Sistemas y Aplicaciones Informáticas

Tema 38. Lenguajes para Definición y Manipulación de Datos en Sistemas de Bases de Datos Relacionales. Tipos. Características. Lenguaje SQL.

1. Ámbito de docencia.

- Sistemas informáticos monousuario y multiusuario (ASI 1).
- Sistemas informáticos multiusuario y en red (DAI 1).
- Instalación y mantenimiento de equipos y sistemas informáticos (ESI 1).

2. Lenguajes de definición de datos (DDL). Tipos y características.

- Los sistemas gestores de bases de datos (SGBD) deben facilitar lenguajes para describir el esquema conceptual de las bases de datos, para realizar especificaciones relativas a su esquema interno y para declarar los esquemas externos necesarios para el desarrollo de las aplicaciones:
  - *Definición del esquema conceptual*. Se utiliza un lenguaje de definición de datos (DDL) para describir los datos de la base de datos, las relaciones entre ellos y sus restricciones de integridad. Los SGBD poseen un compilador de DDL cuya función consiste en procesar las sentencias del lenguaje para identificar las descripciones de los distintos elementos de la base de datos y almacenar la descripción del esquema en el diccionario de datos.
  - *Definición del esquema interno*. En muchos SGBD no se mantiene una separación estricta de niveles, por lo que el administrador de la base de datos y los diseñadores utilizan el DDL para definir el esquema interno. Cuando en un SGBD hay una clara separación entre niveles, para especificar el esquema interno se utiliza un lenguaje de definición de almacenamiento (SDL). Las correspondencias entre el esquema conceptual y el esquema interno se pueden especificar en cualquiera de los dos lenguajes.
  - *Definición del esquema externo*. En una verdadera arquitectura de tres niveles se dispone de un tercer lenguaje, el lenguaje de definición de vistas (VDL), que se utiliza para especificar el esquema externo y su correspondencia con el esquema conceptual.
- Los lenguajes de definición de datos son lenguajes autocontenidos, puesto que no necesitan apoyarse en ningún lenguaje de programación. En la práctica se suele utilizar el mismo DDL para la descripción de los tres niveles de la arquitectura ANSI/SPARC.

3. Lenguajes de manipulación de datos (DML). Tipos y características.

- Una vez creados los esquemas de las bases de datos, los usuarios necesitan un lenguaje que les permita manipular sus datos, realizando consultas, inserciones, eliminaciones y modificaciones. Este lenguaje es el que se denomina lenguaje de manipulación de datos (DML).
- Hay dos tipos de DML que se distinguen por el modo en que acceden a los datos:
  - *Procedimentales*. Manipulan la base de datos registro a registro (lenguajes navegacionales), especificando qué operaciones se deben realizar para obtener los datos. Estos lenguajes acceden a un registro, lo procesan y basándose en los resultados obtenidos acceden a otro registro, que también deben procesar y así sucesivamente hasta que se obtienen los datos deseados. Las sentencias de un DML procedimental deben estar embebidas en un lenguaje de alto nivel, ya que se necesitan sus estructuras (bucles, condicionales, etc.) para obtener y procesar cada registro individual. A este lenguaje se le denomina lenguaje anfitrión (host language). Los SGBD jerárquicos y de red utilizan DML procedimentales.

- *No procedimentales*. Operan sobre conjuntos de registros (lenguajes no navegacionales), especificando qué datos deben obtenerse sin expresar cómo hacerlo. El SGBD traduce las sentencias del DML en uno o varios procedimientos que manipulan los conjuntos de registros necesarios. Esto libera al usuario de tener que conocer cuál es la estructura física de los datos y qué algoritmos se deben utilizar para acceder a ellos. Un DML no procedimental puede actuar de manera autocontenida, es decir, se puede utilizar de manera independiente para especificar operaciones complejas sobre la base de datos de forma concisa. En muchos SGBD se pueden introducir interactivamente instrucciones de DML desde un terminal o bien embeberlas en un lenguaje anfitrión de alto nivel (como COBOL, C, Visual Basic, etc). Las bases de datos relacionales utilizan DML no procedimentales, como SQL (Structured Query Language) o QBE (Query-By-Example).

– En general, los lenguajes de manipulación de datos embebidos en un lenguaje anfitrión son procedimentales, se explotan en tratamientos por lotes y actúan registro a registro, mientras que los autocontenidos suelen ser no procedimentales, se usan en modo conversacional desde un terminal y recuperan conjuntos de registros.

