1º CFGS DESARROLLO DE APLICACIONES MULTIPLATAFORMA

UD 4. El lenguaje SQL - DDL

Módulo: Bases de Datos

Centro de Enseñanza
Gregorio Fernández

Macarena Cuenca Carbajo

1. Introducción

- El **DDL** (*Data Definition Language)* es la parte del lenguaje SQL que realiza la función de definición de datos del SGBD.
- Fundamentalmente se encarga de la creación, modificación y eliminación de los objetos de la base de datos (es decir de los metadatos).
- Vamos a manejar las sentencias **CREATE**, **DROP** y **ALTER**.
- También vamos a trabajar con algo tan importante como las **restricciones**. Gracias a ellas, vamos a asegurar la integridad de la base de datos.

1. Introducción

- Cada usuario de una base de datos posee un **esquema**.
- El esquema suele tener el mismo nombre que el usuario y sirve para almacenar los objetos de esquema, es decir los objetos que posee el usuario.
- Esos objetos pueden ser: tablas, vistas, índices y otros objetos relacionados con la definición de la base de datos.
- Los objetos son creados y manipulados por los usuarios.
- En principio sólo los administradores y los usuarios propietarios pueden acceder a cada objeto, salvo que se modifiquen los privilegios del objeto para permitir el acceso a otros usuarios.
- Hay que tener en cuenta que ninguna instrucción DDL puede ser anulada por una instrucción ROLLBACK, por lo que hay que tener mucha precaución a la hora de utilizarlas. Es decir, las instrucciones DDL generan acciones que no se pueden deshacer (salvo que dispongamos de alguna copia de seguridad).

2. Creación de tablas: CREATE TABLE

- Sintaxis:

```
CREATE TABLE  nombre_tabla
(
   columna1      tipo_dato        [NOT NULL],
   columna2      tipo_dato        [NOT NULL],
  …
) [TABLESPACE espacio_de_tabla];
```

- Donde:
  - **columna1, columna2**: son los nombres de las columnas.
  - **tipo_dato**: indica el tipo de dato (VARCHAR2, NUMBER, …) de cada columna.
  - **TABLESPACE espacio_de_tabla**: señala el TABLESPACE para almacenar la tabla.
  - **NOT NULL**: indica que la columna es obligatoria, es decir, que debe contener alguna información.

2. Creación de tablas: CREATE TABLE

- Observaciones:
  - Las definiciones individuales de columnas se separan mediante comas.
  - No se pone coma después de la última definición de columna.
  - Las mayúsculas y minúsculas son indiferentes a la hora de crear una tabla.
- Si intentamos crear una tabla y ya existe otra tabla con este nombre, aparecerá un mensaje de error.
- Los usuarios pueden consultar las tablas creadas por medio de la vista **USER_TABLES**. Esta vista contiene información acerca de las tablas: nombre de la tabla, nombre del tablespace, número de filas, información de almacenamiento, etc.

2.1. Creación de tablas a partir de una SELECT

- La sentencia **CREATE TABLE** permite crear una tabla a partir de una consulta a otra tabla ya existente.
- La nueva tabla contendrá los datos obtenidos en la consulta.
- Sintaxis:

```
CREATE TABLE nombre_tabla (
        [columna1, columna2, …]
) [TABLESPACE espacio_de_tabla]
  AS consulta;
```

- No es necesario especificar tipos ni tamaño de las columnas, ya que vienen determinados por los tipos y los tamaños de las recuperadas en la consulta.
- La consulta puede contener una subconsulta, una combinación de tablas o cualquier sentencia SELECT válida.
- Las restricciones con nombre no se crean en una tabla desde la otra, sólo se crean aquellas restricciones que carecen de nombre.

3. Vistas (I)

- Vistas que permiten obtener información de los objetos que son **propiedad del usuario**:

- **USER_TABLES:** tablas que son propiedad del usuario.
- **USER_OBJECTS**: objetos que son propiedad del usuario.
- **USER_CATALOG**: tablas, vistas, sinónimos y secuencias propiedad del usuario.

