AUTORES

MARIBEL CAMPOS MONGE

Ingeniera Técnica en Informática de Gestión por la Universidad Politécnica de Valencia (UPV). Profesora de Secundaria de Informática desde 2004. Especialista en Redes Informáticas y Servicios en Red. Coordinadora e Instructora de la academia CISCO.

Experiencia como miembro del Tribunal de Oposiciones de Secundaria de Informática de Valencia. Preparadora de Oposiciones de Secundaria de Informática.

EVA MARÍA CAMPOS MONGE

Ingeniera Superior en Informática por la Universidad Politécnica de Valencia (UPV) superando la intensificación en Gestión. Diploma de Estudios Avanzados y Suficiencia Investigadora en Programación Declarativa e Ingeniería de la Programación por la Universidad Politécnica de Valencia (UPV).

Experiencia como investigadora en el área de Ingeniería del Software en el Departamento de Sistemas Informáticos y Computación (DSIC) de la Universidad Politécnica de Valencia (UPV).

Profesora de Secundaria de Informática desde 2002 especializada en Bases de Datos, Redes Informáticas y Servicios en Red. Miembro del Equipo Directivo. Coordinadora e instructora de Academia CISCO. Preparadora de Oposiciones de Secundaria de Informática.

JORGE LÓPEZ QUEROL

Ingeniero Técnico en Informática de Gestión por la Universidad Politécnica de Valencia (UPV). Maestro de primaria y profesor de secundaria de Informática desde 2008. Especialista en Redes Informáticas y programación. Coordinador e instructor de academia CISCO. Preparador de Oposiciones de Secundaria de Informática.

INTRODUCCIÓN BLOQUE III

Este es el tercero de un total de cuatro libros que forman el temario actualizado de Oposiciones de Secundaria de Informática. Su objetivo fundamental es el de facilitar al opositor la preparación de la prueba escrita.

El contenido de este libro está desarrollado basándose en la legislación actual que regula el contenido de estas pruebas.

Este volumen contiene desde el tema 39 hasta el 55. En estos temas se tratan los conceptos más avanzados de Bases de Datos y el bloque de Ingeniería del Software ofreciendo un contenido totalmente actualizado.

Cada uno de los temas consta de un índice que presenta el esquema general del tema, la introducción, el desarrollo del tema en cuestión, una conclusión y bibliografía/webgrafía.

En la prueba escrita es recomendable que se introduzca el punto de bibliografía/webgrafía al final del tema. En este libro se presenta la bibliografía agrupada por bloques con el fin de facilitar al opositor la tarea de recordarla.

Los temas se presentan de forma acotada para que el opositor sea capaz de desarrollarlo en el tiempo estipulado, asegurando que se tratan todos los puntos de interés con la profundidad adecuada.

Los temas pertenecientes al mismo bloque tienen contenidos en común, lo que permitirá al opositor rentabilizar tiempo de estudio y poder amortizar píldoras de conocimiento aplicables a distintos temas.

Además este volumen viene acompañado de material adicional en el que el lector puede encontrar trucos sobre cómo afrontar el examen, ejemplos para añadir a los temas, contextualización en los ciclos formativos y otros recursos de interés.

TEMA XXXIX. LENGUAJES PARA LA DEFINICIÓN Y MANIPULACIÓN DE DATOS EN SISTEMAS DE BASES DE DATOS RELACIONALES. CARACTERÍSTICAS LENGUAJE SQL

Tema 39

LENGUAJES PARA LA DEFINICIÓN Y MANIPULACIÓN DE DATOS EN SISTEMAS DE BASES DE DATOS RELACIONALES. CARACTERÍSTICAS LENGUAJE SQL

39.1 INTRODUCCIÓN

Las bases de datos se han convertido en parte fundamental de nuestras vidas. Cualquier ciudadano interacciona con bases de datos diariamente sin darse cuenta. Por ejemplo, cuando alguien acude a un cajero a sacar dinero o consultar nuestro saldo, cuando realiza una compra por Internet o al hacer una consulta en Google.

Una base de datos (BD) es un conjunto de datos relacionados entre sí que se adecua a una determinada organización del mundo real. Estos datos son accedidos, insertados y manipulados por medio de un Sistema Gestor de Base de Datos (SGBD), que es el software dedicado a este fin.

