

Lenguajes para la definición y manipulación de datos en sistemas de base de datos relacionales. Tipos. Características. Lenguaje SQL.

TEMA 39 (38 SAI)

ABACUS NT

Índice

- 1. *Introducción.***
- 2. *Sistemas de información.***
 - 2.1. Sistemas de ficheros.**
 - 2.2. Bases de datos.**
 - 2.3. Modelos de datos. Esquemas. Diseño de bases de datos.**
- 3. *Lenguajes para la definición y manipulación de datos en sistemas de base de datos relacionales***
 - 3.1. Lenguajes para la definición de datos**
 - 3.2. Lenguajes para la manipulación de datos**
 - 3.3. Características**
- 4. *Lenguaje SQL***
- 5. *Lenguaje de definición de datos (DDL).***
- 6. *Lenguaje de manipulación de datos (DML)***
 - 6.1. SQL**
 - 6.1.1. Instrucciones de actualización**
 - 6.1.2. Consulta de Datos - Cláusula Select**
- 7. *Conclusión***
 - 7.1. Relación del tema con el sistema educativo actual**
- 8. *Bibliografía***

1. Introducción.

En el presente tema, vamos a estudiar y analizar los aspectos fundamentales relacionados con los lenguajes de definición y manipulación de datos en sistemas de bases de datos relaciones. Se trata de un tema de suma importancia dentro del campo de estudio de las aplicaciones, ya que la mayoría de las aplicaciones almacenan sus datos de forma persistente en una base de datos relacional, por lo que la forma en que estos se definen y manipulen, va a condicionar el **mantenimiento, rendimiento y velocidad de desarrollo** de la aplicación.

2. Sistemas de información.

Un sistema de información es un conjunto de elementos que gestiona la información de una determinada organización. Sus componentes son:

- Datos. Información relevante que almacena y gestiona el sistema de información.
- Hardware. Equipamiento físico que se utiliza para gestionar los datos.
- Software. Aplicaciones que permiten el funcionamiento adecuado del sistema.
- Recursos humanos. Personal que maneja el sistema de información.

Existen dos tipos fundamentales de sistemas de información:

- Orientados al proceso. En estos sistemas de información se crean diversas aplicaciones (software) para gestionar diferentes aspectos del sistema. Cada aplicación realiza unas determinadas operaciones, y almacena y utiliza sus propios datos.
- Orientados a los datos. En esos sistemas los datos se almacenan en una única estructura lógica que es utilizable por las aplicaciones. A través de esa estructura se accede a los datos que son comunes a todas las aplicaciones.

2.1. Sistemas de ficheros.

Los sistemas de ficheros son sistemas de información orientados al proceso que tienen las siguientes características:

- Los ficheros se diseñan para una determinada aplicación.
- Los datos se encuentran almacenados en soportes de almacenamiento secundario, mientras su descripción está separada de los mismos, formando parte de los programas.
- No hay control sobre el acceso y la manipulación de los datos más allá de lo impuesto por los programas de aplicación.

Los inconvenientes que se plantean son los siguientes:

- Hay una ocupación inútil de memoria secundaria.
- Suele aparecer un cierto grado de inconsistencia y duplicación en la información.
- Falta de flexibilidad del sistema de ficheros para adaptarse a las nuevas necesidades.
- Existe cierta dificultad para compartir información.

2.2. Bases de datos.

Una base de datos es un conjunto estructurado de datos relacionados entre sí que reside en soportes de almacenamiento secundario de acceso directo.

Las características que las diferencian de los sistemas de ficheros son las siguientes:

- Además de los datos, se almacenan las relaciones entre ellos y sus restricciones semánticas.
- No debe existir redundancia lógica, aunque se admite redundancia física por eficiencia.
- Las bases de datos han de atender a múltiples usuarios y diferentes aplicaciones.
- Existe independencia tanto física como lógica entre datos y tratamientos.
- La definición y la descripción de los datos están integradas con los mismos datos.
- Incorporan procedimientos de actualización y recuperación que mantienen la integridad, seguridad y confidencialidad de los datos.

