Administración de BBDD Oracle 21c XE

En este capítulo se verá el lenguaje SQL enfocado a la administración de una base de datos. Un subconjunto de SQL, el DDL (*Data Definition Language*) se centra en la ejecución de consultas a una base de datos para crear o modificar la estructura de una base de datos. Se verán las actividades habituales de un administrador de BD, como gestionar el almacenamiento, gestionar los objetos de bases de datos, control de acceso y por último la importación y exportación de datos.

Tabla de contenido

Administración de BBDD Oracle 21c XE

1 Introducción

Toda organización que maneja grandes volúmenes de datos requiere un sistema de gestión de base de datos (SGBD). La persona o grupo de personas encargadas de administrar, supervisar y asegurar el adecuado uso de los datos dentro de un SGBD son los administradores de base de datos (DBA *DataBase Administrator*). Un DBA es la persona con más conocimientos sobre base de datos en una organización. Como tal, debe entender las reglas básicas de la tecnología de base de datos relacional.

1.1 Funciones de un DBA

Las tareas que normalmente debe realizar el DBA son, entre otras:

- ✔ Modelado de Datos y Diseño de BD.- Un DBA debe ser un profesional experto en la recopilación y análisis de las necesidades del usuario para obtener modelos de datos conceptuales y lógicos. Un modelo conceptual de datos describe los requisitos de datos a un nivel muy alto, un modelo de datos lógico proporciona una representación de la BD orientado a la BD relacional. El DBA utiliza técnicas de normalización para ofrecer modelos de datos que reflejen las necesidades de los datos de la empresa.
- ✔ Instalación del software de Oracle.- Normalmente en colaboración y coordinado con el administrador del sistema operativo. Esto incluye motor de la BD, actualizaciones, herramientas administrativas, importación de datos, etc.
- ✔ Integración con aplicaciones.- La mayoría de las empresas hoy en día utilizan

aplicaciones propias o de terceros, muy pocas de estas aplicaciones funcionan de manera aislada. En otras palabras, las aplicaciones tienen que interconectarse unas con otras, generalmente utilizando base de datos como el medio para compartir los datos. Los DBAs a menudo se involucran en los procesos de integrar las aplicaciones existentes con las bases datos que administran. Esto puede incluir la creación de aplicaciones a medida, scripts, etc.

- ✔ Resguardo y recuperación de datos.- Uno de los aspectos más fundamentales del trabajo del DBA es proteger los datos de la organización. Esto incluye hacer copias de seguridad periódicas de los datos y mantenerlos a salvo de la destrucción accidental o intencional. Además, diseñar, implementar y probar un plan de recuperación para que cuando se presenten los problemas, los datos se pueden restaurar rápidamente.

- ✔ Planificación de la capacidad.- En la mayoría de las organizaciones, el número y tamaño de las BDs crece rápidamente. Es la responsabilidad del DBA gestionar el creciente volumen de datos y diseñar los planes apropiados para administrarlos. Esto incluye también la gestión del hardware donde se almacenan los datos.

- ✔ Administración de cambios.- La configuración del servidor de BD, el esquema de BD, el código, y muchas otras facetas del ecosistema de aplicaciones cambian con el tiempo. A menudo es la responsabilidad del DBA realizar el análisis de impacto antes de realizar los cambios dentro de una SGBD. Implementar cambios, hacer pruebas piloto y documentar todos los cambios y procedimientos es parte del trabajo de un DBA.

- ✔ Desarrollo de aplicaciones.- Muchos DBA deben de desarrollar aplicaciones y scripts con el objetivo de automatizar tareas relacionadas con la inserción, eliminación o borrado de información dentro del SGBD. En general, éste debe de colaborar a nivel de integración de sistema con los desarrolladores de aplicaciones, por lo que a veces se ve en la obligación de desarrollar código para casos específicos.

- ✔ Controlar la seguridad de la BD.- Esto incluye la gestión de usuarios, privilegios, roles, auditorias de seguridad.

- ✔ Documentar todo lo relativo a la BD.- La responsabilidad final de un DBA en la administración de la estructura de una BD es la documentación. Es de suma importancia saber que modificaciones han sido efectuadas, como fueron realizadas y cuando fueron establecidas. Una modificación sobre la estructura de la BD pudiera ocasionar un error que no apareciera a corto plazo; una vez que este surja, sin la documentación adecuada sobre las modificaciones realizadas, él diagnostico resultaría extremadamente complicado.

1.2 DDL

El lenguaje de definición de datos (*Data Definition Language*, DDL) es un subconjunto de SQL que permite a los DBA llevar a cabo las tareas de definición de las estructuras que almacenarán los datos, su control de acceso y los procedimientos o funciones que permitan

consultarlos.

El DDL incluye sentencias cuyo efecto es modificar el esquema de la base de datos, añadiendo, cambiando o eliminando las definiciones de tablas y otros objetos de BD. Estas declaraciones se pueden mezclar libremente con otras sentencias SQL, por lo que el DDL no es realmente un lenguaje independiente, sino un subconjunto de SQL.

En las siguientes secciones veremos las sentencias DDL que permiten administrar los objetos de la base de datos.

2 Administración del almacenamiento

Ya vimos en el capítulo de introducción a Oracle que en la BD disponemos de estructuras lógicas y físicas para el almacenamiento de los datos. Una base de datos está formada por una o varias unidades lógicas llamadas TABLESPACES. Cada tablespace está formado por uno o más segmentos los cuales se dividen en extensiones y finalmente en bloques de datos.

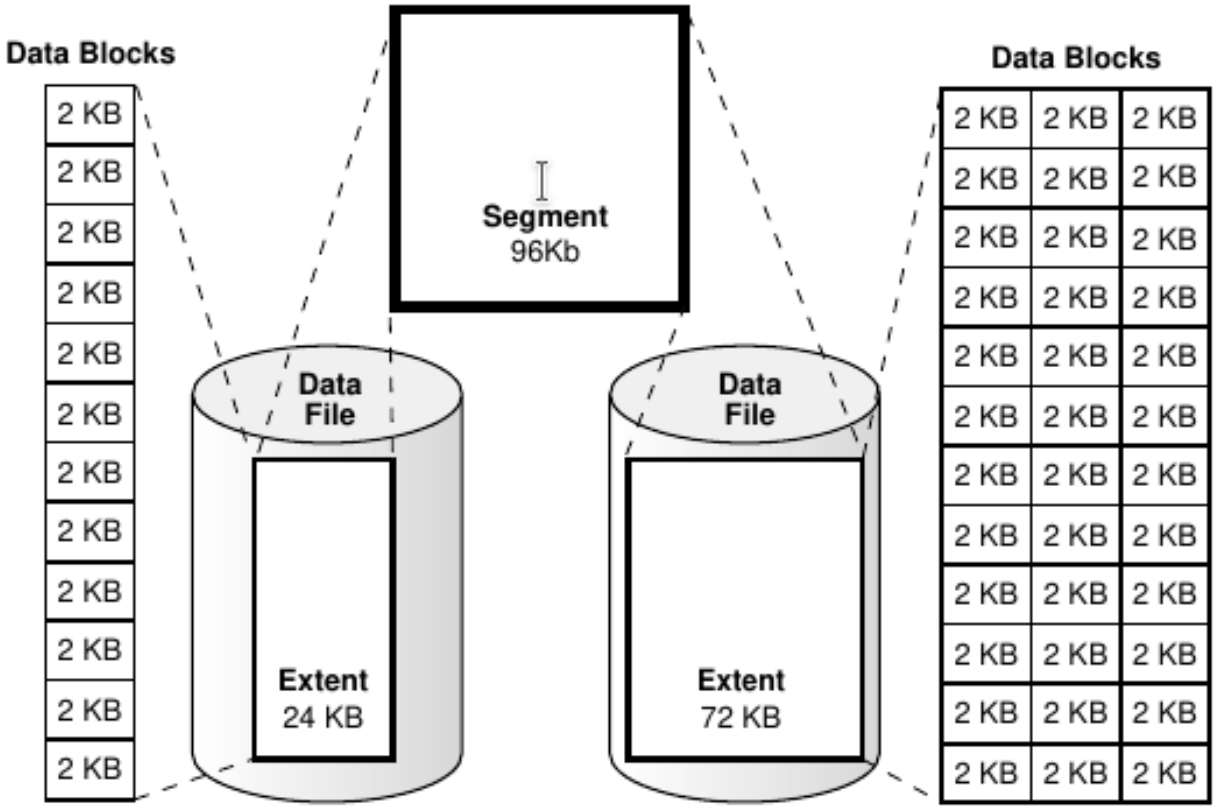

Figura 1.- Estructuras de almacenamiento

Además, cada una de estos TABLESPACES está formado por uno o varios ficheros físicos que son los DATAFILES. Un DATAFILE solamente puede pertenecer a un TABLESPACE. Por lo tanto, los datafiles de una base de datos son todos los datafiles que forman parte de todos los tablespaces de la base. En una BD hay al menos un tablespace, por lo que durante el proceso de creación de la base de datos siempre se indica el

tablespace principal de ésta, que se llama `SYSTEM`.

De igual manera, cuando se crea un tablespace se debe indicar obligatoriamente también el nombre de al menos un datafile que formará parte de ese tablespace. El datafile es un fichero físico al que le tendremos que asignar un directorio, un nombre y un tamaño.

En esta sección veremos la gestión de la estructuras, físicas y lógicas, que emplea una BD Oracle para almacenar la información. Las sentencias DDL que se verán en este apartado gestionan el almacenamiento de la BD y por tanto deben ejecutarse con una sesión del usuario SYS como administrador de la BD ya que cuenta con los privilegios necesarios para ello.

2.1 Creación de TABLESPACE

Utilizar varios TABLESPACES en una BD permite aumentar la flexibilidad y el rendimiento de la misma. Cuando una BD tiene varios TABLESPACES se puede:

- ✔ Separar los datos de usuario de los datos del diccionario de datos para reducir la sobrecarga en la E/S.
- ✔ Separar los datos de una aplicación de los datos de otra previene que las aplicaciones se vean afectadas por paradas de un TABLESPACE.
- ✔ Almacenar los datos en diferentes TABLESPACES en diferentes unidades de disco reduce los cuellos de botella en la E/S.
- ✔ Aumento de la disponibilidad de datos cuando TABLESPACES individuales se ponen fuera de línea mientras que otros permanecen operativos.
- ✔ Se pueden realizar copias de seguridad individuales de los TABLESPACES.

En la mayoríia de los sistemas operativos especificas el tamaño y el nombre del fichero completamente cualificado de los ficheros de datos cuando creas un nuevo TABLESPACE o modificas uno ya existenten añadiendo más ficheros de datos. En este caso sería conveniente crear primero la estructura de directorios y directorios para el esquema de la BD. Si estás creando un nuevo TABLESPACE o modificando uno existente, la BD automáticamente coloca y da formato a los ficheros de datos.

Hay tres tipos de TABLESPACE:

- ✔ TABLESPACE permanente.- Contiene objetos de esquemas de BD permanentes. Estos objetos se almacenan en los ficheros de datos.
- ✔ Undo TABLESPACE.- Es un tipo de TABLESPACE utilizado por Oracle para gestionar los datos undo cuando se ejecuta la BD en modo deshacer automático.
- ✔ TABLESPACE temporal.- El cual contiene objetos del esquema exclusivamente para la duración de la sesión actual. Los objetos de esquema en un TABLESPACE temporal se almacenan en ficheros temporales.

Oracle recomienda que cada esquema de usuario tenga su propio TABLESPACE para

datos y TABLESPACE temporal para las operaciones en la BD.

