su formato es:

  <expresión> [**NOT**] **IN** (lista de valores separados por comas)

- Ejemplos:

```
/* Obtiene los empleados cuyo nº de departamento sea 10 ó 30 */
SELECT apellido FROM Empleados WHERE num_dpto IN (10,30);

/* Obtiene los empleados cuyo oficio no sea ni vendedor ni analista */
SELECT apellido FROM Empleados WHERE oficio NOT IN ('vendedor','analista');
```

7.5. Condiciones: LIKE

- Para comparar cadenas de caracteres que cumplan un patrón no nos sirve el operador =. Para estos casos podemos usar el operador LIKE que permite utilizar los siguientes comodines en las cadenas de comparación:

Operador	Significado
%	Representa cualquier cadena de 0 ó más caracteres
_	Representa un carácter cualquiera

- Su formato es: **WHERE** columna **LIKE** 'caracteres_especiales'
- Ejemplos:

```
/* Obtiene los empleados cuyo apellido empieza por J */
SELECT apellido FROM Empleados WHERE apellido LIKE 'J%';
/* Obtiene los empleados cuyo apellido tenga una R en la segunda posición */
SELECT apellido FROM Empleados WHERE apellido LIKE '_R%';
/* Obtiene los empleados cuyo apellido empieza por A y tengan una O en su interior */
SELECT apellido FROM Empleados WHERE apellido LIKE 'A%O%';
```

7.6. Condiciones: NULL

- Una columna de una fila es NULL si está completamente vacía.
- Para comprobar si el valor de una columna es nulo no podemos utilizar los operadores de igualdad mayor o menor. Para ello utilizamos la expresión:

  columna **IS** [**NOT**] **NULL**

- Ejemplo:

```
/* Obtiene los empleados que no tienen teléfono */
SELECT apellido FROM Empleados WHERE telefono IS NULL;
```

7.7. Condiciones: Precedencia

Orden de precedencia	Operador
1	*, /
2	+, -
3	|| (concatenación)
4	Comparaciones (>, <, …)
5	IS [NOT] NULL, [NOT] LIKE, IN
6	NOT
7	AND
8	OR

8. Funciones

- Los SGBD disponen de funciones que podemos usar en las consultas.
- Estas funciones se usan dentro de expresiones y actúan con los valores de las columnas, variables o constantes.
- Todas las funciones devuelven un resultado que procede de un determinado cálculo.
- La mayoría de funciones precisan que se les envíe datos de entrada (*parámetros* o *argumentos*) que son necesarios para realizar el cálculo de la función.
- Sintaxis de una función:

  nombreFunción[(parámetro1[, parámetro2,...])]

- Si una función no precisa parámetros (como **SYSDATE**) no hace falta colocar los paréntesis.
  a) Generalmente producen dos tipos diferentes de resultados:
  b) Modificación de la información original.
- Devolución de información.
- Se utilizan en cláusulas SELECT, cláusulas WHERE y cláusulas ORDER BY. Además, es posible el anidamiento de funciones.

8. Funciones

- Tipos de funciones:
  - Aritméticas
  - De cadenas de caracteres
  - De manejo de fechas
  - De conversión
  - Otras
- Oracle proporciona una tabla llamada DUAL con la que se permiten hacer pruebas. Esa tabla tiene un solo campo (llamado DUMMY) y una sola fila.

*Pues ver un resumen de las funciones Oracle en "**Material de Apoyo**"*

9. Consultas multitabla: JOIN

- Sintaxis general:

  SELECT columnas de las tablas que aparecen en la cláusula FROM

  FROM tabla1, tabla2

  WHERE tabla1.columna = tabla2.columna ;

- En el apartado FROM se pueden indicar varias tablas separadas por comas.
- Si no ponemos la condición de emparejamiento (join) entre las tablas, se produce un producto cruzado o producto cartesiano, en el que aparecerán todos los registros de tabla1 relacionados con todos los registros de tabla2.
- Es decir, el **producto cartesiano** empareja todas las filas de una tabla con cada fila de la otra tabla.
- El producto cartesiano raras veces es útil, lo que necesitamos es discriminar ese producto para que sólo aparezcan los registros de la tabla1 relacionados con los de la tabla2 correspondientes. A eso se le llama asociar (**join**) tablas.

