

Seguridad y Fiabilidad en los SGBD

Resumen del capítulo:

- 4.1. Seguridad de los Datos
- 4.2. Fiabilidad y Recuperación frente a Fallos
- 4.3. Salvado y Recuperación de BD

Problemas

Dos aspectos muy importantes de todo sistema de información son:

1. Los referentes a la garantía de que no se puede acceder libremente a esa información por parte de cualquier usuario.
2. La confianza que puede tener un usuario en que el sistema es capaz de recuperarse por sí mismo tras un fallo, manteniendo la coherencia de los datos.

Estos dos aspectos se engloban en lo que genéricamente se denomina *seguridad y fiabilidad* de los datos, respectivamente.

La información de la BD debe estar protegida contra los accesos no autorizados, los borrados o alteraciones indebidos y la introducción accidental de información inconsistente. Como sabemos, gran parte de la tarea de protección de los datos frente a inconsistencias se lleva a cabo de forma indirecta gracias al correcto diseño de la propia BD (definiciones de claves primarias y externas, uso de la cláusula *not null* en la definición de atributos, etc.) que, si bien es de gran importancia, no cubre todos los aspectos referentes a la integridad de la información. En este capítulo examinaremos los mecanismos que pueden emplear los SGBD para prevenir el mal uso de la información e impedir que ésta se vuelva inconsistente, analizaremos los distintos mecanismos que emplean los sistemas para recuperarse frente a una caída accidental y mostaremos los principales métodos de copia de seguridad, que nos permitirán reponer los datos perdidos en caso de fallo en un dispositivo de almacenamiento. Para ilustrar esta última sección, haremos uso de los mecanismos proporcionados por Oracle, dado el gran abanico de posibilidades que ofrece.

4.1. SEGURIDAD DE LOS DATOS

A veces, las inconsistencias que se producen en la BD son, por desgracia, provocadas por usuarios *malintencionados*, siendo más fácil prevenir la pérdida accidental de consistencia que el mal uso intencionado.

No es posible proteger la BD de forma absoluta contra operaciones de este tipo. El término **seguridad** normalmente se refiere a la protección contra este tipo de accesos indebidos. A veces no es tan perjudicial el hecho de que se *robe* la información como el estado en el que se deja la BD.

Para asegurar la protección de la BD, deben llevarse a cabo controles a varios niveles:

- **A nivel físico.** El acceso al local donde se encuentren las máquinas que permiten el control del sistema debe ser restringido, esto es, solo debe permitirse la entrada al personal autorizado.
- **A nivel humano.** Debe conocerse bien a los usuarios a los que se conceden ciertos privilegios sobre el sistema; esta precaución reducirá el riesgo de que personas autorizadas permitan el acceso a intrusos.
- **A nivel de sistema operativo.** Aunque el sistema de BD esté bien protegido, es posible que el sistema operativo sirva para obtener acceso a la BD sin estar autorizado. Dado que casi todos los sistemas de BD permiten el acceso remoto a través de terminales o redes de comunicaciones, la seguridad a nivel software en el sistema operativo es tan importante, o más, que la seguridad física.
- **A nivel del sistema de BD.** Se pueden establecer controles a distintos niveles. A nivel de *información*, estableciendo para cada usuario a qué porción de la BD puede acceder, y a nivel de *operación*, estableciendo qué operaciones son las que cada usuario puede realizar sobre los datos (solo consulta, consulta y modificación, borrado...). El sistema es el encargado de garantizar que no se violen estas restricciones.

Todas estas medidas y a todos estos niveles han de mantenerse simultáneamente ya que, de no ser así, un punto de entrada a bajo nivel permitiría violar todas las normas de seguridad en los niveles superiores. Conforme mayor es la importancia de los datos almacenados en la BD, mayor debe ser el tiempo y el esfuerzo dedicados a conservar su seguridad e integridad.

A menudo, todas las precauciones que se toman para evitar el acceso sin autorización a la BD son insuficientes para proteger información de gran importancia (información de los servicios secretos, información bursátil de importancia, etc.). En estos casos, lo que suele hacerse es codificar dicha información de manera que no sea posible entenderla a menos que se conozca el medio de decodificarla. El proceso de codificación de información se denomina **cifrado**. Las claves no son fáciles de encontrar y las que existen no son precisamente sencillas, ya que están basadas en

propiedades matemáticas complejas que quedan fuera del ámbito de este libro. Si se desea más información sobre las técnicas de cifrado y criptografía, véase [19].

En adelante, nos centraremos en los mecanismos de control de seguridad a nivel del sistema de BD, ya que el resto de los niveles queda fuera del ámbito de este libro. Asimismo, haremos referencia al *modelo relacional* de BD aunque los conceptos tratados pueden aplicarse igualmente a cualquier otro modelo de datos.

4.1.1. AUTORIZACIÓN DE USUARIOS

Los mecanismos que garantizan la seguridad de la información de una BD son las autorizaciones. Éstas pueden establecerse a dos niveles:

1. Mediante la identificación del usuario por una contraseña (*password*) que permite su conexión bajo determinadas condiciones.
2. A nivel de la utilización de los datos. Se puede llevar a cabo la asignación de permisos o privilegios mediante dos mecanismos distintos:
 - Un **sistema centralizado** en el que el administrador lleva todo el control de autorizaciones. Este sistema tiene la ventaja de que es fácil de implementar pero tiene el inconveniente de que no existen objetos privados, ya que el administrador es el único que puede crearlos.
 - Un **sistema descentralizado** en el que existe una jerarquía de usuarios que pueden concederse privilegios unos a otros. En este caso pueden existir objetos privados cuyo propietario puede autorizar a otros a realizar una serie de operaciones.

Con este sistema, la concesión, comprobación y anulación de las autorizaciones no pasa por el administrador de la BD sino que se lleva a cabo por el propietario del objeto en cuestión.

Así, por ejemplo, entre los privilegios que un usuario podría conceder sobre sus tablas a otros usuarios estarían los siguientes:

- Usar la tabla para consultas, lo que incluye leer tuplas y definir vistas a partir de dicha tabla. La seguridad se logra si se cuenta con un mecanismo que limite a

los usuarios a su vista personal (visión externa). Es posible también utilizar una combinación mixta de seguridad a nivel de tabla y a nivel de vista, de forma que se limite el acceso del usuario de forma exclusiva a los datos que necesite.

- Insertar nuevas tuplas; este privilegio no permitiría modificar las ya existentes.
- Borrar tuplas. Un usuario puede borrar todas las tuplas de una relación, quedando ésta vacía pero permaneciendo el objeto en el sistema.
- Modificar los datos de todos o algunos de los atributos de la tabla.
- Modificar la estructura de la tabla añadiendo atributos.
- Borrar la tabla, eliminando la estructura y el contenido.
- Definir índices sobre la tabla. Si bien la creación de índices no puede alterar la información contenida en la tabla, sí puede consumir gran cantidad de memoria y de tiempo de CPU cada vez que éstos se reorganicen. Esto obliga a considerar la creación de índices como un privilegio, lo que permite al administrador regular el uso de los recursos del sistema.
- Autorización para conceder privilegios a otros usuarios sobre los permisos concedidos. Hay que tener cuidado con el modo en que se pasa la autorización de unos usuarios a otros, para asegurar que ésta pueda ser anulada en un futuro si fuera necesario. Las autorizaciones de un usuario a otro suelen representarse mediante un *grafo de autorizaciones*, en el que cada nodo representa a un usuario U_i y una arista de U_i a U_j indica que el usuario U_i concede algún tipo de autorización al usuario U_j .

Para ilustrar el problema, supongamos que el administrador U_0 concede una serie de privilegios (los mismos) a los usuarios U_1 , U_2 y U_3 . Supongamos, además, que el usuario U_5 recibe autorizaciones tanto de U_1 como de U_2 mientras que U_4 recibe autorización solo de U_1 (véase el grafo de la Figura 4.1).

Si el administrador decide revocar la autorización de U_1 , también será necesario revocar en cascada las autorizaciones de los usuarios que *dependen* de él, en este caso, las de U_4 y U_5 . Sin embargo, este último conservará los privilegios dado que también fue autorizado por U_2 y este usuario aun conserva sus privilegios. En el caso de que U_2 fuera también desautorizado, U_5 perdería todos sus privilegios.

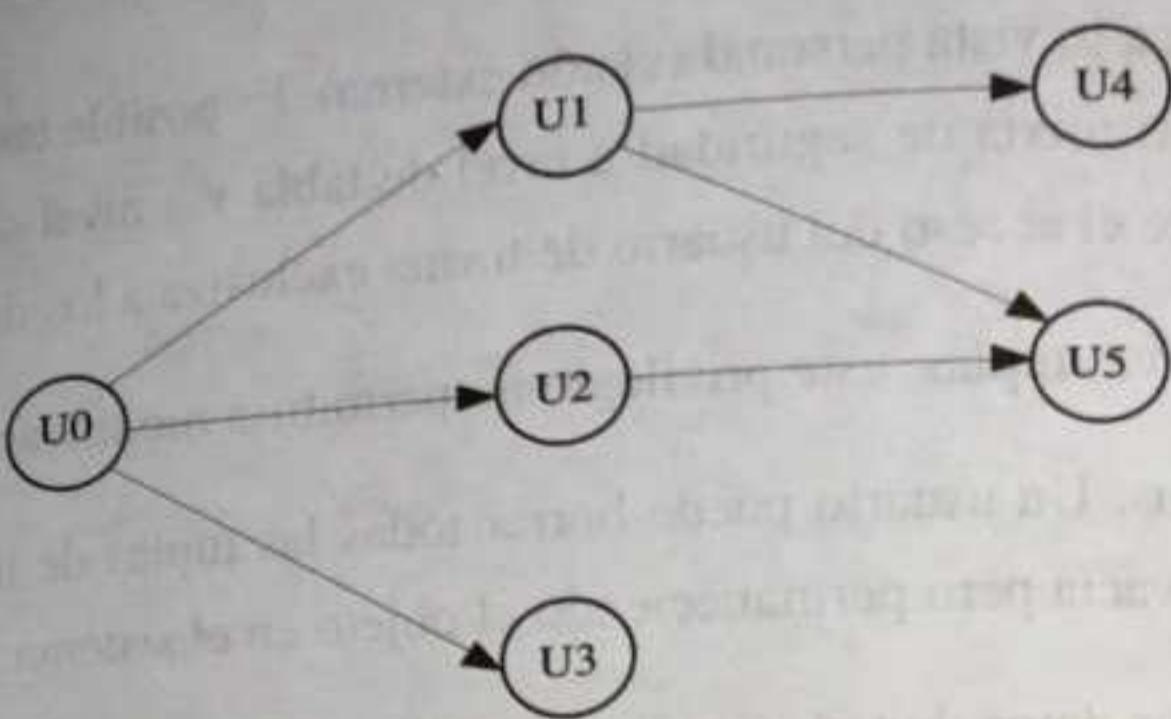


FIGURA 4.1 Grafo de autorizaciones.