Cada BD está basada en un modelo de datos concreto. El modelo relacional es el más utilizado en el entorno empresarial donde se desea que la información sea íntegra en todo momento y esté segura. Este modelo se basa en el álgebra relacional. Al basarse en un modelo matemático ofrece una base más robusta que los modelos de datos anteriores. Además, ofrece mucha más flexibilidad.

Los lenguajes diseñados para las bases de datos relacionales son los siguientes:

Lenguaje comercial	Lenguaje teórico en el que se basa
QBE (Query By Example)	Cálculo relacional de dominios
QUEL	Cálculo relacional de tuplas
SQL	Álgebra relacional

SQL fue el único lenguaje que logró ser estandarizado por ISO, y así se convirtió en el lenguaje de bases de datos relacionales por excelencia.

SQL es un lenguaje declarativo, es decir, el usuario indica con sus órdenes SQL qué quiere obtener o qué operación quiere realizar, pero no cómo. Es el SGBD el encargado de realizar la búsqueda de la información solicitada entre los datos almacenados en la BD.

El lenguaje SQL tiene la característica de que se puede utilizar de varias maneras:

- **Interactiva**. En este modo las órdenes SQL se pueden ejecutar directamente en el SGBD. Se trata de órdenes SQL puras, sin código de otros lenguajes.
- **Embebido**: en este modo el SQL está integrado entre órdenes de otros lenguajes de programación. Por ejemplo, se puede realizar un programa en Java que tenga interacción con una BD y se intercalen entre las instrucciones Java, instrucciones SQL.
- **Procedural**: Algunos de los principales SGBDs crearon un lenguaje procedural para añadirle aún más potencia al lenguaje SQL. Por ejemplo, ORACLE creó PL/SQL, Postgre el PL/psSQL, etc. Estos lenguajes añaden la posibilidad de definir procedimientos almacenados directamente en la BD, cursores.

ôComo SQL es el lenguaje de BD relacionales por excelencia, esta unidad se explicará basándose siempre en él.

El lenguaje SQL se ha definido históricamente en:

- DDL: Lenguaje de definición de datos
- DML: Lenguaje de manipulación de datos
- DCL: Lenguaje de control de datos

Sin embargo, con el paso del tiempo hay bibliografía que hace una división más exhaustiva y el DML lo divide en lenguaje de manipulación de datos y lenguaje de consulta de datos dándole a la consulta la importancia que tiene al ser la herramienta más potente de SQL. Además, existe el lenguaje de transacción de datos que pertenece al SQL extendido y permite la ejecución de varias acciones como si fuera una sola.

Con el avance del uso de las bases de datos y la necesidad de poder añadir comportamiento a la base de datos, cada SGBD desarrolló un lenguaje procedimental que permiten la definición de procesos o disparos en el propio sistema gestor. ORACLE fue el pionero con su PL/SQL.

Sublenguajes de SQL y sus principales operaciones

39.2 LENGUAJE DE DEFINICIÓN DE DATOS (DDL)

El lenguaje de definición de datos es el conjunto de instrucciones que permiten la definición y actualización de las estructuras de la BD. Por tanto, permitirán la definición de tablas, de las columnas de las tablas, los dominios de datos, restricciones, vistas…

El lenguaje de definición de datos consta de tres instrucciones generales:

- CREATE
- ALTER
- DROP

Que permiten la creación, la modificación y el borrado de la estructura en cuestión.

39.2.1 Definición del esquema

El esquema de BD es como una especie de contenedor que va a englobar todos los componentes de la BD:

- Dominios de datos
- Tablas
- Vistas
- Restricciones
- …

```
CREATE SCHEMA <nombre_esquema> AUTHORIZATION
<nom_susuario>
[<elemento de esquema>…];
```

39.2.2 Definición de dominio

La definición de un dominio con un nombre determinado establece el conjunto de datos posible para los campos que se definan en ese dominio.

```
CREATE DOMAIN <nombre de dominio> [ AS] <tipo de datos>
[DEFAULT <opción por defecto>]
[ CONSTRAINT <nombre de restricción> CHECK
(condición_restricción)];
```

Para cada dominio se establece un nombre de dominio que tendrá un tipo de datos predefinido. Además, en el campo opción por defecto se puede indicar el valor que adoptará el dominio en caso de no indicarse ningún valor, y puede ser un valor literal, un valor tiempo/fecha, o los valores de las variables USER, SYSTEM o NULL.

