TEMA III
EL LENGUAJE ESTRUCTURADO DE CONSULTA - SQL

Introducción

Aun cuando se describe a SQL como un lenguaje de consulta, en realidad es mucho más que eso, porque dispone de muchas otras posibilidades además de las de consultar una base de datos. Entre éstas se incluyen las relativas a definir la estructura de datos, modificar los datos de la base de datos y especificar restricciones de seguridad. Cada posibilidad tiene su conjunto de instrucciones propias que se expresan, respectivamente, en el lenguaje de definición de datos (DDL), el lenguaje de manejo de datos (DML) y el lenguaje de control de datos (DLC). SQL también contiene un conjunto de palabras y de símbolos que se utilizan para especificar los resultados de los comandos. A estos se los llama expresiones de valores, operadores lógicos, predicados, expresiones de tablas, funciones agregadas y subconsultas. Cada uno tiene su sintaxis correspondiente.

COMPONENTES DEL SQL

El Catálogo o Diccionario de Datos

Una base de datos relacional que funcione y que emplee el lenguaje de datos SQL, también contendrá un catálogo o diccionario de datos. Aun cuando el catálogo es esencial para conseguir una utilización eficaz del SQL, su forma exacta es una característica del sistema de gestión de base de datos (manejado a nivel motor), y no una parte del SQL. El catálogo es una base de datos del sistema, que contiene información sobre las tablas de la base, las vistas, los derechos de acceso, la identificación de los usuarios, etc., que se puede consultar utilizando las instrucciones SELECT de SQL (obviamente, no cualquier usuario, sino uno con jerarquía de DBA). En un sistema relacional, el catálogo contiene tablas con ese tipo de información, además de las tablas que contienen la base de datos almacenada por el usuario y a la que éste tiene acceso. La estructura del catálogo depende del implementador, y por consiguiente no se puede describir de una manera detallada y general, pero coinciden en poseer tablas que almacenen las bases de datos, tablas que almacenen las tablas de las bases de datos, tablas que almacenen los índices de las tablas de las bases de datos, tablas que almacenen las restricciones de los atributos de las bases de datos, etc. Su finalidad, es proporcionar al usuario información relativa al contenido de la base de datos. EL usuario normalmente no puede aplicar al catálogo, los comandos **UPDATE**, **INSERT** o **DELETE** porque el funcionamiento de la base de datos depende él, y por consiguiente, cualquier cambio que se lleve a cabo por el usuario, podría destruir la integridad de la base de datos.

Los comandos de SQL

El lenguaje SQL, consta de un conjunto de comandos y de reglas para usarlos. Los más comunes son:

ALTER TABLE
BEGIN TRANSACTION
COMMENT
COMMIT TRANSACTION
CREATE DATABASE
CREATE INDEX
CREATE SCHEMA
CREATE TABLE
CREATE VIEW
DELETE
DROP
GRANT
INSERT
REVOKE
ROLLBACK
SELECT
UPDATE

Las palabras reservadas

A las palabras claves que forman los comandos de SQL, así como a las palabras calificativas que se usan con ellos, se las designa como palabras reservadas, y el usuario no las puede utilizar como identificadores. Es decir, estas palabras clave no se pueden usar como nombres de tablas, de vistas o de columnas sin aplicarles símbolos especiales definidos por el implementador, con el fin de que se distingan de sus correspondientes palabras claves. Estas palabras serán desarrolladas posteriormente.

Los tipos de Datos

SQL funciona con tres tipos principales datos:

- **Cadenas alfanuméricas** (CHAR, CHARACTER VARING)
- **Numérico**
    - **Entero**
        - Largo (INTEGER)
        - Corto (SMALLINT)
    - **Decimal** (DECIMAL)
- **Fecha** (DATE)

Otros tipos de datos tienen soporte adecuado en algunas bases de datos relacionales comerciales aunque hoy en día están totalmente difundidos. Las cadenas de caracteres (tipo CHAR), pueden incluir números (por ejemplo fechas), sin embargo, no se pueden efectuar operaciones aritméticas en ninguna cadena de caracteres. Todos los tipos de datos numéricos son números y se puede operar con ellos de forma algebraica. Algunos motores, son sensibles al tipo de letra permitiendo la distinción entre mayúsculas y minúsculas.

SQL incluye las siguientes expresiones de valores:

Suma	**(+)**
Resta	**(-)**
Multiplicación	**(*)**
División	**(/)**

La utilización de estas expresiones, se explicará en relación con otras expresiones, funciones y operadores seguidamente.

