

La manipulación de datos.
Operaciones. Lenguajes.
Optimización de consultas.

TEMA 36

ABACUS NT

Oposiciones 2021

Índice

- 1. Introducción.**
- 2. Álgebra relacional**
- 3. Lenguajes para la definición y manipulación de datos**
 - 3.1. Lenguajes para la definición de datos**
 - 3.2. Lenguajes para la manipulación de datos**
 - 3.3. Características**
- 4. Lenguaje de manipulación de datos (DML)**
 - 4.1. SQL**
 - 4.1.1. Instrucciones de actualización
 - 4.1.2. Consulta de Datos - Cláusula Select
 - 4.2. PL/SQL**
 - 4.2.1. Transacciones PL/SQL
 - 4.3. Sistemas NoSQL**
 - 4.3.1. Manipulación de Datos en MongoDB
 - 4.3.2. BSON-JSON
 - 4.3.3. Manipulación de Datos
- 5. Optimización de consultas**
 - 5.1. Aspectos generales**
 - 5.2. Transformaciones lógicas**
 - 5.3. Evaluación de los componentes de una consulta**
 - 5.4. Selección del plan de acceso**
- 6. Conclusiones**
 - 6.1. Sistema educativo**
- 7. Bibliografía**

1. Introducción.

SQL (Structured Query Language). Es un lenguaje **declarativo** que **incluye** tanto el DML, como el DDL. Hoy en día es un **estándar de facto**, y es utilizado por la mayoría de los SGBD relacionales comerciales, como Oracle, o MySQL, entre otros.

Pero SQL, también es un **estándar de iure**. Así, en 1986, ANSI (American National Standards Institute) e ISO (International Standards Organization) elaboran un estándar común para el lenguaje SQL, el SQL86 o SQL1. En 1992 se mejoró significativamente, obteniéndose SQL92 o SQL2, y en 1999, se publicó un nuevo estándar, SQL99 o SIQ3 , que incluye, según los autores Abraham Silberschatz, Henry F. Korth, y S. Sudarshan, en su libro Fundamentos de bases de datos (2014), las siguientes características para el soporte a la orientación a objetos:

- Relaciones anidadas
- Tipos complejos
- Herencia
- Tipos de referencia
- Funciones y procedimientos

2. Álgebra relacional

El lenguaje DML está basado en el álgebra relacional. Las *operaciones básicas* del álgebra relacional son:

Selección: Extraer las tuplas especificadas de una relación (filas).

Proyección: Extraer atributos especificados de una relación (columnas).

Producto: Construye una relación de otras dos combinando de todas las formas posibles pares de tuplas, una de cada relación.

Unión: Construye una relación uniendo las tuplas que aparecen en una o ambas relaciones.

Intersección: Construye una relación con las tuplas que aparecen en ambas relaciones a la vez.

Diferencia: Construye una relación con las tuplas que aparecen en la primera y no en la segunda.

Enlace: Construye una relación de otras dos enlazando todas los posibles pares de tuplas, de forma que se cumpla una condición específica.

División: Tomando dos relaciones, una binaria (dos atributos) y otra unaria (uno sólo), construye una relación consistente en todos los valores de la relación binaria que tienen un enlace en la otra relación.

Mediante este conjunto de ocho operaciones, se puede construir cualquier relación a partir de las ya dadas. El lenguaje resultante es altamente amigable y fácil de aprender. Plantea, sin embargo, el inconveniente de que es difícil el determinar la manera óptima de crear una relación con estas primitivas, ya que suelen existir varias opciones posibles, con costes distintos de utilización del ordenador.

3. Lenguajes para la definición y manipulación de datos

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.

Generalmente, los DDL de los diferentes SGBD son lenguajes muy simples basados en una gramática muy sencilla.

La representación de los datos compilación es almacenada en denominado **Diccionario de Datos**.

el DDL cuenta con un **sublenguaje** encargado del control y seguridad de los datos, el cual se denomina **Lenguaje de Control de Datos (DCL)** y permite el control del acceso de a la información almacenada en el diccionario de datos (definición de privilegios y tipos de acceso), así como el control de seguridad de los datos.

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.

Al igual que el DDL, el DML está basado en un modelo de datos y, por tanto, los SGBD basados en distintos modelos de datos tienen diferente DML. Se trata también de un lenguaje basado en una gramática completa, sencilla y, generalmente, fácil de entender por usuarios no expertos.