Podría darse incluso el caso de que dos usuarios se dieran mutuamente autorización (como se muestra en el grafo de la Figura 4.2) de manera que si el administrador revoca la autorización a uno de ellos, éste la conserve a través del otro.

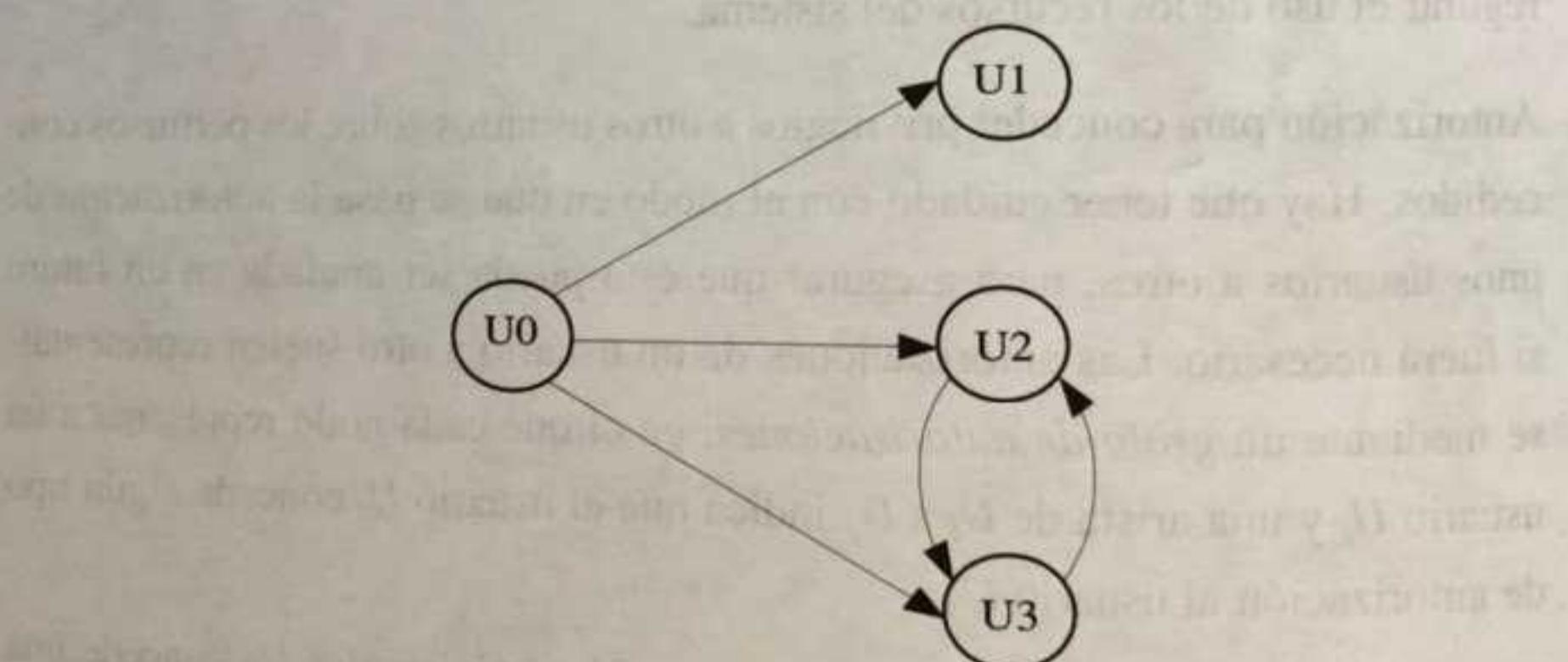


FIGURA 4.2 Grafo de autorizaciones incorrecto.

Para evitar esto, podría comprobarse que todas las aristas del grafo formen parte de una *única* ruta con origen en el administrador. Siguiendo esta norma, el grafo anterior quedaría como se muestra en la Figura 4.3, en el que una revocación de autorización a U_3 provoca la inmediata revocación a U_2 .

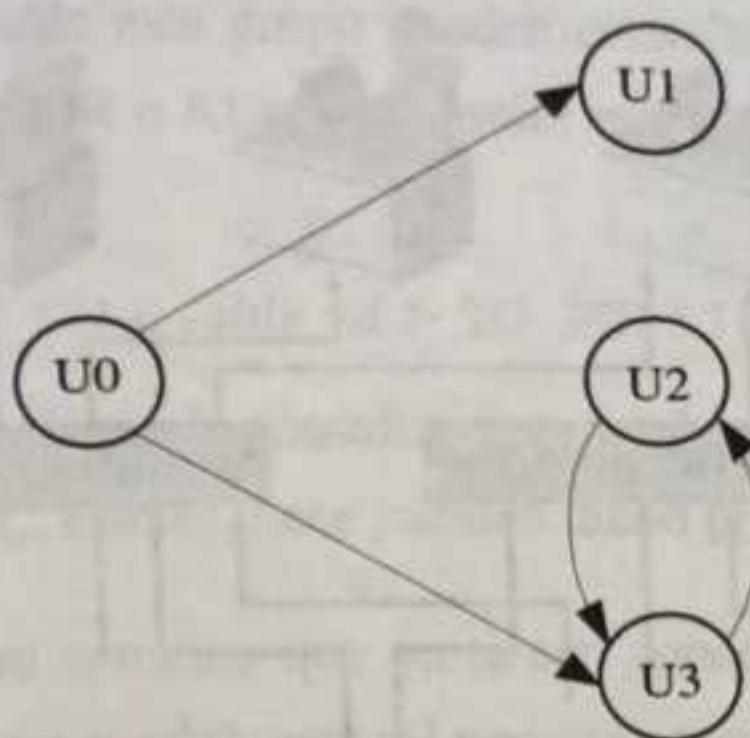


FIGURA 4.3 Grafo de autorizaciones reestructurado.

4.1.2. GESTIÓN DE PRIVILEGIOS Y ROLES EN ORACLE

Un **privilegio** es el derecho de un usuario a ejecutar una determinada sentencia de SQL o a acceder a un objeto (tabla, vista, programa...) que es propiedad de otro usuario. Algunos de estos derechos son: conectarse a la BD, crear una tabla, ejecutar un programa, consultar una tabla, etc. Los privilegios se conceden a los usuarios para que éstos puedan realizar las tareas propias de su trabajo, pero nunca deberá concederse a un usuario un privilegio que no sea absolutamente necesario para el desarrollo del mismo. A continuación se detallan los mecanismos que ofrece Oracle para la gestión de dichos privilegios. Si se desea más información, véanse [11] y [17].

◦ *Los privilegios pueden concederse de dos formas:*

- De forma explícita a un usuario concreto. Por ejemplo, se le puede conceder el privilegio de insertar registros en la tabla **EMPLEADOS** al usuario *tintin*.
- De forma indirecta, definiendo un *rol* que luego se asigna a uno o varios usuarios. Un rol consiste en la agrupación de varios privilegios bajo un nombre. Lo normal es que un administrador que tenga que gestionar muchos usuarios no administre los privilegios de forma individualizada, sino que agrupe varios privilegios según su funcionalidad. En la Figura 4.4 se muestra un ejemplo en el que aparecen dos tipos de funcionalidades: *programador* y *administrativo*, y varios usuarios.

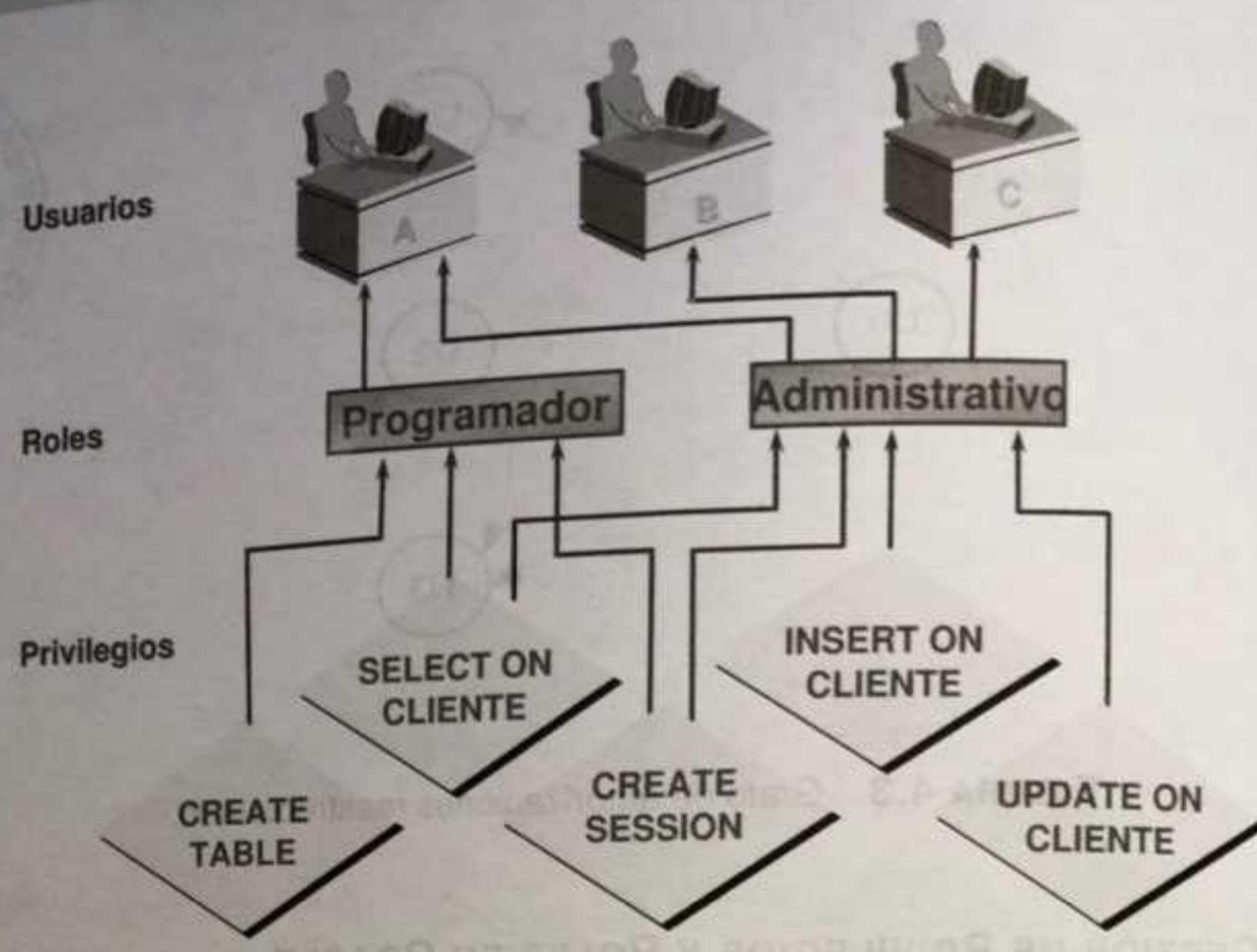


FIGURA 4.4 Asignación de privilegios a usuarios a través de roles.