2.3. Modelos de datos. Esquemas. Diseño de bases de datos.

Un modelo de datos es una colección de conceptos para la descripción de los datos, las relaciones entre ellos y las restricciones que deben cumplir. Un esquema es una descripción de una base de datos mediante un modelo de datos. Los modelos de datos se clasifican en:

- Modelos conceptuales. Describen los datos con un alto nivel de abstracción, utilizando entidades (concepto del mundo real), atributos (propiedad de interés de una entidad) y relaciones (interacción entre dos o más entidades). Son independientes de la base de datos a utilizar. Por ejemplo, el modelo entidad/relación y el modelo orientado a objetos.
- Modelos lógicos. Representan los datos valiéndose de estructuras de registros de varios tipos, formados por un número determinado de campos. Son dependientes de la base de datos a utilizar. Por ejemplo, el modelo relacional, el modelo de red y el modelo jerárquico.
- Modelos físicos. Los modelos físicos describen cómo se almacenan los datos en cuanto al formato de los registros, la estructura de los ficheros y los métodos de acceso utilizados.

El diseño de bases de datos se estructura en tres pasos:

- Diseño conceptual. Recibe como entrada la especificación de requerimientos y su resultado es el esquema conceptual, que es una descripción de alto nivel de la estructura de la base de datos mediante un modelo conceptual, y que es independiente del SGBD que se utilice.
- Diseño lógico. Recibe como entrada el esquema conceptual y da como resultado un esquema lógico, que es una descripción de la estructura de la base de datos mediante un modelo lógico, y que puede ser procesado por el SGBD que se utilice.

- Diseño físico. Recibe como entrada el esquema lógico y da como resultado un esquema físico, que es una descripción mediante un modelo físico de las estructuras de almacenamiento y de los métodos usados para tener un acceso efectivo a los datos.

3. Lenguajes para la definición y manipulación de datos en sistemas de base de datos relacionales

Según el autor **C.J Date**, en su libro *Introduction to Database Systems* (2003), “una base de datos es un conjunto de datos persistentes que es utilizado por los sistemas de aplicación de alguna empresa dada” (entendiéndose como empresa una organización).

Para trabajar con dicho conjunto de datos, la arquitectura ANSI/X3/SPARC diferencia dos tipos de lenguajes:

- **Lenguaje de definición de datos:**
- **Lenguaje de manipulación de datos**

A continuación, vamos a ver en qué consiste cada uno de ellos

3.1. Lenguajes para la definición de datos

El lenguaje de descripción o definición de datos (DDL, Data Definition Language), propio de cada SGBD, permite al desarrollador de la base de datos especificar los elementos de datos que integran su estructura, y las relaciones que existen entre ellos, las reglas de integridad semántica, las características de tipo físico y las vistas lógicas de los usuarios.

3.2. Lenguajes para la manipulación de datos

Una vez descrita la estructura de la base de datos, los usuarios utilizarán un lenguaje de manipulación de datos (DML, Data Manipulation Language) para realizar las siguientes operaciones sobre dicha estructura:

- **Recuperar información:** Consultar la base de datos.
- **Actualizar la información:** La actualización supondrá tres tipos de operaciones
 - Inserción
 - Borrado
 - Modificación de los datos.

A su vez, dentro de los DML, podemos destacar los siguientes ejemplos comerciales:

- **SQL (Structured Query Language):** Lo estudiaremos en el siguiente apartado.
- **QBE (Query By Example):** Creado a principios de los años 70. Hay varias **implementaciones** de este lenguaje, como el original de IBM (Sistema QBE), o QBE de Microsoft (en Access). Se fundamenta en el cálculo relacional orientado a dominios.

- **QUEL (Query Language):** Basado en el cálculo relacional orientado a tuplas.