En esta sección nos centraremos en los tablespaces permanente y temporal. Para crear un nuevo TABLESPACE se utiliza la sentencia `CREATE TABLESPACE`. Hay que tener el privilegio `CREATE TABLESPACE` para poder ejecutarla. En el caso de un TABLESPACE permanente la sintaxis es:

```
CREATE TABLESPACE <nombre_tablespace>
DATAFILE 'nombre_fichero' SIZE <tamaño>
    AUTOEXTEND {OFF | ON NEXT <tamaño>
    MAXSIZE {UNLIMITED | <tamaño>}

    [, 'nombre_fichero' SIZE <tamaño>
    AUTOEXTEND {OFF | ON NEXT <tamaño>
    MAXSIZE {UNLIMITED | <tamaño>} ]

    [, …]
```

El parámetro `<tamaño>` se define así:

```
entero { K | M | G | T | P | E }
```

Donde:

- ✔ `<nombre_tablespace>` .- Nombre del TABLESPACE.
- ✔ `DATAFILE 'nombre_fichero', …` .- El nombre del primer DATAFILE del TABLESPACE. El nombre del fichero debe ser un path absoluto. Si es un nombre de archivo relativo entonces Oracle lo colocará en el directorio por defecto establecido en el parámetro de inicialización del servidor `DB_CREATE_FILE_DEST`. Pueden especificarse varios ficheros de datos separados por comas.
- ✔ `SIZE <tamaño>` .- Tamaño de los DATAFILES especificados.
- ✔ `AUTOEXTEND { OFF | ON NEXT <tamaño> }` .- Clausula que indica si el DATAFILE aumenta o no de tamaño y en el caso de que lo haga hay que espeficiar los tamaños de la extensión del tamaño del archivo.
- ✔ `MAXSIZE { UNLIMITED | <tamaño> }` .- Tamaño máximo del archivo de datos cuando se incluye una clausula `AUTOEXTEND`.

Se pueden especificar más de un DATAFILE separando por comas la definición de cada uno. La sintaxis completa de la sentencia `CREATE TABLESPACE` incluye más opciones, como encriptación o compresión de datos. Para una mayor referencia consultar la documentación de Oracle.

Por ejemplo, para crear un TABLESPACE podemos escribir esta sentencia

```
CREATE TABLESPACE 'ts_user1' DATAFILE
    '/opt/oracle/oradata/XE/XEPDB1/ts_user1/ts_user1.dbf'
    SIZE 100M
```

```
    AUTOEXTENT ON NEXT 100M MAXSIZE UNLIMITED;
```

Para crear un TABLESPACE temporal tenemos la siguiente sintaxis:

```
CREATE TEMPORARY TABLESPACE <nombre_tablespace>
TEMPFILE 'nombre_fichero' SIZE <tamaño>
    AUTOEXTEND {OFF | ON NEXT <tamaño>
    MAXSIZE {UNLIMITED | <tamaño>}

    [, 'nombre_fichero' SIZE <tamaño>
    AUTOEXTEND {OFF | ON NEXT <tamaño>
    MAXSIZE {UNLIMITED | <tamaño>} ]

    [, …]
```

Como se puede apreciar a la sintaxis anterior se ha añadido la cláusula `TEMPORARY` y en lugar de crear `DATAFILE` se crea un `TEMPFILE`. El resto de parámetros y opciones son similares a lo que hemos visto.

Por ejemplo, para crear un TABLESPACE temporal para un usuario emplearíamos la siguiente sentencia

```
CREATE TABLESPACE 'ts_temp1'
TEMPFILE '/opt/oracle/oradata/XE/XEPDB1/ts_temp1/ts_temp1.dbf'
    SIZE 50M
    AUTOEXTENT ON NEXT 50M MAXSIZE UNLIMITED;
```

Una vez creados los TABLESPACES permanente y temporal ya pueden asignarse a los usuarios mediante las sentencias `CREATE USER` o `ALTER USER` que se verán más adelante en este mismo capítulo.

2.2 Gestionar un TABLESPACE

Una vez creado un TABLESPACE podemos realizar una serie de operaciones de mantenimiento sobre los mismos. Para cambiar las opciones de un TABLESPACE disponemos de la sentencia `ALTER TABLESPACE` que tiene la siguiente sintaxis:

```
ALTER TABLESPACE <nombre_TS>
[ADD DATAFILE 'nombre_fichero' SIZE <tamaño>
    AUTOEXTEND {OFF | ON NEXT <tamaño>
    MAXSIZE { UNLIMITED | <tamaño>}
    [, ...]
]
[DROP DATAFILE 'nombre_fichero']
[RENAME DATAFILE 'nombre_antiguo' [,…] TO 'nombre_nuevo' [,…] ]
[RENAME TO <nuevo_nombre>]
[OFFLINE | ONLINE]
[READ ONLY | READ WRITE]
```

Donde:

- ✔ `<nombre_TS>` .- Nombre del TABLESPACE a modificar.

- ✔ `ADD DATAFILE` .- Añade un nuevo `DATAFILE` al `TABLESPACE`. La sintaxis a continuación es similar a la creación de un `DATAFILE` en la sentencia `CREATE TABLESPACE`.
- ✔ `DROP DATAFILE 'nombre_fichero'` .- Elimina el datafile indicado.
- ✔ `RENAME DATAFILE 'nombre_antiguo' TO 'nombre_nuevo'` .- Renombra un datafile.
- ✔ `RENAME TO <nuevo_nombre>'` .- Renombra el tablespace.
- ✔ `OFFLINE | ONLINE` .- Pone el `TABLESPACE` fuera de línea o en línea. Esto se verá más adelante en esta misma sección.
- ✔ `READ ONLY | READ WRITE` .- Establece el `TABLESPACE` en solo lectura, o en lectura y escritura. Esto se verá más adelante en esta misma sección.

Con la sentencia anterior podemos:

- ✔ Aumentar el tamaño del tablespace añadiendo más datafiles.
- ✔ Borrar datafiles de un tablespace.
- ✔ Cambiar el nombre de los datafiles de un tablespace.
- ✔ Cambiar el nombre de un tablespace.
- ✔ Poner los TABLESPACES fuera de línea o en línea
- ✔ Configurar los TABLESPACES en modo solo lectura o lectura y escritura.

2.2.1 Tablespace online y offline

Un tablespace puede estar en dos estados: Online y Offline. Que un tablespace esté online significa que está disponible para operar en él, y por tanto, los objetos de esquema creados dentro del tablespace están disponibles para su consulta y operación. Por otro lado, si el tablespace está offline quiere decir que no se puede utilizar, y consecuentemente los objetos de esquema contenidos en el tablespace no están disponibles. Cuando creamos un tablespace, se crea por defecto en estado online y, por lo tanto, podemos crear en dicho tablespace objetos como índices, tablas, etc.

Generalmente un tablespace se pone offline para realizar copias de seguridad o cualquier operación de mantenimiento sobre la BD o sobre las aplicaciones de BD. También, cuando se desea cambiar el nombre y la ubicación de los datafiles del tablespace.

Para cambiar el estado de un tablespace a offline disponemos de la siguiente sentencia

```
ALTER TABLESPACE <nombre_TS> OFFLINE;
```

Para devolverle el estado online al tablespace disponemos de la siguiente sentencia

```
ALTER TABLESPACE <nombre_TS> ONLINE;
```

2.2.2 Tablespaces de solo lectura

Convertir un tablespace a solo lectura evita operaciones de escritura sobre los datos del tablespace. El principal propósito de los tablespaces de solo lectura es eliminar la necesidad de realizar copias de seguridad de grandes porciones de datos estáticos. Los tablespaces de solo lectura también proveen una forma de proteger datos históricos que los usuarios no pueden modificar. Cuando se pasa un tablespace a solo lectura no se pueden hacer actualizaciones de datos en las tablas que contiene, independientemente del nivel de privilegios de usuario.

Para pasar un tablespace a solo lectura se escribe la siguiente sentencia:

```
ALTER TABLESPACE <nombre_TS> READ ONLY;
```

Si queremos ponerlo otra vez en modo lectura y escritura tenemos que ejecutar la siguiente sentencia:

```
ALTER TABLESPACE <nombre_TS> READ WRITE;
```

2.2.3 Borrar tablespaces

Podemos borrar un tablespace y su contenido de la base de datos si este no va a ser requerido. Una vez es borrado, los datos en el tablespace no son recuperables. Se recomienda encarecidamente hacer una copia de seguridad previa para prevenir el borrado accidental de datos.

Opcionalmente se puede indicar que se borren también los datafiles asociados al tablespace. Si no se hace, se tendrán que borrar manualmente desde el sistema operativo.

Se puede borrar el tablespace estándo online u offline, pero se recomienda este último para que no haya transacciones activas en el tablespace.

Para borrar un tablespace ejecutar la siguiente sentencia:

```
DROP TABLESPACE <nombre_TS>
    INCLUDING CONTENTS
    { AND DATAFILES | KEEP DATAFILES }
    CASCADE CONSTRAINT;
```

Donde:

- ✔ `<nombre_TS>` .- Nombre del TABLESPACE a borrar.
- ✔ `INCLUDE CONTENTS` .- Elimina el contenido del tablespace. Si el tablespace contiene objetos de esquema y no se indica esta cláusula, entonces la operación de borrado devuelve un error.
- ✔ `AND DATAFILES` .- Se solicita el sistema operativo que borre los datafiles asociados

al tablespace. Si no se indica esta cláusula también se borran los datafiles por defecto cuando se usa la cláusula `INCLUDE CONTENTS`.

- ✔ `KEEP DATAFILES` .- Si se emplea la cláusula `INCLUDE CONTENTS` y no queremos borrar los datafiles entonces usamos `KEEP DATAFILES`.
- ✔ `CASCADE CONSTRAINT` .- Elimina todas las restricciones de integridad referencial desde tablas que están fuera del tablespace que referencian a claves primarias de tablas dentro del tablespace borrado. Si se omite esta cláusula y existen restricciones de integridad referencial, entonces la operación de borrado devuelve un error y el tablespace no se borra.

2.3 Gestión de DATAFILES

Los DATAFILES son los ficheros físicos en los que se almacenan los objetos de esquema que forman parte de un TABLESPACE. Un DATAFILE pertenece solamente a un TABLESPACE y a una instancia de base de datos. Un TABLESPACE puede estar formado por uno o varios DATAFILES. Cuando se crea un DATAFILE, se debe indicar su nombre, su ubicación o directorio, el tamaño que va a tener y el TABLESPACE al que va a pertenecer. Además, al crearlos, ocupan ya ese espacio aunque se encuentran totalmente vacíos, es decir, Oracle reserva el espacio para poder ir llenándolo poco a poco con posterioridad. Por supuesto, si no hay sitio suficiente para crear un fichero físico del tamaño indicado, se producirá un error y no se creará dicho fichero.

Cuando se van creando objetos de esquema en un TABLESPACE, éstos físicamente se van almacenando en los DATAFILES asignados a dicho TABLESPACE, es decir, cuando creamos una tabla y vamos insertando datos en ella, estos datos realmente se reparten por los ficheros físicos o DATAFILES que forman parte del TABLESPACE. No se puede controlar en qué fichero físico se almacenan los datos de un TABLESPACE. Si un TABLESPACE está formado por 2 DATAFILES y tenemos una tabla en ese TABLESPACE, a medida que vamos insertando filas éstas se almacenarán en cualquiera de los dos DATAFILES indistintamente, es decir, unas pueden estar en un DATAFILE y otras en otro.

Como un DATAFILE está siempre asociado a uno y solo un TABLESPACE, las dos sentencias que nos permiten crear DATAFILES son:

- ✔ `CREATE TABLESPACE … ADD DATAFILE …`
- ✔ `ALTER TABLESPACE … ADD DATAFILE ...`

Ambas se han visto en apartados anteriores. Los siguientes apartados tratarán sobre las operaciones más habituales en la gestión de los DATAFILES.

2.3.1 Habilitar / Deshabilitar la autoextensión de los datafiles

Podemos crear DATAFILES cuyo tamaño se entiende automáticamente cuando lo necesitan. Establecer un DATAFILE que se extiende automáticamente ofrece las siguientes ventajas:

- ✔ Evita una intervención inmediata cuando el TABLESPACE agota su espacio.

- ✔ Asegura que las aplicaciones de BD no pararán o se suspenderán por fallos en la colocación de extensiones del TABLESPACE.