Los conectores lógicos y los predicados

El lenguaje SQL también proporciona los conectores lógicos **AND, OR** y **NOT** siendo su utilización la de la lógica proposicional. Un predicado o proposición, es toda expresión declarativa que se puede evaluar, y cuyo resultado puede ser "verdadero", "falso" o "desconocido". Este resultado se consigue aplicando el predicado a una fila dada en una tabla. Los predicados que se incluyen en SQL son:

comparación	**(=, <>, <, >, <=, =>)**
entre	**(...BETWEEN...AND...)**
incluido (o no) en	**(IN, NOT IN)**
como	**LIKE**
nulos	**NULL**
cuantificadores	**(ALL, SOME, ANY)**
existenciales	**EXISTS, (NOT EXISTS)**

El Lenguaje de Definición de datos (DDL)

Un esquema de una base de datos, debe ser especificado por un conjunto de definiciones. Estas se pueden expresar en el lenguaje de definición de datos de SQL, que consta de las instrucciones de definición de datos siguientes:

ALTER TABLE
CREATE DATABASE
CREATE SCHEMA
CREATE TABLE
CREATE INDEX
CREATE VIEW
DROP DATABASE
DROP SCHEMA
DROP TABLE
DROP VIEW
DROP INDEX

Cuando se completan las instrucciones de definición de datos con las cláusulas y predicados adecuados y se ejecutan, el resultado es un conjunto de tablas e índices. Los nombres de éstos se almacenan en las tablas del diccionario de datos o catálogo.

Creación de la Base de Datos

Para preparar el entorno en el cual se desarrollarán las distintas tablas, debe crearse un elemento que las contenga que se denomina base de datos. La instrucción que permite su creación es:

CREATE DATABASE nombre_de_la_base [adicionales]

Dentro de adicionales, pueden ir distintos elementos que dependerán del motor sobre el que se ejecute la orden. Estos elementos pueden ser si llevará o no archivo de *log*; qué tipo de *log* será, en qué *dbspace* o *device* se alojarán los datos, en cuáles el *log*, etc. A los efectos de lograr generalidad, solamente será considerada la primer parte de la orden, sin considerar los adicionales.

Creación de las Tablas

La instrucción que se usa para crear una tabla, debe contener el nombre de la tabla, los nombres de las columnas, los tipos de datos y el tamaño de los datos que hay que introducir. La sintaxis es como sigue:

```
CREATE TABLE nombre_de_la_tabla
(nombre_columna_tipo1     tipo_de_datos(tamaño_de_los_datos),
nombre_columna_tipo2      tipo_de_datos(tamaño_de_los_datos),
...
nombre_columna_tipoN      tipo_de_datos(tamaño_de_los_datos));
```

Obsérvese que la descripción completa de las columnas, va entre paréntesis. El formato completo de la orden que permite crear y definir una tabla base, es:

```
CREATE TABLE  dueño.nombre_de_la_tabla (
        Col_1  tipo_dato (tamaño)
               [NOT NULL] [DEFAULT valor_por_default]
               [CONSTRAINT nom_constraint_de_columna]
               [UNIQUE] [PRIMARY KEY]
               [REFERENCES nom_tabla (columna/s)]
               [CHECK (condición)]
        Col_n …..,

[CONSTRAINT nombre_constraint_de_tabla]
     [UNIQUE (columna/s)]
     [PRIMARY KEY (columna/s)]
     [CHECK (condición)]
```

[FOREIGN KEY (*columna/s****) REFERENCES*** *nom_tabla (columna/s)****]***

)

Dueño es el nombre de quien creó la tabla y que es utilizado para entrar en la base de datos como usuario. El nombre de la columna puede ser cualquier cadena alfanumérica con ciertos símbolos especiales y palabras reservadas del SQL. Los tipos de datos dependerán del motor que se utilice. Dependiendo de la implementación y de cómo está definido el servidor, por defecto se puede asumir que la columna acepta o no valores nulos, para independizarse de tal situación se debe especificar. El resto de los parámetros adicionales se refieren a restricciones para esa columna. De manera similar, pueden indicarse limitaciones o restricciones a nivel de tabla. Por ejemplo, si se especifica **UNIQUE**, entonces, esa columna no puede contener ningún valor duplicado. Si se especifica **NOT NULL**, entonces hay que poner un valor distinto de **NULL** en cada fila de esa columna. Si se especifica **UNIQUE** para una columna en particular, entonces habrá que especificar también **NOT NULL** para esa columna. Si se especifica tanto **UNIQUE** como **NOT NULL** en una columna, estas dos limitaciones hacen que la columna sea adecuada para su uso como clave primaria.