3.3. Características

Como hemos visto anteriormente, para trabajar con las bases de datos vamos a tener dos tipos de lenguajes: los DML y los DDL, éstos, a su vez, se pueden clasificar atendiendo a las siguientes características:

- **Embebido/autocontenido:** Un lenguaje autocontenido no necesita de otro, mientras que un lenguaje embebido se inserta en el seno de un programa escrito en otro lenguaje, denominado anfitrión.
- **Más o menos procedimental:** Un lenguaje de manipulación de datos es más procedimental, cuanto con más detalles es preciso especificar el procedimiento necesario para acceder a la base de datos.
- **Diferido/Conversacional:** Algunos DML se utilizan en modo diferido, esto es, en tratamiento por lotes, pero en la actualidad la mayoría permiten su uso en modo conversacional, es decir, interactivo, desde un terminal, o interfaz gráfica.
- **Navegacional/Especificación:** Existen DML llamados navegacionales que recuperan o actualizan los datos registro a registro, debiéndose indicar el camino que se ha de recorrer, hasta llegar al registro buscado. Otros, actúan sobre el conjunto de registros, como por ejemplo SQL.
- **Compilación/Interpretación:** Los lenguajes de compilación necesitan de un compilador, el cual se encarga de generar código máquina a partir de un código fuente. Por el contrario, un lenguaje interpretado es el que es ejecutado paso a paso, sin ser necesaria una traducción previa a la ejecución.

4. Lenguaje SQL

5. Lenguaje de definición de datos (DDL).

Cualquier lenguaje de bases de datos está formado por un DDL y por un DML. Existe un lenguaje considerado como estándar para manipular bases de datos relacionales: SQL (Structured Query Language). Estudiaremos el DDL y el DML que utiliza el SQL. Las características principales de SQL son su sencillez, su carácter estándar y ser un lenguaje declarativo o no procedural (para que SQL realice una acción concreta no se le dice cómo tiene que hacerla, sino lo que queremos obtener).

El lenguaje de definición de datos (DDL) proporciona órdenes para definir, eliminar y modificar tablas, así como para crear índices y vistas. Las órdenes SQL más importantes del DDL son:

Creación de una tabla

CREATE sirve para crear objetos de la base de datos, entre estos objetos tenemos tablas, vistas etc.

```

CREATE TABLE nombre_tabla
(nombrecoll tipocol1
 [CONSTRAINT nombre_restricción]
 [not NULL]
 [PRIMARY KEY]
 [UNIQUE]
 [DEFAULT valor]
 [check <condici>]
 [REFERENCES Nombre_tabla_ref (colref) [ON DELETE CASCADE]] ,...
 [Restricciones de la tabla]
)
[tablespace nombre-tablespace];

```

Donde:

- **Nombre tabla:** Es el nombre de la nueva tabla. Debe ser único dentro de la BD y además debe identificar su contenido, el nombre de la tabla puede ser una cadena de 1 a 30 caracteres alfanuméricos (0-9, a-z, subrayado, \$, #) empezando siempre por un carácter alfabético.
- **nombreCol:** Es el nombre de una columna de la tabla. Los nombres de columna deben cumplir las reglas de los identificadores y deben ser únicos en la tabla.
- **tipoCol:** Especifica el tipo de datos de la columna. Se permiten los tipos de datos del sistema o definidos por el usuario. Especifica el tipo de datos de la columna. Se permiten los tipos de datos del sistema o definidos por el usuario.

Cuando almacenamos datos las tablas se ajustan a una serie de restricciones predefinidas, por ejemplo, que una columna no pueda tomar valores negativos o que una columna tenga que almacenarse en mayúsculas.

La integridad hace referencia al hecho de que los datos de la BD han de ajustarse a restricciones antes de almacenarse en la BD y una restricción de integridad será una regla que restringe el rango de valores de una o más columnas de una tabla.

Existe otro tipo de integridad que es la integridad referencial, que garantiza que los valores de una o varias columnas de una tabla dependan de los valores de otro o otras columnas de otra tabla.

Las restricciones se definen dentro de la orden CREATE TABLE y para ello se utiliza la cláusula CONSTRAINT. Una vez creadas las restricciones se pueden añadir, modificar o borrar a través de la orden ALTER TABLE.