4. Lenguaje SQL.

4.1. Características. Sublenguajes.

– El lenguaje de consulta estructurado (SQL) es un lenguaje de base de datos normalizado, utilizado por los SGBD para realizar determinadas operaciones sobre los datos de una base de datos o sobre la estructura de la misma. Está formado por comandos, operadores, cláusulas y funciones que se combinan en sentencias para crear, actualizar y manipular las bases de datos.

– Se dice que SQL es estructurado porque trabaja con conjuntos de resultados abstractos como unidades completas. Un conjunto de resultados es el esquema básico de una tabla: N filas x N columnas. Este esquema se trata como un todo y es la idea principal de SQL. Se trata de un lenguaje estándar ANSI/ISO cuyas revisiones más recientes son SQL92, SQL99 y SQL2005.

– Existen tres sublenguajes dentro de SQL:

- *Lenguaje de manipulación de datos (DML)*. Permite al usuario consultar datos almacenados en la base de datos, así como actualizarla añadiendo, suprimiendo o modificando datos.
- *Lenguaje de definición de datos (DDL)*. Utilizado para definir, modificar y borrar las tablas en las que se almacenan los datos y las relaciones entre éstas.
- *Lenguaje de control de datos (DCL)*. Sirve para definir la protección y la seguridad de los datos, y la manera de compartirlos concurrentemente en un entorno multiusuario.

4.2. Estructuras lógicas.

4.2.1. Tablas. Vistas.

– Una tabla es la estructura de datos que contiene los datos en una base de datos relacional. Una tabla se compone de filas y columnas. Una tabla puede representar una única entidad que se desee representar en el sistema. También puede representar una relación entre dos entidades.

– Una vista es la representación lógica de otra tabla o combinación de tablas. No tiene datos propios, si no que los obtiene a partir de las tablas en las que se basa, que pueden ser tablas reales u otras vistas. Todas las operaciones realizadas sobre las vistas afectan realmente a las

tablas reales en las que se basan. Las vistas se utilizan para proporcionar diferentes representaciones de los datos que residen en otras tablas u otras vistas.

4.2.2. Columnas. Filas. Valores.

- Cada columna de una tabla representa un solo atributo de la entidad, y tiene asignado un tipo de dato. Las columnas se identifican por su nombre, sin importar el orden dentro de la tabla.
- Las filas almacenan los datos de una tabla, y cada una de ellas representa una ocurrencia de la entidad o de la relación representada en la tabla. No existen filas duplicadas en una tabla, y siempre existe un atributo o grupo de atributos que identifica de forma unívoca cada una de las filas, denominado clave primaria. El orden de las filas dentro de una tabla carece de importancia.
- La intersección de una fila y una columna determinadas es un valor. Puede contener un dato del tipo de datos al que pertenece una columna, o puede contener un valor NULL (desconocido o ausente). Una operación devuelve NULL si cualquiera de los operandos es NULL.

4.3. Tipos de datos.

- **CHAR (n)**. Almacena cadenas de caracteres de longitud fija. Si se introduce un valor con un número de caracteres $m < n$, las posiciones restantes se rellenan con espacios.
- **VARCHAR (n)**. Almacena cadenas de caracteres de longitud variable. Si se introduce un valor con un número de caracteres inferior a $m < n$, la columna ocupará m caracteres.
- **NUMBER (p,s)**. Almacena un número decimal de p dígitos con signo y s dígitos a la derecha del punto decimal. Se utiliza para almacenar valores numéricos en punto flotante con una precisión máxima de 38 dígitos (lo que incluye números desde 1.0E-129 hasta 9.99E124) y una escala que va desde –84 hasta 127.
- **DATE**. Se usa para almacenar información del siglo, año, mes, día, hora, minuto y segundo. El formato de fecha puede ser alterado en cualquier momento. Internamente una fecha se almacena como el número de días desde cierto punto de inicio. Esto permite que las fechas puedan ser tratadas en operaciones aritméticas normales.
- **RAW**. Permiten almacenar información en formato binario. Se pueden utilizar tanto para almacenar grandes cantidades de datos (hasta 4 GB) como para almacenar directamente cualquier tipo de fichero o para transportar datos de una base de datos a otra, ya que el formato binario es el único formato común entre cualquier sistema informático.