La creación de Vistas

Para crear una vista, se usa el mismo comando inicial que el que se utilizara para crear una tabla, pero a partir de aquí el proceso es ligeramente distinto por dos razones:

- Las columnas ya están creadas en la tabla de la que se extraerá la vista; por lo tanto, el tipo de datos de la comuna y su longitud serán los mismos que en la tabla de la base y no hará falta especificarlos en el comando **CREATE VIEW**.

- Para crear una vista hay que seleccionar (**SELECT**) las columnas que se quiere extraer de la tabla o tablas base. El seleccionar las columnas representa manejar los datos en vez de definirlos, por lo que esta función entra dentro de las reglas de manejo de datos.

CREATE VIEW nombre_de_la_vista ((nombre_columnas_de_la_vista))
AS (SELECT columna(s) FROM tabla(s) WHERE condición(es));

La creación de Índices

El comando de SQL que se utiliza para crear un índice en una tabla específica es:

CREATE INDEX *nombre_del_índice*
ON *nombre_de_la_tabla(nombre_de_la_columna1, …*
…, nombre_de_la_columnaN);

Pueden crearse tantos índice como se deseen en una tabla cualquiera. Puede tener un índice para cada columna de la tabla, así como un índice para una combinación de columnas. Cuántos índices sean creados y de qué tipo para una determinada tabla, dependerán de los tipos de consul-

tas esperadas que se dirigirán a la base de datos y además del tamaño de la misma. El crear demasiados índices puede presentar tantos inconvenientes como el crear muy pocos.

Los índices, tiene dos fines principales:

- Mejorar el rendimiento reduciendo las entradas y salidas del disco. Los índices de una base de datos, realizan una función análoga a las de los índices de un libro: aceleran la recuperación de la información que hay en las tablas resultantes de la unión.

- Asegurar la unicidad. Puede crearse un índice único de una columna. Después, si se hace algún intento para insertar una fila que duplique el valor que ya está en esa columna, la inserción será rechazada. Por consiguiente, un **UNIQUE INDEX** actúa como una comprobación de la unicidad de la columna.

Aun cuando la utilización del SQL exige el uso de un índice, no requiere una clave. Las claves, sin embargo, son parte de las bases de datos relacionales, y por lo tanto, deben poder ser definidas dentro de la tabla indicándose cuáles son las claves primarias, cuáles las claves alternativas y cuáles las claves ajenas. Una clave candidata es una columna o concatenación de columnas que pueden ser utilizadas para identificar de forma unívoca una fila dentro de una tabla. A una de las claves candidatas se la puede designar como la clave primaria. Una clave ajena, es una columna de una tabla que existe como clave primaria en otra (integridad referencial).

Si la implementación de interés, utiliza claves, entonces el crear un índice en cada clave, mejorará de forma evidente el rendimiento de la consulta, pero no asegurará la integridad referencial dentro de la base. Es por eso que se recomienda trabajar con integridad referencial y poder activar de esta manera uno de los elementos más importantes que proveen las bases de datos relacionales. La forma de desenvolverse de los índices dentro de la base, dependerá de si la tabla requiere de muchas inserciones o cambios en su cantidad de registros, o bien, si tan sólo es consultada, en cuyos casos, puede mejorarse o empeorarse la *performance* del acceso a los datos. La utilización de un índice al momento de realizar una consulta, es una elección que corresponde al motor quien determina si los datos que se pretenden recuperar requieren del uso del índice.

Si se posee una tabla, o si se le ha otorgado acceso a una tabla, se puede crear un índice de ella. La sintaxis para crear un índice **UNIQUE** es:

CREATE UNIQUE [CLUSTERED|NONCLUSTERED] INDEX nombre_del_indice
ON nombre_de_la_tabla (nombre(s)_de_la(s)_columna(s));

en donde se especifican las columnas de las que se quiere hacer el índice. Puede hacerse un índice de tantas columnas de la tabla como se desee, pero el número de columnas indexadas dependerá del objetivo de la tabla: si se va a actualizar la tabla con frecuencia, entonces más índices implican más gastos generales. Si, por otra parte, se trata de una tabla de sólo lectura, o si se la actualiza con poca frecuencia, entonces sería aconsejable tener más índices.

Como una tabla puede tener varios índices, a la hora de hacer una consulta, el motor de la base, determinará cuál de ellos utilizar, optimizando la rapidez para obtener los resultados.

Las dos formas principales de acceder a los datos son realizando la exploración completa de una tabla de forma secuencial (table scan), o utilizando un índice. La utilización del índice normalmente proporcionará un mejor rendimiento, porque la mayoría de las instrucciones de SQL se crean para recuperar sólo unas cuantas filas especificadas de la tabla. Sin embargo, si la recuperación consiste en una gran parte de la tabla, la utilización del índice, sólo aumentará los gastos. Para aquellos casos en los que se espera que una tabla permanezca relativamente estática (no se inserten, modifiquen o borren registros), puede ser conveniente que el orden de los registros de la tabla física, coincida con el índice que se define. Los índices que cumplen con esta característica, reciben en nombre de ***clustered***, y esta opción se coloca en la sintaxis de declaración del índice.