Para especificar un DATAFILE de autoextensión automática usamos las siguientes sentencias

```
CREATE TABLESPACE <nombre_TS>
    ADD DATAFILE <nombre_DF>
         SIZE <tamaño>
         AUTOEXTEND ON NEXT <tamaño> MAXSIZE <tamaño>
```

El valor de `NEXT` es el tamaño mínimo de los incrementos que se añaden al fichero cuando se extiende. El valor de `MAXSIZE` es el tamaño máximo que puede tomar el DATAFILE.

También se puede añadir el DATAFILE anterior sobre un tablespace existente con la sentencia `ALTER TABLESPACE`.

De forma análoga podemos deshabilitar la autoextensión automática de un DATAFILE con la siguiente sentencia

```
ALTER DATABASE <nombre_TS>
    DATAFILE <nombre_DF> AUTOEXTEND OFF;
```

2.3.2 Cambiar tamaño de DATAFILES

Podemos aumentar o disminuir manualmente el tamaño de un fichero de datos utilizando la sentencia `ALTER DATABASE`. Por tanto, podemos añadir más espacio a la base de datos sin añadir más ficheros de datos.

Reducir manualmente el tamaño de los ficheros de datos nos permite recuperar espacio sin utilizar en la base de datos. Esto es útil cuando hemos sobreestimado los requerimientos de espacio.

Para cambiar el tamaño de un fichero de datos ejecutar la siguiente sentencia:

```
ALTER DATABASE DATAFILE <nombre_DF> RESIZE <tamaño>
```

El valor `<tamaño>` tiene la misma semántica que hemos visto anteriormente en este capítulo.

2.3.3 Cambiar el nombre y la ubicación de los datafiles

Podemos cambiar el nombre y la ubicación de los ficheros de datos de un TABLESPACE. Estos cambios solamente quedan registrados en la base de datos, pero Oracle no cambia ni el nombre del fichero de datos ni su ubicación actual. Estas operaciones hay que hacerlas a nivel de sistema operativo.

El cambio de nombre de los ficheros de datos requiere un proceso en varios pasos:

1. Poner el tablespace en modo OFFLINE.

```
ALTER TABLESPACE <nombre_TS> OFFLINE;
```

2. Cambiar el nombre de los ficheros de datos utilizando el sistema operativo.
3. Ejecutar la sentencia `ALTER TABLESPACE` con la cláusula `RENAME DATAFILE` para cambiar los nombres de los ficheros en la base de datos. Especificar siempre el path absoluto de los ficheros tal y como aparecen en la columna `FILE_NAME` de la tabla `DBA_DATA_FILES`.

```
ALTER TABLESPACE <nombre_TS>
    RENAME DATAFILE <nombre_actual> [,…] TO <nombre_nuevo> [,…];
```

4. Hacer una copia de la base de datos
5. Poner el TABLESPACE en modo ONLINE.

```
ALTER TABLESPACE <nombre_TS> ONLINE;
```

Para cambiar la ubicación de los ficheros de datos seguimos el siguiente proceso:

1. Si no conocemos el nombre y tamaño de los ficheros de datos podemos obtener esta información haciendo una consulta al diccionario de datos.

```
SELECT FILE_NAME, BYTES FROM DBA_DATA_FILES
WHERE TABLESPACE_NAME = <nombre_TS>;
```

2. Poner el TABLESPACE a modo OFFLINE

```
ALTER TABLESPACE <nombre_TS> OFFLINE;
```

3. Copiar los ficheros de datos a sus nuevas ubicaciones y cambiarles el nombre utilizando el sistema operativo.
4. Cambiar el nombre de los ficheros de datos en la base de datos. Para ello se emplea la sentencia ALTER TABLESPACE…

```
ALTER TABLESPACE <nombre_TS>
    RENAME DATAFILE <nombre_DF_actual>,… TO <nombre_DF_nuevo>,
…;
```

5. Hacer una copia de la BD. Después de hacer cualquier cambio estructural en la base de datos, hacer inmediatamente un backup completo.
6. Poner el tablespace en modo online.

```
ALTER TABLESPACE <nombre_TS> ONLINE;
```

3 Administración de objetos de esquema

Este apartado incluye como crear y mantener objetos de esquema.

3.1 Gestión de tablas

Las tablas son la unidad básica de almacenamiento de datos en una BD Oracle. Los datos se almacenan en filas y columnas. Podemos definir una tabla con un nombre y un conjunto de columnas. Para cada columna definimos un nombre y un tipo de datos. Una fila es una colección de información de columnas correspondiente a un registro simple. Podemos también especificar reglas de integridad para cada columna denominadas restricciones de integridad las cuales deben cumplirse para confirmar la inserción o actualización de los datos en una columna.

Esta sección describe las líneas básicas a seguir cuando gestionamos tablas. Estas nos facilitarán la gestión de las tablas y mejorarán el rendimiento al crear una tabla, asi como cargar, actualizar y consultar los datos de la tabla.

3.1.1 Crear una tabla

La sentencia DDL SQL que crea un tabla la vimos en un capítulo anterior y reproducimos aquí la sintaxis:

```
CREATE TABLE <tabla> (
    <definición de columna1>,
    <definición de columna2>,
    …
    <definición de columna N>
    <constraint1>,
    <constraint2>,
    …
    <constraint N>
)
TABLESPACE <tablespace>
```

Donde:

- ✔ `<definición de columna>` consiste en definir la estructura de una columna de la tabla.
- ✔ `<constraint>` consiste en definir una restricción de la tabla.
- ✔ `TABLESPACE <tablespace>` es el tablespace donde se almacenará la tabla.

La cláusula `<definición de columna>` tiene la siguiente sintaxis

```
<nombre_columna> <tipo_datos> [DEFAULT <expr>]
[<constraint_en_linea>]
```

Donde:

- ✔ `<nombre_columna>` .- Es el nombre único de la columna en la tabla.
- ✔ `<tipo_datos>` .- El el tipo de datos Oracle de la columna.

- ✔ `DEFAULT <expr>` .- Valor por defecto de la columna cuando se añade una fila y no se indica explícitamente un valor para esta columna. `<expr>` debe ser una expresión válida en Oracle que puede incluir pseudocolumnas, operadores y operandos.
- ✔ `<constraint_en_línea>` .- Consiste en definir una restricción de integridad en la columna afectada.

La cláusula `<constraint_en_linea>` tiene la siguiente sintaxis

```
[CONSTRAINT <nombre>]
    NOT NULL |
    UNIQUE |
    PRIMARY KEY |
    <referencia> |
    CHECK (<condición>)
```

Donde:

- ✔ `CONSTRAINT <nombre>` .- Es el nombre del constraint único en el esquema. Se puede omitir.
- ✔ `NOT NULL` .- La columna no puede tomar valores nulos.
- ✔ `UNIQUE` .- La columna toma valores únicos.
- ✔ `PRIMARY KEY` .- La columna es clave primaria de la tabla
- ✔ `<referencia>` .- Consiste en la definición de la columna como clave externa.
- ✔ `CHECK <condición>` .- Restricción que deben cumplir los valores de la columna.

Si queremos definir un constraint a nivel de tabla, es decir fuera de la definición de columna tendremos que indicar la cláusula `<constraint>` con la siguiente sintaxis:

```
[CONSTRAINT <nombre>]
    UNIQUE (<columna>, …) |
    PRIMARY KEY (<columna>, …) |
    FOREIGN KEY (<columna>, …) <referencia> |
    CHECK (<condición>)
```

Donde:

- ✔ `CONSTRAINT <nombre>` .- Es el nombre del constraint único en el esquema. Se puede omitir.
- ✔ `UNIQUE (<columna>,…)` .- Lista de columnas cuya combinación de valores es única en cada fila de la tabla.
- ✔ `PRIMARY KEY (<columna>, …)` .- Las columnas son clave primaria de la tabla.
- ✔ `FOREIGN KEY (<columna>, …) <referencia>` .- Las columnas son clave externa de la tabla. La cláusula `<referencia>` es la definición de la tabla a la que referencia.

- ✔ `CHECK <condición>` .- Restricción que deben cumplir los valores de las columnas afectadas.

Para definir una `<referencia>` de clave externa se emplea la siguiente sintaxis

```
REFERENCES <tabla> (<lista columnas>) ON DELETE {CASCADE|SET
NULL}
```

Donde:

- ✔ `REFERENCES <tabla> (<lista columnas>)` .- Es la tabla y columnas de clave primaria que se referencia.
- ✔ `ON DELETE {CASCADE | SET NULL}` .- Indica si al borrar una fila en la tabla referenciada se borran o ponen a null los valores de clave externa de esta tabla.

3.1.1.1 Cláusula DEFAULT en definición de columna

Como hemos visto antes en la sintaxis de la sentencia `CREATE TABLE` podemos definir un valor por defecto para las columnas de una tabla. Cuando se realiza una inserción de una nueva fila en la tabla, si no se especifica un valor para la columna, entonces tomará el valor por defecto definido en la creación de la tabla en lugar del valor `NULL`.

Para especificar los valores por defecto de una columna debemos escribir una expresión válida de Oracle, la cual puede contener operadores y literales, incluido pseudocolumnas o columnas de la propia tabla como operandos de la expresión ya que esta característica está desde la versión 12 de Oracle. Vamos a ver unos ejemplos útiles de uso de la cláusula `DEFAULT` en la definición de valores por defecto para las columnas.

En el siguiente ejemplo vemos como rellenar por defecto valores de tipo `DATE` y `TIMESTAMP` con fecha y hora por defecto.

```
CREATE TABLE tabla2 (
    id              NUMBER(5,0) PRIMARY KEY,
    ahora           TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    fecha           DATE DEFAULT SYSDATE NOT NULL
) TABLESPACE datos;
```

Como podemos ver en el ejemplo anterior utilizamos las pseudocolumnas `CURRENT_TIMESTAMP` y `SYSDATE` para obtener los valores de marca de tiempo y fecha actuales.

3.1.1.2 Columnas autoincrementadas

Una columna autoincrementada es una columna de tipo numérico cuyos valores se incrementan uno a uno para generar un valor único. Suelen ser empleadas para columnas de clave primaria.

Para generar valores numéricos autoincrementados podemos utilizar un trigger, que se verá en un capítulo posterior a éste. Sin embargo, desde la versión 12 de Oracle se han incluido dos características para generar automáticamente este tipo de valores. Consiste en:

1. Definir la columna como identidad.
2. Utilizar una secuencia como valor por defecto

El primer caso consiste en definir la cláusula `IDENTITY` en la definición de columna. La sintaxis de esta cláusula es

```
GENERATED
[ ALWAYS | BY DEFAULT [ ON NULL ] ]
AS IDENTITY [(opciones)]
```

Donde:

- ✔ `GENERATED ALWAYS AS IDENTITY` .- Indica que siempre tiene que generar un valor autoincrementado, el siguiente en la secuencia, por lo que no hay que incluir un valor de columna en la sentencia `INSERT`, sea nulo o no.
- ✔ `GENERATED BY DEFAULT AS IDENTITY` .- Indica que se va generar un nuevo valor autoincrementado, el siguiente en la secuencia, si la sentencia `INSERT` no incluye un valor específico para la columna. Si lo incluye, se usa éste en lugar de generar uno autoincrementado. Por supuesto, si el valor especificado para la columna es `NULL` entonces se genera un error.
- ✔ `GENERATED BY DEFAULT ON NULL AS IDENTITY` .- Indica que se va generar un nuevo valor autoincrementado, el siguiente en la secuencia, si la sentencia `INSERT` no incluye un valor específico para la columna. Si lo incluye, se usa éste en lugar de generar uno autoincrementado. A diferencia del anterior que carece de la opción `ON NULL` si el valor especificado para la columna es `NULL` entonces se genera un nuevo valor autoincrementado en la secuencia para la columna y no se genera un error.