4.4. Operadores.

4.4.1. Aritméticos.

- Permiten la realización de operaciones matemáticas con los atributos de cada fila y con valores constantes. Utilizan valores numéricos y devuelven un valor numérico.
- Son los siguientes: suma (+), resta (-), multiplicación (*) y división (/). El orden de estos operadores de mayor a menor prioridad es multiplicación, división, suma y resta, aunque puede alterarse este orden utilizando los paréntesis, de forma que toda expresión encerrada entre paréntesis se ejecuta primero.

4.4.2. Lógicos.

- Permiten la realización de operaciones lógicas con valores lógicos (verdadero o falso).

- Producen un valor lógico que viene determinado por las tablas de verdad correspondientes a cada uno de los operadores.

A	B	A AND B	A OR B	NOT A
F	F	F	F	V
F	V	F	V	
V	F	F	V	F
V	V	V	V	

4.4.3. De comparación.

- Permiten la realización de operaciones de comparación con los atributos de cada fila y con valores constantes. Utilizan valores del mismo tipo y devuelven un valor lógico verdadero si la comparación entre los valores es cierta, y falso si la comparación entre los valores no es cierta.
- Existen varios tipos de operadores de comparación:
  - *Para un valor*. IS NULL devuelve verdadero si el valor del atributo es NULL, en caso contrario devuelve falso.
  - *Entre dos valores.* Igual (=), distinto (<>, !=, ^=, ¬=), mayor que (>), menor que (<), mayor o igual que (>=), menor o igual que (<=).
  - *Entre un valor y un conjunto de valores:*
    - **ANY (lista o subconsulta)**. Compara un valor con cada valor incluido en una lista o devuelto por una subconsulta. Debe ir precedido de los operadores =, !=, >, >=, <, <=. Es equivalente a un OR lógico entre todos los elementos.
    - **ALL (lista o subconsulta)**. Compara un valor con todos los incluidos en una lista o devueltos por una subconsulta. Debe ir precedido de los operadores =, !=, >, >=, <, <=. Es equivalente a un AND lógico entre todos los elementos.
    - **IN (lista o subconsulta)**. Pertenencia a un conjunto de valores incluidos en una lista o devueltos por una subconsulta. Es equivalente a =ANY.
    - **BETWEEN *x* AND *y***. Pertenencia a un conjunto de valores comprendidos entre *x* e *y*. Es equivalente a >= *x* AND <= *y*.
    - **EXISTS (subconsulta)**. Devuelve verdadero si la subconsulta devuelve al menos una fila, en caso contrario devuelve falso.
    - **LIKE 'patrón'**. Compara un valor formado por una cadena de caracteres con un patrón de comparación. Utiliza los comodines '%' (sustituye de 0 a N caracteres) y '_' (sustituye un único carácter).

4.4.4. De álgebra relacional.

- **UNION**. Unión de dos relaciones con las mismas columnas. Da como resultado una relación formada por las filas que se encuentran en cualquiera de las dos relaciones y en ambas a la vez. Con la cláusula ALL se incluyen también las filas repetidas entre ambas relaciones.
- **INTERSECT**. Intersección de dos relaciones con las mismas columnas. Da como resultado una relación formada por las filas que se encuentran en ambas relaciones a la vez.
- **MINUS**. Diferencia de dos relaciones con las mismas columnas. Da como resultado una relación formada por las filas que se encuentran en la primera relación y no se encuentran en la segunda.

4.4.5. Otros operadores.

- *****. Selecciona todas las columnas de una tabla en una consulta.
- **ALL**. Fuerza a devolver valores duplicados en una consulta.
- **DISTINCT**. Elimina valores duplicados en una consulta.
- ||. Concatena una cadena de caracteres con otra.

4.5. Funciones.

4.5.1. Numéricas.

- **ABS (n)**. Devuelve el valor absoluto de *n*.
- **CEIL (n)**. Devuelve el menor entero mayor o igual que *n*.
- **FLOOR (n)**. Devuelve el mayor entero menor o igual que *n*.
- **MOD (m, n)**. Devuelve el resto de la división entre *m* y *n*.
- **POWER (m, n)**. Devuelve la potencia de *m* elevado a *n*.
- **SQRT (n)**. Devuelve la raíz cuadrada de *n* (NULL si *n*<0).