La modificación de tablas

Como pueden crearse situaciones nuevas o presentarse datos nuevos para que la base de datos los almacene, puede que no sea suficiente la definición inicial de la tabla. SQL permite modificar **(ALTER)** una tabla mediante la adición de una columna a las columnas existentes. También se puede cambiar la anchura de una columna que ya existe. Este comando presenta leves diferencias entre los distintos fabricantes y algunos no incluyen ciertas posibilidades.

Para añadir otra columna, la sintaxis es:

ALTER TABLE ***nombre_de_la_tabla***
ADD COLUMN ***nombre_de_la_columna tipo_de_datos;***

Para cambiar el ancho de una columna ya existente, la sintaxis es:

ALTER TABLE ***nombre_de_la_tabla***
ALTER COLUMN ***nombre_de_la_columna***
TYPE ***tipo_de_datos nueva_anchura;***

Para modificar el nombre de una columna:

ALTER TABLE ***nombre_tabla***
RENAME ***nombre_viejo*** **TO** ***nombre_nuevo;***

Para eliminar una columna:

ALTER TABLE ***nombre_de_la_tabla***
DROP COLUMN ***nombre_de_la_columna;***

Para eliminar o agregar restricciones:

ALTER TABLE ***nombre_de_la_tabla***
ADD CONSTRAINT ... ***(igual definición que en la tabla);***

ALTER TABLE ***nombre_de_la_tabla***
DROP CONSTRAINT ***nombre_constraint;***

Estas restricciones pueden ser tanto de columna como de tabla.

La supresión de una Base de Datos

Para suprimir una base de datos, se utiliza el comando **DROP DATABASE** seguido del nombre de la base:

DROP DATABASE ***nombre_de_la_base***

Cuando se suprime una base de datos mediante el comando anterior de SQL, todas las tablas, todas las vistas e índices definidos para las tablas de ella, son suprimidos automáticamente sin posibilidad alguna de recuperación anulando el efecto del comando (excepto de las copias de respaldo).

La supresión de Tablas

Para suprimir una tabla de una base de datos, se utiliza el comando **DROP TABLE** seguido del nombre de la tabla:

DROP TABLE ***nombre_de_la_tabla***

Cuando se suprime una tabla mediante el comando anterior de SQL, todas las vistas e índices definidos en esa tabla, quedan suprimidos automáticamente.

La supresión de Vistas

El comando **DROP VIEW** se utiliza para eliminar una vista de la base de datos. Su sintaxis es:

DROP VIEW nombre_de_la_vista

Cuando se suprime una vista no se suprime automáticamente la o las tablas desde la que se obtuvo, pero queda suprimida la vista en el diccionario de datos de la base.

La supresión de Índices

Para suprimir un índice se utiliza el comando **DROP INDEX** seguido del nombre del índice.

DROP INDEX ***nombre_del_índice***
ON ***nombre_de_la_tabla*****;**

Si tiene índices definidos con el mismo nombre en tablas distintas, hay que usar el **ON**, para distinguir el índice que se quiere suprimir. Otras implementaciones exigen que se especifique el nombre de la tabla y separado por un punto el nombre del índice.

Cuando se suprime un índice, no se suprimen las tablas o las vistas en las que se basa el índice.

Observación :

siempre que se realice algún tipo de operación sobre alguna tabla, previamente debe seleccionarse la base de datos en la que está contenida.

EL LENGUAJE DE MANEJO DE DATOS

Después de crear las tablas y sus objetos asociados con las instrucciones DDL que se han visto, las instrucciones de manejo de datos harán posible realizar manipulaciones sobre los mismos, incluyendo la inserción, la actualización, el borrado y la consulta.

La Inserción

La mayoría de los sistemas efectúan la carga inicial de grandes lotes de datos en la base de datos durante una operación general de carga. La instrucción **INSERT** de SQL, se usa normalmente para agregar nuevas filas individuales o grupales de datos a los que ya existe en la tabla de la base. La sintaxis es:

INSERT INTO ***nombre_de_la_tabla (nombre_de_columna1,***
nombre_de_columna2, ...)
VALUES ***('valor1', 'valor2', ...);***

En este caso la forma usada permite insertar una sola fila o parte de una sola fila.