4. Integridad

- Cuando almacenamos datos en tablas, éstos se deben de ajustar a una serie de restricciones predefinidas. Por ejemplo, que una columna no pueda almacenar valores negativos, que una cadena de caracteres se deba almacenar en mayúsculas o que una columna no pueda ser cero.
- La **integridad** hace referencia al hecho de que los datos de la base de datos han de ajustarse a restricciones antes de almacenarse en ella.
- Existe otro tipo de integridad, que es la **integridad referencial**, la cual garantiza que los valores de una columna (o columnas) de una tabla (clave ajena) dependan de los valores de otra columna (o columnas) de otra tabla (clave primaria).

4.1. Restricciones

- La orden CREATE TABLE permite definir distintos tipos de restricciones sobre una tabla:
  1. Claves primarias
  2. Claves ajenas
  3. Obligatoriedad
  4. Valores por defecto
  5. Verificación de condiciones.
- Para definir las restricciones se usa la cláusula **CONSTRAINT**.
- Ésta puede restringir una sola columna (**restricción de columna**) o un grupo de columnas de una misma tabla (**restricción de tabla**).

4.1. Restricciones. Sintaxis

- Sintaxis CREATE TABLE con **restricción de columna**:

```
CREATE TABLE nombre_tabla (
columna1        tipo_dato
        [CONSTRAINT nombre_restricción]
        [NOT NULL] [UNIQUE] [PRIMARY KEY] [DEFAULT valor]
        [REFERENCES Nombretabla [(columna1 [, columna2])]
                [ON DELETE CASCADE]]
        [CHECK (condición)],
columna2        tipo_dato
        [CONSTRAINT nombre_restricción]
        [NOT NULL] [UNIQUE] [PRIMARY KEY] [DEFAULT valor]
        [REFERENCES Nombretabla [(columna1 [, columna2])]
                [ON DELETE CASCADE]]
        [CHECK (condición)],
         ...
) [TABLESPACE espacio_de_tabla];
```

4.1. Restricciones. Sintaxis

- Sintaxis CREATE TABLE con **restricción de tabla** :

```
CREATE  TABLE  nombre_tabla  (
      columna1    tipo_dato,
      columna2    tipo_dato,
      columna3    tipo_dato,
...
[CONSTRAINT nombre_restricción]
          { [UNIQUE] | [PRIMARY KEY] (columna1 [,columna2])},
[CONSTRAINT nombre_restricción]
                    [FOREIGN KEY] (columna1 [, columna2])
                [REFERENCES Nombretabla [(columna1 [, columna2])]
                [ON DELETE CASCADE]],
[CONSTRAINT nombre_restricción]
          [CHECK (condición)],
   ...
) [TABLESPACE espacio_de_tabla];
```

4.2. Restricciones: PRIMARY KEY

- Una **clave primaria** es una columna o un conjunto de columnas que identifican unívocamente a cada fila.
- Debe ser única, no nula y obligatoria.
- Para definirla usamos la restricción **PRIMARY KEY**.
- Como máximo podemos definir una clave primaria por tabla.
- Cuando se crea una clave primaria, automáticamente se crea un **índice** que facilita el acceso a la tabla.

4.3. Restricciones: FOREIGN KEY

- Una clave ajena está formada por una o varias columnas que están asociadas a una clave primaria de otra o de la misma tabla.
- Se pueden definir tantas claves ajenas como sea preciso, y pueden estar o no en la misma tabla que la clave primaria.
- El valor de la columna o columnas que son claves ajenas debe ser NULL o igual a un valor de la clave referenciada (**regla de integridad referencial**).

4.4. Restricciones: NOT NULL

- Restricción de obligatoriedad.
- Asociada a una columna significa que no puede tener valores nulos, es decir, que ha de tener obligatoriamente un valor.

4.5. Restricciones: DEFAULT

- En el momento de crear una tabla podemos asignar valores por defecto a las columnas con esta clausula.
- De esta forma proporcionamos un valor por omisión cuando el valor de la columna no se especifica en la cláusula INSERT.
- En la especificación DEFAULT no se puede hacer referencias a columnas o a funciones PL/SQL.