Los SGBD también son capaces de procesar peticiones de DML que se han formulado desde programas escritos en otros lenguajes de programación

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 gráfico.
- **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 de manipulación de datos (DML)

4.1. SQL

4.1.1. 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 llenar se indica si no queremos llenar 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 triger de tipo DELETE asociado a la tabla con la que estemos trabajando, mientras que TRUNCATE no disparará ningún triger.

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.

4.1.2. 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 cursor.
- 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

4.2. PL/SQL

En SQL el trabajo con la base de datos se ha hecho de forma interactiva: el usuario introduce una orden y Oracle proporciona una respuesta. Esta forma de trabajar no resulta operativa en un entorno de producción, porque todos los usuarios no conocen ni utilizan SQL, y además suelen producirse frecuentes errores.

Para superar estas limitaciones, Oracle incorpora un gestor PL/SQL en el servidor de la BD y en algunas de sus herramientas (Forms, Reports, Graphics, etc.). Este lenguaje incorpora todas las características propias de los lenguajes de tercera generación: manejo de variables, estructura modular (procedimientos y funciones), estructuras de control (bifurcaciones, bucles y demás estructuras), control de excepciones, y una total integración en el entorno Oracle.

Los programas creados con PL/SQL se pueden almacenar en la base de datos como un objeto más de ésta; así se facilita a todos los usuarios autorizados el acceso a estos programas, y en consecuencia, la distribución, instalación y mantenimiento de software. Además, los programas se ejecutan en el servidor, suponiendo un significativo ahorro de recursos en los clientes y de disminución del tráfico de red.

El uso del lenguaje PL/SQL es también imprescindible para construir disparadores de bases de datos, que permiten implementar reglas complejas de negocio y auditoria en la BD.

PL/SQL soporta todos los comandos de consulta y manipulación de datos, aportando sobre SQL las estructuras de control y otros elementos propios de los lenguajes procedimentales de tercera generación. Su unidad de trabajo es el bloque, formado por un conjunto de declaraciones, instrucciones y mecanismos de gestión de errores y excepciones.

4.2.1. Transacciones PL/SQL

Oracle es un sistema de base de datos puramente transaccional, de tal forma, que la instrucción BEGIN TRANSACTION no existe.

Una transacción es un conjunto de sentencias SQL que se ejecutan en una base de datos como una única operación, confirmándose o deshaciéndose todo el conjunto de

sentencias SQL. La transacción puede quedar finalizada (con las sentencias apropiadas) o implícitamente (terminando la sesión).

Durante la transacción, todas las modificaciones que hagamos sobre base de datos, no son definitivas, más concretamente, se realizan sobre un tablespace especial que se denomina tablespace de ROLLBACK, o RBS (RollBack Segment). Este tablespace tiene reservado un espacio para cada sesión activa en el servidor, y es en ese espacio donde se almacenan todas las modificaciones de cada transacción. Una vez que la transacción se ha finalizado, las modificaciones temporales almacenadas en el RBS, se vuelcan al tablespace original, donde está almacenada nuestra tabla. Esto permite que ciertas modificaciones que se realizan en varias sentencias, se puedan validar todas a la vez, o rechazar todas a la vez.

Dentro de una transacción se pueden crear los llamados “punto de control” mediante la sentencia:

```
SAVEPOINT Nombre_punto_control;
```

Las sentencias de finalización de transacción son:

- **COMMIT**: la transacción termina correctamente, se vuelcan los datos al tablespace original y se vacía el RBS.
- **ROLLBACK**: se rechaza la transacción y el vacía el RBS. Cualquier cambio realizado desde que se inició la transacción se deshace, quedando la base de datos en el mismo estado que antes de iniciarse la transacción.

A la hora de hacer un ROLLBACK o un COMMIT se podrá hacer hasta cierto punto con la sintaxis:

```
COMMIT TO punto_control;  
ROLLBACK TO punto_control;
```

Cuando tenemos abierta una sesión (WorkSheet de Oracle por ejemplo), los cambios que realizamos no son visibles a otra sesión hasta que no hagamos un COMMIT. Este se puede realizar de forma manual, ejecutando el comando COMMIT; o bien, de forma automática, cuando cerramos la sesión.

En una transacción los datos modificados no son visibles por el resto de usuarios hasta que se confirme la transacción.