4.5.2. Alfanuméricas.

- **CHR (n)**. Devuelve el carácter con código ASCII n.
- **UPPER (char)**. Convierte la cadena a mayúsculas.
- **LOWER (char)**. Convierte la cadena a minúsculas.
- **LPAD (char1, n [, char2])**. Rellena char1 por la izquierda hasta llegar a los n caracteres con los caracteres especificados en char2 (blancos por defecto).
- **RPAD (char1, n [, char2])**. Lo mismo que LPAD, pero por la derecha.
- **LTRIM (char1 [, char2])**. Elimina de la cadena char1 los primeros caracteres que coincidan con la cadena char2. Por defecto, char2 es un espacio en blanco.
- **RTRIM (char1 [, char2])**. Elimina de la cadena char1 los últimos caracteres que coincidan con la cadena char2. Por defecto, char2 es un espacio en blanco.
- **ASCII (char)**. Devuelve el código ASCII del carácter.
- **LENGTH (char)**. Devuelve la longitud de la cadena.

4.5.3. De conversión de tipo de datos.

- **TO_CHAR (n [, formato])**. Convierte un número a una cadena de caracteres.
- **TO_NUMBER (char [, formato])**. Convierte una cadena de caracteres a formato numérico.
- **TO_CHAR (d [, formato])**. Convierte una fecha a una cadena de caracteres.
- **TO_DATE (char [, formato])**. Convierte una cadena de caracteres a formato de fecha.

4.5.4. De agrupación.

- Estas funciones actúan sobre un conjunto de valores, devolviendo un solo registro. Los valores pueden ser los correspondientes a una determinada columna, a una lista o a una subconsulta.
- Todas estas funciones permiten incluir el modificador DISTINCT delante del conjunto de valores para que omita los repetidos. Son las siguientes:
  - *AVG (valores).* Devuelve la media del conjunto de valores, ignorando los valores nulos.
  - *COUNT (valores).* Devuelve el número de filas cuyo valor es no nulo.
  - *MAX (valores).* Devuelve el máximo valor del conjunto de valores.

- *MIN (valores)*. Devuelve el mínimo valor del conjunto de valores.
- *SUM (valores)*. Devuelve la suma del conjunto de valores.

4.5.5. De fechas.

- **ADD_MONTHS (d, n)**. Suma un número positivo o negativo *n* de meses a una fecha *d*.
- **LAST_DAY (d)**. Retorna el último día de mes de la fecha *d*.
- **MONTHS_BETWEEN (d1, d2)**. Retorna la diferencia en meses entre dos fechas *d1* y *d2*.

4.6. Sentencias de DML.

4.6.1. Sentencia SELECT.

- La sentencia SELECT devuelve un único conjunto de datos con cualquier tipo de condición, agrupación u ordenación, por lo que podrá ser aplicada en cualquier lugar donde se espere un conjunto de resultados. La sintaxis básica es:

```
SELECT columnas
FROM tablas
WHERE condición
GROUP BY columnas de agrupación
HAVING condición agrupada
ORDER BY columnas de ordenación
```

- Todas las cláusulas son opcionales excepto SELECT y FROM:
  - *SELECT*. Selecciona las columnas, constantes, expresiones, pseudocolumnas o funciones SQL que se desean mostrar en el resultado separadas por coma. Cualquiera de ellas puede cualificarse con un nombre adicional utilizando la sintaxis {*columna* | *constante* | *pseudocolumna* | *función SQL}* *nombre*.
  - *FROM*. Indica el o los conjuntos de resultados que intervienen en la consulta. Normalmente se utilizan tablas, pero se admite cualquier tipo de conjunto (tabla, select, vista…). Si apareciese más de una tabla, deben ir separadas por coma. Al igual que a las columnas, también se puede cualificar a las tablas utilizando la sintaxis *tabla nombre*.
  - *WHERE*. Indica qué condiciones deben cumplirse para que una fila entre dentro del conjunto de resultados devuelto. Para construir las condiciones se podrán utilizar todos los operadores lógicos y de comparación. Una condición en la que un operando sea NULL siempre se evalúa como falso, y por tanto ese registro queda automáticamente excluido.
  - *GROUP BY*. Se utiliza para tratar valores que es necesario procesar como un grupo, considerando los registros que tienen el mismo valor en ciertos campos, para contarlos, sumarlos, hacer la media… Todas las columnas incluidos en la cláusula SELECT fuera de funciones de agrupación deben estar dentro de la cláusula GROUP BY.
  - *HAVING*. Se utiliza para aplicar condiciones sobre agrupaciones. Sólo puede aparecer si se ha incluido la cláusula GROUP BY.
  - *ORDER BY*. Se utiliza para ordenar ascendente (ASC) o descendentemente (DESC) las filas del conjunto de resultados final. Dentro de esta cláusula podrá aparecer cualquier expresión que pueda aparecer en el SELECT.