Existen algunos *roles predefinidos*¹ en el sistema, entre los que se encuentran:

- CONNECT: Este rol permite al usuario conectarse a la BD en unas determinadas condiciones y contiene, como mínimo, el privilegio `create session`. Un usuario que lo tenga concedido puede:
 - Conectarse a la BD Oracle.
 - Consultar y/o manipular las tablas públicas.
 - Crear vistas.
 - Exportar tablas.

pero no podrá crear tablas ni índices.

ORACLE tiene definido un usuario especial llamado PUBLIC. Este usuario representa al grupo de usuarios que tiene el privilegio CONNECT sobre la BD.

¹ El contenido particular de cada rol depende de la versión de Oracle utilizada y puede conocerse consultando las tablas correspondientes del catálogo.

Todos los usuarios de este grupo pueden consultar las tablas del diccionario cuyo prefijo sea USER o ALL y cualquier tabla en la que su propietario haya ejecutado el comando:

```
GRANT select ON <table_id> TO PUBLIC;
```

que concede el privilegio de consulta sobre la tabla *table_id* a todos los usuarios del grupo PUBLIC, esto es, hace *público* dicho privilegio.

- RESOURCE: Para usuarios que ya tienen concedido el rol CONNECT (no tiene sentido de otro modo), este rol permite crear tablas e índices.
- DBA: Este rol, además de incluir por defecto a los dos anteriores, permite, entre otras posibilidades:
 - Acceder a los datos de cualquier usuario.
 - Conceder y revocar privilegios a los usuarios de la BD.
 - Llevar a cabo operaciones de mantenimiento sobre el sistema, como añadir índices, hacer copias de seguridad, ampliar memoria a los usuarios, etc.
 - Exportar parcial o totalmente la BD.

La sentencia de creación de roles en SQL es la siguiente:

```
CREATE ROLE <role_id>;
```

y la de asignación de privilegios a un rol ya creado es la sentencia GRANT, cuya sintaxis veremos a continuación.

En Oracle existen dos tipos distintos de privilegios: los privilegios sobre el sistema y los privilegios sobre objetos concretos.

4.1.2.1. Privilegios sobre el sistema

Un privilegio sobre el sistema es un derecho a realizar una acción genérica sobre el mismo, como por ejemplo, la creación de tablas o la posibilidad de hacer un backup de la BD. Este tipo de privilegios suele concederse solo a personal administrativo y programadores de aplicaciones porque no es habitual que usuarios terminales

necesiten realizar este tipo de operaciones. En la Tabla 4.1 se recogen algunos de los privilegios sobre el sistema que tienen mayor interés.

TABLA 4.1 Privilegios sobre el sistema.

Grupo	Privilegio	Descripción
TABLE	CREATE TABLE	Crear tablas
	ALTER ANY TABLE	Modificar cualquier tabla
	BACKUP ANY TABLE	Hacer copia de seguridad a cualquier tabla
	DROP ANY TABLE	Borrar cualquier tabla
	SELECT ANY TABLE	Consultar cualquier tabla
	UPDATE ANY TABLE	Actualizar cualquier tabla
INDEX	CREATE INDEX	Crear índices sobre las propias tablas
	CREATE ANY INDEX	Crear índices sobre cualquier tabla
	DROP ANY INDEX	Eliminar índices de cualquier tabla
VIEW	CREATE ANY VIEW	Crear vistas sobre cualquier tabla
	DROP ANY VIEW	Eliminar cualquier vista
PROCEDURE	CREATE/EXECUTE ANY PROCEDURE	Crear o ejecutar módulos
PRIVILEGE	GRANT ANY PRIVILEGE	Conceder cualquier privilegio a otro usuario
ROLE	CREATE ROLE	Definir roles
	GRANT ANY ROLE	Conceder cualquier rol a otro usuario
SESSION	CREATE SESSION	Conectar usuarios a la BD
USER	CREATE USER	Crear usuarios asignándoles espacio y privilegios
TABLESPACE	CREATE TABLESPACE	Crear nuevos tablespaces
	ALTER TABLESPACE	Modificar parámetros de un tablespace
	DROP TABLESPACE	Eliminar tablespaces
DATABASE	ALTER DATABASE	Modificar la BD

Como vemos en la lista de privilegios de la Tabla 4.1, la palabra reservada ANY aparece con frecuencia e indica que se dispone del privilegio en cuestión sobre cualquier esquema.

4.1.2.2. Privilegios sobre los objetos

Un privilegio sobre un objeto es un derecho a realizar una acción particular sobre un objeto *específico* de la BD: una tabla, una vista, una función... Por ejemplo, el privilegio de borrar tuplas de la tabla `clientes` es un privilegio sobre un objeto. Un usuario tiene concedidos de forma automática todos los privilegios sobre los objetos de su propiedad, incluido el derecho a conceder privilegios sobre sus objetos a otros usuarios. En la Tabla 4.2 se recogen los privilegios que se pueden conceder sobre objetos concretos.

TABLA 4.2 Privilegios sobre objetos.

Privilegio	Descripción
ALL [PRIVILEGES]	Todos los privilegios sobre objetos
DELETE	Eliminar registros
EXECUTE	Ejecutar un objeto (función, procedimiento, paquete)
INDEX	Crear un índice sobre una tabla
INSERT	Insertar registros en una tabla
REFERENCES	Definir claves externas que refieren este objeto
SELECT	Consultar una tabla
UPDATE	Modificar registros de una tabla

Las sentencias de SQL que sirven para conceder privilegios son las siguientes:

- Concesión de privilegios y roles del sistema.

`GRANT { <system_priv> } TO { <user> | <role> | PUBLIC } [WITH ADMIN OPTION]`

La cláusula WITH ADMIN OPTION es opcional e indica que el receptor del privilegio especificado puede, a su vez, concederlo a terceros; esto es, administrar el privilegio. Si se deroga el privilegio concedido al usuario original, no se le retira de forma automática a los usuarios que lo obtuvieron a través de él.

- Concesión de privilegios y roles sobre objetos.

GRANT { <object_priv>
ALL PRIVILEGES } ON <object_id> TO
{ <user>
<role>
PUBLIC } [WITH GRANT OPTION]

La cláusula WITH GRANT OPTION es opcional e indica que el receptor del privilegio especificado puede, a su vez, concederlo a terceros. Al contrario de lo que sucede con los privilegios del sistema, si se deroga el privilegio concedido al usuario original, se les retirará en cascada a todos los que lo obtuvieron por medio de él.

- *Revocación de privilegios y roles del sistema*

REVOKE { <system_priv>
<system_role> } FROM { <user>
<role>
PUBLIC }

- *Revocación de privilegios y roles sobre objetos*

REVOKE { <object_priv>
ALL PRIVILEGES } ON <object_id> FROM
{ <user>
<role>
PUBLIC }

donde la notación empleada tiene el siguiente significado:

- <system_priv>: Consiste en una lista de privilegios del sistema separada por comas. Consultar Tabla 4.1.
- <system_role>: Consiste en la especificación de uno o varios roles del sistema previamente creados.
- <object_priv>: Consiste en una lista de privilegios de objeto separados por comas. Consultar Tabla 4.2.
- <object_id>: Consiste en una lista de nombres de objeto sobre los que recae el privilegio en cuestión: tabla, vista, procedure...

Como puede observarse, la identificación de una lista de roles aparece como conjunto de privilegios a concederse (a la izquierda) y como beneficiario de privilegios tras la

cláusula TO. Esto se debe a que el rol, una vez creado, debe ser dotado de funcionalidad mediante la concesión de privilegios al mismo. Una vez realizada esta tarea, el rol se comporta como un grupo de privilegios que puede ser asignado a otros usuarios o roles.

4.1.3. EJEMPLOS

Para ilustrar el uso de la sintaxis descrita vamos a considerar que existen los usuarios *coyote*, *tintiny* y *filemon*, las tablas *emp* y *cliente* y el procedimiento *balance*.

- Concesión del privilegio de consulta sobre la tabla *emp* a todos los usuarios con derecho de conexión.

```
grant select on emp to public;
```

- Conceder la posibilidad de modificar la columna *trabajo* de la tabla *emp* al usuario *coyote* y que éste, a su vez, pueda concederlo a terceros.

```
grant update(trabajo) on emp to coyote  
with grant option;
```

- Conceder al usuario *filemon* los privilegios de crear tablas, crear procedimientos y crear índices sobre cualquier esquema de la BD.

```
grant create any table, create any procedure,  
create any index to filemon.
```

- Crear un rol llamado *gestion* que permita a los usuarios que lo obtengan, las operaciones de consulta (SELECT), inserción (INSERT), modificación (UPDATE) y borrado (DELETE) sobre las tablas *emp* y *cliente*.

```
create role gestion;  
grant select, insert, update, delete on emp, cliente  
to gestion;
```

- Conceder a *tintin* el rol *gestion* y la posibilidad de ejecutar el procedimiento *balance*.

- Derogar el privilegio de crear procedimientos al usuario filemon.
`grant gestion to tintin;`
`grant execute on balance to tintin;`
`revoke create procedure from filemon;`
- Impedir a los usuarios con el rol gestion el borrado de filas sobre la tabla emp;
`revoke delete on emp from gestion;`

4.1.3.1. Organización de privilegios en el catálogo de la BD

En esta sección veremos algunas vistas del catálogo relacionadas con la concepción de roles y privilegios, para ilustrar cómo y dónde se almacena toda esta información que es mantenida y gestionada por el sistema.