10. Consultas agrupadas: GROUP BY

- La sentencia SELECT posibilita agrupar uno o más conjuntos de filas. El agrupamiento se lleva a cabo mediante la cláusula GROUP BY por las columnas especificadas y en el orden especificado.
- Sintaxis general:

```
SELECT ...
FROM …
GROUP BY columnal, columna2 , columna3, ...
HAVING condición
ORDER BY ...
```

- Así por ejemplo, podremos saber cuál es el salario medio de cada departamento.

10. Consultas agrupadas: GROUP BY

- Los datos seleccionados en la sentencia SELECT deben ser:
  - una constante
  - una función de grupo (SUM, COUNT, AVG, ...) ó
  - una columna expresada en el GROUP BY.
- Del mismo modo que existe la condición de búsqueda WHERE para filas individuales, también hay una condición de búsqueda para grupos de filas: **HAVING**.
- La cláusula **HAVING** se emplea para controlar cuál de los conjuntos de filas se visualiza. Se evalúa sobre la tabla que devuelve el GROUP BY. No puede existir esta cláusula sin GROUP BY.
- Además, cuando usamos la cláusula **ORDER** BY con columnas y funciones de grupo hemos de tener en cuenta que ésta se ejecuta detrás de las cláusulas WHERE, GROUP BY y HAVING.
- En ORDER BY podemos especificar:
  - funciones de grupo.
  - columnas de GROUP BY o
  - su combinación.

10. Consultas agrupadas: GROUP BY

- Resumiendo, la evaluación de las cláusulas en tiempo de ejecución se efectúa en el siguiente orden:
  1. **WHERE** Selecciona las filas.
  2. **GROUP BY** Agrupa estas filas.
  3. **HAVING** Filtra los grupos. Selecciona y elimina los grupos.
  4. **ORDER BY** Clasifica la salida. Ordena los grupos.

11. Combinaciones externas: OUTER JOIN

- Existe una variedad de combinación de tablas que se llama **OUTER JOIN** y que nos permite seleccionar algunas filas de una tabla aunque éstas no tengan correspondencia con las filas de la otra tabla con la que se combina.
- El formato es el siguiente:

```
SELECT tabla1.columa1, tabla1.columa2, tabla2.columa1,
FROM tabla1, tabla2
WHERE tablal.columa1 = tabla2. columna1 (+)
```

- Esta SELECT seleccionará todas las filas de la tabla tabla1, aunque no tengan correspondencia con las filas de la tabla tabla2.
- Se denota con el símbolo **(+)** detrás de la columna de la tabla2 (que es la tabla donde no se encuentran las filas) en la cláusula WHERE. Se obtendrá una fila con las columnas de tabla1 y el resto de columnas de la tabla2 se rellena con NULL.

12. Combinaciones especiales: UNION, INTERSECT, MINUS

- Los **operadores relacionales UNION, INTERSECT** y **MINUS** son *operadores de conjuntos*.
- Los conjuntos son las filas resultantes de cualquier sentencia SELECT válida que permiten combinar los resultados de varias SELECT para obtener un único resultado.
- El formato de SELECT con estos operadores es el siguiente:

  SELECT … FROM … WHERE …

  Operador de conjuntos

  SELECT … FROM … WHERE …

12.1. Combinaciones especiales: UNION, INTERSECT, MINUS

Operador UNION

- Combina los resultados de dos consultas. Las filas duplicadas que aparecen se reducen a una fila única.
- Formato:

```
SELECT columna1, columna2, … FROM tabla1 WHERE condición
UNION
SELECT columna1, columna2, … FROM tabla2 WHERE condición;
```

- **UNION ALL** combina los resultados de dos consultas pero a diferencia de **UNION**, no elimina duplicados.