INSERT INTO ***nombre_de_la_tabla (nombre_de_columna1,***
nombre_de_columna2, ...)
(subconsulta)**;**

En esta segunda forma, el resultado de evaluar la subconsulta, se inserta en el listado de las columnas de la tabla mencionada. Esta forma se usa normalmente cuando se insertan varias filas. Si una de las columnas no aparece en la lista, ésta deberá tener un valor por default para que se acepte la inserción.

Si la lista de valores está en la misma secuencia que la secuencia de las columnas de la tabla, y hay un valor para cada columna de la tabla, entonces, se puede omitir la lista de los nombres de las columnas. En caso contrario, se debe especificar los nombres de las columnas como se describió. Los valores que se inserten deberán ajustarse al tipo de datos de la columna en la que se van a insertar. Los valores CHAR y los tipo DATE deben ir entre comillas, mientras que los valores NUMERIC y NULL, simplemente se indican.

La actualización

El comando **UPDATE** se usa para actualizar los valores en las filas existentes. Su forma general es:

```
UPDATE  nombre_de_la_tabla
    SET   columna1 = valor_nuevo,
          columna2 = valor_nuevo,
          ...
          columnaN = valor_nuevo,
   [WHERE     condición];
```

La cláusula **SET** del comando indica qué columnas son las que hay que actualizar y qué valores hay que poner en ellas.

El comando **UPDATE** actúa en todas las filas que satisfacen la condición especificada por la cláusula **WHERE**. La cláusula **WHERE** es opcional, pero si se omite, se actualizarán todas las filas de la tabla.

Se pueden actualizar múltiples columnas en cada fila, con un único comando **UPDATE**, poniendo una lista de columnas múltiples después de la cláusula **SET**. La cláusula **WHERE** de un comando **UPDATE** puede contener una subconsulta, tema que se verá posteriormente.

El Borrado

El comando **DELETE** se usa para suprimir físicamente filas de una tabla. Su forma general es la siguiente:

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

No se pueden borrar parcialmente las filas; por consiguiente, no se tiene que especificar los nombres de las columnas en el comando. La cláusula **WHERE** puede ser compleja y puede que incluya condiciones múltiples, conectores, y/o subconsultas. Por ejemplo, si se desea borrar todas las filas de una tabla, entonces se omite la cláusula **WHERE** y el comando queda:

```
DELETE  FROM   nombre_de_la_tabla;
```

Esta orden, suprime todas las filas y dejará solamente las especificaciones de las columnas y el nombre de la tabla.

El comando **TRUNCATE TABLE** elimina todos los datos de la tabla indicada sin dejar rastros en el log, y consecuentemente no pueden deshacerse los cambios (el borrado). Por este motivo es más rápido pero así también de peligroso.

La recuperación de datos

La estructura básica de la consulta SQL consta de las dos cláusulas:

SELECT ***columna1, columna2, ...,columnaN***
FROM ***nombre_de_la_tabla***

La cláusula **SELECT** hace un listado de las columnas que se desean obtener en el resultado de la consulta, es decir, la lista objeto. Esta cláusula corresponde a la operación de proyección del álgebra relacional.

El comando **SELECT** y la cláusula **FROM**, son necesarios para cualquier consulta de SQL. **SELECT** y **FROM** deben aparecer antes que cualquier otra cláusula en una consulta. Para recuperar solamente un conjunto de filas que se hayan especificado, se indicará la característica común de las filas que se quieren obtener mediante la cláusula **WHERE**.

La cláusula **WHERE** corresponde al predicado de selección del álgebra relacional. Consta de un predicado que incluye columnas de la tabla o tablas que aparecen en la cláusula **FROM**.

Como se indicara, el comando **SELECT** y la cláusula **FROM** se necesitan para cada consulta SQL, y deben aparecer antes que cualquier otra cláusula, mientras que **WHERE** es opcional, y en caso de omitirla, devuelve todas las filas de la tabla..

El comando CREATE VIEW

Aun cuando **CREATE** es un comando de definición de datos, el comando **CREATE VIEW** se incluye en esta lista de instrucciones de manejo de datos porque solamente se le puede completar usando el comando **SELECT**. En realidad, la especificación de la vista es realmente una consulta SQL. Puede usar cualquier consulta SQL válida con un comando **CREATE VIEW**; lo que no está permitido hacerse es utilizar la cláusula **ORDER BY** en la vista. Si se desea que las filas estén ordenadas en una cierta forma, debe hacerse con una consulta aparte dirigida a la vista una vez que ésta ya se haya creado.