Una cláusula CONSTRAINT puede restringir una sola columna, se habla en este caso de restricción de columna o puede restringir un grupo de columnas de una tabla, en este caso se llama restricción de tabla.

- **CONSTRAINT.** Es una palabra clave opcional que indica el principio de la definición de una restricción para la columna o tabla: PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY o CHECK. Las restricciones son propiedades especiales que exigen la integridad de los datos y pueden crear índices para la tabla y sus columnas.
- **nombre_restricción.** Los nombres de restricción deben ser únicos en una base de datos.
- **NULL | NOT NULL.** Son palabras clave que determinan si se permiten o no valores NULL en la columna. Si la restricción es NOT NULL significa que dicha columna no puede tener valores nulos, es decir, ha de tener un valor obligatoriamente; en caso contrario causa una excepción.
- **PRIMARY KEY.** Es una restricción que indica qué columna o columnas formarán la clave primaria de la tabla. Sólo se puede crear una restricción PRIMARY KEY por cada tabla.
- **UNIQUE.** Es una restricción que proporciona la integridad de entidad para la columna o columnas indicada a través de un índice único. Una tabla puede tener varias restricciones UNIQUE.
- **DEFAULT.** Especifica el valor suministrado para la columna cuando no se ha especificado explícitamente un valor durante la inserción.
- **REFERENCES.** Es una restricción que proporciona integridad referencial para los datos de una columna o columnas. Las restricciones REFERENCES requieren que cada valor de la columna o columnas existan en la columna o columnas de referencia correspondiente de la tabla a la que se hace referencia. Las restricciones REFERENCES pueden hacer referencia sólo a columnas que sean restricciones PRIMARY KEY en la tabla de referencia.
- **Nombre_tabla_ref.** Es el nombre de la tabla a la que hace referencia la restricción REFERENCES.
- **(colref [,...n]).** Es una columna o lista de columnas de la tabla a la que hace referencia la restricción REFERENCES.
- **ON DELETE CASCADE.** Especifica qué acción tiene lugar en una fila de la tabla creada, si esa fila tiene una relación referencial y la fila a la que hace referencia se elimina en la tabla primaria. En nuestro caso si se elimina una fila de la tabla primaria, también se eliminan las filas de la tabla desde donde se hace referencia. Cuando la restricción de Integridad Referencial se realiza sobre la definición de un campo en la sentencia CREATE TABLE solo se utiliza la cláusula REFERENCES, no se utiliza la cláusula FOREIGN KEY; esta última se utiliza cuando la restricción se crea a nivel de tabla.
- **CHECK.** Es una restricción que exige la integridad del dominio al limitar los valores posibles que se pueden escribir en una o varias columnas.

Borrado de una tabla (estructura y datos)

```
Drop table nombre_tabla [CASCADE CONSTRAINT];
```

Al borrar una tabla, se borra tanto su estructura como sus datos, sus índices asociados y los privilegios concedidos sobre estas también se borran, las vistas creadas directa o indirectamente sobre esta tabla son desactivadas de forma automática por ORACLE, pero no borradas.

Cada usuario puede borrar sus propias tablas, pero no puede borrar las de otro usuario al menos que tenga concedido un permiso adecuado. Si se hace referencia a la clave primaria de esta tabla mediante restricciones FOREIGN KEY o REFERENCES, la cláusula CASCADE CONSTRAINT permite suprimir estas restricciones de integridad referencial en las tablas ‘descendientes’.

Borrado de los registros de una tabla

Con la orden TRUNCATE se eliminan todas las filas de una tabla y se puede liberar espacio utilizado por esta tabla. Es una orden del lenguaje DDL y por tanto no se puede anular. Tampoco activa disparadores DELETE por lo que es más rápido que una orden DELETE. Su sintaxis es:

```
Truncate table nombre_table [{DROP | REUSE} STORAGE];
```

Con DROP STORAGE se desasigna todo el espacio. Con DROP REUSE mantendrá reservado el espacio para nuevas filas.