4.6. Restricciones: CHECK

- Con esta restricción podemos hacer que los valores de las columnas de tablas estén limitados dentro de un rango o el cumplimiento de ciertas condiciones.
- Actúa como una cláusula WHERE.
- Puede hacer referencia a una o a más columnas, pero no a valores de otras filas.
- En una cláusula CHECK no se pueden incluir subconsultas ni las pseudocolumnas SYSDATE, UID y USER.

4.7. Restricciones: UNIQUE

- Restricción de unicidad.
- Evita valores repetidos en la misma columna.
- Puede contener una o varias columnas.
- Es similar a la restricción PRIMARY KEY, salvo que son posibles varias columnas UNIQUE definidas en una tabla y admite valores NULL.
- Al igual que en PRIMARY KEY, cuando se define una restricción UNIQUE se crea un índice automáticamente.

5. Vistas (II)

- Vistas que permiten obtener información sobre las **restricciones**:

- **USER_CONSTRAINTS: definiciones de restricciones de tablas propiedad del usuario.**
- **ALL_CONSTRAINTS: definiciones de restricciones sobre tablas a las que puede acceder el usuario.**
- **DBA_CONSTRAINTS: todas las definiciones de restricciones sobre todas las tablas.**

- **USER_CONS_COLUMNS: información sobre las restricciones de columnas en tablas del usuario.**
- **ALL_CONS_COLUMNS: información sobre las restricciones de columnas en tablas a las que puede acceder el usuario.**
- **DBA_CONS_COLUMNS: información sobre todas las restricciones de columnas.**

6. Supresión de tablas: DROP TABLE

- Sintaxis:

  DROP TABLE [usuario].nombre_tabla [**CASCADE CONSTRAINTS**];

- Donde:
  - **CASCADE CONSTRAINTS** elimina las restricciones de integridad referencial que hagan referencia a la clave primaria de la tabla borrada.
- Cada usuario puede borrar sus propias tablas, sólo el administrador de la base de datos o algún usuario con privilegios puede borrar las tablas de otro usuario.
- Al borrar una tabla también se suprimen los índices y los privilegios asociados a ella.
- Las vistas y los sinónimos creados a partir de esta tabla dejan de funcionar, pero siguen existiendo en la base de datos.

6.1. Sentencia TRUNCATE TABLE

- Sintaxis:

  TRUNCATE TABLE [usuario].nombre_tabla;

- Permite suprimir todas las filas de una tabla y liberar el espacio ocupado para otros usos sin que desaparezca la definición de la tabla de la base de datos.
- Es una orden que no genera información de retroceso (ROLLBACK), es decir, una sentencia TRUNCATE no se puede anular, como tampoco activa disparadores DELETE.
- Por eso, la eliminación de filas con la orden TRUNCATE es más rápida que con DELETE.
- No se puede truncar una tabla cuya clave primaria sea referenciada por la clave foránea de otra tabla. Antes de truncar la tabla hay que desactivar la restricción.

7. Modificación de tablas: ALTER TABLE

- Sintaxis:

```
ALTER  TABLE  nombre_tabla
   [ADD (columna1 [, columna2] … )]
   [MODIFY (columna1 [, columna2] …)]
   [DROP COLUMN (columna1 [, columna2] …)]
   [ADD CONSTRAINT restricción]
   [DROP CONSTRAINT restricción]
   [DISABLE CONSTRAINT restricción]
   [ENABLE CONSTRAINT restricción];
```

7.1. Modificación de tablas: añadir columnas

- Sintaxis:

```
ALTER TABLE nombre_tabla
  ADD (columna1 [, columna2] … )
```

- Conseraciones:
  - Si la columna no está definida como NOT NULL, se le puede añadir en cualquier momento.
  - Si la columna está definida como NOT NULL hay que seguir los siguientes pasos:
    1. Añadir la columna sin especificar NOT NULL.
    2. Dar valor a la columna para cada una de las filas.
    3. Modificar la columna a NOT NULL.

7.2. Modificación de tablas: modificar columnas

- Sintaxis:

```
ALTER  TABLE  nombre_tabla
   MODIFY (columna1 [, columna2] … )
```

- Consideraciones:
  - Se puede aumentar la longitud de una columna en cualquier momento.
  - Al disminuir la longitud de una columna que tiene datos no se puede dar menor tamaño que el máximo valor almacenado.
  - Es posible aumentar o disminuir el número de posiciones decimales en una columna de tipo NUMBER.
  - La opción MODIFY ... NOT NULL sólo será posible cuando la tabla no contenga ninguna fila con valor nulo en la columna que se modifica.