Si alguna de las tablas afectadas por la transacción tiene triggers, las operaciones que realiza el trigger están dentro del ámbito de la transacción, y son confirmadas o deshechas conjuntamente con la transacción. Durante la ejecución de una transacción,

una segunda transacción no podrá ver los cambios realizados por la primera transacción hasta que esta se confirme.

4.3. Sistemas NoSQL

Los sistemas de bases de datos NoSQL crecieron con las principales redes sociales, como Google, Amazon, Twitter y Facebook. Estas tenían que enfrentarse a desafíos con el tratamiento de datos que las tradicionales SGBDR no solucionaban. Con el crecimiento de la web en tiempo real existía una necesidad de proporcionar información procesada a partir de grandes volúmenes de datos que tenían unas estructuras horizontales más o menos similares. Estas compañías se dieron cuenta de que el rendimiento y sus propiedades de tiempo real eran más importantes que la coherencia, en la que las bases de datos relacionales tradicionales dedicaban una gran cantidad de tiempo de proceso.

En ese sentido, a menudo, las bases de datos NoSQL están altamente optimizadas para las operaciones recuperar y agregar, y normalmente no ofrecen mucho más que la funcionalidad de almacenar los registros (p.ej. almacenamiento clave-valor). La pérdida de flexibilidad en tiempo de ejecución, comparado con los sistemas SQL clásicos, se ve compensada por ganancias significativas en escalabilidad y rendimiento cuando se trata con ciertos modelos de datos.

Tipos de Bases de Datos NoSQL

- **Bases de datos clave – valor:** Es el modelo más popular y más sencilla. En este tipo de sistema, cada elemento está identificado por una llave única, lo que permite la recuperación de la información de forma muy rápida, información que habitualmente está almacenada como un objeto binario (BLOB).

Algunos ejemplos de este tipo son Cassandra, BigTable o HBase.

- **Bases de datos documentales:** Son las bases de datos NoSQL más versátiles. Se pueden utilizar en gran cantidad de proyectos, incluyendo muchos que tradicionalmente funcionarían sobre bases de datos relacionales.

Algunos ejemplos de este tipo son MongoDB o CouchDB.

- **Bases de datos en grafo:** La información se representa como nodos de un grafo y sus relaciones con las aristas del mismo, de manera que se puede hacer uso de la teoría de grafos para recorrerla. Para sacar el máximo rendimiento a este tipo de bases de datos, su estructura debe estar totalmente normalizada, de forma que cada tabla tenga una sola columna y cada relación dos.

Algunos ejemplos de este tipo son Neo4j, InfoGrid o Virtuoso.

- Bases de datos orientadas a objetos: La información se representa mediante objetos, de la misma forma que son representados en los lenguajes de programación orientada a objetos (POO) como ocurre en JAVA, C# o Visual Basic .NET.

Algunos ejemplos de este tipo de bases de datos son Zope, Gemstone o Db4o.

Como ejemplo para este tema , y debido a la diversidad en que el Universo NoSQL se definen los datos, nos centramos en un de los sistemas NoSQL más conocidos y utilizados en la actualidad: MongoDB.

4.3.1. Manipulación de Datos en MongoDB

MongoDB es un tipo de base de datos de esquema flexible. Esto significa que ha de ser el usuario el que el que determina y declara el esquema de cada una de las tablas justo en el momento de realizar las inserciones. Esta funcionalidad la comparten en cierta forma muchas bases de datos NoSQL.

Esto es bueno y malo a la vez. Por un lado, se tiene la flexibilidad de añadir campos cuando son necesarios sin haberlos tenido que haberlos definido de antemano, o obviar algunos en los casos en los que no sean necesarios, pero a la vez se obliga al programador a ser mucho más metódico en su trabajo, puesto que el sistema no avisará de un posible error o despiste en el código.

Las bases de datos en MongoDB residen en un host que puede alojar varias bases de datos de forma simultánea almacenadas de forma independiente. Cada una de estas bases de datos puede contener una o más colecciones, a partir de las cuales se almacenarán los objetos o documentos.

4.3.2. BSON-JSON

BSON es un formato binario que se utiliza para almacenar la información en MongoDB. BSON es la codificación binaria del formato JSON. Esta codificación ha sido elegida porque presenta ciertas ventajas a la hora de almacenar los datos, como la eficiencia o la compresión.