Modificar la estructura de una tabla

Para modificar la estructura de una tabla se utiliza el comando ALTER TABLE.

```
ALTER TABLE Tabla
{ [ADD      ( Columna Tipodato [restricción de columna] [...] ) ]
[MODIFY ( Columna Tipodato[restricción de columna]] [...] ) ]
[ADD CONSTRAINTS restricción]
[DROP CONSTRAINTS restricción]};
```

Añadir o modificar columnas (nombre, tipo, valor por defecto, restricción NOT NULL)

```
alter table nombre_table {ADD | MODIFY} ( columna tipo [restricción,...])
```

Eliminación de columnas

```
alter table nombre_table DROP COLUMN nombre_columna
```

Añadir restricción de tabla

```
alter table nombre_tabla ADD CONSTRAINT nombre_restricción restricción
```

Eliminar una restricción.

```
alter table nombre_table DROP CONSTRAINT nombre_restricción
```

Activación y desactivación de una restricción.

```
alter table nombre_table [ENABLE VALIDATE|ENABLE NOVALIDATE|DISABLE] nombre_restricción
```

Donde:

- ENABLE VALIDATE activa la restricción si el conjunto de filas ya presentes en la tabla cumplen dicha restricción.
- ENABLE NOVALIDATE activa la restricción para las siguientes instrucciones de manipulación de datos.
- DISABLE desactiva la restricción.

Hay que tener en cuenta que si la tabla está vacía, al añadir una columna con la restricción NOT NULL no habrá ningún error, pero si tiene filas no permitirá añadir una columna con esta opción.

VISTAS

Las vistas son tablas virtuales ‘que contienen’ el resultado de una consulta SELECT, tienen la misma estructura que una tabla cualquiera, es decir, están organizadas por filas y columnas. Una de las principales ventajas de utilizar vistas procede del hecho de que la vista no almacena los datos, sino que hace referencia a una o varias tablas de origen mediante una consulta SELECT, consulta que se ejecuta cada vez que se hace referencia a la vista. De esta forma, cualquier modificación que se realice sobre los datos de las tablas de origen es inmediatamente visible en la vista, cuando ésta vuelve a ejecutarse. Su sintaxis es:

```
CREATE [OR REPLACE] VIEW Nombre_vista
[(Lista de columnas)]
AS SELECT[...]
```

La opción REPLACE, lo que hace es, reemplazar la vista en el caso de que esta ya exista. Las vistas se utilizan de forma análoga a las tablas, permitiendo realizar consultas sobre las vistas, también se pueden realizar sentencias DML sobre las vistas, sin embargo, las modificaciones, borrados e inserciones están restringidas a vistas que estén definidas sobre una única tabla.

Borrado de una vista.

La orden para borrar una vista es DROP VIEW. Su sintaxis es:

```
DROP VIEW Nombre_vista
```

Creación de un índice

Los índices sirven para mejorar el rendimiento de las consultas. El optimizador de Oracle los utiliza implícitamente y se actualizan de forma automática al actualizar las filas.

En general, los índices se crean sobre todas las claves externas y sobre los criterios de búsqueda actuales.

```
CREATE [unique] INDEX nombre_indice
ON nombre_tabla (columnas [{asc | desc}] [,.....])
[TABLESPACE Nombre_Tablespace]
```

Borrado de un índice

Cuando se borra una tabla, automáticamente se borran los índices asociados a ella. Los índices ocupan espacio dentro de la BD como si de una tabla se tratara y por esa razón se aconseja tener solo como índices aquellas columnas por las cuales se realizan consultas de forma periódica. Para borrar un índice se utiliza la orden:

```
drop index nombre_indice;
```

6. Lenguaje de manipulación de datos (DML)

6.1. SQL

6.1.1. Instrucciones de actualización

Las instrucciones de actualización son aquellas que no devuelven ningún registro, sino que son las encargadas de acciones como añadir, borrar y modificar registros. En este post veremos las órdenes INSERT, DELETE y UPDATE.

- **INSERT**: sirve para insertar registros en una tabla
- **DELETE**: permite eliminar registros de una tabla.
- **UPDATE**: permite modificar registros de una tabla.