También puede definirse una restricción que deba cumplir el dominio. La restricción que se defina en este nivel hará que cada dato que se incluya en un campo definido en este dominio deba cumplir la condición establecida.

39.2.3 Definición de tabla

La tabla es la estructura que representa las relaciones del esquema relacional.

Cada fila de la tabla se denomina tupla y debe ser única, y para ello es obligatorio definir un identificador que ayude a evitar duplicados. Este identificador es la **clave primaria**. Toda tabla debe definirse indicando qué atributo o conjunto de atributos conforman su clave primaria.

Los atributos de la relación se corresponden con las columnas de la tabla.

La sintaxis para la creación de una tabla es la siguiente:

```
CREATE TABLE <nombre de tabla>(
<nombre de columna> { <tipo de datos> | <nombre de dominio>}
[ DEFAULT <valor por defecto de la columna>]
[ NOT NULL | CHECK (condición restricción) | UNIQUE | PRIMARY KEY | REFERENCES
<nombre tabla ajena> |
<restricción de tabla>
);
```

Como se puede observar en la definición de la tabla se indicará el nombre de la tabla y, entre paréntesis, se indicarán las columnas con su tipo de datos o dominio asociado. Además, se pueden indicar condiciones en ese atributo en particular. Por ejemplo, puede indicarse si ese valor no puede tener valor nulo, si su valor debe ser único (no puede repetirse en distintas tuplas), si se trata de la clave primaria o una clave ajena.

La clave ajena es como una especie de puntero entre atributos de diferentes tablas que se utiliza para definir las relaciones entre las distintas tablas del esquema de nuestra BBDD. Si solo es un atributo puede definirse como una restricción de columna indicando a qué columna de qué tabla hará referencia Además, podrá indicarse si, en

caso de modificación o borrado del atributo al que hace referencia la modificación causará el borrado del valor de esta clave ajena, su modificación para adaptarse a este nuevo valor, la actualización por un valor por defecto o, simplemente, no podrá modificarse el atributo al que hace referencia si tiene alguna clave ajena apuntando a él. El atributo o atributos que forman la clave ajena siempre referencian al atributo o atributos que forman la clave primaria de la tabla referenciada.

Además de la definición de las distintas columnas junto con sus restricciones, dentro de la definición de la tabla también se pueden definir restricciones de tabla que pueden ser:

- La definición de una clave primaria

```
PRIMARY KEY (lista columnas que la conforman separadas por coma)
```

- La definición de una clave ajena

```
FOREIGN KEY < (lista columnas que la conforman separadas por coma)>
REFERENCES <columnas y tablas referenciadas>
[(ON DELETE | ON UPDATE) (RESTRICT | CASCADE |SET NULL |SET DEFAULT)
```

  En este caso se define el conjunto de columnas que conforman la clave ajena, y se indica a qué atributos de qué tabla hace referencia, así como el comportamiento que tendrá dicha clave ajena si la tupla o fila a la que hace referencia es borrada o modificada.

- La definición de una clave única

```
UNIQUE (lista columnas que la conforman separadas por coma)
```

- Restricción de valor NO NULO

```
NOT NULL (lista columnas que la conforman separadas por coma)
```

- Restricción general

```
CHECK (condiciones que deben cumplirse);
```

La orden para el borrado de una tabla sigue la siguiente sintaxis:

```
DROP TABLE <nom_tabla>;
```

También se puede modificar la estructura de la tabla añadiendo, eliminando o modificando columnas. La sintaxis es la siguiente:

```
ALTER TABLE <nom_tabla>
ADD COLUMN <definición de columna>;

ALTER TABLE <nom_tabla>
DROP COLUMN <nombre_columna>;

ALTER TABLE <nom_tabla>
MODIFY COLUMN <definición de columna>;
```

39.2.4 Definición de vista

La vista es una estructura que permite mostrar al usuario el resultado de una consulta (SELECT) como si fuera una tabla. La definición de una vista sigue la siguiente sintaxis:

```
CREATE VIEW <nombre de tabla>
(lista columnas que la conforman separadas por coma) AS <expresión de consulta
(SELECT)>
[ WITH CHECK OPTION];
```

Las vistas son muy útiles, entre otras cosas, para que cada perfil de usuario definido en el SGBD tenga una visión parcial de los datos almacenados, así se puede mostrar y ocultar la información según el uso de ese tipo de usuario.