La Tabla 4.3 muestra algunas de las vistas del catálogo que recogen la información sobre concesiones de privilegios y roles tanto a usuarios como a roles.

TABLA 4.3 Vistas del catálogo relacionadas con privilegios y roles.

Vista	Descripción
DBA_ROLES	Todos los roles que existen
DBA_ROLE_PRIVS	Roles asignados a usuarios o a roles
ROLE_ROLE_PRIVS	Roles asignados a roles
DBA_SYS_PRIVS	Privilegios del sistema asignados a usuarios o a roles
ROLE_SYS_PRIVS	Privilegios del sistema asignados a roles
ROLE_TAB_PRIVS	Privilegios de tablas asignados a roles
DBA_TAB_GRANTS_MADE	Privilegios de tablas asignados a usuarios

A continuación detallamos el esquema de algunas de ellas por su interés. El lector interesado puede encontrar una descripción exhaustiva de las mismas en [9] y [10].

- DBA_SYS_PRIVS: En esta tabla se almacenan los privilegios del sistema concedidos a los usuarios. Sus columnas son:

- GRANTEE: Nombre del usuario que recibe el privilegio.
- PRIVILEGE: Nombre del privilegio.
- ADMIN OPTION: Si se concede con la opción de administrarlo, esto es, de concederlo a otros usuarios.

```
SELECT * FROM DBA_SYS_PRIVS;
```

GRANTEE	PRIVILEGE	ADM
PETER	CREATE SESSION	NO
USER1	CREATE SESSION	NO
USER2	CREATE TABLE	NO
USER1	CREATE TABLE	YES
USER2	CREATE PROCEDURE	NO

- DBA_TAB_GRANTS_MADE. En esta tabla se almacenan los privilegios concedidos a usuarios sobre objetos (tablas y vistas) que son propiedad de otro. Algunas de sus columnas son:

- GRANTOR: Nombre del usuario que concede el privilegio. No tiene por qué ser el propietario, pero sí una persona autorizada.
- TABLE_NAME: Nombre de la tabla.
- OWNER: Propietario de la tabla.
- GRANTEE: Nombre del usuario que recibe el privilegio.
- UPDATE_PRIV: Si se concede o no permiso para actualizar el objeto.
- DELETE_PRIV: Ídem para borrar el objeto.
- INSERT_PRIV: Ídem para insertar en el objeto.
- SELECT_PRIV: Ídem para consultar el contenido del objeto.

```
SELECT GRANTOR, TABLE_NAME, OWNER, GRANTEE,
       UPDATE_PRIV FROM BA_TAB_GRANTS_MADE;
```

GRANTOR	TABLE_NAME	OWNER	GRANTEE	UPDATE_PRIV
PETER	VENTAS	PETER	OPC	NO
OPC	VENTAS	OPC	X234123	YES
USER2	MUSICA	USER1	OPC	YES
USER1	PIEZAS	USER1	USER2	NO
....

4.2. FIABILIDAD Y RECUPERACIÓN FREnte A FALLOS

Un sistema de computadores, como cualquier otro dispositivo mecánico o eléctrico, está sujeto a fallos. Las causas que provocan un fallo en el sistema pueden ser de diversa naturaleza y origen: la rotura de un disco, un corte de electricidad, sobrecarga del sistema (demasiados procesos ejecutándose), errores software, etc.

◊ *Clasificaremos todos los posibles tipos de fallos que pueden producirse en un sistema en cuatro categorías:*

1. **Errores lógicos.** La ejecución actual no puede continuar debido a algún problema interno, como por ejemplo una entrada inválida, un desbordamiento (overflow) o se ha excedido el límite de los recursos. Estos errores no tienen que ver con los componentes físicos del sistema, sino con la ejecución de algún programa.
2. **Errores del sistema.** El sistema ha entrado en un estado de bloqueo que puede producirse, por ejemplo, cuando dos usuarios intentan actualizar simultáneamente el mismo registro, o el sistema no puede atender un número excesivo de procesos en un momento dado, etc. En esta situación, la transacción que se estaba ejecutando queda anulada, pero puede volver a ejecutarse más tarde cuando el sistema se haya recuperado.
3. **Caída del sistema.** El sistema deja de responder debido a un corte en la electricidad o a un fallo del hardware, causando la pérdida de la información contenida en memoria. Sin embargo, el contenido de las unidades de almacenamiento externo permanece intacto.

- 4. Fallo en las unidades de almacenamiento externo.** Un bloque de disco pierde su contenido debido a la rotura de la cabeza o al deterioro del propio soporte magnético.

Las principales consecuencias que pueden derivarse de estos fallos son las siguientes:

- Una caída del sistema puede interrumpir la ejecución de una transacción dejando los datos en un estado inconsistente (véase el Apartado 4.2.1). Pensemos, por ejemplo, que un determinado usuario está ejecutando una transferencia bancaria de 1000 euros entre las cuentas 12000345 y 120000897 con las siguientes sentencias SQL:

```
begin transaction  
    UPDATE cuentas SET saldo=saldo-1000  
        WHERE num_cuenta=12000345;
```

***** *fallo del sistema* *****
UPDATE cuentas SET saldo=saldo+1000
WHERE num_cuenta=12000897;

```
end transaction
```

y que se produce una caída del sistema tras la ejecución de la primera sentencia... ¿Cuáles serían las consecuencias? Pues se habrán cargado 1000 euros a la primera cuenta pero no se habrá realizado el ingreso en la segunda, esto es, *han desaparecido 1000 euros*. Piense en la magnitud del problema si cuando se produce el fallo hay cientos de transacciones a medio ejecutar.

Para evitar este problema, se establece el concepto de **transacción** como una colección de operaciones teóricamente inseparables que realizan una única función lógica. Las transacciones deben ser atómicas en el sentido de que, o se ejecutan *todas* sus sentencias o no se ejecuta *ninguna*, con el fin de no producir situaciones inconsistentes en el sistema. Para ello, es necesario que los SGBD vengan provistos de mecanismos de recuperación responsables de restaurar, si es posible, la información perdida y de dejar la BD en un estado consistente (el que tuviera antes del fallo).

- La rotura de una unidad de almacenamiento y, por tanto, la pérdida de su contenido. Este problema no tiene solución, pero sí puede prevenirse haciendo

copias periódicas de seguridad en distintos medios de almacenamiento. En la Sección 4.3 se comentan los distintos mecanismos de copia de seguridad, sus ventajas e inconvenientes y cuándo y cómo conviene usarlos.

4.2.1. TRANSACCIONES

Como acabamos de ver, una transacción es una unidad lógica de procesamiento que puede estar constituida por varias sentencias. Es, en definitiva, una unidad de programa que consulta y actualiza datos sin violar ninguna de las restricciones de consistencia de la BD. Es decir, si la BD era consistente antes de ejecutarse la transacción, debe seguir siéndolo tras su ejecución.

Toda transacción debe cumplir los siguientes requisitos:

- *Atomicidad*: Todas las operaciones asociadas a una transacción deben ejecutarse por completo o no ejecutarse ninguna de ellas.
- *Consistencia*: La ejecución de la transacción no podrá violar ninguna restricción de integridad.
- *Aislamiento*: Aunque se ejecuten varias transacciones concurrentemente, la ejecución individual de cada una de ellas no debe interferir en la ejecución de las otras.
- *Persistencia*: Tras la ejecución con éxito (*commit*) de una transacción, los cambios realizados sobre los datos deben permanecer, incluso si se produce un fallo después.

A estas cuatro conocidas propiedades de las transacciones se las denomina en su conjunto propiedades ACID (Atomicity, Consistency, Isolation y Durability) [3], [5], [20].

El programador de la aplicación en la que se inscribe la transacción es el responsable de asegurar que la transacción sea consistente; sin embargo, el asegurar su atomicidad es responsabilidad del SGBD, en concreto del gestor de transacciones. Cuando una transacción no termina con éxito (*aborts*) no debe verse afectado el estado de la BD y, por tanto, ésta deberá recuperar el estado anterior a la ejecución de dicha transacción.

Una transacción puede encontrarse en uno de los siguientes estados:

- *Activa*, mientras se ejecuta con normalidad.
- *Parcialmente ejecutada*, tras la ejecución de la última sentencia.
- *Parcialmente abortada*, cuando se ha descubierto que no se puede seguir ejecutando por causa de algún error o sentencia incorrecta.
- *Ejecutada*, cuando finaliza completamente con éxito.
- *Abortada*, cuando tras un fallo se restaura la BD al estado anterior a la ejecución de la misma.

Una transacción empieza en el estado *activa*. Cuando llega a su última sentencia entra en el estado *parcialmente ejecutada* y en ese momento, aunque ya se han ejecutado todas sus sentencias, todavía es posible que tenga que abortarse, ya que puede que las modificaciones realizadas no se hayan escrito aun en el disco y se produzca un fallo hardware que impida su terminación con éxito. Debe tenerse especial cuidado con transacciones que contengan salidas de datos por pantalla o por impresora donde un fallo no permite el *borrado*.

En la mayoría de los sistemas lo que se hace es almacenar en disco temporalmente todos los datos que vayan a ser escritos en dispositivos externos y llevar a cabo las escrituras reales solo cuando la transacción haya sido ejecutada.

Una transacción entra en el estado *parcialmente abortada* cuando no se puede proseguir con su ejecución normal. En este caso, la transacción debe *abortarse*. Posteriormente, el sistema puede optar o bien por reiniciarla (siempre que el error producido no tenga que ver con las sentencias que contiene), o bien por eliminarla. Esta última decisión se toma cuando la transacción contiene algún error interno que solo puede corregirse volviendo a escribirla correctamente. En la Figura 4.5 se muestran las posibles transiciones entre estados que pueden producirse durante la ejecución de una transacción.

4.2.2. GESTIÓN DE BLOQUES Y BUFFERS

Como ya hemos comentado, la naturaleza de los fallos que pueden presentarse en un sistema es muy diversa y cada fallo necesita ser tratado de forma específica.