12.2. Combinaciones especiales: UNION, INTERSECT, MINUS

Operador INTERSECT

- El operador **INTERSECT** devuelve las filas que son comunes en ambas consultas. Todas las filas duplicadas serán eliminadas antes de la generación del resultado final.
- Formato:

```
SELECT columna1, columna2, … FROM tabla1 WHERE condición
INTERSECT
SELECT columna1, columna2, … FROM tabla2 WHERE condición;
```

12.3. Combinaciones especiales: UNION, INTERSECT, MINUS

Operador MINUS

- El operador **MINUS** devuelve aquellas filas que están en la primera SELECT y no en la segunda. Las filas duplicadas del primer conjunto se reducirán a una fila única antes de que empiece la comparación con el otro conjunto.
- Formato:

```
SELECT columna1, columna2, … FROM tabla1 WHERE condición
MINUS
SELECT columna1, columna2, … FROM tabla2 WHERE condición;
```

12.4. Combinaciones especiales: UNION, INTERSECT, MINUS

Reglas para la utilización de operadores de conjuntos

- Las columnas de las dos consultas se relacionan en orden, de izquierda a derecha.
- Los nombres de columna de la primera sentencia SELECT no tienen por qué ser los mismos que los nombres de columna de la segunda.
- Las SELECT necesitan tener el mismo número de columnas.
- Los tipos de datos deben coincidir, aunque la longitud no tiene que ser la misma.

13. Subconsultas

- A veces, para responder una consulta necesitamos los datos devueltos por otra consulta.
- Por ejemplo, si queremos obtener los datos de los empleados que tengan el mismo oficio que 'PEPE', hemos de averiguar el oficio de 'PEPE' (primera consulta). Una vez conocido este dato, podemos averiguar qué empleados tienen el mismo oficio que 'PEPE' (segunda consulta).
- Este problema se puede resolver usando una **subconsulta**, que no es más que una sentencia SELECT dentro de otra SELECT.
- Una **subconsulta** consistirá en incluir una SELECT como parte de una cláusula WHERE.
- Formato:

```
SELECT …
FROM …
WHERE columna operador (SELECT …
                        FROM …
                        WHERE …);
```

→ La subconsulta se ejecutará primero y, posteriormente el valor extraído es "introducido" en la consulta principal.

Centro de Enseñanza
Gregorio Fernández

13.1. Test en subconsultas

Test de comparación en subconsultas (>,<,<>,<=,>=,=)

- Compara el valor de una expresión con un valor único producido por una subconsulta.
- Ejemplo: Obtener aquellos apellidos de empleados cuyo oficio es igual al oficio de 'GIL':

```
SELECT apellido FROM Empleados WHERE oficio =
  (SELECT oficio FROM Empleados WHERE apellido ='GIL' );
```

13.2. Test en subconsultas

Test de pertenencia a un conjunto devuelto por una subconsulta (IN)

- Comprueba si el valor de una expresión coincide con uno del conjunto de valores producido por una subconsulta.
- Ejemplo: Obtener aquellos apellidos de empleados cuyo oficio sea alguno de los oficios que hay en el departamento 20:

```
SELECT apellido FROM Empleados WHERE oficio IN
(SELECT oficio FROM Empleados WHERE num_dpto=20);
```

13.3. Test en subconsultas

Test de existencia (EXISTS, NOT EXISTS)

- Examina si una subconsulta produce alguna fila de resultados. El test es TRUE si devuelve filas, si no es FALSE.
- Ejemplo: Listar los departamentos que tengan empleados:

```
SELECT nombre_dpto, num_dpto FROM Departamentos WHERE EXISTS
    (SELECT * FROM Empleados WHERE
        Empleados.num_dpto=Departamentos.num_dpto);
```

- Para calcular los que no tengan empleados se usa el operador **NOT EXISTS**.