Inserción de registros

La sentencia INSERT nos permite introducir nuevas filas en una tabla de la base de datos. Su sintaxis más simple es:

```
Insert into tabla ([<lista_campos>]) Values ([<lista de="de" valores="valores">])
```

donde tabla representa la tabla a la que queremos añadir el registro y los valores que siguen a la cláusula VALUES son los valores que damos a los distintos campos del registro. Si no se especifica la lista de campos, la lista de valores debe seguir el orden de todos los campos de la tabla.

La lista de campos a rellenar se indica si no queremos rellenar todos los campos. Los campos no llenados explícitamente con la orden INSERT, se llenan con su valor por defecto (DEFAULT) o bien con NULL si no se indicó valor alguno.

```
Insert into tabla ([<lista_campos>])
```

```
Select .....
```

En esta segunda sintaxis se permite añadir registros a una tabla obteniéndolos mediante una consulta SELECT. Por supuesto el tipo de los campos y el orden de estos debe coincidir con los de la lista de campos o con los de la tabla destino si estos últimos no se indican.

Borrado de registros

La sentencia DELETE nos permite borrar filas de una tabla de la base de datos. Su sintaxis más simple es:

```
Delete [from] tabla [Where <condición>]
```

La sentencia DELETE es de tipo DML mientras que la sentencia TRUNCATE es de tipo DDL; la diferencia está en dos aspectos:

- DELETE puede borrar 0, 1 o más registros de una tabla, mientras que TRUNCATE los borra todos.
- DELETE puede disparar un trigger de tipo DELETE asociado a la tabla con la que estemos trabajando, mientras que TRUNCATE no disparará ningún trigger.

Una vez que se han eliminado los registros utilizando la sentencia DELETE, no puede deshacer la operación. Si desea saber qué registros se eliminarán, primero examine los resultados de una consulta de selección que utilice el mismo criterio y después ejecute la consulta de borrado. Mantenga copias de seguridad de sus datos en todo momento. Si elimina los registros equivocados podrá recuperarlos desde las copias de seguridad. Hay que tener en mucho cuidado con la restricción de ON DELETE CASCADE.

Modificación de registros

La sentencia UPDATE nos permite modificar filas de una tabla de la base de datos. Su sintaxis más simple es:

```
Update tabla Set columna1= valor1 [, columna2= valor2, .....] [Where <condición>]
```

Se modifican las columnas indicadas en el apartado SET con los valores indicados. La cláusula WHERE permite especificar qué registros serán modificados.

Su segundo tipo de sintaxis es:

```
Update tabla Set columna1=(Sentencia SELECT) [Where <condición>]
```

Este tipo de actualizaciones sólo son válidas si la Sentencia SELECT devuelve un único valor, que además debe de ser compatible con la columna que se actualiza.

Sentencia MERGE

Este comando sirve para actualizar los valores de los registros de una tabla a partir de valores de registros de otra tabla o consulta. Permite pues combinar los datos de dos tablas a fin de actualizar la primera. Su sintaxis es:

```
MERGE INTO tabla alias
USING (instrucción SELECT) alias
ON (condición de unión)
WHEN MATCHED THEN
    UPDATE SET col1=valor1 [col2=valor2]
WHEN NOT MATCHED THEN
    INSERT (listaDeColumnas)
VALUES (listaDeValores)
```

MERGE compara los registros de ambas tablas según la condición indicada en el apartado ON. Compara cada registro de la tabla con cada registro del SELECT. Los apartados de la sintaxis significan lo siguiente:

- **tabla** es el nombre de la tabla que queremos modificar.

- **USING.** En esa cláusula se indica una instrucción SELECT que muestre una tabla que contenga los datos a partir de los cuales se modifica la tabla.
- **ON.** permite indicar la condición que permite relacionar los registros de la tabla con los registros de la consulta SELECT.
- **WHEN MATCHED THEN.** El UPDATE que sigue a esta parte se ejecuta cuando la condición indicada en el apartado ON sea cierta para los dos registros actuales.
- **WHEN NOT MATCHED THEN.** El INSERT que sigue a esta parte se ejecuta para cada registro de la consulta SELECT que no pudo ser relacionado con ningún registro de la tabla.