EL LENGUAJE DE CONTROL DE DATOS (DCL)

El lenguaje de control de datos (DCL), consta de un grupo de instrucciones SQL que permiten al administrador de la base controlar el acceso a la misma y establecer procedimientos para proteger la seguridad de los datos. Estas consideraciones alcanzan una mayor importancia en un sistema multiusuario en el que el uso concurrente podría conducir a crear la confusión entre los usuarios y a la desorganización del sistema. Estos comandos que ejercitan al control de acceso y ejecución de procedimientos de la base de datos son **GRANT** y **REVOKE**. Los comandos de SQL que protegen la integridad en cuanto a concurrencia y compleción de transacciones de los datos son **COMMIT TRANSACTION** y **ROLLBACK TRANSACTION**. La mayor parte de las implementaciones comerciales, también añaden limitaciones para la auditoría, el bloqueo y la validación de los índices de las bases de datos.

El Control de Acceso

Incluso en los sistemas monousuario puede ser necesario controlar el acceso a la base de datos y cambiar los privilegios de acceso de vez en cuando. Los comandos **GRANT** y **REVOKE** proporcionan una amplia gama de posibilidades de control de acceso. Estos comandos pueden ser aplicados a distintos objetos de la base de datos tales como: tablas, vistas, columnas y procedimientos almacenados. Los permisos considerados son: *select, update, insert, delete, references y execute*. En la siguiente tabla, se ven cuáles permisos son aplicables a los distintos objetos:

Permiso	Objeto
SELECT	Tabla, Vista o Columna
UPDATE	Tabla, Vista o Columna
INSERT	Tabla, Vista
DELETE	Tabla, Vista
REFERENCES	Tabla, Columna
EXECUTE	Stored procedure

El comando GRANT

El comando **GRANT** se puede usar para permitir el acceso completo a la base de datos o grados de acceso limitado. Por ejemplo, según se puede ver en los sistemas comerciales que hay actualmente en el mercado, el tipo de acceso que se otorga puede venir definido en los siguientes términos: solamente ver las tablas de la base de datos, ver las tablas pero no poder cambiarlas, ver solamente las vistas, ver y cambiar vistas, o ver todas las tablas y vistas, pudiendo hacer cambios en ellas y concediendo privilegios a otros usuarios. En este contexto, las vistas combinadas con los comandos de acceso, pueden salvaguardar las columnas que se especifiquen en una tabla. Para salvaguardar columnas, se crea una vista de la tabla, la cual contendrá solamente las columnas de la tabla que no son confidenciales. Después se concede acceso a la vista en vez de a la tabla en sí. La sintaxis del comando **GRANT** es la siguiente:

GRANT privilegio_nivel_base TO usuario [WITH GRANT OPTION];

Este comando permite otorgar permisos en el ámbito de base de datos completa. O bien

GRANT privilegio_nivel_tabla ON nombre_de_la_tabla
TO usuario [WITH GRANT OPTION];

Este comando permite otorgar permisos a nivel de tabla. En ambos casos, ***usuario*** es el nombre que ha de introducir el usuario al que se le ha concedido el privilegio cada vez que acceda al sistema.

Algunos tipos de acceso pueden comportar el privilegio de otorgar acceso a otros usuarios. Por ejemplo, si las tablas de la base de datos van identificadas con el nombre del creador de la tabla, entonces ese usuario (si se le ha otorgado el privilegio adecuado) puede conceder acceso a otros usuarios en sus tablas. En el siguiente ejemplo el propietario de la base actualmente acti-

va, ha dado acceso a todas las tablas de la base al usuario **user1** permitiéndole además que éste a su vez, le pueda conceder permisos a otros usuarios. El comando es el siguiente:

GRANT ALL PRIVILEGES TO user1 WITH GRANT OPTION

Suponiendo que el usuario **user2** (que trabaja bajo la supervisión del usuario **user1**) necesita acceder a las tablas de la base **tabla1** y **tabla2**; como el administrador o dueño de la base datos le dio a **user1** el derecho de conceder permisos (a través de **WITH GRANT OPTION** en la instrucción de concesión de privilegios), éste puede conceder al usuario **user2** los siguientes privilegios:

GRANT ALL PRIVILEGES ON tabla1, tabla2 TO user2;

Con este comando, **user1** le ha dado al usuario **user2** el derecho de actualizar (**UPDATE**), insertar (**INSERT**) y borrar (**DELETE**) filas en las tabla **tabla1** y **tabla2**. También tiene el privilegio de eliminar ambas tablas, crear índices en ellas y usar todas las instrucciones de manejo de datos. La instrucción **GRANT** no le da al usuario **user2** el derecho de pasar sus privilegios a otros usuarios, porque no se incluyó la cláusula **WITH GRANT OPTION**.