Cabe resaltar que en alguna bibliografía la definición de vista se clasifica dentro del lenguaje de control de datos alegando al mecanismo de seguridad que ofrecen pudiendo mostrar parcialmente los datos a los usuarios según sus perfiles.

39.2.5 Definición de restricción de integridad

Se pueden definir restricciones de integridad de carácter general que deben cumplirse en todo momento. Estas restricciones pueden implicar a una o varias tablas.

```
CREATE ASSERTION <nombre de restricción>
CHECK (<condición de búsqueda>)[<atributos de     restricción>];
```

39.2.6 Definición de índices

Un índice es una estructura de datos que permite el acceso a ciertos datos de la BD más rápidamente.

```
CREATE INDEX <nom_índice> ON <nom_tabla> (lista_columnas);
```

Siempre debe encontrarse un equilibrio porque, aunque un índice mejora sustancialmente la velocidad de las consultas realizadas accediendo a las columnas que contiene, también requiere tiempo de computación cada vez que se hacen modificaciones en la tabla sobre la que está definido.

La sintaxis para eliminar el índice es la siguiente:

```
DROP INDEX nombre_índice;
```

39.2.7 Secuencias

Las secuencias permiten la generación de números a medida que se necesiten. Es una manera de simular un campo autonumérico.

La sintaxis en SQL es:

```
CREATE SEQUENCE nombresec
[INCREMENT BY numero]
[START WITH numero]
[MAXVALUE numero]
[MINVALUE numero]
[CYCLE | NOCYCLE]
[ORDER | NOORDER];
```

39.3 LENGUAJE DE MANIPULACIÓN DE DATOS (DML)

El lenguaje de manipulación de datos es el conjunto de instrucciones que permiten la definición y actualización de los datos de la BD. Por tanto, permitirán insertar datos, borrarlos o modificarlos.

El lenguaje de definición de datos consta de tres instrucciones generales:

- INSERT
- UDPATE
- DELETE

39.3.1 Inserción de datos

```
INSERT INTO nombre_tabla (nom_col1, nom_col2, …, nom_colN)
VALUES (valor_col1, valor_col2, … , valor_colN);
```

Inserta en la tabla "nombre_tabla", en las columnas cuyos nombres se indican entre paréntesis, los valores indicados en VALUES. Cada valor se asignará a la columna correspondiente de las indicadas, a la primera columna el primer valor y así sucesivamente.

Existe una variación más avanzada de la inserción de datos en la que en lugar de indicar directamente el valor que va a insertarse en cada una de las columnas, estos valores son resultado de una consulta a la BD.

```
INSERT INTO nombre_tabla (nom_col1, nom_col2, …, nom_colN)
<cláusula SELECT>;
```

39.3.2 Modificación de datos

```
UPDATE nombre_tabla
SET (nom_col1=valor_col1, …, nom_colN=valor_colN) [WHERE <condición>];
```

Actualizará los registros que cumplan la condición indicada en la cláusula WHERE, modificando el valor de las columnas indicadas por el nuevo valor establecido en la cláusula SET.

Si se ejecutara el UPDATE sin cláusula WHERE las modificaciones indicadas se realizarían en todas las filas de la tabla "nombre_tabla".

39.3.3 Borrado de datos

```
DELETE FROM nombre_tabla
[WHERE <condición>];
```

Se borrarán las filas de la tabla "nombre_tabla" que cumplan la condición establecida en la cláusula WHERE.

Si se ejecutara el DELETE sin cláusula WHERE se eliminarían todos los registros de la tabla "nombre_tabla" dejándola sin datos.

39.4 LENGUAJE DE CONSULTA DE DATOS (DQL)

El lenguaje de consulta de datos es el conjunto de instrucciones que permiten la consulta de los datos de la BD. Este lenguaje se puede considerar también dentro del lenguaje de manipulación de datos, aunque no realiza modificaciones en sí mismas, simplemente consulta los datos.

Consta de una única instrucción: SELECT. Se trata de la orden con mayor potencial de todo SQL. Debido a la complejidad se va a presentar de manera incremental presentando primero las consultas más sencillas y completándola poco a poco.

39.4.1 Consultas sencillas

```
SELECT [ALL | DISTINCT] <lista_columnas_separadas_por_comas
| *>
FROM <lista_tablas_separadas_por_comas> [WHERE <condición>]
[ORDER_BY <lista_columnas_separadas_por_comas>;
```

Cláusula SELECT

- Debe aparecer obligatoriamente en la consulta.
- Indica, separado por comas, las columnas que se quieren obtener.

```
SELECT columna1, columna2, …, columnaN FROM tabla1;
```

- Pueden indicarse también operaciones matemáticas realizadas sobre los valores obtenidos.