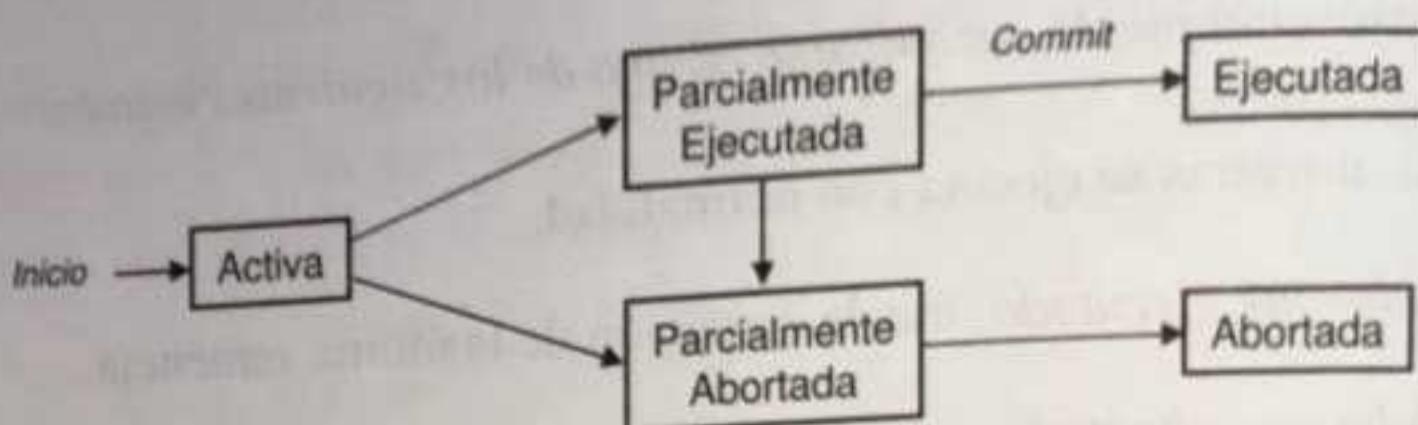


FIGURA 4.5 Transiciones y estados posibles de una transacción.

Los fallos más difíciles de subsanar son aquellos que han producido una pérdida de información. Para poder recuperar el sistema, hay que tener información precisa de todos los posibles modos de fallo de cada uno de los dispositivos de almacenamiento y considerar cómo afectan estos fallos al contenido de la BD (por ejemplo, si se rompe una controladora no tiene por qué producirse una pérdida de información).

◦ Con respecto a la posibilidad de que surja algún problema, suelen tomarse las siguientes medidas:

- Asegurar que se mantiene suficiente información en el sistema para permitir una recuperación en el caso de producirse un fallo.
- Tras producirse el fallo, llevar a cabo una serie de acciones que aseguren la consistencia de la BD y la atomicidad de las transacciones: mecanismos de recuperación de transacciones.

Los distintos dispositivos de almacenamiento *reaccionan* de modo diferente a los posibles fallos, como vemos a continuación.

- **Memoria volátil.** Nos referimos aquí a la memoria principal y a la memoria caché. Estas memorias son las más rápidas porque sus datos son directamente accesibles desde el procesador. Como contrapartida, la información que reside en este tipo de memorias se pierde cuando se produce una caída del sistema.
- **Memoria no volátil.** Nos referimos aquí a los discos y cintas magnéticas. Los discos se suelen utilizar para el almacenamiento continuo de datos (por la facilidad y la rapidez en el acceso) mientras que las cintas se utilizan para realizar copias de seguridad (son más económicas y ocupan poco espacio). Este tipo

de memoria tampoco asegura la permanencia de la información tras un fallo, si bien es mucho más *duradera* que la memoria volátil. Un ejemplo de fallo sobre estos dispositivos puede ser la rotura de una cabeza de disco, que con alta probabilidad producirá la pérdida de información.

Toda la información de la BD reside en almacenamiento no volátil y la ejecución de transacciones produce múltiples transferencias de datos entre la memoria principal y el disco.

Los distintos casos que se pueden dar cuando se transfiere un bloque de datos entre la memoria y el disco son:

- *Transferencia realizada con éxito:* La información transferida llegó completa y correcta a su destino.
- *Fallo parcial:* Se produjo un fallo durante la transferencia y el bloque de disco donde se realizó la escritura contiene información incorrecta.
- *Fallo total:* Se produjo un fallo antes de iniciarse la transferencia, de manera que el bloque destino permanece intacto.

Si ocurre un fallo durante la transferencia de datos, el sistema debe detectarlo inmediatamente y utilizar procedimientos que dejen el bloque destino en un estado consistente. Para hacerlo, una de las formas es mantener dos bloques físicos por cada bloque lógico de la BD, de manera que la información se escribe primero en uno de los bloques físicos; si esta escritura se ha llevado a cabo con éxito, se escribe la misma información en el otro bloque y la transferencia se dará por válida después de terminar con éxito esta segunda escritura. Este procedimiento garantiza que todas las escrituras, o bien acaben con éxito, o bien no produzcan cambio alguno. Durante la recuperación de datos se examina cada par de bloques físicos; si los dos son iguales y no existen errores detectables, no será necesario tomar medidas. Si durante la lectura de uno de los bloques se produce un fallo, entonces se sustituirá su contenido por el valor del otro bloque.

Este procedimiento puede extenderse para permitir el uso de un número arbitrario de copias de cada bloque. Aunque un número elevado de copias reduce mucho la probabilidad de un fallo, también es cierto que consume mucho tiempo y recursos.

La información que reside en los discos está organizada en *bloques físicos* mientras que la que reside en memoria principal se aloja en una zona de la misma denominada *buffer*. Supondremos que ningún dato ocupa más de un bloque, lo cual es bastante realista para la mayor parte de las aplicaciones.

Las transferencias de bloques entre memoria principal y disco se llevan a cabo por medio las siguientes dos operaciones:

- *lee_bloque(X)*, que transfiere el bloque físico que contiene el dato X al buffer de la memoria principal, y
- *escribe_bloque(X)*, que transfiere el buffer en el que reside el dato X (recuerde que pueden tenerse varios buffer abiertos simultáneamente) al disco, sustituyendo al bloque físico antiguo.

Por otro lado, las transacciones que se ejecutan sobre el sistema transfieren datos desde las variables de los programas a la BD y desde la BD a las variables de los programas. Estas transferencias se llevan a cabo mediante las operaciones:

- *lee(X, x_i)*, que asigna el contenido de X a la variable local x_i . Si el bloque en el que reside el dato X no está en memoria principal, entonces habrá de ejecutarse previamente *lee_bloque(X)*. Finalmente, se asigna el valor de X (que ahora se encuentra en el buffer) a la variable x_i .
- *escribe(X, x_i)*, que asigna a X el valor de la variable local x_i . Al igual que en el caso anterior, si el bloque que contiene el dato X no está en memoria, habrá de ejecutarse *lee_bloque(X)* y posteriormente asignar a X el valor de x_i .

En la Figura 4.6 pueden verse de forma esquemática estas cuatro operaciones.

El buffer no se graba en disco a menos que sea necesario, esto es, cuando el gestor de memoria necesite el espacio que ocupa, por una nueva lectura de bloque o porque el SGBD deba reflejar de inmediato los cambios realizados. Cuando una transacción necesita acceder a un elemento X por primera vez, el sistema deberá ejecutar *lee_bloque(X)*. Tras esta recuperación, se llevarán a cabo todas las operaciones sobre x_i descritas en la transacción. La operación *escribe_bloque(X)* no tiene por qué ejecutarse inmediatamente después, ya que el bloque que está actualmente en el

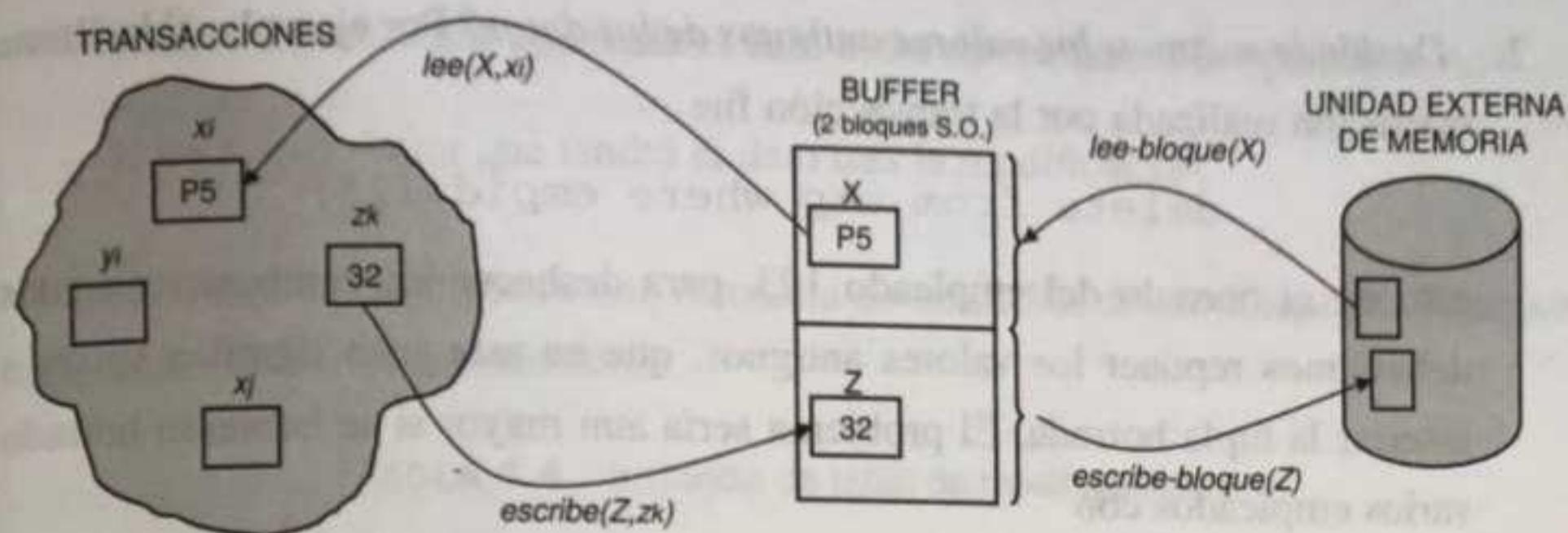


FIGURA 4.6 Transferencias entre la memoria de las transacciones, los buffers de datos y el disco.

buffer y que contiene a X, puede contener otros datos necesarios para la siguiente transacción. Nótese que si en este momento se cayera el sistema, el nuevo valor que se ha asignado a X no se grabaría en el disco y, por tanto, se perdería.