Básicamente BSON y JSON son los formatos con los que trabaja MongoDB: JSON es el formato con el que se presenta la información a los usuarios y a las aplicaciones y BSON el formato que utiliza MongoDB de forma interna.

4.3.3. Manipulación de Datos

En MongoDB un documento es un registro compuesto por pares “campo: valor”. Estos documentos son muy parecidos a los objetos JSON.

Los valores de los campos pueden ser otros documentos, arreglos-arrays y arreglos-arrays de documentos. Las ventajas de tener este tipo de estructura de datos como registros es

que principalmente podemos reducir las “relaciones” entre tablas que comúnmente usamos en bases de datos relacionales cuando hacemos consultas con “joins” o uniones de datos, ya que resulta muy costoso. Otra ventaja es que de cierta forma esta estructura se corresponde con tipos de datos en varios lenguajes de programación lo cual resulta conveniente para desarrollar código de forma más ágil. Además, también nos permite guardar grandes volúmenes de datos sin tanto costo.

5. Optimización de consultas

5.1. Aspectos generales

Los diferentes pasos y partes del motor de SGBD para **procesar una consulta** expresada en un lenguaje de alto nivel, como SQL, son:

- **Análisis léxicos:** Se identifican los símbolos o **tokens** del lenguaje en la consulta.
- **Análisis sintáctico:** Se analiza si la consulta cumple con la **gramática** del lenguaje.
- **Validación:** Se valida que la consulta tenga sentido desde un punto de vista **semántico**.
- **Optimización de la consulta:** A continuación, se crea una representación interna de la consulta, normalmente en forma de estructura de **árbol de consulta**, y el optimizador se encarga de seleccionar, de entre todas las posibles **estrategias de ejecución**, cuál cree que es la más adecuada para ejecutar dicha consulta, produciendo un **plan de ejecución**.
- **Generación de código:** Se genera el código necesario para ejecutar la consulta seleccionada.
- **Procesados de base de datos en tiempo de ejecución:** Obtiene el resultado final.

5.2. Transformaciones lógicas

Para una misma consulta pueden existir varias expresiones semánticamente idénticas. Es por ello que el optimizador transforma las expresiones en otras equivalentes, con el objetivo de facilitar la optimización. Para ello, se suelen aplicar los siguientes heurísticos:

- Convertir las **secuencias de proyecciones** en una sola proyección.
- Convertir las **secuencias de restricciones** en una sola restricción.
- Mover, dentro de las expresiones, las operaciones **selectivas**, como la proyección y la selección, de forma que se ejecuten antes de que las operaciones **constructivas**, como la combinación o el producto cartesiano.

5.3. Evaluación de los componentes de una consulta

El optimizador debe estimar y comparar la ejecución de consultas en base a una función. Para ello, es importante saber que el costo real de ejecución de una consulta tiene, según los autores Ramez Elmasri y Shamkant B. Navathe, en su libro Fundamentos de Sistemas de Bases de Datos (2007), los siguientes componentes:

- **Costo de acceso al almacenamiento secundario:** Es el costo de buscar, leer y escribir bloques de datos que residen en el almacenamiento secundario.
- **Costo de almacenamiento intermedio:** coste del almacenamiento de información en ficheros intermedios necesarios para generar una estrategia de ejecución de consultas.
- **Costo de cómputo:** Es el costo de ejecutar operaciones en la memoria durante la ejecución de la consulta.
- **Costo de uso de memoria:** Éste es el costo relacionado con el número de buffers de memoria que se necesitan durante la ejecución de la consulta.
- **Costo de comunicación:** Es el costo de enviar la consulta y sus resultados entre el sitio de la base de datos y el terminal donde se originó la consulta.

Dependiendo del tipo de base de datos, un factor tendrá más peso que otro. Es por ello que, es difícil diseñar una función ponderada asignándole los pesos adecuados. Por esta razón, muchas funciones de costo consideran solo un factor, el acceso a disco.

5.4. Selección del plan de acceso

Una vez estimado el coste de cada uno de los posibles planes de acceso para cada árbol equivalente, hay que elegir el menor de todos ellos.

En muchas ocasiones, se aplican algoritmos de acotación y poda, para que, una vez se sepa que un plan va a superar el coste de otro plan, se detenga el cálculo de su estimación.

6. Conclusiones

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.

6.1. Sistema educativo

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)

7. 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)