Para ilustrar las opciones anteriores vamos a ver un ejemplo. Disponemos de la siguiente tabla

```
CREATE TABLE prueba_identidad (
  id          NUMBER GENERATED ALWAYS AS IDENTITY,
  descripcion VARCHAR2(30)
);
```

La columna `id` es numérica autoincrementada. En este caso indicamos que tiene que generar siempre el valor de la columna cuando se inserta una nueva fila. Ahora insertamos filas y vemos el resultado

```
-- Nueva fila y solo se indica
-- un valor para la columna descripcion
SQL> INSERT INTO prueba_identidad (descripcion) VALUES ('Solo la
descripción');

1 row created.

-- Nueva fila y se indica NULL para id
```

```
-- y un valor para descripción
SQL> INSERT INTO prueba_identidad (id, description) VALUES
(NULL, 'ID=NULL y DESCRIPCIÓN');

INSERT INTO prueba_identidad (id, descripcion) VALUES (NULL,
'ID=NULL y DESCRIPCIÓN')
                                                    *
ERROR at line 1:
ORA-32795: cannot insert into a generated always identity column

-- Nueva fila y se indica un valor para ID
-- y un valor para descripción
SQL> INSERT INTO prueba_identidad (id, descripcion) VALUES (999,
'ID=999 y DESCRIPCIÓN');

INSERT INTO prueba_identidad (id, descripcion) VALUES (999,
'ID=999 y DESCRIPCIÓN');
                                                    *
ERROR at line 1:
ORA-32795: cannot insert into a generated always identity column
```

Como se puede observar el valor de la columna autoincrementado siempre es generado por Oracle y no importa si indicamos un valor para esta columna, nulo o no, se genera un error. Ahora volvemos a definir la tabla pero en esta ocasión definimos la columna autoincrementada de otra forma. En este caso indicamos que tiene que generar el valor de la columna cuando no se indica uno en la inserción de una nueva fila.

```
CREATE TABLE prueba_identidad (
  id          NUMBER GENERATED BY DEFAULT AS IDENTITY,
  descripcion VARCHAR2(30)
);
```

Ahora insertamos filas y vemos el resultado.

```
SQL> INSERT INTO prueba_identidad (descripcion) VALUES ('Solo
descripción');

1 row created.

SQL> INSERT INTO prueba_identidad (id, descripcion) VALUES (999,
'ID=999 y descripción');

1 row created.

SQL> INSERT INTO prueba_identidad (id, descripcion) VALUES
(NULL, 'ID=NULL y descripción');
INSERT INTO prueba_identidad (id, descripcion) VALUES (NULL,
'ID=NULL y descripción')
                                                    *
ERROR at line 1:
ORA-01400: cannot insert NULL into
("TEST"."PRUEBA_IDENTIDAD"."ID")
```

Como se puede observar en los ejemplos anteriores, el valor de la columna `id` se autogenera cuando no se especifica un valor para la columna. Si se especifica uno se inserta éste. Solo en el caso de que se quiera introducir un valor `NULL` se genera un error.

Por último, vamos a definir la columna autoincrementada con la opción `ON NULL`.

```
CREATE TABLE prueba_identidad (
  id          NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY,
  descripcion VARCHAR2(30)
);
```

Ahora insertamos nuevas filas y observamos el resultado

```
SQL> INSERT INTO prueba_identidad (descripcion) VALUES ('Solo
descripción');

1 row created.

SQL> INSERT INTO prueba_identidad (id, descripcion) VALUES (999,
'ID=999 y descripción');

1 row created.

SQL> INSERT INTO prueba_identidad (id, descripcion) VALUES
(NULL, 'ID=NULL y descripción');

1 row created.
```

En esta ocasión todas las filas se han podido insertar. Ahora, cuando no se indica un valor o éste es NULL para la columna autogenerada, se genera el siguiente número en la secuencia. Solo en el caso de que se establezca un valor esplícitamente en la columna se utiliza éste en lugar del autogenerado. Si hacemos un listado de las filas insertadas veremos lo siguiente

```
SQL> SELECT * FROM prueba_identidad;

        ID DESCRIPCION
---------- ------------------------------
         1 Solo descripción
       999 ID=999 y descripción
         2 ID=NULL y descripción
```

Para poder usar este tipo de columna el usuario que crea la tabla necesita el privilegio `CREATE SEQUENCE` o de lo contrario obtendrá un error por privilegios insuficientes.

3.1.1.3 Utilizar secuencias en valores por defecto de columna

A partir de la versión 12c podemos utilizar pseudocolumnas como valores por defecto de columnas en una tabla. Por ejemplo

```
CREATE SEQUENCE spedido START WITH 1 NOMAXVALUE;
```

```
CREATE TABLE pedido (
    npedido   NUMBER(6,0) DEFAULT spedido.NEXTVAL PRIMARY KEY,
    fecha     DATE DEFAULT SYSDATE,
    cliente   VARCHAR2(15) NOT NULL     REFERENCES cliente(nif)
) TABLESPACE to_datos;
```

Cuando insertamos nuevas filas veremos lo siguiente

```
INSERT INTO pedido (cliente) VALUES ('30000001A');

INSERT INTO pedido (npedido, cliente)
VALUES (999, '30000001A');

SELECT npedido, fecha FROM pedido;

   NPEDIDO FECHA
---------- --------
         1 22/12/19
       999 22/12/19

2 rows selected.
```

Como se puede observar, si no indicamos valor de columna autoincrementada se genera el de la secuencia.

De hecho, podemos jugar con las pseudocolumnas `NEXTVAL` y `CURRVAL` para insertar filas en tablas que mantienen una relación maestro-detalle. Como en el siguiente ejemplo

```
CREATE SEQUENCE spedido START WITH 1 NOMAXVALUE;

CREATE TABLE pedido (
    npedido   NUMBER(6,0) DEFAULT spedido.NEXTVAL PRIMARY KEY,
    fecha     DATE DEFAULT SYSDATE,
    nif       VARCHAR2(15) NOT NULL REFERENCES cliente(nif)
) TABLESPACE to_datos;

CREATE TABLE lpedido (
    npedido        NUMBER(6,0) DEFAULT spedido.CURRVAL,
    nlinea         NUMBER(6,0),
    referencia     VARCHAR2(15) NOT NULL
                   REFERENCES articulo(referencia),
    unidades       NUMBER(4) DEFAULT 1 NOT NULL,
    PRIMARY KEY (npedido,nlinea)
) TABLESPACE to_datos;
```

En esta ocasión al insertar filas de ambas tablas se genera un nuevo número de pedido para la columna `npedido` de la tabla `pedido` y este valor se emplea para introducir las líneas del pedido. Por supuesto, hay que insertar el pedido y sus líneas consecutivamente, o de lo contrario el número de pedido de la secuencia puede ser diferente.

3.1.2 Cambiar la definición de una tabla

Para cambiar la definición de una tabla disponemos de la sentencia `ALTER TABLE`. Las razones para cambiar la definición de una tabla pueden ser:

- ✔ Modificar las características físicas.
- ✔ Mover una tabla a un nuevo segmento o tablespace.
- ✔ Añadir, borrar o modificar definiciones de columna (tipo de datos, longitud, valor por defecto, restricciones de integridad NOT NULL, etc.
- ✔ Añadir, borrar o modificar restricciones de integridad asociados a la tabla.
- ✔ Habiliar o deshabilitar restricciones de integridad o triggers asociados a la tabla.
- ✔ Cambiar el nombre de la tabla.
- ✔ Poner una tabla en modo solo lectura, o de nuevo a lectura y escritura.

La sintaxis de la sentencia `ALTER TABLE` es:

```
ALTER TABLE <tabla>
<modificación de columnas>
<modificación de constraints>
[RENAME TO <nuevo_nombre>]
[READ ONLY | READ WRITE]
[ {ENABLE | DISABLE} {TABLE LOCK | ALL TRIGGERS} ]
```

Donde:

- ✔ `ALTER TABLE <tabla>` .- Indica la tabla a modificar.
- ✔ La clásula `<modificación de columna>` es para modificar la definición de columnas. Se verá en detalle en el siguiente apartado.
- ✔ La cláusula `<modificación de constraint>` es para modificar la definición de un constraint. Se verá en detalle en un apartado posterior.
- ✔ `RENAME TO <nuevo_nombre>` .- Cambia el nombre de la tabla al nuevo nombre que se indica.
- ✔ `READ ONLY | READ WRITE` .- Establece la tabla como solo lectura, o la devuelve a su estado de lectura y escritura. Una tabla en modo solo lectura no permite sentencias DML que modifiquen sus datos, aunque si permite sentencias DDL para modificar su definición.
- ✔ `{ENABLE | DISABLE} {TABLE LOCK | ALL TRIGGERS}` .- Habilita/deshabilita el bloqueo de tabla o los triggers de la tabla. El bloqueo de tabla se hace para permitir realizar operaciones DDL sobre la tabla sin que haya ninguna transacción activa. Sin embargo, antes de poner la tabla en bloqueo Oracle tiene que esperar a que se hayan confirmado todas las transacciones activas lo que podría llevar mucho tiempo.

3.1.2.1 Modificar las columnas de una tabla

Para modificar las columnas de una tabla (añadir nuevas columnas, cambiar la definición de una columna existente en la tabla, borrar una columna o cambiarle el nombre) disponemos de la cláusula `<modificación de columnas>` de la sentencia `ALTER TABLE`. Su sintaxis es la siguiente:

```
[ADD ( <definición de columna>, … )]
[MODIFY (<definición de columna>, … )]
[DROP { COLUMN <columna> | (<columna>, … ) }
        [CASCADE CONSTRAINT] ]
[RENAME COLUMN <columna> TO <nuevo_nombre>]
```

Donde:

- ✔ `ADD ( <definición de columna>, … )` .- Se añaden nuevas columnas cada una con su definición completa tal y como se vió en la sentencia `CREATE TABLE`.
- ✔ `MODIFY (<definición de columna>, … )` .- Se cambia la definición de una columna que debe existir. Cualquier cambio en la definición de columna que omita una parte (tipo de datos, valor por defecto o constraint) permanecerá sin cambios.
- ✔ `DROP { COLUMN <columna> | (<columna>, … ) } [CASCADE CONSTRAINTS]` .- Elimina las columnas que se indiquen. Si se indica `CASCADE CONSTRAINTS` entonces se borran todas las restricciones de clave externa definidas en otras tablas y que hagan referencia a claves primarias y únicas de las columnas que se borran.
- ✔ `RENAME COLUMN <columna> TO <nuevo_nombre>` .- Se cambia el nombre de la columna.

Generalmente se emplea una sentencia `ALTER TABLE` con una de estas cláusulas, no se combinan para evitar complejidad en la sentencia..

3.1.2.2 Modificar los constraints de una tabla

La sentencia `ALTER TABLE` contiene la clásula `<modificación de constraints>` para cambiar los constraint de la tabla. Podemos añadir nuevos constraints, modificar los existentes, habilitarlos/deshabilitarlos, cambiarles el nombre o borrarlos. La sintaxis de esta cláusula es:

```
[ADD <constraint>]
[MODIFY  { CONSTRAINT <constraint> |
           PRIMARY KEY |
           UNIQUE (<columna>, … ) }
         {ENABLE | DISABLE} ]
[RENAME CONSTRAINT <constraint> TO <nuevo_nombre]
[DROP    { PRIMARY KEY [CASCADE] |
           UNIQUE (<columna>, …) |
           CONSTRAINT <constraint> [CASCADE] ]
```

Donde:

- ✔ `ADD <constraint>` .- Se añade un nuevo constraint a la tabla. La sintaxis de <constraint> es similar a la que hemos visto en la sentencia `CREATE TABLE`.
- ✔ `MODIFY { CONSTRAINT <constraint> | PRIMARY KEY | UNIQUE (<columna>, … ) } {ENABLE | DISABLE}` .- Se cambia el estado del constraint indicado (por nombre, clave primaria o clave única) a `DISABLE` (deshabilitado), lo que significa que no tiene efecto, o `ENABLE` (habilitado) para que esté en vigor.
- ✔ `RENAME CONSTRAINT <constraint> TO <nuevo_nombre>` .- Se cambia el nombre del constraint.
- ✔ `DROP { PRIMARY KEY [CASCADE] | UNIQUE (<columna>, …) | CONSTRAINT <constraint> [CASCADE]` .- Elimina la clave primaria, clave única o constraint por nombre que se indique. Si se indica `CASCADE` entonces se borran todas las restricciones de integridad que dependen del constraint eliminado.