4.2.3. RECUPERACIÓN DE TRANSACCIONES

Si seguimos al pie de la letra un esquema de transferencia de datos como el que se muestra en la Figura 4.6, las transacciones irían colocando en el buffer correspondiente los datos ya modificados conforme éstos se fueran produciendo, a la espera de que dicho buffer sea grabado en disco próximamente. Si durante la ejecución de una transacción (como, por ejemplo, la transacción bancaria de el Apartado 4.2) el sistema cayera dejando parte de los datos modificados y parte sin modificar, al recuperarse, éste debería deshacer los cambios realizados y dejarlo todo tal y como se encontraba antes de que la transacción comenzara su ejecución, garantizando así su atomicidad. Esta vuelta atrás supondría volver a localizar los datos que ya fueron modificados y asignarles los valores antiguos, lo que generaría dos importantes problemas:

1. *La relocalización de los datos ya modificados* por parte de transacciones no acabadas, supone un gran número de operaciones de E/S, que son las más costosas, lo que haría que la recuperación del sistema tras una caída consumiera un tiempo inaceptable.

2. ¿De dónde sacamos los valores antiguos de los datos? Por ejemplo, si la última operación realizada por la transacción fue

```
delete from emp where empid=123;
```

esto es, el borrado del empleado 123, para deshacer los cambios realizados deberíamos reponer los valores antiguos, que en este caso significa volver a insertar la tupla borrada. El problema sería aun mayor si se hubiesen borrado varios empleados con

```
delete from emp where ciudad='Granada';
```

4.2.3.1. La Tabla de Modificaciones

Para solucionar los dos problemas planteados en el apartado anterior, los SGBD utilizan una tabla en memoria donde se va guardando información precisa de todas las modificaciones (log) que las transacciones pretenden llevar a cabo sobre la BD, con el fin de poder recuperar, de forma eficiente, un estado anterior en caso de que se produzca un fallo.

La información que se guarda es solo la referente a operaciones que producen modificaciones en los datos, esto es, las que involucran una escritura en disco (insertar, actualizar y borrar). La tabla donde se registra dicha información se denomina **tabla de modificaciones** (TM) o **bitácora**, cuyos registros contienen, como mínimo, los siguientes campos:

- *Código de transacción*. Cada transacción está identificada por un código interno único que normalmente se asigna usando el reloj del sistema.
- *Estado actual* en que se encuentra la transacción (start, commit o abort).
- *Operación* que se está llevando a cabo por la transacción en ese instante. Las únicas operaciones de interés para la tabla de modificaciones son las que alteran el contenido de la BD, esto es, aquellas que provocan una operación de E/S.
- *Marca de tiempo*: opcionalmente puede aparecer algún campo temporal que indica en qué instante se modifica el estado de una transacción.
- *Nombre del dato*. Se refiere al identificador del atributo (el nombre del atributo varía...)

- **Valor antiguo.** Valor que tenía el dato antes de realizarse la operación.
- **Valor nuevo.** Valor que tendrá el dato tras la modificación.

En la Tabla 4.4 se muestra una instancia de una tabla de modificaciones durante la ejecución de varias transacciones.

TABLA 4.4 Instancia de tabla de modificaciones.

T_id	Hora	Estado	Operación	Dato	V _A	V _N
T ₁	9:05	start				
T ₂	9:06	start				
T ₁	9:08		update	Salario	145.000	170.000
T ₃	9:10	start				
T ₂	9:15		update	Nombre	Perea	Perez
T ₃	9:17		insert	Empleado		(Abad...)
T ₂	9:18	commit				
T ₁	9:19		delete	Empleado	Nombre=Ruiz	
T ₃	9:20	commit				

Algunos sistemas incluyen también otra información de interés sobre las transacciones, como puede ser: el usuario que la ha ejecutado, identificador de la sesión, dirección física de los bloques modificados...

El protocolo que el sistema ha de seguir para usar con eficacia la información registrada en la tabla de modificaciones es el siguiente:

- Cuando una transacción T_i comienza su ejecución, se anota en la tabla de modificaciones la información $\langle T_i \text{ start} \rangle$, indicando que dicha transacción acaba de comenzar.
- Durante su ejecución, cualquier operación de modificación que se encuentre irá precedida de la escritura del registro correspondiente.
- Cuando T_i finaliza (se ha ejecutado su última sentencia), se escribe el registro $\langle T_i \text{ commit} \rangle$ en la TM, indicando que su ejecución ha finalizado con éxito.

Para ilustrar cómo se registra la información en la TM conforme se ejecutan simultáneamente varias transacciones, consideraremos el siguiente ejemplo de ejecución.

Ejemplo 4.1

- Sean tres transacciones T_1 , T_2 y T_3 que se llevan a cabo de forma concurrente y que actúan sobre diversos átomos. En la parte izquierda de la tabla, se muestra el código de cada una de ellas y se han tomado como valores iniciales para las variables: $A = 2, B = 16, W = 5, X = 10, Y = 20$.

T_1	T_2	T_3	T_i	Estado	Op.	D	V_A	V_N
lee(A,r)			T_1	start				
$r := r^*2$	lee(W,n)		T_2	start				
escribe(A,r)	lee(X,t)		T_1		update	A	2	4
			T_1	commit				
	$t := t + n$	lee(A,s)	T_3	start				
		escribe(X,t)	T_2		update	X	10	15
	lee(Y,t)	$s := s^*5$	T_3					
	$t := t - n$	escribe(A,s)	T_3		update	A	4	20
		lee(B,s)	T_2	commit				
		$s := s^*5$						
		escribe(B,s)	T_3		update	B	16	90
			T_2		update	Y	20	15
		escribe(Y,t)	T_3	commit				

En la parte derecha de la tabla se muestran los registros log que se almacenarían durante la ejecución de las transacciones.

4.2.3.2. Modificación retardada de la BD

Dado que la información de esta tabla se utiliza para reconstruir la BD en caso de fallo, será necesario grabarla en almacenamiento estable con cierta frecuencia, como mínimo tras cada inserción de un registro $< T_i \text{ commit} >$. Esto nos garantiza que, antes de que se hagan efectivas las modificaciones indicadas en las transacciones, habrá una copia almacenada de los cambios a realizar y las operaciones $\text{escribe}(X, x_i)$.

y *escribe_bloque(X)* se llevarán a cabo de forma segura garantizando la persistencia de los datos.

A partir del momento en que se escribe $\langle T_i \text{ commit} \rangle$ en la tabla de modificaciones (y ésta se almacena en disco), comienza la **ejecución real** de la transacción, esto es, se empiezan a transferir los datos modificados al buffer y, de éste, al disco. Esta etapa de ejecución real se va haciendo al ritmo que impone el propio sistema y se denomina **modificación retardada de la BD**.

Esta técnica garantiza la atomicidad de las transacciones, ya que consiste en grabar todos los cambios a realizar en la tabla de modificaciones, pero aplazando su ejecución hasta que la transacción esté parcialmente ejecutada (tras la última sentencia). De esta forma, si se produce una caída del sistema durante la ejecución de varias transacciones, cuando el sistema se recupere no tendrá más que mirar la última copia almacenada de la tabla de modificaciones y deshacer las modificaciones para aquellas transacciones que quedaron a medias (no tienen $\langle T_i \text{ commit} \rangle$). Como dichas transacciones *nunca* llegaron a escribir nada en la BD, deshacerlas consistirá simplemente en borrar sus registros de la tabla de modificaciones, quedando ignoradas definitivamente.

Pero ¿qué ha pasado con las transacciones que tienen $\langle T_i \text{ commit} \rangle$? Para estas transacciones no se tiene constancia en la tabla de modificaciones de si sus cambios fueron o no guardados físicamente en los ficheros por lo que, para garantizar su terminación con éxito, el sistema vuelve a actualizar los datos aun cuando algunos o muchos de ellos ya se encuentren modificados, esto es, se *rehace* la transacción.

Por tanto, el esquema de recuperación del sistema tras un fallo se basa en la utilización de dos procedimientos, que son:

- UNDO(T_i) (deshacer), anula o borra todos los registros relativos a T_i de la tabla de modificaciones.
- REDO(T_i) (rehacer), que asigna los nuevos valores a todos los datos actualizados por T_i .

En resumen, después de ocurrir un fallo, el sistema debe determinar para cada transacción si ésta debe deshacerse o, por el contrario, debe volver a ejecutarse. Esta decisión se toma en base al siguiente criterio:

- UNDO(T_i): La transacción T_i debe deshacerse si la tabla de modificaciones contiene la información $\langle T_i \text{ start} \rangle$ pero no contiene la información $\langle T_i \text{ commit} \rangle$.
- REDO(T_i): La transacción T_i debe volver a ejecutarse si la tabla de modificaciones contiene ambos registros, $\langle T_i \text{ start} \rangle$ y $\langle T_i \text{ commit} \rangle$.

4.2.3.3. Puntos de verificación

Cuando ocurre un fallo en el sistema es necesario consultar la tabla de modificaciones para determinar cuáles son las transacciones que deben volver a ejecutarse y cuáles deben deshacerse. En principio, es necesario revisar toda la tabla para poder determinarlo, ya que depende del último estado en que quedara la transacción antes de producirse el fallo.

Esta consulta plantea dos inconvenientes:

- El proceso de búsqueda en la tabla de modificaciones puede llegar a ser muy lento si había registradas muchas transacciones.
- La mayor parte de las transacciones que necesitan volver a ejecutarse, ya habrán escrito sus actualizaciones en la BD, por lo que parece absurdo volver a hacerlo si bien no causaría ningún daño.

Para reducir el tiempo dedicado a estas tareas, existen los llamados *puntos de verificación* o *checkpoint*, cuya utilidad veremos a continuación.

Periódicamente, el sistema lleva a cabo la siguiente secuencia de operaciones:

1. Graba en disco todos los registros de la tabla de modificaciones que actualmente residen en memoria principal.
2. Graba en los ficheros de la BD (datafiles) los bloques que contienen los datos ya modificados.
3. Incluye en la tabla de modificaciones un registro de estado $\langle \text{checkpoint} \rangle$. Este registro informa al sistema de que, en ese punto, se produjo la grabación de los datos modificados hasta el momento por transacciones ya acabadas (han realizado *commit*).

4. Graba la tabla de modificaciones en disco para que la copia almacenada de la misma contenga el <checkpoint>.

Gracias a esta mejora, después de un fallo solo será necesario volver a ejecutar las transacciones que terminaron después del último punto de verificación. Este proceso consiste en buscar hacia atrás en la TM hasta encontrar un registro <checkpoint> y, a partir de ahí, buscar hacia adelante el siguiente registro < T_i commit>.