  Ejemplo[1]**:**

```
SELECT precio*1,16 FROM factura;
```

- En el caso de que se quieran obtener todas las columnas de la tabla se utiliza el comodín "*".

```
SELECT * FROM tabla1;
```

- Si la consulta es sobre varias tablas se puede indicar el nombre de la tabla en cada una de las columnas para que no haya lugar a error de la siguiente manera:

```
SELECT TABLA1.columna1, TABLA2.columna1 FROM tabla1, tabla2;
```

- En esta cláusula pueden indicarse alias para las columnas.

  Ejemplo:

```
SELECT precio*1,21 AS precio_con_iva FROM factura;
```

- Por defecto muestra todas las filas que cumplen con la condición (ALL), pero puede indicarse que sólo se desean obtener filas que son distintas. Para ello se utiliza la cláusula DISTINCT.

  Ejemplo:

```
SELECT DISTINCT departamento FROM empleados;
```

- Es la cláusula encargada de la operación de PROYECCIÓN.

1 Se van a presentar ejemplos triviales que se considera que el lector puede interpretar perfectamente sin necesidad de explicar el contexto.

Cláusula FROM

- Debe aparecer obligatoriamente en la consulta.
- Indica la/s tabla/s de las que se quieren obtener los datos (véase ejemplos del apartado anterior).
- Pueden indicarse alias para las tablas.

  Ejemplo:

```
SELECT e.nombre, c.nombre FROM EMPLEADO e, CIUDAD c;
```

- Cuando se indican varias tablas separadas por comas se está realizando la operación de PRODUCTO CARTESIANO.

Cláusula WHERE

- Es opcional.
- Indica la condición que deben cumplir los datos para mostrarlos.

  Ejemplo:

```
SELECT * FROM empleado
WHERE tipo_contrato = ”indefinido”;
```

- Admite los operadores lógicos AND, OR y NOT para poder construir condiciones más complejas.
- Puede contener otra consulta anidada.
- Siempre debe ser una condición que para cada una de las filas de la consulta tenga un valor cierto o falso. De esta manera sólo se mostrarán los datos para los que la condición tenga valor verdadero, es decir, que cumplen la condición.
- Permite utilizar los operadores de comparación =, >, >=, <, <=, != o <>
- Es la cláusula encargada de la operación de SELECCIÓN.

Cláusula INNER JOIN

Cuando se unen varias tablas en una misma consulta, pueden simplemente ponerse separadas por comas en la cláusula FROM y se conseguirá el producto cartesiano y normalmente se enlazan mediante las claves ajenas en el WHERE.

La cláusula INNER JOIN (o simplemente JOIN) ya contempla la relación entre ambas tablas y en lugar de relacionar sus claves ajenas dentro del WHERE se hace en la cláusula dedicada a tal efecto, ON.

```
SELECT e.nombre, c.nombre
FROM EMPLEADO e JOIN CIUDAD c
ON e.ciudad=c.codigo;
```

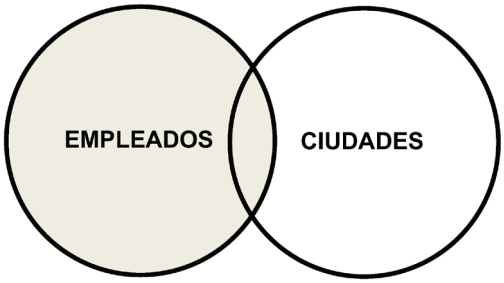
En este ejemplo se mostrarán los nombres de empleado junto con el nombre de la ciudad asociada. En caso de que algún empleado no tenga ciudad asignada ese empleado no se mostrará, tampoco las ciudades que no tengan ningún empleado asociado.

Cláusula LEFT JOIN

En la cláusula LEFT JOIN tiene prioridad la tabla que se indica a la izquierda de la orden. De la tabla que tiene preferencia se mostrarán todas las filas, y de la tabla de la derecha se muestra el valor con el que está relacionada y, en caso de no haber, se mostrará NULL.

```
SELECT e.nombre, c.nombre
FROM EMPLEADO e LEFT JOIN CIUDAD c
ON e.ciudad=c.codigo;
```

En este ejemplo se mostrarán los nombres de todos los empleados junto al nombre de la ciudad con la que está relacionado, y en caso de no tener ciudad, se mostrará el nombre del empleado junto al valor NULL.