7.3. Modificación de tablas: eliminar columnas

- Sintaxis:

```
ALTER  TABLE  nombre_tabla
   DROP COLUMN (columna1 [, columna2] … )
```

- Hay que tener en cuenta que no se pueden borrar todas las columnas de una tabla y tampoco se pueden eliminar claves primarias referenciadas por claves ajenas.

7.4. Modificación de tablas: añadir y eliminar restricciones

- Sintaxis:

ALTER TABLE nombre_tabla ADD CONSTRAINT nombre_restricción

ALTER TABLE nombre_tabla DROP CONSTRAINT nombre_restricción

7.5. Modificación de tablas: activar y desactivar restricciones

- Sintaxis:

```
ALTER TABLE nombre_tabla DISABLE CONSTRAINT  nombre_restricción
ALTER TABLE nombre_tabla ENABLE CONSTRAINT  nombre_restricción
```

- Por defecto, las restricciones se activan al crearlas. Se pueden desactivar añadiendo la cláusula DISABLE al final de la restricción.

8. Vistas

- Una vista es una **tabla lógica** que no contiene información, sino que su información está basada en la que contienen otras tablas, llamadas **tablas base**, y siempre refleja los datos de estas tablas.
- Una vista es una sentencia SQL.
- Las vistas tienen la misma estructura que una tabla: filas y columnas, y se tratan de forma semejante a una tabla.
- Se pueden realizar todas las operaciones que se realizan con ellas y al estar basadas en tablas, también las rigen todas las restricciones de integridad de sus tablas base.
- Usos:
  - Ocultar la complejidad del modelo de datos.
  - Simplificar el acceso a los datos.
  - Tener acceso de una manera simple a consultas muy complejas.
  - Presentar los datos en una perspectiva diferente de las tablas en las que se basan.
  - Dar un nivel de seguridad adicional, al restringir a los usuarios el acceso a ciertas filas de datos o columnas de las tablas en las que se basan.

8.1. Creación de vistas: CREATE VIEW

- Sintaxis:

```
CREATE [OR REPLACE] [FORCE] VIEW nombre_vista  [(columna1  [,columna2])]
AS  consulta
[WITH {CHECK OPTION |READ ONLY} CONSTRAINT nombre_restricción];
```

- Donde:
  - **columna1, columna2**: son los nombres de las columnas que va a contener la vista. Si no se ponen, se asumen los nombres de columna devueltos por la consulta.
  - **OR REPLACE:** crea de nuevo la vista si ya existía.
  - **FORCE:** fuerza la creación del a vista, aunque las tablas en las que se basa no existan.
  - **WITH CHECK OPTION**: es la opción de comprobación para una vista. Con ella, SQL comprueba automáticamente cada operación INSERT, UPDATE y DELETE sobre la vista para asegurarse que las filas resultantes satisfagan la condición de definición de la vista.
  - **WITH READ ONLY**: indica sólo se puede hacer SELECT en la vista.

8.2. Borrado de vistas: DROP VIEW

- Sintaxis:

  DROP VIEW nombre_vista

9. Índices

- Un **índice** es una estructura de datos definida sobre una o varias columnas de una tabla, que permite localizar de forma rápida las filas de la tabla en base al contenido de la columna indexada, además de permitir recuperar las filas de la tabla ordenadas por esa columna.
- Cuando se define un índice sobre una columna, los registros que se recuperen utilizando el índice aparecerán ordenados por el campo indexado.
- El espacio en disco requerido para almacenar un índice es menor que el espacio de almacenamiento de la tabla, puesto que los índices generalmente contienen solamente los campos clave con los que la tabla será ordenada, y excluyen el resto de los detalles de la tabla.
- La definición de los índices de la base de datos es tarea del administrador de la base de datos, ya que es una tarea compleja.