Como ejemplo, considérese la transacción T_1 que se muestra en la Tabla 4.5 y que tiene commit antes de un punto de verificación.

TABLA 4.5 Tabla de modificaciones con checkpoint.

T_id	Estado	Op.	Dato	V. anterior	V. nuevo
T_1	start				
T_2	start				
T_1		update	Salario	145.000	170.000
T_3	start				
T_1		insert	Empleado		(Abad,125.000)
T_1	commit				
	checkpoint				
T_3	commit				
T_2		update	Nombre	Perea	Perez

Cualquier modificación realizada por T_1 sobre la BD, o bien se llevó a cabo en su momento, o bien se habrá realizado dentro de las tareas propias del punto verificación. Por tanto, si se produce un fallo, no será necesario realizar la operación REDO para T_1 .

En general, el criterio a seguir será:

- Ejecutar REDO para aquellas transacciones que tienen commit tras un registro <checkpoint> (T_3), y
- Ejecutar UNDO para aquellas transacciones que no tienen commit (T_2).

4.2.4. GESTIÓN DE TRANSACCIONES EN ORACLE

El SGBD Oracle utiliza el mecanismo de tabla de modificaciones, a la que se denomina **redo log buffer** y que se almacena físicamente en el **redo log file**. Oracle utiliza, como mínimo, dos *redo log buffer* para que uno de ellos pueda seguir registrando transacciones mientras el otro está siendo grabado en disco.

Para Oracle, una transacción es una secuencia de sentencias SQL que es tratada de forma atómica. Hasta que una transacción no ha sido ejecutada, ninguno de sus cambios se hace visible al resto de los usuarios.

Las sentencias SQL que permiten a un usuario confirmar o abortar sus transacciones son **COMMIT** y **ROLLBACK**, respectivamente, y serán discutidas ampliamente más adelante².

Oracle considera que una transacción comienza cuando se encuentra la primera sentencia DML (*insert, update, delete...*) de SQL. Sin embargo, la da por finalizada cuando se da una de las siguientes situaciones:

- Se hace una llamada a los procedimientos **COMMIT** o **ROLLBACK**.
- Se hace una llamada a un comando del DDL (**CREATE, DROP, RENAME...**). En este caso la transacción anterior es la que se da por finalizada.
- El usuario se desconecta de Oracle. En este caso, la última transacción se confirma.
- Se produce una terminación anormal de un proceso. En este caso, la transacción actual se aborta y se restaura la BD.

En el primero de los casos, estas operaciones se hacen de forma explícita, pero en los casos restantes, es el sistema el que automáticamente confirma o aborta la transacción en curso.

La descripción de las sentencias SQL más importantes para el control de transacciones se detalla a continuación.

²Para más información véase [16].

- COMMIT. Esta sentencia lleva a cabo las siguientes tareas:

- Hace permanentes todos los cambios que se han llevado a cabo durante la transacción actual, esto es, graba en disco el contenido actual del buffer de redo log (TM).
- Da por finalizada la transacción.
- Libera los recursos acaparados por la transacción.

- ROLLBACK. Esta sentencia se utiliza para deshacer las operaciones llevadas a cabo hasta el momento por una transacción, siempre que ésta no haya sido confirmada previamente.

Esta sentencia lleva a cabo las siguientes tareas:

- Aborta la transacción en curso.
 - Deshace todos los cambios registrados por la transacción abortada.
 - Libera los recursos acaparados por la transacción.
- SAVEPOINT <savepoint_id>. Esta sentencia marca ciertos puntos de una transacción en los que se guarda el estado de los datos. Cada marca tiene un identificador único, de manera que pueda ser posteriormente referenciado con el fin de restaurar la BD a ese instante; es decir, ofrece la posibilidad de volver atrás a dicho estado cuando se ejecute un ROLLBACK, en lugar de anular totalmente la transacción. En este caso, la sintaxis de la sentencia ROLLBACK adopta la forma:

ROLLBACK [TO] <savepoint_id>;

Un ejemplo del uso de estas sentencias durante una transacción interactiva es el siguiente:

```
/* Cambiar el nombre al departamento 'Ventas' por el de  
'Ventas Extranjero' en la tabla empleados */  
  
UPDATE empleados SET dpto='Ventas Extranjero'  
WHERE dpto='Ventas';  
  
SAVEPOINT venta_mayor;
```

```

/* Actualizar el valor del depto. a 'Ventas Nacionales'
   para las tuplas en las que el jefe es 'Ruiz' */
```

```

UPDATE empleados SET dpto='Ventas Nacionales'
   WHERE jefe='Ruiz';
SAVEPOINT venta_exterior;
```

```

/* Actualizar el valor del jefe a 'Ruiz' donde el jefe
   es 'Perez'. La modificacion hecha anteriormente al
   departamento no se veria actualizada para las
   nuevas tuplas de 'Ruiz', por lo que volvemos atras.*/
```

```

UPDATE empleados SET jefe='Ruiz'
   WHERE jefe='Perez';
ROLLBACK venta_mayor;
```

```

/* Se ejecutan de nuevo las dos ultimas sentencias en
   el orden correcto y se graban las modificaciones. */
```

```

UPDATE empleados SET jefe='Ruiz'
   WHERE jefe='Perez';
UPDATE empleados SET dpto='Ventas Nacionales'
   WHERE jefe='Ruiz';
COMMIT;

```

4.3. SALVADO Y RECUPERACIÓN DE BD

Hemos visto que el uso de la tabla de modificaciones es un mecanismo que permite garantizar tanto la atomicidad como la persistencia. Sin embargo, la persistencia de los datos puede todavía verse seriamente comprometida si se produce un fallo en el dispositivo físico que los almacena.

Por este motivo, hacer copias del contenido de la BD cada cierto intervalo de tiempo debe ser una práctica obligada para garantizar que existirá al menos una versión reciente si ésta se daña o se destruye. Una rotura de disco, el borrado accidental de datos o cambios en los ficheros de configuración pueden ser catastróficos. Si se ha

previsto un protocolo de salvaguarda de la BD, ésta se podrá reconstruir cargando la copia de seguridad y rehaciendo todas las transacciones que se procesaron después de la realización de dicha copia. La copia de seguridad de la BD suele llevarla a cabo el administrador del sistema apoyado por herramientas que le facilitan esta tarea.

◇ *La mayoría de los sistemas de BD permiten dos tipos de copia de seguridad:*

1. **Copia Física.** Es la forma más común. Se suelen usar las utilidades y los comandos que ofrece el propio S.O. (backup, dump, tar, cpio, copy...). Este tipo de copia tiene la ventaja de que la mayoría de los S.O. permiten manejar directamente dispositivos de almacenamiento masivo (cintas, discos, RAIDs, etc.). Este tipo de copia no se diferencia, en lo esencial, de la copia de seguridad realizada para cualquier tipo de directorio o grupo de archivos independientes y, por tanto, garantiza compatibilidad con los dispositivos físicos instalados, ya que fueron configurados especialmente para el S.O. que los soporta. Se aconseja dedicar estos dispositivos en exclusiva al SGBD y garantizar espacio de sobra.

La copia realizada debe ser *completa y consistente*, esto es, debe realizarse una copia de todos los ficheros de datos, de control, de instalación, de configuración... cuando los usuarios no se encuentren conectados lanzando transacciones (¿qué sucedería si los usuarios siguieran ejecutando transacciones mientras se realiza la copia?). La mejor forma de garantizar que la copia será perfecta es haciendo una copia *off-line* (la BD no está levantada), cuando sea posible. En la Figura 4.7 se esquematiza este tipo de copia de seguridad.

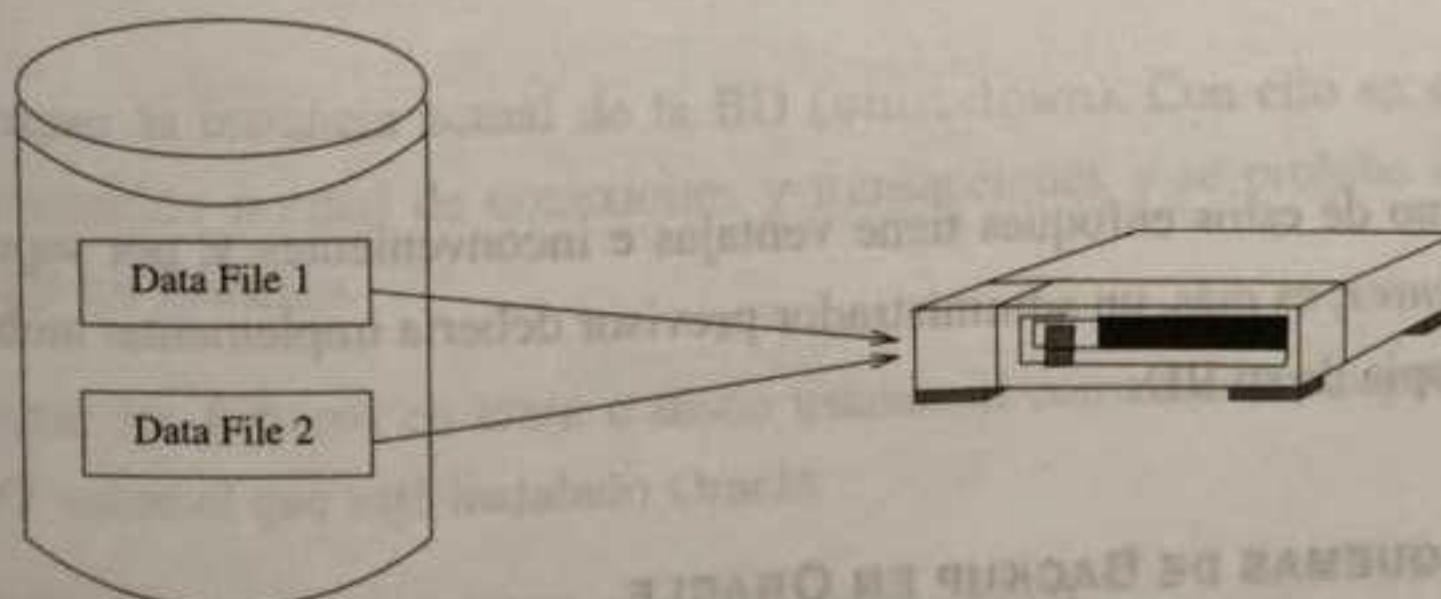


FIGURA 4.7 Copia de seguridad de ficheros de datos completos.