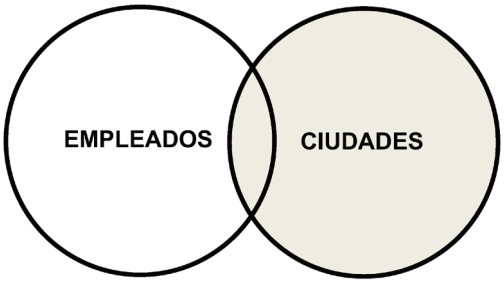


Cláusula RIGHT JOIN

En la cláusula RIGHT JOIN tiene prioridad la tabla que se indica a la derecha de la orden. Es simétrica a la LEFT JOIN. De la tabla que tiene preferencia se mostrarán todas las filas, y de la tabla de la derecha se muestra el valor con el que está relacionada y, en caso de no haber, se mostrará NULL.

```
SELECT e.nombre, c.nombre
FROM EMPLEADO e JOIN CIUDAD c
ON e.ciudad=c.codigo;
```

En este ejemplo se mostrarán los nombres de todos los empleados junto al nombre de la ciudad con la que está relacionado, y en caso que alguna ciudad no tenga ningún empleado asociado, se mostrará el nombre de la ciudad junto al valor NULL.



Cláusula ORDER BY

- Permite ordenar el resultado de la consulta en función de las columnas indicadas en esta cláusula.
- Obviamente solo puede contener un subconjunto de las columnas establecidas en la cláusula SELECT. Puede ser desde una sola columna hasta todas las especificadas (nunca ninguna otra que no aparezca en el SELECT).

  Ejemplo:

```
SELECT * FROM empleado
WHERE tipo_contrato = ”indefinido”
ORDER BY apellidos;
```

En esta consulta se obtienen los empleados cuyo tipo de contrato es indefinido ordenados de manera ascendente por el contenido del campo apellidos.

- En caso de indicarse varias columnas la ordenación se realizará en primer lugar por la primera columna indicada, a continuación, por la segunda y así sucesivamente.

  Ejemplo:

```
SELECT * FROM empleado
WHERE tipo_contrato = ”indefinido”
ORDER BY edad, apellidos;
```

  En esta consulta se obtienen los empleados cuyo tipo de contrato es indefinido ordenados de manera ascendente por edad, y dentro de cada edad por el contenido del campo apellidos.

- Por defecto se ordena de manera ascendente, aunque puede incluirse “DESC” si se desea que se ordene de manera descendente.

  Ejemplo:

```
SELECT * FROM empleado
WHERE tipo_contrato = ”indefinido”
ORDER BY edad DESC;
```

  En esta consulta se obtienen los empleados cuyo tipo de contrato es indefinido ordenados de manera descendente por edad.

39.4.2 Consultas complejas

Funciones resumen

SQL permite utilizar funciones de resumen que facilitan el cálculo de ciertos datos que pueden ser de utilidad para el usuario final. Por ejemplo, se puede obtener el número de empleados mayores de 50 años, la media de salario de los empleados de un determinado departamento, el salario máximo de la empresa…

Para ello se utilizan funciones en la cláusula SELECT, por ejemplo:

Ejemplo:

```
SELECT AVG(salario)
FROM empleados
WHERE dpto=”ventas”;
```

Ejemplo:

```
SELECT COUNT(*)
FROM empleados
WHERE edad>50;
```

Algunos ejemplos de estas funciones resumen son: AVG que obtiene la media aritmética, SUM que obtiene la suma, COUNT que obtiene el número de filas que cumplen la condición del WHERE, MIN que obtiene el mínimo de la selección, MAX que obtiene el máximo…

Consultas agrupadas

Las consultas agrupadas o resumen permiten obtener datos de la totalidad de una consulta, pero en determinados casos puede ser de utilidad conseguir datos de resumen agrupados en función de un determinado criterio.

Por ejemplo, se podría querer realizar una consulta que obtuviera la media de salario de los empleados de cada departamento.

Para este tipo de situaciones se utiliza la cláusula GROUP BY.