3.1.3 Borrar una tabla

Para borrar una tabla disponemos de la sentencia `DROP TABLE`. Su sintaxis es:

```
DROP TABLE <tabla> [CASCADE CONSTRAINT] [PURGE]
```

Donde:

- ✔ `DROP TABLE <tabla>` .- Se indica la tabla a borrar.
- ✔ `CASCADE CONSTRAINT` .- Se borran todas las restricciones de integridad referencial (claves externas) que se refieren a clave primaria o claves únicas en la tabla borrada. Si se omite esta cláusula y existen tales restricciones de integridad la ejecución de la sentencia devuelve un error y la tabla no se borra.
- ✔ `PURGE` .- Se indica que al borrar la tabla se libera el espacio asociado a ella y, por tanto, la tabla no se lleva a la papelera de reciclaje. Esto dejaría sin posibilidad de recuperar la tabla en caso de borrado accidental.

3.1.4 La papelera de reciclaje

Cuando borra una tabla Oracle no libera su espacio de almacenamiento inmediatemente, sino que la renombra y coloca en la papelera de reciclaje junto con sus objetos dependientes. Desde la papelera de reciclaje podemos recuperar tablas borradas.

Podemos ver los objetos borrados en la papelera de reciclaje haciendo una consulta a la vista `USER_RECYCLEBIN` para usuarios o `DBA_RECYCLEBIN` para el administrador de la BD.

```
SELECT * FROM USER_RECYCLEBIN;
```

Al ejecutar la consulta anterior veremos que Oracle utiliza un nombre para cada objeto borrado que comienza por BIN$ y a continuación un código alfanumérico. Hay una columna que nos muestra el nombre original del objeto y el tipo de objeto.

Para recuperar una tabla borrada podemos utilizar la sentencia `FLASHBACK TABLE`. Su sintaxis es:

```
FLASHBACK TABLE <tabla>, …
TO BEFORE DROP
[RENAME TO <nuevo_nombre>, … ];
```

Donde:

- ✔ `FLASHBACK TABLE <tabla>, …` .- Lista de tablas que se van a recuperar. Puede usarse el nombre original de la tabla dado por el usuario o el nombre generado por el sistema al incluirla en la papelera de reciclaje cuando se borró.
- ✔ `TO BEFORE DROP` .- Se recuperan las tablas y sus objetos dependientes al estado en el que se encontraban antes de ser borradas.
- ✔ `[RENAME TO <nuevo_nombre>, … ]` .- Se cambia el nombre a las tablas que se recuperan para evitar conflicto con tablas actuales con el mismo nombre.

A la hora de recuperar tablas de la papelera de reciclaje debemos tener en cuenta lo siguiente:

- ✔ La sentencia `FLASHBACK TABLE` recupera las tablas, sus índices, sus triggers y sus constraints excepto los constraints de integridad referencial (claves externas). Para estos objetos emplea el nombre generado por el sistema, en lugar de sus nombres originales. Si queremos que tengan otro nombre más familiar tienes que renombrarlos antes de recuperarlos.
- ✔ No se recuperan tablas que han sido purgadas con la sentencia `PURGE` o que el sistema ha borrado definitivamente al reclamar espacio de almacenamiento. En esta situación, lo primero que se borra son los índices por lo que es posible que al recuperar una tabla no se recuperen todos sus índices.
- ✔ La recuperación se realiza en una sola transacción. Si ocurre un fallo en la recuperación de una tabla de la lista, entonces no se recupera ninguna.
- ✔ Si en la papelera hay varias tablas con el mismo nombre entonces se recupera la borrada más reciente. Si queremos recuperar una en concreto que no es la más reciente podemos indicar el nombre de la tabla generado por el sistema o ejecutar sucesivas sentencias `FLASHBACK TABLE` con el mismo nombre del usuario hasta recuperar la que queramos.
- ✔ Oracle intenta preservar el nombre de tabla original. Si el nombre de la tabla que se recupera es el mismo que el de otra tabla creada en el mismo esquema desde que se borró la anterior ocurrirá un error, salvo en el caso de emplear la cláusula `RENAME TO` que permite cambiar el nombre a la tabla recuperada.

Por otro lado, también es posible eliminar definitivamente una tabla borrándola de la papelera de reciclaje, para lo que podemos usar la sentencia `PURGE`. En este caso, la tabla no podría ser recuperada posteriormente con `FLASHBACK TABLE` sino con una

restauración de una copia de seguridad. La sintaxis de la setencia `PURGE` es:

```
PURGE    { TABLE <tabla> |
           INDEX <índice> |
           REBYCLEBIN |
           TABLESPACE <nombre_TS> [USER <usuario>] }
```

Donde:

- ✔ `TABLE <tabla>` .- Tabla a eliminar definitivamente. Puede ser su nombre original o el generado por el sistema. Si hay más de uno con el mismo nombre se elimina el que lleva más tiempo borrado.
- ✔ `INDEX <índice>` .- Índice a eliminar definitivamente. Puede ser su nombre original o el generado por el sistema.
- ✔ `RECYCLEBIN` .- Elimina toda la papelera de reciclaje del usuario. Se borran definitivamente todos los objetos del esquema del usuario y se libera el espacio de almacenamiento que ocupaban.
- ✔ `TABLESPACE <nombre_TS> [USER <usuario>]` .- Elimina todos los objetos en el TABLESPACE especificado. Si se emplea la cláusula `USER <usuario>` entonces eliminaría todos los objetos del esquema del usuario especificado dentro del TABLESPACE.

3.2 Gestión de Índices

Un índice es una estructura de datos asociado a una tabla que permite que las consultas SQL se ejecuten con mayor rapidez. El índice es independiente de la tabla a la que está asociada, por tanto, necesita espacio de almacenamiento. La base de datos mantiene automáticamente el índice de una tabla cuando se insertan, actualizan o borran filas. Sin un índice, el acceso a una tabla sería mucho más lento.

Básicamente, un índice ordena una tabla en base a una columna o un conjunto de columnas denominadas claves de indexación. Los índices pueden admitir valores duplicados en las claves de indexación, o solo permitir valores únicos.

La cuestión que todo DBA tiene que hacerse es: ¿cuándo crear un índice? Por defecto, Oracle crea un índice para la clave primaria (PRIMARY KEY) y cada clave única (UNIQUE) definidas en la tabla. Por otro lado, no se crean índices sobre las claves externas. Sin embargo, es posible crear más índices sobre otras columnas. Oracle recomienda crear un índice en las siguientes situaciones:

- ✔ Si se consulta con frecuencia datos de una tabla en base el valor de una columna.
- ✔ Cuando se ejecutan muchas consultas multitabla (joins) para las columnas utilizadas en el join. En este sentido puede ser recomendable crear índices para las columnas que actúan como clave externa.
- ✔ Tablas pequeñas no necesitan índices.

- ✔ Indexar por una columna que tiene muchos valores distintos.
- ✔ Indexar por una columna que tiene un amplio rango de valores.
- ✔ Indexar por una columna que tiene muchos valores nulos y se consulta con frecuencia las filas con algún valor en dicha columna. Es preferible indicar en la clausula `WHERE columna > -9.99 * power(10,125)` que utilizar `WHERE columna IS NOT NULL`. En el primer caso utiliza el índice y en el segundo no.

Una tabla puede tener tantos índices como se quiera. No hay límite, pero cuantos más índices tenga más tiempo se tarda en ejecutar las sentencias que actualizan datos, ya que el índice tiene que actualizarse también. Otra cuestión es el TABLESPACE donde se almacena el índice. Podemos elegir en que TABLESPACE se creará el índice, que puede ser el mismo que el de la tabla asociada, pero si está en uno diferente el rendimiento se mejora.

3.2.1 Crear un índice

Para crear un índice usamos la sentencia `CREATE INDEX` que tiene la siguiente sintaxis:

```
CREATE [UNIQUE] INDEX <indice>
ON <tabla> (<columna>, …)
TABLESPACE <nombre_TS>
```

Donde:

- ✔ `UNIQUE` .- Especifica que los valores de las columnas sobre las que se construyen el índice son únicos.
- ✔ `INDEX <índice>` .- Nombre del índice único en el esquema.
- ✔ `ON <tabla>` .- Tabla asociada al índice que estamos creando.
- ✔ `(<columna>, …)` .- Lista de columnas separadas por coma que serán la clave de indexación.
- ✔ `TABLESPACE <nombre_TS>` .- Indica el TABLESPACE donde se almacenará el índice creado.

3.2.2 Borrar un índice

Si un índice no va a usarse es mejor eliminarlo para evitar que ralentice las operaciones de actualización en los datos de la tabla. Las razones para borrar un índice son:

- ✔ El índice ya no es necesario.
- ✔ El índice no proporciona una mejora en el rendimiento de la ejecución de las consultas a la tabla.
- ✔ Las aplicaciones no utilizan el índice.
- ✔ El índice está muy fragmentado y es mejor borrarlo antes de reconstruirlo.

No pueden borrarse los índices asociados a claves primarias o únicas de una tabla.

El borrado de un índice se realiza con la sentencia `DROP INDEX`. Su sintaxis es:

```
DROP INDEX <indice>
```

3.3 Gestión de vistas

Una vista es una representación lógica de una o una combinación de tablas. En esencia, una vista es una consulta almacenada. Los datos de una vista derivan de las tablas en las que está basada. Estas tablas, llamadas tablas base podrían ser tablas o incluso otras vistas. Todas las operaciones realizadas sobre una vista afectan a las tablas de la vista. Podemos usar una vista para casi lo mismo que una tabla, como consultar, actualizar, borrar o insertar filas.

Para crear una vista se emplea la sentencia `CREATE VIEW` con la siguiente sintaxis:

```
CREATE OR REPLACE VIEW <vista>
AS <sentencia SELECT>
[WITH { READ ONLY | CHECK OPTION } [CONSTRAINT <constrain>]]
```

Donde:

- ✔ `CREATE OR REPLACE <vista>` .- Crear la vista o al reemplaza si ya existiera.
- ✔ `AS <sentencia SELECT>` .- Sentencia SELECT sobre la que se basa la consulta.
- ✔ `WITH READ ONLY` .- La consulta es de solo lectura, por lo que solamente se pueden ejecutar sentencias `SELECT` con ella.
- ✔ `WITH CHECK OPTION` .- La consulta solo admite inserciones o actualizaciones de nuevas filas siempre que estas modificaciones resulten una fila o filas que estarían dentro de la vista.
- ✔ `[CONSTRAINT <constraint>]` .- Nombre que se asigna al constraint establecido con las cláusulas `WITH READ ONLY` o `WITH CHECK OPTION`.

La sentencia `SELECT` en la que se basa la vista no puede contener pseudocolumnas. Si se especifica * como lista de columnas y después de la creación de la vista se añaden nuevas columnas, estas no se incluirán en la vista.

Una vista es actualizable si cumple los siguientes requisitos:

- ✔ Cada columna de la vista se corresponde con una columna de una simple tabla.
- ✔ No pueden ser consultas agrupadas o que contengan subconsultas.

Podemos modificar una vista con la misma sentencia anterior añadiendo la cláusula `OR REPLACE`. En este caso nos permitiría volver a definir la sentencia `SELECT` en la que se basa la vista. Sin embargo, si lo que pretendemos es modificar algún constraint de la vista podemos usar la sentencia `ALTER VIEW`.

Para borrar una vista usamos la sentencia `DROP VIEW` como sigue:

```
DROP VIEW <vista>
```

El borrado solo afecta a la vista, no a las tablas subyacentes.