Si el uso que **user2** puede hacer de las tablas, implica usar sólo las columnas de **col1_tabla1, y col1_tabla2** el comando sería:

```
GRANT ALL PRIVILEGES (col1_tabla1, col1_tabla2)
      ON tabla1, tabla2 TO user2;
```

Este comando le permite al usuario **user2**, ejecutar cualquier operación de manejo de datos (**SELECT, UPDATE, INSERT, DELETE**) en las tablas de **tabla1** y **tabla2**, pero sólo en las columnas especificadas de esas tablas. En realidad, el comando **DELETE** no puede ejecutarlo pues no se pueden borrar parcialmente filas.

Si es necesario que varios usuarios, pero no todos, usen ciertas tablas de una base, se puede nombrar a todos estos usuarios en una misma instrucción **GRANT**. Por ejemplo:

GRANT ALL PRIVILEGES ON tabla1 TO user3, user4, user5;

En otros casos, podría ser conveniente o necesario hacer que algunas tablas estén disponibles para todo el personal de una organización. En lugar de conceder acceso a cada usuario por su nombre, es más eficaz que el administrador de la base haga esta operación con un solo comando. Esta es la finalidad de la concesión llamada **PUBLIC**. Cualquier tabla a la que se otorga acceso **PUBLIC** está disponible para cualquier usuario de la organización que tenga un identificador para entrar en la base de datos. El acceso **PUBLIC** puede ser sólo para leer la tabla, o para la operación que se desee autorizar. El caso de sólo lectura sobre la tabla rubro, se expresa en el siguiente ejemplo:

GRANT SELECT ON tabla1 TO PUBLIC;

Si se debe posibilitar que alguien pueda cambiar algún atributo de **tabla1**, entonces este privilegio se puede otorgar -privilegio de **UPDATE**- a todos con la instrucción:

GRANT UPDATE ON tabla1 TO PUBLIC;

Si son solamente algunos los usuarios que dispondrán de esa capacidad, se puede otorgar a **PUBLIC** permisos de solamente lectura y específicamente a los usuarios preestablecidos indicar expresamente el permiso de **UPDATE** sobre la o las tablas de interés. Adicionalmente, si todos los usuarios pueden insertar nuevos registros en la tabla de rubros, la instrucción sería:

GRANT INSERT ON tabla1 TO PUBLIC;

La sintaxis completa del comando **GRANT** es:

```
GRANT {ALL [PRIVILEGES] | lista_de_permisos }
      ON {      nombre_de_tabla [(lista_de_columnas)]
              | nombre_de_vista [(lista_de_columnas)]
              | nombre_procedimiento_almacenado }
      TO { PUBLIC | lista_de_usuarios | nombre_del_rol }
      [WITH GRANT OPTION]
```

Asociado a esto, deben verse los **grupos** y la asociación de los usuarios a esos **grupos**. También otro elemento relacionado son los **logines** conceptos que se escapan al contenido del tema.

El comando REVOKE

El revocar el acceso, es simplemente lo opuesto a concederlo. Esto se logra utilizando el término **REVOKE** nombrando a continuación el tipo de privilegio que se concedió previamente y escribiendo el nombre del usuario cuyo acceso se suspende. Si se han concedido distintos tipos de acceso, entonces se pueden revocar algunos o todos los niveles mediante revocación selectiva. La sintaxis es:

REVOKE privilegios_especificados FROM nombre_del_usuario;

Por ejemplo:

REVOKE ALL PRIVILEGES FROM user2;

Dependiendo cómo se haya creado la interfaz de usuario, esto puede que deje o no deje en vigencia las posibilidades de acceso que ella ha concedido a otros miembros de la organización.

La sintaxis completa del comando **REVOKE** es:

```
REVOKE [GRANT OPTION FOR] {ALL [PRIVILEGES] | lista_de_permisos }
      ON {      nombre_de_tabla [(lista_de_columnas)]
              | nombre_de_vista [(lista_de_columnas)]
              | nombre_procedimiento_almacenado }
      FROM { PUBLIC | lista_de_usuarios | nombre_del_rol }
```

El control de Integridad

Los controles de integridad **COMMIT TRANSACTION** y **ROLLBACK TRANSACTION** están incluidos en las normas **ANSI** del lenguaje de bases de datos SQL y constituyen los controles mínimos necesarios para mantener la integridad de la base de datos. Los sistemas comerciales multiusuario emplean controles adicionales tales como los comandos **LOCK** (bloqueo) para evitar que los valores cambien mientras un usuario los está editando, consultando o está trabajando con ellos. Estos comandos **LOCK** a veces incluyen previsiones de **SHARED LOCK** (bloqueo compartido) de forma que varios usuarios puedan acceder a las tablas a la vez sin tener que esperar mucho tiempo.