9.1. Creación de índices

Al crear una tabla

- Cuando se define una clave primaria o una columna UNIQUE durante la creación de una tabla o su mantenimiento, Oracle crea automáticamente un índice de tipo UNIQUE.
- Un índice UNIQUE es aquel que no permite valores duplicados en la columna indexada.

Sobre una tabla existente

```
CREATE [UNIQUE] INDEX nombre_indice  [(columna1  [,columna2])]
ON
nombre_tabla (columna1 [,columna2]…) [REVERSE];
```

Donde:

- **UNIQUE:** indica que columnas indexadas no pueden contener duplicados.
- **columna1, columna2:** columna de la tabla sobre la cuál se creará el índice.
- **REVERSE:** indica a Oracle que invierta los bytes del valor indexado, lo cual, puede mejorar la distribución de los datos y el procesamiento cuando se insertan muchos valores secuenciales.

9.2. Monitorizar el uso de índices

- Para saber si un índice se está utilizando, podemos consultar las estadísticas sobre el uso de índices.
- Para ello en primer lugar, deberemos activar la monitorización del índice que queramos, utilizando el siguiente comando SQL:

ALTER INDEX nombre_índice **MONITORING USAGE;**

- De la misma forma, para desactivar la monitorización ejecutaremos el comando SQL:

ALTER INDEX nombre_índice **NOMONITORING USAGE;**

9.3. Ventajas e inconvenientes de los índices

- **Ventajas**
  - La utilización de índices puede **mejorar el rendimiento de las consultas**, ya que se reduce la E/S global en el disco.
- **Inconvenientes**
  - Las tablas utilizadas para almacenar los índices **ocupan espacio**.
  - Los índices **consumen recursos** ya que cada vez que se realiza una operación de actualización, inserción o borrado en la tabla indexada, se tienen que actualizar todas las tablas de índice definidas sobre ella. Por estos motivos no es buena idea definir índices indiscriminadamente.

9.4. Consideraciones sobre los índices

1. Hay que evitar crear demasiados índices en tablas que se **actualizan con mucha frecuencia** y procurar definirlos con el menor número de columnas posible.
2. Es conveniente utilizar un número mayor de índices para mejorar el rendimiento de consultas en tablas con **pocas necesidades de actualización**, pero con **grandes volúmenes de datos**.
3. Crear índices en **tablas pequeñas** puede no ser una solución óptima, porque puede provocar que el optimizador de consultas tarde más tiempo en realizar la búsqueda de los datos a través del índice que en realizar un simple recorrido de la tabla.

10. Secuencias

- Permiten generar valores enteros secuenciales únicos.
- Generalmente se usan para asignar valores a las claves primarias de las tablas.
- Sintaxis:

```
CREATE SEQUENCE nombre_secuencia
    [ START WITH entero
    INCREMENT BY entero
    MAXVALUE entero
    MINVALUE entero
    CYCLE | NOCYCLE ];
```

- Donde:
  - **START WITH:** indica el valor inicial desde el que comenzará la generación de números secuenciales. Si no se especifica, se inicia con el valor indicado en MINVALUE.
  - **INCREMENT BY**: es la diferencia entre los números de la secuencia. Si no se indica, por defecto es 1.
  - **MAXVALUE**: valor máximo para la secuencia.

10. Secuencias

- **MINVALUE**: valor mínimo de la secuencia. Si se omite será 1.
- **CYCLE**: indica que cuando la secuencia llegue al máximo valor se reinicie, comenzando con el mínimo valor nuevamente.Si se omite, por defecto la secuencia se crea NOCYCLE.

- Una vez creada la secuencia, para recuperar sus valores usaremos las pseudocolumnas **CURRVAL** y **NEXTVAL** de la siguiente forma:

  nombre_secuencia.**NEXTVAL;**

  nombre_secuencia.**CURRVAL;**

- Si queremos eliminar una secuencia de la base de datos usaremos la sentencia:

  DROP SEQUENCE nombre_secuencia;

11. Vistas (III)

- Vistas que permiten obtener información sobre las **vistas, índices** y **secuencias**:

- **USER_VIEWS: información sobre las vistas del usuario.**
- **USER_INDEXES: información sobre los índices del usuario.**
- **USER_SEQUENCES: información sobre las secuencias del usuario.**