3.4 Gestión de secuencias

Una secuencia es un objeto de la BD que genera números enteros únicos que puede usarse para generar valores de clave primaria automáticamente. Para crear una secuencia empleamos la sentencia `CREATE SEQUENCE` como sigue:

```
CREATE SEQUENCE <secuencia>
[INCREMENT BY <entero>]
[START WITH <entero>]
[MAXVALUE <entero> | NOMAXVALUE]
[MINVALUE <entero> | NOMINVALUE]
[CYCLE | NOCYCLE]
[ORDER | NOORDER]
```

Donde:

- ✔ `CREATE SEQUENCE <secuencia>` .- Nombre de la secuencia
- ✔ `INCREMENT BY <entero>` .- Especifica el intervalo entre números de la secuencia. Puede ser positivo o negativo, pero nunca 0. Por defecto es 1.
- ✔ `START WITH <entero>` .- Especifica el primer número de la secuencia a generar. Se emplea para indicar un valor mayor que el mínimo en una secuencia ascendente o menor que el máximo en una secuencia descendente.
- ✔ `MAXVALUE <entero> | NOMAXVALUE` .- Valor máximo de la secuencia a generar o sin valor máximo (por defecto).
- ✔ `MINVALUE <entero> | NOMINVALUE` .- Valor mínimo de la secuencia a generar o sin valor mínimo (por defecto).
- ✔ `CYCLE | NOCYCLE` .- Indica si al alcanzar el valor máximo en secuencias ascendentes, o mínimo en secuencias descendentes, continua generando valores comenzando por su valor mínimo en el primer caso y máximo en el segundo. `NOCYCLE` es por defecto.
- ✔ `ORDER | NOORDER` .- Genera los números en orden o sin orden (por defecto).

Podemos modificar una secuencia con la sentencia `ALTER SEQUENCE` utilizando los mismos parámetros de la sentencia `CREATE SEQUENCE` para modificarlos.

Una vez se genera una secuencia podemos utilizar las pseudocolumnas `CURRVAL` y `NEXTVAL` para acceder a los valores actual y siguiente de la secuencia. Cada vez que se emplea la pseudocolumna `NETXVAL` se genera un nuevo número. Mientras que con `CURRVAL` obtenemos el último número generado en la secuencia. Por ejemplo para insertar

filas en dos tablas con una relación maestro/detalle podríamos ejecutar estas dos sentencias `INSERT`.

```
CREATE SEQUENCE SPEDIDO
INCREMENT BY 1
ORDER;

-- Introducimos una cabecera de pedido
-- con un nuevo número de pedido obtenido de la secuencia
INSERT INTO pedido VALUES(SPEDIDO.NEXTVAL, '30000001A',
    TO_DATE ('1/10/2018','DD/MM/YYYY') );

-- Introducimos una línea de pedido con el mismo
-- número de secuencia que el pedido al que pertenece.
INSERT INTO linea_pedido VALUES (SPEDIDO.CURRVAL, 1, 'mag0001',
4, 2);
```

En el ejemplo anterior vemos que para generar un nuevo número de pedido usamos `SPEDIDO.NEXTVAL`, el cual utilizamos también para introducir una línea de pedido. En este caso habría que utilizar `SPEDIDO.CURRVAL` el cual no genera un nuevo número en la secuencia sino que devuelve el último número generado.

Finalmente para eliminar una secuencia usamos la sentencia `DROP SEQUENCE` como sigue:

```
DROP SEQUENCE <secuencia>
```

3.5 Gestión de sinónimos

Un sinónimo es un alias que se crea para un objeto del esquema. Un sinónimo se emplea para ocultar al usuario el verdadero nombre del objeto y también para reducir la complejidad de las sentencias SQL. También se pueden modificar los nombres originales de los objetos y no es necesario modificar las aplicaciones si están utilizan el sinónimo.

Los sinónimos pueden ser públicos (visibles para todos los usuarios de la BD) o privados (visibles solo para el usuario del esquema).

Para crear un sinónimo usamos la sentencia `CREATE SYNONYM` como sigue:

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM <sinónimo> FOR <objeto>
```

Donde:

- ✔ `CREATE [OR REPLACE] [PUBLIC] SYNONYM <sinónimo>` .- Nombre del sinónimo a crear. Si especificamos `PUBLIC` es un sinónimo público.
- ✔ `FOR <objeto>` .- Especifica el objeto para el que se crea el sinónimo. Si éste es público tiene que estar cualificado con el nombre del esquema al cual pertenece. Los obetos que pueden tener sinónimos son: tablas, vistas, secuencias, procedimientos o funciones almacenadas.

Por ejemplo, vamos a crear un sinónimo para una tabla de clientes que tiene un nombre muy complicado, como `DB_19_AB_70`.

```
CREATE SYNONYM cliente FOR DB_19_AB_70;
```

A partir de ahora se puede emplear `cliente` como sinónimo de la tabla `DB_19_AB_70` en cualquier sentencia SQL.

Para borrar un sinónimo usamos la sentencia `DROP SYNONYM` como sigue:

```
DROP [PUBLIC] SYNONYM <sinónimo>;
```

Se borra el sinónimo, no el objeto subyacente.

3.6 Gestión de clusters

Un cluster es un método opcional para almacenar datos en una tabla. Un cluster se compone de un grupo de tablas que comparten los mismos bloques de datos. Las tablas se agrupan porque tienen columnas comunes que se usan juntas con frecuencia.

Por ejemplo, la tabla `EMP` y `DEPT` tienen una columna común que es `deptno`. Cuando se crea un cluster de estas dos tablas Oracle almacena físicamente todas las filas de ambas tablas en el mismo bloque de datos.

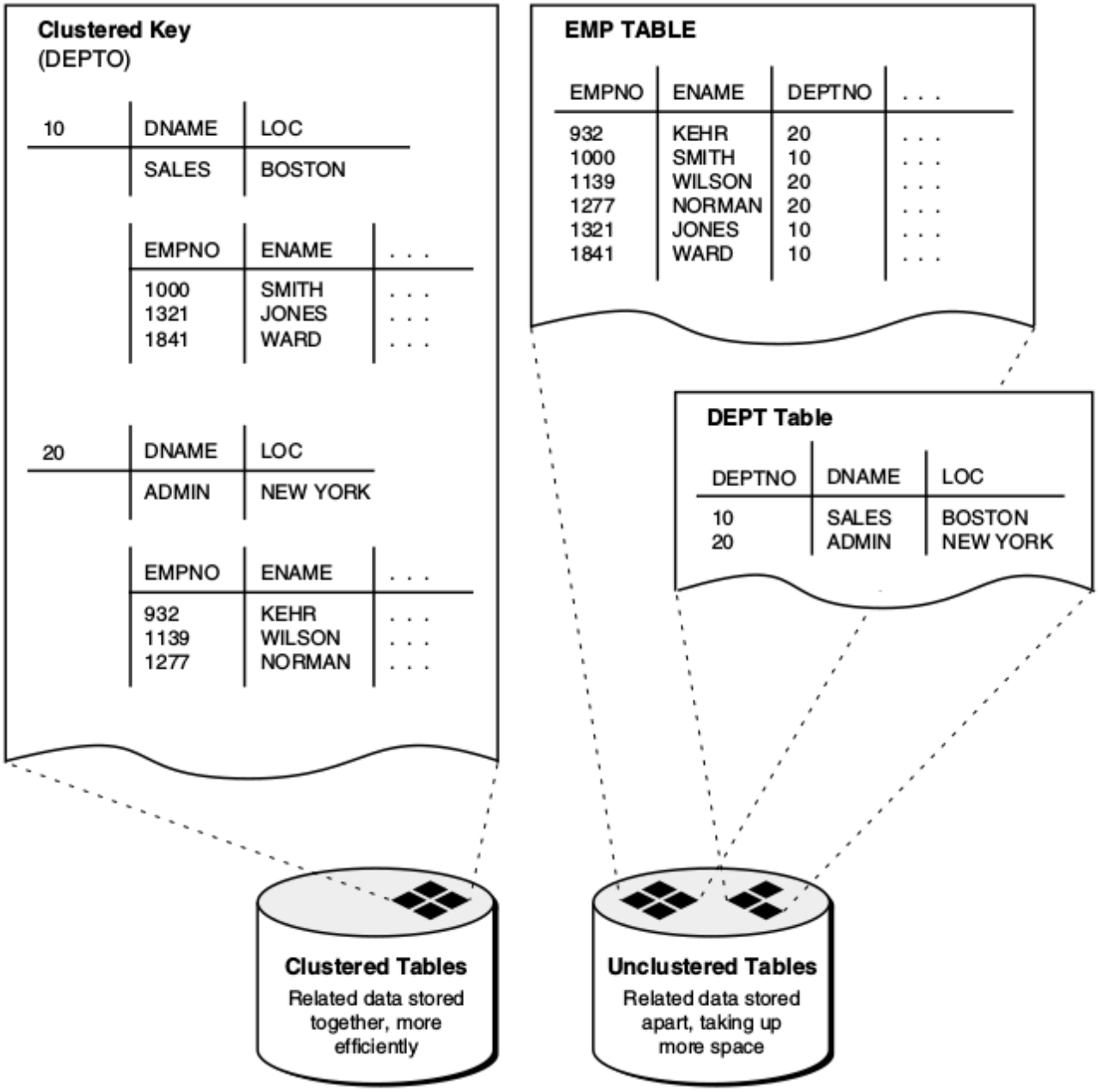

Figura 2.- Cluster de dos tablas

Los beneficiones de un cluster son:

- ✔ Al hacer joins de tablas relacionadas en un cluster se reduce las operaciones de E/S y los tiempos de acceso.
- ✔ La clave del cluster es la columna, o grupo de columnas, que las tablas tienen en común. La clave de cluster se define al crear el cluster. Cada clave de cluster se almacena una vez en el cluster y en el índice del cluster independientemente del número de filas de las tablas diferentes contengan el valor. Por tanto, se requiere menos espacio para almacenar tablas en un cluster que si estuvieran en tablas separadas.

Después de crear un cluster se pueden crear las tablas del cluster. Sin embargo, antes de insertar filas en las tablas del cluster hay que crear un índice del cluster. El uso de un índice de cluster no afecta la creación de índices adicionales en ls tablas del cluster. Se pueden crear y borrar como siempre.

No es conveniente utilizar un cluster para tablas que son accedidas con frecuencia de forma individual.

Las tablas apropiadas para crear un cluster son:

- ✔ Aquellas que se consultan con mucha frecuencia y tienen pocas operaciones de actualización de datos (inserciones y borrados).
- ✔ Los registros de las tablas se consultan con frecuencia a través de joins.

3.6.1 Crear un cluster

Para clear un cluster utilizamos la sentencia CREATE CLUSTER que tiene la siguiente sintaxis.

```
CREATE CLUSTER <cluster> (<columna>, … )
TABLESPACE <nombre_TS>
```

Donde:

- ✔ `CREATE CLUSTER <cluster>` .- Especifica el nombre del cluster que se está creando.
- ✔ `(<columna>, …)` .- Conjunto de columnas que forman la clave del cluster
- ✔ `TABLESPACE <nombre_TS>` .- Nombre del TABLESPACE donde se almacena el cluster.

Una vez se ha creado el cluster podemos añadir tablas al mismo. Para ello, en la sentencia `CREATE TABLE` tenemos que añadir la claúsula `CLUSTER`. Su sintaxis sería así

```
 CREATE TABLE <tabla> (
    <definición de columna1>,
    <definición de columna2>,
    …
    <definición de columna N>
    <constraint1>,
    <constraint2>,
    …
    <constraint N>
)
CLUSTER <cluster>(<columna>)
```

Podemos apreciar en la sintaxis de `CREATE TABLE` que en la claúsula `CLUSTER` tenemos que indicar el cluster al que pertenece la tabla y la columna común con las otras tablas que se añadan al cluster. No se debe indicar la cláusula `TABLESPACE` ya que esta no está permitida en una tabla de cluster y ya se índico en la definición del cluster.

Una vez se han creado las tablas que componen el cluster hay que crear el índice del cluster. Para ello utilizamos la sentencia `CREATE INDEX` con la cláusula `ON CLUSTER` como se muestra a continuación.