13.4. Test en subconsultas

Test de comparación cuantificada (ANY y ALL)

- Se usan en conjunción con los operadores de comparación (>, <, <>, <=, >=, =).
- **ANY** compara el valor de una expresión con cada uno del conjunto de valores producido por una subconsulta, si alguna de las comparaciones individuales da como resultado TRUE, ANY devuelve TRUE, si la subconsulta no devuelve nada devolverá FALSE.
- Ejemplo: Obtener los datos de los empleados cuyo salario sea igual a algún salario de los empleados del departamento 30:

```
SELECT * FROM Empleados WHERE salario = ANY
        (SELECT salario FROM Empleados WHERE num_dpto=30);
```

13.5. Test en subconsultas

Test de comparación cuantificada (ANY y ALL)

- **ALL** compara el valor de una expresión con cada uno del conjunto de valores producido por una subconsulta, si todas las comparaciones individuales da como resultado TRUE, ALL devuelve TRUE, en caso contrario devuelve FALSE.
- Ejemplo: Obtener los datos de los empleados cuyo salario sea menor a cualquier salario de los empleados del departamento 30:

```
SELECT * FROM Empleados WHERE salario < ALL
        (SELECT salario FROM Empleados WHERE num_dpto=30);
```

14. Sentencia INSERT

- Permite añadir filas de datos en una tabla.
- Su sintaxis es la siguiente:

```
INSERT INTO tabla [(columna1 [, columna2] …)]
VALUES (valor1 [, valor2] …);
```

- Donde:
  - **tabla:** es la tabla en la que se van a insertar las filas.
  - **[(columna1 [, columna2] …)]:** son las columnas donde se van a introducir los valores. Si no se especifican, se consideran, por defecto, todas las columnas de la tabla.
  - **(valor1 [, valor2] …):** son los valores que se van a dar a las columnas. Éstos se deben corresponder con cada una de las columnas que aparecen. Además, deben coincidir con el tipo de dato definido para cada columna.
  - Cualquier columna que no se encuentre en la lista de columnas recibirá el valor NULL, siempre y cuando no esté definida como NOT NULL, en cuyo caso INSERT fallará.
  - Si no se da la lista de columnas, se han de introducir valores en todas las columnas.

14. Sentencia INSERT: Multifila

- La sentencia INSERT “sencilla” solo inserta una fila, pero si añadimos a INSERT una sentencia SELECT, se añaden tantas filas como devuelva la consulta.
- El formato es el siguiente:

```
INSERT INTO tabla1 [(columna1 [, columna2] …)]
SELECT (columna1 [, columna2] … | *);
FROM tabla2 [cláusulas de SELECT];
```

- Si las columnas no se especifican en la cláusula INSERT, por defecto, se consideran todas las columnas de la tabla.

15. Sentencia UPDATE

- Para actualizar los valores de las columnas de una o varias filas de una tabla utilizamos la sentencia **UPDATE**, cuyo formato es el siguiente:

```
UPDATE tabla
SET columna1=valor1, … , columnaN=valorN
WHERE condición;
```

- Dónde:
  - **tabla:** es la tabla cuyas columnas se van a actualizar.
  - **SET:** indica las columnas que se van a actualizar y sus valores.
  - **WHERE:** selecciona las filas que se van a actualizar. Si se omite, la actualización afectará a todas las filas de la tabla.

15. Sentencia UPDATE: con SELECT

- Podemos incluir una subconsulta en una sentencia UPDATE que puede estar contenida en la cláusula WHERE o puede formar parte de SET.
- En la clausula **WHERE**:

```
UPDATE tabla
SET columna1=valor1, … , columnan=valorn
WHERE columna test (SELECT columna …);
```

- Cuando la subconsulta forma parte de **SET**, debe seleccionar una única fila y el mismo número de columnas (con tipos de datos adecuados) que las que hay entre paréntesis al lado de SET. Dos formatos:

```
UPDATE tabla
SET (columna1, columna2, …) = (SELECT columna1, columna2, …)
WHERE condición;

UPDATE tabla
SET columna1 = (SELECT columna1 …), columna2 = (SELECT columna2 …)
WHERE condición;
```

16. Sentencia DELETE

- Permite eliminar una o varias filas de una tabla.
- La cláusula WHERE es esencial para eliminar sólo aquellas filas deseadas. Sin ella, DELETE borrará todas las filas de la tabla.
- La condición puede incluir una subconsulta.
- Sintaxis:

```
DELETE [FROM] tabla
WHERE condición;
```