La cláusula SELECT de las consultas que tienen GROUP BY sólo pueden contener constantes, funciones resumen o una columna que forme parte del GROUP BY.

Ejemplo:

```
SELECT AVG(salario)
FROM empleados GROUP
BY dpto;
```

En esta consulta se obtiene la media de salario de los empleados agrupada por el departamento al que pertenecen.

A la consulta podría añadírsele que muestre el departamento en cuestión:

Ejemplo:

```
SELECT AVG(salario) , dpto
FROM empleados
GROUP BY dpto;
```

Pero no podría añadirse en la cláusula SELECT una columna que no aparezca en el GROUP BY. Por ejemplo, no podría añadirse la columna nombre.

Condiciones para las consultas agrupadas

Existen ocasiones en las que se desea realizar una consulta que requiere una agrupación de los datos como se ha visto en el apartado anterior, y además se desea que solo se muestren los grupos que cumplen una condición.

Por ejemplo, se puede querer obtener la media de los salarios de cada departamento cuyo número de empleados sea mayor de 10.

Para este tipo de caso se utiliza la cláusula HAVING que es el equivalente a la cláusula WHERE pero mucho menos potente y donde solo se pueden establecer condiciones sobre los grupos.

Ejemplo:

```
SELECT AVG(salario) , dpto
FROM empleados
GROUP BY dpto HAVING
COUNT(*)>10;
```

39.5 LENGUAJE DE CONTROL DE DATOS (DCL)

El lenguaje de control de datos es el conjunto de instrucciones que establecer el control de acceso a los datos.

El lenguaje de control de datos consta de dos instrucciones generales:

- GRANT
- REVOKE

39.5.1 Autorización de permisos

Con la orden GRANT el propietario de un objeto le concede un privilegio sobre él a otro usuario. Este privilegio puede concederse con opción de concesión o sin ella.

```
GRANT privilegio_o_derecho ON
<nombre_objeto>
TO {nombre_usuario | PUBLIC | nombre_rol} [WITH GRANT OPTION];
```

Los privilegios o derechos que pueden concederse a otro/s usuario/s pueden ser, por ejemplo:

- SELECT
- INSERT
- UPDATE
- DELETE
- ALL PRIVILEGES
- CREATE VIEW
- …

El nombre de objeto suele ser una tabla o una columna.

Si se especifica la opción WITH GRANT OPTION significa que el usuario al que se le otorgan los permisos podrá otorgárselos a otros usuarios posteriormente.

39.5.2 Eliminación de permisos

Con la orden REVOKE pueden eliminarse permisos sobre un determinado objeto a un usuario o grupo de usuarios. Su sintaxis es muy parecida a la de la orden GRANT.

```
REVOKE privilegio_o_derecho
ON <nombre_objeto>
FROM {nombre_usuario | PUBLIC | nombre_rol};
```

En este caso se eliminará el permiso indicado sobre el objeto indicado desde el nombre de usuario o rol establecido en la orden.

39.6 LENGUAJE DE CONTROL DE TRANSACCIONES (TCL)

Una transacción en BD es un conjunto de órdenes que se ejecutan como un todo, es decir, se ejecutan de forma atómica. Si en el transcurso de la ejecución de las órdenes se produce algún error se volverá al estado del instante justo anterior al comienzo de la transacción y no se habrá ejecutado nada, sin embargo, si todas las operaciones se ejecutan con éxito, los cambios quedarán persistentes en la BD.

Para que un conjunto de órdenes se ejecute como una transacción debe definirse el principio de esta con una orden.

```
SET TRANSATION {READ ONLY | READ WRITE};
```

A continuación, se ejecutarán todas las operaciones que forman parte de la transacción, y al final se ejecutará la orden que hace que los cambios sean permanentes:

```
COMMIT;
```

Si se produce algún error en la ejecución de la transacción se ejecuta una orden que hace que los cambios que se habían producido hasta ahora se deshagan:

```
ROLLBACK;
```

39.7 LENGUAJES PROCEDIMENTALES

Las bases de datos relacionales junto con su lenguaje SQL tuvo un auge tremendo e hizo que fuera utilizado de forma masiva en la gran mayoría de las aplicaciones. Sin embargo, SQL es un lenguaje que sólo permite el mantenimiento de las estructuras de las BD y de sus datos, así como consultarlos. Con la utilización únicamente de SQL no se podían construir aplicaciones porque carece de estructuras de programación como los condicionales, bucles, cursores, variables locales y globales, procedimientos almacenados, etc.