6.1.2. Consulta de Datos - Cláusula Select

La instrucción DML más utilizada es la de consulta de datos SELECT. Su función principal es la de recuperar filas de la tabla o tablas. Además, esta sentencia es capaz de realizar las siguientes funciones:

- Obtener datos para la creación de una tabla.
- Realizar operaciones estadísticas.
- Definir cursos.
- Realizar operaciones totalizadoras sobre grupos de valores que tienen los mismos valores en ciertas columnas.
- Se puede utilizar como sub-consulta para formar parte de una condición.

La sintaxis básica es:

```
SELECT select_list
      FROM table_source
      [WHERE search_condition]
      [GROUP BY group_by_expression]
      [HAVING search_condition]
      [ORDER BY order_expression [ASC | DESC] ]
```

Vamos a ir viendo las diferentes cláusulas que componen la sentencia SELECT:

- Cláusula **SELECT** : Especifica qué columnas o expresiones han de ser devueltas por la consulta
- Cláusula **FROM**: En esta cláusula se indican la tabla o tablas a las que vamos a tener acceso.
- Cláusula **WHERE**: Selecciona aquellas filas que cumplen la condición especificada
- Cláusula **GROUP BY**: Agrupa las filas seleccionadas por la cláusula WHERE
- Cláusula **HAVING**: Especifica una condición de selección para un grupo
- Cláusula **ORDER BY**: Ordena las filas seleccionadas por la cláusula WHERE

7. Conclusión

A modo de síntesis, se podría destacar el éxito en el sector productivo de los SGBD, cuya utilidad hoy en día, ya nadie cuestiona. Estos ofrecen, en su mayoría, lenguajes de manipulación de datos declarativos, con los que los programadores de aplicaciones pueden abstraerse de cómo se realizan las consultas, e incluso, de cómo se optimizan. En este sentido cabe destacar SQL, como el lenguaje más utilizado, aunque también merece fijarse en el surgimiento de nuevas propuestas, a nivel comercial, de nuevos lenguajes y la existencia de herramientas de mapeo, como los ORM, para mapear SQL con lenguajes orientados a objetos.

7.1. Relación del tema con el sistema educativo actual

Este tema es aplicado en el aula en los módulos profesionales siguientes, con las atribuciones docentes indicadas (PES/SAI):

Formación profesional básica

- Operaciones auxiliares para la configuración y la explotación(TPB en Informática de Oficina/ TPB en informática y Comunicaciones) (PES/SAI)
- Ofimática y archivo de documentos (TPB en Informática de Oficina) (PES/SAI)

Grado Medio

- Aplicaciones ofimáticas (GM de SMR) (PES/SAI)

Grado Superior

- Gestión de bases de datos (ASIR) (PES)
- Bases de Datos (DAW/DAM) (PES)

Bachillerato:

- 4º ESO – Tecnología de la Información y la comunicación (PES)
- Bachillerato – Tecnologías de la Información y la Comunicación (PES)

8. Bibliografía

- C.J. Date: **Introducción a los sistemas de bases de datos** Pearson, 2001.
- Elmasri, R.A. y Navathe S.B: "**Fundamentos de Sistemas de Bases de Datos**".Addison-Wesley, 3^a Edic, 2002.
- Olga Pons, J M Medina, M.A. Vila. **Introducción a los Sistemas de Bases de Datos** Edt Paraninfo (2005)

- Korth, H.F. y Silberschatz: "**Fundamentos de Bases de Datos**". McGraw -Hill, 4^a Edic., 2002.
- Garcia-Molina, H.; Ullman, J.D.; Widom, J. **Database systems: the complete book** - Pearson Education Limited, 2013.
- Abraham Silberschatz, Henry F. Korth, y S. Sudarshan, **Fundamentos de bases de datos** Edt. Mc Graw-Hill (2014)
- <https://elbauldelprogramador.com/> (2020)
- www.Unir.net (2020)