4.6.2. Sentencia INSERT.

- Permite introducir nuevas filas en una tabla de base de datos. La sintaxis básica es:

```
INSERT INTO tabla {(campos)}
VALUES (lista de valores)
```

- Los nombres de los campos detrás del nombre de tabla son opcionales y si no se ponen se supondrá todos los campos de la tabla en su orden original. Si se ponen, se podrán indicar cualquier número de columnas en cualquier orden.
- La lista de valores es el registro que se insertará en la tabla. Los tipos de datos deben coincidir con la definición dada en la cláusula INTO o con la definición de la tabla si omitimos dicha cláusula. Las columnas que no se incluyan en el INTO, se inicializarán con NULL si no se ha definido valor en el DEFAULT.
- Existe otra sintaxis que se denomina INSERT masivo:

```
INSERT INTO tabla {(campos)}
SELECT …
```

- Este tipo de INSERT permite introducir un gran número de registros en una sola sentencia. Al igual que con el INSERT normal, los tipos de datos de la sentencia SELECT deben coincidir con la definición de la cláusula INTO.

4.6.3. Sentencia DELETE.

- Permite eliminar filas en una tabla de base de datos conforme a una condición. Su sintaxis es:

```
DELETE {FROM} tabla
{WHERE condición}
```

- Si se omite la cláusula WHERE se borrarán todas las filas de la tabla. Las condiciones pueden ser las mismas que las aplicadas en una sentencia SELECT. En la cláusula FROM no puede haber más de una tabla.

4.6.4. Sentencia UPDATE.

- Se encarga de modificar registros ya existentes en una tabla. Su sintaxis es la siguiente:

```
UPDATE tabla
SET campo1 = valor1, campo2 = valor2, . . .
{WHERE condición}
```

- El valor puede ser tanto un valor discreto, un valor dependiente de otra una columna o una subselect que retorne un conjunto de resultados de una fila y una columna. Si se omite la cláusula WHERE, se actualizarán todas las filas de la tabla.

4.7. Sentencias de DDL.

4.7.1. Sentencia CREATE TABLE.

- Crea una tabla en base de datos. La sintaxis básica es:

```
CREATE TABLE nombre_tabla (
COLUMNA TIPO [NOT NULL], . . .
{CONSTRAINT nombre_clave_primaria
{UNIQUE | PRIMARY} KEY (columnas_clave)}
{CONSTRAINT nombre_clave_foránea
```

```
FOREIGN KEY (columnas) REFERENCES tabla (columnas)
{ON DELETE | MODIFY} {RESTRICT | CASCADE | SET NULL}}
{CONSTRAINT nombre_condición CHECK (condición)})
```

- Los campos que van a formar parte de la clave se tienen que definir como NOT NULL ya que no tiene sentido que éstas columnas carezcan de valor.
- Las cláusulas de restricción (CONSTRAINT) sirven para limitar el rango de valores válidos para una columna o grupo de columnas de una tabla. Las sentencias INSERT, UPDATE y DELETE obligan a la evaluación de las cláusulas de restricción, que deben ser satisfechas para que las sentencias se ejecuten con éxito. Básicamente, las restricciones se usan para:
  - Imponer que los valores de un atributo dado no pueden ser nulos (NOT NULL).
  - Imponer que el valor de una columna debe ser único en toda la tabla (UNIQUE KEY).
  - Identificar una columna o grupo de ellas como clave primaria de la tabla (PRIMARY KEY).
  - Identificar una columna o grupo de ellas como clave ajena de otra tabla (FOREIGN KEY).
  - Obligar a que los valores de una columna cumplan una condición (CHECK).
  - Cumplir las restricciones de integridad referencial en el caso de borrado o modificación de la clave ajena (ON {DELETE | MODIFY} {RESTRICT | CASCADE | SET NULL}).

4.7.2. Sentencia CREATE INDEX.

- Crea un índice sobre una tabla. Los índices se utilizan principalmente para permitir un acceso rápido a los datos y forzar la unicidad de filas en una tabla. La sintaxis básica es:

```
CREATE {UNIQUE} INDEX nombre_índice
ON tabla (columnas_indexadas)
```

- La cláusula UNIQUE no permite que el conjunto de campos indexados se repita en la tabla. Normalmente se crea de manera automática un índice cuando se define la clave primaria.