Para potenciar los SGBD y dotarlos de esa operabilidad de la que carecían, los distintos SGBDs fueron introduciendo lenguajes propietarios que permitían aumentar aún más la potencia del SGBD de manera que incorporaban estas estructuras y con ellas podían realizarse procedimientos y funciones almacenadas.

Estos procedimientos se almacenan directamente en la BD y las aplicaciones que la utilizan pueden hacer uso de estos.

SGBD	LENGUAJE
ORACLE	PL/SQL
POSTGRESQL	PL/pgSQL
MariaDB	SQL/PSM
Microsoft MySQL	Transact SQL

ORACLE fue el primer SGBD que incorporó este lenguaje, que llamó PL/SQL. Más tarde otros SGBDs se unieron como PostgreSQL que creó PL/pgSQL, y en 1998 ANSI incorporó a su estándar el lenguaje procedural llamándolo SQL/PSM que es el utilizado por MySQL.

A continuación, se presenta la definición de dos de los elementos más importantes utilizando la sintaxis PL/SQL del SGBD ORACLE.

39.7.1 Definición de disparadores (triggers)

Un trigger o disparador define una serie de acciones que se desencadenan al realizar cierta operación en la base de datos (por ejemplo, al insertar, borrar o modificar ciertos datos en una tabla de la BD). Es una especie de script escrito en SQL que se ejecutará cuando se cumpla la condición que le haga disparar.

La sintaxis para la definición de un trigger es la siguiente:

```
CREATE TRIGGER <nom_disparador>
{BEFORE | AFTER}
{INSERT | DELETE | UPDATE [OF lista_atributos]}
ON nom_tabla
[REFERENCING {OLD [ROW] [AS] nombre_referencia
|NEW[ROW][AS] nombre_referencia
| OLD_TABLE [AS] nombre_referencia
| NEW_TABLE [AS] nombre_referencia}] [FOR EACH {ROW | STATEMENT}]
[WHEN (condición)]
{sentencia_SQL | bloque SQL/PSM | CALL procedi_SQL}
```

39.7.2 Definición de procedimiento almacenado

Los procedimientos almacenados permiten la definición de funciones o secuencias de instrucciones en una misma rutina de ejecución. Permiten programar de forma imperativa sin necesidad de utilizar otra herramienta de programación. Suelen utilizarse para definir rutinas siempre relacionadas con el agrupamiento de acciones sobre la BD.

```
CREATE [OR REPLACE] PROCEDURE nombre_procedimiento
    [ (parametro [,parametro]) ]
IS
    [sección de declaración]
BEGIN
Sección ejecutable
[EXCEPTION
Sección de excepción]
END [nombre_procedimiento];
```

39.7.3 Definición de cursor

Un cursor es el conjunto de información que devuelve una consulta. Se suele utilizar dentro de procedimientos almacenados.

Se definen asociándolos a una SELECT en la sección de declaración del procedimiento almacenado.

```
CURSOR nombre_cursor IS <sentencia_SELECT>;
```

Una vez definidos pueden abrirse en cualquier momento durante la sección de ejecución del procedimiento.

```
OPEN nombre_cursor;
```

Solo si está abierto puede recorrerse. Con la orden FETCH se va recorriendo el cursor, esto permite volcar el contenido de esa fila en una variable del procedimiento, etc.

```
FETCH nombre_cursor INTO nombre_variable;
```

Una vez utilizado el cursor, debe cerrarse.

```
CLOSE nombre_cursor;
```

39.8 CONCLUSIÓN

Las bases de datos relacionales son las bases de datos más utilizadas para el almacenamiento de datos en los sistemas de información. Cualquier administrador de BD debe conocer los principios del modelo relacional, así como SQL. SQL sigue siendo el estándar utilizado para operar con bases de datos relacionales, por lo que conocer sus operaciones básicas es esencial para poder trabajar con ellas.

Cualquier persona que conozca el SQL estándar está preparada para poder trabajar con la mayoría de los SGBDs relacionales del mercado (Oracle, PostgreSQL, MySQL…)

Por otra parte, hace ya muchos años que los SGBD incorporaron los lenguajes procedimentales. De esta manera, los SGBD aumentan en potencia procedimental y hacen posible que las aplicaciones Ad-hoc puedan ser realizadas utilizando prácticamente solo el SGBD como software.