```
CREATE [UNIQUE] INDEX <índice>
ON CLUSTER <cluster>
TABLESPACE <nombre_TS>
```

En el siguiente ejemplo se crea un cluster con dos tablas: emp y dept, que contienen los empleados y los departamentos de una organización. Vemos como se crea inicialmente el cluster, luego se añaden las tablas y finalmente el índice del cluster.

```
CREATE CLUSTER emp_dpto (deptno NUMBER(3))
TABLESPACE users;

CREATE TABLE dept (
    deptno NUMBER(3) PRIMARY KEY,
    descripcion VARCHAR(40) NOT  NULL
)
CLUSTER emp_dpto(deptno);

CREATE TABLE emp (
    empno NUMBER(5) PRIMARY KEY,
    ename VARCHAR2(15) NOT NULL,
    deptno NUMBER(3) REFERENCES dept
)
CLUSTER emp_dpto (deptno);

CREATE INDEX emp_dept_index
ON CLUSTER emp_dpto
TABLESPACE gc_datos;
```

3.6.2 Borrar un cluster

Se puede borrar un cluster si las tablas del cluster no se necesitan. Cuando se borra un cluster, desaparecen las tablas del cluster y el correspondiente índice. El espacio ocupado se libera y está disponible en el TABLESPACE.

Para borrar un cluster se emplea la sentencia `DROP CLUSTER` que tiene la siguiente sintaxis

```
DROP CLUSTER <cluster>
[INCLUDING TABLES]
[CASCADE CONSTRAINT];
```

Donde:

- ✔ `DROP CLUSTER <cluster>` .- Especifica el nombre del cluster que se pretende borrar.
- ✔ `[INCLUDING TABLES]` .- Si el cluster contiene tablas las elimina también. Si el cluster contiene tablas debemos incluir la cláusula `INCLUDING TABLES` para que

también las borre o de lo contrario la sentencia `DROP CLUSTER` fallará.

- ✔ `[CASCADE CONSTRAINT]` .- Se eliminan las restricciones de clave externa que una tabla pudiera tener con el cluster. Si no se indica esta cláusula y alguna tabla externa referenciara al cluster entonces el borrado del cluster fallará.

Por ejemplo, la siguiente sentencia elimina el cluster que se creó en el apartado anterior.

```
DROP CLUSTER emp_dept INCLUDING TABLES;
```

Se puede borrar el índice del cluster como cualquier otro índice de tabla, pero no se podrá acceder al cluster hasta que tenga un nuevo índice. El borrado y recreación de un índice de cluster se emplea para como parte de un procedimiento para reconstruir un índice de cluster fragmentado.

4 Control de acceso

Cada BD debe desarrollar su política de seguridad la cual establece métodos para proteger la BD ante pérdida, accidental o maliciosa, de los datos o daños a la infraestructura de la BD.

Cada BD tiene un administrador de la seguridad, que en el caso de BD pequeñas puede ser el mismo administrador de la BD, pero en grandes sistemas de BD será una persona o un grupo de personas a cargo de la seguridad de la BD.

En los siguientes apartados veremos la gestión de usuarios y los privilegios que se asignan a dichos usuarios para que solamente pueden acceder a la información necesaria de la BD.

4.1 Gestión de usuarios

Cada BD Oracle tiene una lista de usuarios válidos. Para acceder a la BD de datos un usuario ejecuta una aplicación de base de datos y conecta a la instancia de la BD utilizando un nombre de usuario definido en la BD. Oracle nos permite establecer una seguridad para los usuarios de diferentes maneras. Cuando creamos una cuenta de usuario podemos especificar límites en el uso de recursos para esa cuenta de usario como parte del dominio de seguridad del usuario.

4.1.1 Crear usuarios

Una cuenta de usuario se crea con la sentencia `CREATE USER`. Generalmente, el administrador de la BD es el úncio que puede crear usuarios. La sintaxis de `CREATE USER` es:

```
CREATE USER <usuario> IDENTIFIED BY <clave>
[DEFAULT TABLESPACE <nombre_TS>]
[TEMPORARY TABLESPACE <nombre_TS>]
[QUOTA {<tamaño> | UNLIMITED} ON <nombre_TS>, […] ]
```

```
[PASSWORD EXPIRE];
[ACCOUNT { LOCK | UNLOCK }]
[CONTAINER = { ALL | CURRENT }]
```

Donde:

- ✔ `CREATE USER <usuario>` .- Nombre de la cuenta de usuario que se está creando. Si se está creando un usuario común debe empezar por los caracteres del prefijo establecido en el parámetro de inicialización `COMMON_USER_PREFIX`. Por defecto es `C##`. Si el usuario es un usuario local, entonces su nombre no puede comenzar por el prefijo anterior.
- ✔ `IDENTIFIED BY <clave>` .- Se especifica la clave que debe usar el usuario para autenticarse cuando abra sesión con la BD.
- ✔ `DEFAULT TABLESPACE <nombre_TS>` .- TABLESPACE por defecto del usuario. Cuando el usuario crea objeto del esquema se almacenará en este TABLESPACE por defecto si no indica ninguno para el objeto creado. Si no indicamos esta cláusula y el usuario tampoco indica un TABLESPACE para almacenar el objeto que crea, se almacenará en el TABLESPACE por defecto de la BD que es SYSTEM.
- ✔ `TEMPORARY TABLESPACE <nombre_TS>` .- TABLESPACE temporal por defecto del usuario. Cuando el usuario ejecuta sentencias SQL con ordenaciones o joins necesita segmentos temporales que se almacenan en un TABLESPACE temporal. Si no se indica el usuario tiene el TABLESPACE temporal por defecto de la BD. Si no lo tiene indicado, entonces se emplea TEMP.
- ✔ `QUOTA {<tamaño> | UNLIMITED} ON <nombre_TS>, […]` .- Asigna cuota de espacio de almacenamiento en el TABLESPACE indicado `<tamaño>`. Si se especifica `UNLIMITED` entonces el usuario dispone de todo el espacio del TABLESPACE. Esta cláusula se puede repetir completa separadas por comas para indicar diferentes cuotas en diferentes TABLESPACES. Hasta que un usuario no tenga asigna cuota en un TABLESPACE no podrá crear objetos en su esquema o añadir datos en tablas almacenadas en otros TABLESPACES.
- ✔ `PASSWORD EXPIRE` .- Se fuerza al usuario a cambiar su clave antes de poder conectarse a la BD.
- ✔ `ACCOUNT { LOCK | UNLOCK }` .- Si se especifica `LOCK` la cuenta de usuario se deshabilita y no puede acceder. Si se especifica `UNLOCK`, la cuenta de usuario se habilita y puede acceder. Por defecto es `UNLOCK`.
- ✔ `CONTAINER = { ALL | CURRENT }` .- Esta cláusula se aplica cando estamos conectados a un CDB. Sin embargo no es necesario especificar esta cláusula ya que su valor por defecto es el único permitido.
  - ➢ Para crear un usuario común debemos estar conectados a la raíz. Opcionalmente podemos especificar `CONTAINER = ALL`, lo que es por defecto cuando estamos conectado a la raíz.

- Para crear un usuario local debemos estar conectados a un PDB. Opcionalmente podemos especificar `CONTAINER = CURRENT`, lo que es por defecto cuando estamos conectados a un PDB.

Para crear un usuario conviene antes crear dos TABLESPACES, permanente el primero y temporal el segundo, para el usuario. Podemos asignar más de un usuario al mismo TABLESPACE, dependerá de los requisitos de la organización. También resulta conveniente asignar un perfil para establecer límites de los recursos que puede utilizar el usuario.

En el siguiente ejemplo creamos el usuario y le asignamos TABLESPACES.

```
CREATE TABLESPACE usuario01
DATAFILE 'usuario01_01.dbf'
    SIZE 100M
    AUTOEXTEND ON NEXT MAXSIZE UNLIMITED;

CREATE TEMPORARY TABLESPACE tmp_us01
DATAFILE 'tmp_us01_01.dbf'
    SIZE 50M
    AUTOEXTEND ON NEXT MAXSIZE UNLIMITED;

CREATE USER pepe IDENTIFIED BY clave1234
DEFAULT TABLESPACE usuario01
QUOTA UNLIMITED ON usuario01
DEFAULT TEMPORARY TABLESPACE tmp_us01;
```

4.1.2 Modificar usuarios

La sentencia `ALTER USER` permite cambiar las propiedades de un usuario de la BD. Su sintaxis es:

```
ALTER USER <usuario> IDENTIFIED BY <clave>
[DEFAULT TABLESPACE <nombre_TS>]
[TEMPORARY TABLESPACE <nombre_TS>]
[QUOTA {<tamaño> | UNLIMITED} ON <nombre_TS>, […] ]
[DEFAULT ROLE { <role>, […] | ALL} ]
[PASSWORD EXPIRE]
[ACCOUNT { LOCK | UNLOCK } ];
```

Donde:

- ✔ `ALTER USER <usuario>` .- Nombre de la cuenta de usuario que vamos a modificar.
- ✔ `IDENTIFIED BY <clave>` .- Se modifica la clave de acceso del usuario.
- ✔ `DEFAULT TABLESPACE <nombre_TS>` .- Se modifica el TABLESPACE por defecto del usuario.
- ✔ `TEMPORARY TABLESPACE <nombre_TS>` .- Se modifica el TABLESPACE temporal por defecto del usuario.
- ✔ `QUOTA {<tamaño> | UNLIMITED} ON <nombre_TS>, […]` .- Asigna cuota de espacio de almacenamiento en el TABLESPACE indicado `<tamaño>`. Si se especifica

`UNLIMITED` entonces el usuario dispone de todo el espacio del TABLESPACE. Esta cláusula se puede repetir completa para indicar diferentes cuotas en diferentes TABLESPACES.

- ✔ `PASSWORD EXPIRE` .- Se fuerza al usuario a cambiar su clave antes de poder conectarse a la BD.
- ✔ `DEFAULT ROLE <role>, …` .- Se especifica el conjunto de roles que se habilitan por defecto al usuario cuando abre sesión. Estos roles tienen que haber sido concedidos previamente al usuario con la sentencia `GRANT`. Los privilegios asignados a un role concedido con `GRANT` pero sin habilitar no tienen efecto.
- ✔ `ACCOUNT {LOCK | UNLOCK}` .- Bloquea, o desbloquea, la cuenta de usuario. Una cuenta de usuario bloqueada no puede abrir sesión con la BD.

4.1.3 Borrar usuarios

Para borrar un usuario empleamos la sentencia `DROP USER` que tiene la siguiente sintaxis:

```
DROP USER <usuario> [CASCADE];
```

Donde:

- ✔ `DROP USER <usuario>` .- Nombre de la cuenta de usuario a eliminar.
- ✔ `[CASCADE]` .- Se elimina todos los objetos del esquema del usuario. Si el esquema contiene tablas referenciadas por restricciones de clave externa desde otras tablas, estas restricciones también se borran.

4.2 Privilegios y roles

Un privilegio de usuario es un derecho para ejecutar un tipo específico de sentencia SQL, o el derecho para acceder a un objeto que pertenece a otro usuario. Un role agrupa privilegios u otros roles para facilitar la concesión de múltiples privilegios o roles a los usuarios.

Existen tres categorías de privilegios o roles:

- ✔ Privilegios de sistema.- Permiten realizar tareas administrativas en la BD. Se asignan solamente a cuentas de usuario de confianza.
- ✔ Privilegios de objeto.- Privilegios específicos que se aplican a un tipo concreto de objeto de esquema.
- ✔ Roles de usuario.- Se agrupan un conjunto de privilegios o roles para concederlos o revocarlos juntos a los usuarios.

Los privilegios se conceden a los usuarios para que puedan realizar las tareas necesarias, es decir, se concede un privilegio a un usuario que necesita ese privilegio para que pueda realizar su trabajo. Conceder más privilegios de los necesarios puede

comprometer la seguridad. Por ejemplo, nunca se concede el privilegio SYSDBA a un usuario que no realiza tareas administrativas.

Los usuarios reciben los privilegios de dos formas:

- ✔ Se pueden conceder privilegios a usuarios específicos. Por ejemplo, podemos asignar al usuario `pepe` un privilegio para insertar filas en una tabla.
- ✔ Podemos asignar privilegios a un role, y después asignar el role a uno o a varios usuarios. Por ejemplo podemos crear un role para seleccionar, insertar, actualizar y borrar filas en una tabla y luego asignar ese role a los usuarios `pepe` y `manolo`.

Para conceder privilegios se emplea la sentencia `GRANT` y para revocarlos se emplea la sentencia `REVOKE`.

4.2.1 Privilegios de sistema

Los privilegios de sistema son asignados por el administrador de la BD. También por cualquier usuario que tiene asignado el privilegio de sistema `ADMIN OPTION` o `GRANT ANY PRIVILEGE`.

Para conceder un privilegio de sistema tenemos la siguiente sintaxis de la sentencia `GRANT`:

```
GRANT { <privilegio_sistema> | <role> | ALL PRIVILEGES }, …
TO { <usuario> | <role> | PUBLIC }, …
[WIDTH ADMIN OPTION];
```

Donde:

- ✔ `GRANT <privilegio_sistema>` .- Hay que indicar el privilegio de sistema que se asigna. En la página 18-42 de *Oracle Database SQL Language Reference* aparece una lista con los privilegios del sistema y su significado.
- ✔ `GRANT <role>` .- Se concede el role (grupo de privilegios) al usuario o a otro role. En este caso el usuario, o el role, asume todos los privilegios del role, es decir, se le conceden en una sola operación todos los privilegios del role.
- ✔ `GRANT ALL PRIVILEGES` .- Se conceden todos los privilegios al usuario.
- ✔ `TO <usuario>` .- Se indica el usuario al que se asigna el privilegio.
- ✔ `TO <role>` .- Se asigna el privilegio al role indicado.
- ✔ `TO PUBLIC` .- Se asigna el privilegio a todos los usuarios de la BD.
- ✔ `WIDTH ADMIN OPTION` .- Indica que el usuario al que se concede el privilegio puede, a su vez, concederlo a otros usuarios. También podría quitar el privilegio o role asignado a otros usuarios.

4.2.2 Privilegios de objeto

Para conceder privilegios de objeto también usamos la sentencia `GRANT` con la siguiente sintaxis:

```
GRANT { <privilegio_objeto> | ALL [PRIVILEGES] }
    [(<columna>,… )] , …
ON [<esquema>].<objeto>
TO { <usuario> | <role> | PUBLIC }, …
[WIDTH GRANT OPTION];
```

Donde:

- ✔ `GRANT <privilegio_objeto>` .- Hay que indicar el privilegio de objeto que se asigna. En la página 18-54 de *Oracle Database SQL Language Reference* aparece una lista con los privilegios de objeto y su significado.
- ✔ `GRANT ALL [PRIVILEGES]` .- Se conceden todos los privilegios de objeto al usuario.
- ✔ `(<columna>, …)` .- Se conceden los privilegios de objeto sobre las especificadas columnas de tabla o vista al usuario. En este caso los privilegios de objeto que se pueden conceder son aquellos que permiten realizar una operación sobre una columna, como `INSERT`, `REFERENCES` o `UPDATE`. Si no se indica una lista de columnas entonces se aplica el privilegio a toda la tabla o vista.
- ✔ `ON [<esquema>].<objeto>` .- Se indica el objeto del esquema sobre el que se concede los privilegios de objeto. Si no se es el propietario del objeto se necesita el privilegio de sistema `GRANT ANY OBJECT PRIVILEGE` para poder conceder privilegios de objeto sobre objetos que son propiedad de otros usuarios o tener concedido el privilegio con la opción `WITH GRANT OPTION`. Cuando se omite el nombre del esquema se asume que el objeto pertenece al esquema del usuario que ejecuta la sentencia `GRANT`.
- ✔ `TO <usuario>` .- Se indica el usuario al que se asigna el privilegio de objeto.
- ✔ `TO <role>` .- Se asigna el privilegio de objeto al role indicado.
- ✔ `TO PUBLIC` .- Se asigna el privilegio de objeto a todos los usuarios de la BD.
- ✔ `WIDTH GRANT OPTION` .- Indica que el usuario al que se concede el privilegio puede, a su vez, concederlo a otros usuarios.

Los objetos de esquema más habituales a los que se concede privilegios son:

- ✔ Tablas
- ✔ Vistas
- ✔ Secuencias
- ✔ Procedimientos o funciones

- ✔ Sinónimos

4.2.3 Roles

La sentencia `CREATE ROLE` se emplea para crear un role, que consiste en un conjunto de privilegios que se asignan juntos a usuarios u otros roles. En general, se crea un role al que se asignan un conjunto de privilegios y, posteriormente, se asigna el role a los usuarios, los cuales tienen concedidos todos los privilegios del role.

Un role contiene todos los privilegios concedidos al role y todos los privilegios de otros roles concedidos a él. La sintaxis de `CREATE ROLE` es:

```
CREATE ROLE <role>
IDENTIFIED BY <password>;
```

Donde:

- ✔ `<role>` .- Nombre del role que se crea.
- ✔ `<password>` .- Clave del role. Esta clave debe utilizarla el usuario que quiera adquirir el role con la sentencia `SET ROLE`.

Existe una lista de roles predefinidos. Consultar las páginas 247 a la 251 del documento *Oracle Database Security Guide* que se pueden obtener de la web de Oracle.

Cuando un usuario conecta con la BD Oracle, se le habilitan todos los privilegios explícitamente concedidos y los roles por defecto del usuario. Estos roles se le asignaron cuando se ejecutó la sentencia `ALTER USER` con la cláusula `DEFAULT ROLE <role>, …` Durante la sesión, el usuario o la aplicación de BD puede utilizar la sentencia `SET ROLE` tantas veces como quiera para habilitar o deshabilitar roles actualmente habilitados para la sesión. La sintaxis es:

```
SET ROLE <role>
IDENTIFIED BY <password>;
```

Donde:

- ✔ `<role>` .- Nombre del role que se se desea habilitar.
- ✔ `<password>` .- Clave del role. Esta clave fue la que se estableció cuando se creó el role con la sentencia `CREATE ROLE`.

Si el usuario no tiene un role por defecto asignado, entonces dispone de los privilegios concedidos en todos los roles que tenga asignados.

4.2.4 Revocar privilegios

Para revocar privilegios, de sistema u objeto, y roles a los usuarios utilizamos la setencia `REVOKE`. La sintaxis para revocar privilegios de sistema o roles es la siguiente:

```
REVOKE { <privilegio_sistema> | <role> | ALL PRIVILEGES }, …
```

```
FROM { <usuario> | <role> | PUBLIC }, …
```

Donde:

- ✔ `REVOKE <privilegio_sistema>` .- Hay que indicar el privilegio de sistema que se revoca. En la página 18-40 de *Oracle Database Security Guide* aparece una lista con los privilegios del sistema y su significado.
- ✔ `REVOKE <role>` .- Se deshabilita el role (grupo de privilegios) al usuario o a otro role. En este caso al usuario, o role, se le revocan todos los privilegios del role, es decir, se le revocan en una sola operación todos los privilegios del role.
- ✔ `REVOKE ALL PRIVILEGES` .- Se revocan todos los privilegios al usuario.
- ✔ `FROM <usuario>` .- Se indica el usuario al que se revoca el privilegio.
- ✔ `FROM <role>` .- Se revoca el privilegio al role indicado.
- ✔ `FROM PUBLIC` .- Se revoca el privilegio a todos los usuarios de la BD.

Si lo que queremos es revocar privilegios de objeto tendremos que utilizar la sentencia `REVOKE` con la siguiente sintaxis.

```
REVOKE { <privilegio_objeto> | ALL [PRIVILEGES] } , …
ON [<esquema>].<objeto>
FROM { <usuario> | <role> | PUBLIC }, …
```

Donde:

- ✔ `REVOKE <privilegio_objeto>` .- Hay que indicar el privilegio de objeto que se revoca. En la página 18-48 de *Oracle Database Security Guide* aparece una lista con los privilegios de objeto y su significado.
- ✔ `GRANT ALL [PRIVILEGES]` .- Se revocan todos los privilegios de objeto al usuario.
- ✔ `ON [<esquema>].<objeto>` .- Se indica el objeto del esquema sobre el que se revocan los privilegios de objeto. Si no se es el propietario del objeto se necesita el privilegio de sistema `GRANT ANY OBJECT PRIVILEGE` para poder revocar privilegios de objeto sobre objetos que son propiedad de otros usuarios o tener concedido el privilegio con la opción `WITH GRANT OPTION`. Cuando se omite el nombre del esquema se asume que el objeto pertenece al esquema del usuario que ejecuta la sentencia `REVOKE`.
- ✔ `FROM <usuario>` .- Se indica el usuario al que se revoca el privilegio de objeto.
- ✔ `FROM <role>` .- Se revoca el privilegio de objeto al role indicado.
- ✔ `FROM PUBLIC` .- Se revoca el privilegio de objeto a todos los usuarios de la BD que les fue concedido con una sentencia `GRANT … TO PUBLIC`. Privilegios concedidos directamente o a traves de roles no se pueden revocar con `PUBLIC`.

5 Consultar información de la BD

A continuación se exponen un conjunto de sentencias `SELECT` al diccionario de datos para obtener datos útiles de la BD. Estas consultas hay que hacerlas con el usuario `SYS`.

5.1 Consultas al diccionario de datos

Ver las sesiones actuales abiertas con la BD

```
SELECT osuser, username, machine, program
FROM v$session
ORDER BY osuser
```

Ver las tablas de usuarios

```
SELECT table_name FROM user_tables ORDER BY table_name;
```

Ver las tablas de un usuario concreto

```
SELECT table_name FROM dba_tables
WHERE owner ='JUAN'
ORDER BY table_name;
```

Ver los constraints de usuario

```
SELECT * FROM user_constraints;
```

Ver los índices de usuarios

```
SELECT * FROM user_indexes;
```

Reglas de integridad y columna a la que afectan:

```
SELECT constraint_name, column_name FROM sys.all_cons_columns;
```

Usuarios de Oracle

```
SELECT * FROM dba_users;
```

Ver los privilegios de usuario

```
SELECT * FROM USER_SYS_PRIVS;
SELECT * FROM USER_TAB_PRIVS;
SELECT * FROM USER_ROLE_PRIVS;
```

Ver los TABLESPACES

```
SELECT tablespace_name, status FROM dba_tablespaces;
```

Ver los datafiles de un TABLESPACE

```
SELET file_name, file_id, relative_fno, bytes FROM
dba_data_files
WHERE TABLESPACE_NAME = <nombre_TS>;
```

5.2 Consultas de los parámetros del servidor (init.ora y spfile.ora)

Ver los parámetros del archivo init.ora

```
SELECT name, value FROM v$parameter ORDER BY name;
```

Ver los parámetros del archivo spfile.ora

```
SELECT name, value FROM v$spparameter ORDER BY name;
```

Ver un parámetro

```
SHOW PARAMETER <parámetro>
```

5.3 Ver propiedades de la BD

```
SELECT property_name, property_value FROM database_properties;
```

6 Bibliografía

DUARTE, Eugenio, *¿Cuáles son las funciones de un Administrador de Base de Datos?* [acceso noviembre 2017]. Disponible en <http://blog.capacityacademy.com/2013/02/18/cuales-son-las-funciones-de-un-administrador-de-base-de-datos-parte-1-de-2/>

WIKIPEDIA, Lenguaje de definición de datos. *[a*cceso noviembre 2017]. Disponible en <https://es.wikipedia.org/wiki/Lenguaje_de_definici%C3%B3n_de_datos>

HALL, Tim, *AutoNumber And Identity Functionality (Pre 12c) [a*cceso diciembre 2017]. Disponible en <https://oracle-base.com/articles/misc/autonumber-and-identity>

KRISHNAMURTHY, Usha, *Oracle Database Express Edition – SQL Reference Guide – 21c Release*. ORACLE 2022

HUEY, Patricia, *Oracle Database Security Guide, 21c* ORACLE 2022