4.7.3. Sentencia CREATE VIEW.

- Crea una vista sobre una tabla o conjunto de tablas. La sintaxis básica es:

```
CREATE {OR REPLACE} {FORCE} VIEW nombre_vista AS SELECT …
```

- La cláusula REPLACE permite sobrescribir una definición existente con otra nueva definición. La cláusula FORCE permite crear una vista aunque las tablas de la subconsulta no existan.

4.7.4. Sentencias DROP y ALTER.

- La sentencia DROP sirve para borrar objetos de la base de datos. Toda sentencia CREATE tiene su equivalente DROP para eliminar el objeto creado. Todas tienen la misma sintaxis:

```
DROP tipo_objeto nombre_objeto_a_borrar
```

- La sentencia ALTER sirve para modificar objetos de la base de datos. En el caso de las tablas, la sentencia ALTER TABLE permite alterar una tabla añadiendo columnas o restricciones, modificando definiciones de columnas o eliminando restricciones. Su sintaxis es la siguiente:

```
ALTER TABLE [usuario.] tabla
[ADD ( {nombre_columna | nombre_restricción}
[,{nombre_columna | nombre_restricción } ] … ) ]
[MODIFY (nombre_columna [,nombre_columna] …) ]
[DROP CONSTRAINT nombre_restricción] …
```

4.8. Sentencias de DCL.

4.8.1. Sentencias de control de transacciones.

- Una transacción es una secuencia atómica de acciones en una base de datos ejecutadas por un usuario. Cada transacción que parte de un estado consistente, si se ejecuta completamente, debe dejar la base de datos en otro estado consistente. El SGBD asegura que la ejecución de transacciones de forma concurrente es igual a alguna ejecución en serie de esas transacciones.
- Durante la transacción, todas las modificaciones que se hacen sobre base de datos no son definitivas, se realizan sobre un espacio reservado para cada transacción. Una vez que la transacción finaliza, las modificaciones temporales se vuelcan a la tabla original.
- Existen tres sentencias para el control de transacciones:
  - *SAVEPOINT nombre_savepoint*. Identifica un punto en una transacción que permite recuperar con posterioridad el estado de la base de datos justo en ese punto.
  - *COMMIT*. Esta sentencia se utiliza para hacer permanentes los cambios de la transacción actual, eliminar los bloqueos y los savepoints realizados, y finalizar la transacción.
  - *ROLLBACK [SAVEPOINT nombre_savepoint]*. Se utiliza para deshacer acciones realizadas en la transacción actual. Sin savepoint deshace todos los cambios, elimina los bloqueos y los savepoints, y finaliza la transacción. Con savepoint, deshace todo lo realizado hasta ese punto, incluyendo bloqueos y savepoints posteriores.

4.8.2. Sentencias de control de acceso a datos.

- Un privilegio es un permiso dado a un usuario para que realice ciertas operaciones, que pueden ser sobre el sistema o sobre un objeto determinado. Un rol es una agrupación de permisos de sistema y de objeto al que se puede asignar un conjunto de usuarios.
- El modo de asignar un privilegio es a través de la instrucción GRANT y el modo de cancelar un privilegio es a través de la instrucción REVOKE. La sintaxis básica de ambas instrucciones es:

```
GRANT [privilegios_sistema | privilegios_objeto | roles]
[ON objeto_base_datos]
TO [usuarios | roles |PUBLIC]
{WITH ADMIN OPTION};

REVOKE [privilegios_sistema | privilegios_objeto | roles]
[ON objeto_base_datos]
FROM [usuarios | roles |PUBLIC];
```

- Es posible asignar o eliminar más de un privilegio de sistema o rol, separándolos por comas. También es posible asignar o eliminar uno o varios privilegios a varios usuarios, separándolos por comas. Si se asigna o elimina el privilegio a un rol, se asignará o eliminará a todos los usuarios que tengan ese rol. Si se asigna o elimina el privilegio a PUBLIC, se asignará o eliminará a todos los usuarios actuales y futuros de la base de datos.
- La cláusula WITH ADMIN OPTION permite que el privilegio/rol que hemos concedido, pueda ser concedido a otros usuarios por el usuario al que estamos asignando.