El comando COMMIT TRANSACTION

El uso del comando **COMMIT TRANSACTION** le permite al usuario decidir cuándo hacer permanentes en la base de datos los cambios que se están efectuando. En este contexto, es conveniente emplear el concepto de transacción.

Transacciones:

Una transacción es una unidad de operación u operación atómica compuesta por un conjunto de instrucciones SQL (*select, insert, delete y update*) que son ejecutadas en una cierta secuencia y en la que es absolutamente necesario que todas sean completadas satisfactoriamente para que la unidad de operación (transacción) también sea completada y de esa manera confirmada. En caso contrario, todos los cambios parciales efectuados, no son confirmados y las modificaciones en la base quedan sin efecto y todo queda como en el origen mediante la utilización del **log**. Si sólo se completase parte de la secuencia y los cambios se registren en la base de datos, ésta estaría (o podría estarlo), en estado de desequilibrio o ser incorrecta. Por ejemplo, si se venden productos y estos productos no se dan de baja de la lista de *stock* remanente, entonces la lista de control de *stock*, contendrá más productos que los que hay realmente. De forma análoga, si por ejemplo, se hace un ingreso a la cuenta de un acreedor y la cantidad de este ingreso no se deduce de un saldo del banco, entonces el saldo del banco será incorrecto. Si una transacción exige que se realicen varios cambios en varias columnas o en varias tablas, puede ser que el registrar ese cambio parcialmente cause inconsistencias en la base de datos. Por ejemplo, introducir en la base el acto de confeccionar una factura, puede exigir:

1. Deducir la cantidad y el tipo de producto del *stock*;
2. Confeccionar el remito de los productos adquiridos;
3. Confeccionar la correspondiente factura refiriendo al remito;
4. Asentar la factura en el libro de IVA Ventas;
5. Realizar el asiento en el plan de cuentas correspondiente;
6. Actualizar la cuenta corriente del cliente;
7. Añadir el envío a la lista que se debe distribuir en el día.

Hasta que no se hayan reflejados estos siete eventos en las tablas correspondientes de la base, ésta contendrá información inconsistente o incompatible. Si se produjese una avería en el sistema en el curso de esta secuencia de eventos, puede resultar muy laborioso, e incluso imposi-

ble, encontrar las inconsistencias y restaurar los valores anteriores y reales que se poseían en las tablas, perdiendo de esa forma la consistencia de la base. Por esta razón SQL le proporciona al usuario la opción de hacer fijos los cambios en la base de datos sólo cuando la transacción se ha completado. Esto se hace con el comando:

COMMIT TRANSACTION;

Una transacción o parte de una transacción que todavía no se ha hecho fija (**COMMIT**-ted) sólo la ve el usuario que la está introduciendo. No afecta a la base de datos hasta que no se ejecuta la instrucción **COMMIT**. Antes de ejecutar la instrucción **COMMIT TRANSACTION**, se puede deshacer (**ROLLBACK TRANSACTION**) o descartar. Una vez que una transacción se ha hecho fija, no se la puede deshacer. Si hay que cambiarla o corregirla, habrá que hacer ese cambio por medio de otra instrucción de SQL como **UPDATE** o **DELETE**.

El comando ROLLBACK TRANSACTION

Si en el curso de la introducción de una transacción se comete un error, o si no se puede completar la transacción por alguna razón, el usuario tal vez desee eliminar los cambios con el fin de evitar inconsistencias en la base. Esta es la finalidad del comando **ROLLBACK TRANSACTION**. La sintaxis es similar al comando anterior:

ROLLBACK TRANSACTION;

En el caso de producirse un fallo del sistema, la integridad de la base de datos, puede quedar asegurada mediante un dispositivo automático **ROLLBACK WORK** que elimine las transacciones incompletas y que por lo tanto evite que se las introduzca en la base de datos.

ACID

En bases de datos se denomina ACID a un conjunto de características necesarias para que una serie de instrucciones puedan ser consideradas como una transacción. Así pues, si un RDBMS es "ACID compliant" quiere decir que el mismo cuenta con las funcionalidades necesarias para que sus transacciones tengan las características ACID. En concreto ACID es un acrónimo de **A**tomicity, **C**onsistency, **I**solation and **D**urability: Atomicidad, Consistencia, Aislamiento y Durabilidad.

Atomicidad: es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias.

Consistencia: Integridad. Es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper las reglas y directrices de integridad de la base de datos. La propiedad de consistencia sostiene que cualquier transacción llevará a la base de datos desde un estado válido a otro también válido.

Aislamiento: es la propiedad que asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones sobre la misma información sean independientes y no generen ningún tipo de error.

Durabilidad: es la propiedad que asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema.

Cumpliendo estos 4 requisitos un sistema gestor de bases de datos puede ser considerado ACID Compliant.