2. **Copia Lógica.** Este tipo de copia de seguridad se hace desde el mismo SGBD mediante el uso de utilidades propias. Se trata de salvar parcial o totalmente los objetos de la BD, pero sin almacenar su estructura interna, ni las tablas de modificaciones, ni archivos de configuración... La idea de este tipo de copia es poder guardar tablas concretas o exportarlas a otras BD compatibles, ya que resulta muy tedioso recuperar una sola tabla u objeto de un backup completo de S.O.

En Oracle y sistemas similares, las exportaciones objeto a objeto se llevan a cabo gracias a la utilidad EXPORT/IMPORT, que permite, además, realizar el volcado de tablas de otros puntos de la red que no tengan dispositivos adecuados. En la Figura 4.8 se muestra esquemáticamente este tipo de copia de seguridad.

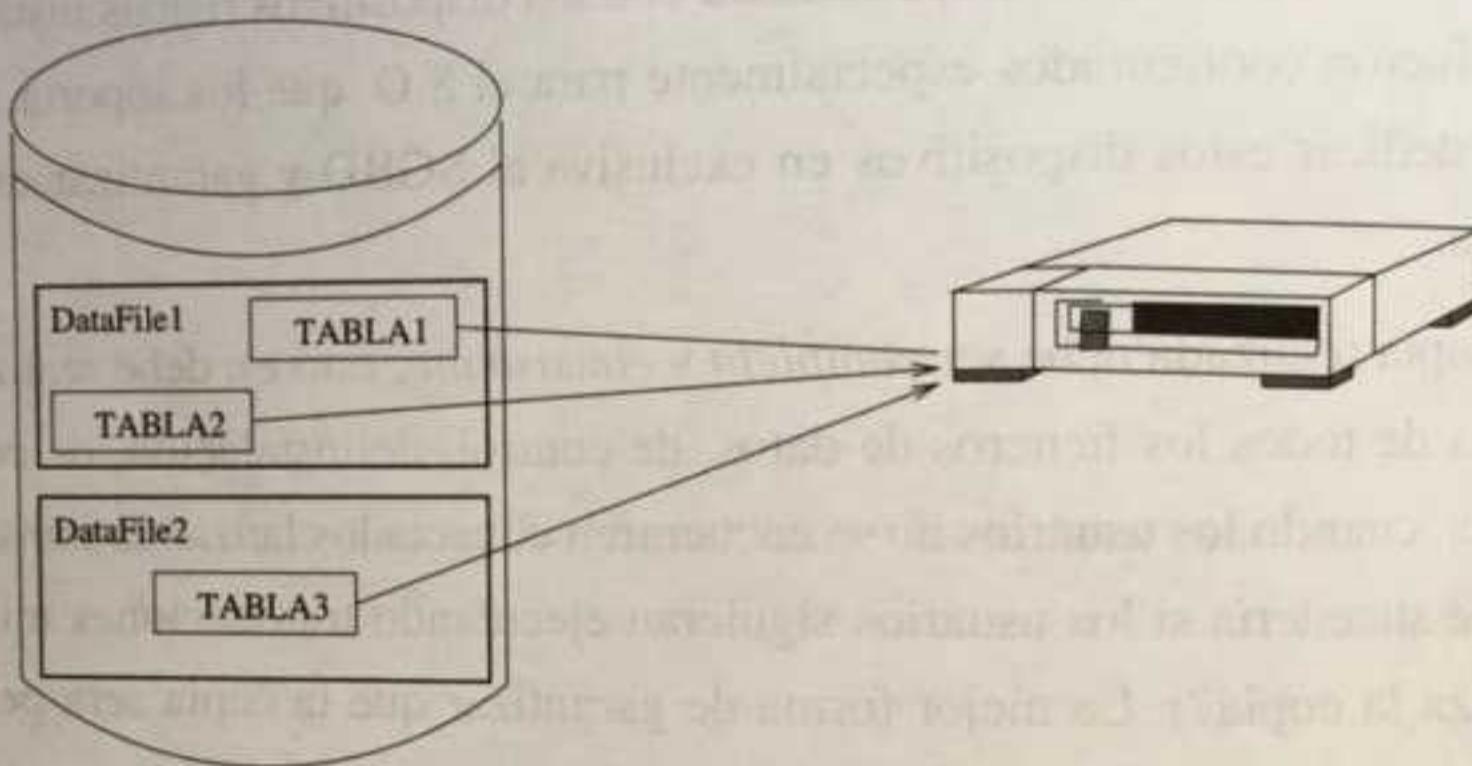


FIGURA 4.8 Copia de seguridad de tablas seleccionadas.

Cada uno de estos enfoques tiene ventajas e inconvenientes y, por supuesto, *no son excluyentes*; es más, un administrador previsor debería implementar ambas estrategias de copia de su BD.

4.3.1. ESQUEMAS DE BACKUP EN ORACLE

En este apartado haremos un breve resumen de las diferentes formas en que puede hacerse una copia de seguridad de una BD Oracle e introduciremos algunos de los

comandos SQL necesarios para su ejecución. Si se desea profundizar más en algunos de los mecanismos en particular, véase [6].

4.3.1.1. Copia Física

Oracle permite dos formas de llevar a cabo el volcado físico de los datos, que se conocen como salvado en frío (off-line) y salvado en caliente (on-line), que pasamos a describir a continuación.

Salvado en frío

Como ya se comentó en la sección anterior, una copia física captura los datos y la estructura global de la BD desde el S.O., esto es, a bajo nivel, por lo que durante la ejecución de la copia, los ficheros que se están transfiriendo no podrán ser accedidos bajo ningún concepto por usuarios y/o aplicaciones, garantizándose así que la copia será consistente.

Debido a la gran cantidad de información de control (tamaño del bloque, cabezas, marcas, apuntadores...) que contienen los ficheros transferidos, una copia de este tipo es muy sensible a la versión Oracle y configuración utilizadas. Por este motivo, los datos se tendrán que recuperar en una instalación exacta a la que generó los datos copiados. Durante la ejecución de este tipo de copia, la base de datos deberá estar detenida.

Los pasos a seguir para llevar a cabo una copia de seguridad en frío son³:

1. Detener la instancia actual de la BD (shutdown). Con ello se garantiza la terminación normal de conexiones y transacciones y se prohíbe el acceso a todos los usuarios.
2. Copiar los ficheros en cinta o disco usando el comando correspondiente del S.O. sobre el que esté instalado Oracle.
3. Iniciar la instancia de la BD de nuevo (startup).

³En general, se aconseja el empleo de utilidades específicas.

Es obvio que, durante el tiempo que dure la copia de ficheros, el sistema ha estado completamente inaccesible para los usuarios, lo cual es un grave inconveniente para organizaciones o empresas de funcionamiento continuo que no se pueden permitir paralizar las aplicaciones.

Los ficheros que obligatoriamente deben salvase son:

- Ficheros de datos de todos los tablespaces (*datafiles*). Estos ficheros tienen, como ya sabemos, la extensión .dbf.
- El fichero de control (*control file*). Este fichero lleva la extensión .ctl y suele haber más de una copia por razones de seguridad, ya que contiene información actual de la estructura, la versión, la organización y el estado de la BD (cuántos tablespaces hay, qué ficheros de datos tienen asociados, qué relaciones se almacenan en cada fichero, fechas de volcado, información de sincronización, etc.), por lo que se trata de un fichero vital para el funcionamiento correcto del sistema.
- Las tablas de modificaciones almacenadas (*redo log files*). Estos ficheros suelen llevar la extensión .log. Es obvio que con la información de los datafiles no se tiene la última versión de la BD, ya que habrá transacciones ya finalizadas que aun no habrán transferido sus modificaciones a los mismos.

Aunque con la información anterior ya se cuenta con una versión completa y consistente de la BD, es recomendable tener también una copia relativamente reciente de los siguientes ficheros:

- Los ficheros init.ora y config.ora que almacenan la información específica de la instancia de BD, datos de arranque y configuración.
- Software de aplicaciones de Oracle cuando éste se ha actualizado o se usan parches.

La ejecución de una copia de seguridad en frío puede hacerse mediante la siguiente secuencia de comandos:

```
# Detener de forma inmediata la BD y salirnos  
SQL> shutdown immediate;  
SQL> exit;  
  
# Ejecutar la copia desde S.O.  
C:\> copy c:\oracle\oradata\*.* D:\backup  
  
# Levantar la BD  
SQL> startup;
```

Es recomendable disponer de unos *scripts* que automaticen el proceso de copia de modo que no se deje de salvar algún fichero por olvido.

Recuperación a partir de una copia en frío

La última copia completa realizada será utilizada en caso de que, o bien la unidad en la que se encuentre actualmente la BD o bien alguno de los datafiles, tenga un fallo irrecuperable y, por tanto, no permita el acceso a los datos que contiene. El procedimiento de recuperación consiste en sustituir todos los ficheros actuales, dañados o no, por los que se encuentran en la copia de seguridad y abrir la BD. Oracle detecta automáticamente la ausencia de algún fichero de datos y lo comunica mediante un mensaje de error la primera vez que iniciemos la instancia tras la sustitución. Es obvio que se han perdido todas las modificaciones practicadas por las transacciones desde que se realizó la copia hasta el momento del fallo y las transacciones que produjeron dichas modificaciones tendrán que ser lanzadas de nuevo manualmente para poner la BD al día.

La recuperación tras un fallo hardware puede hacerse mediante la siguiente secuencia de comandos:

```
# Detener de forma inmediata la BD si esta hubiera #  
# logrado levantarse #  
SQL> shutdown immediate;  
SQL> exit;  
  
# Ejecutar la copia desde S.O.  
C:\> copy D:\backup\*.* C:\oracle\oradata
```

```
# Levantar la BD
SQL> startup;
```

Salvado en caliente u on-line

Este tipo de copia de seguridad⁴ tiene gran utilidad para grandes BD o BD de uso continuo, ya que permite realizar copias parciales sin necesidad de *detener* la BD.

En el momento en el que se inicia la copia, el sistema consulta la tabla de modificaciones para ver qué transacciones han finalizado ya con éxito (commit), con el fin de salvar también sus actualizaciones. A partir de ese instante, cualquier transacción que estuviera a medio ejecutar o recién iniciada, no será tenida en cuenta en la copia, pero ¿podrá proseguir con normalidad? La respuesta es que sí. La idea es ir grabando todas las modificaciones que se estén llevando a cabo por las transacciones en curso de la misma forma que hemos venido haciendo con la tabla de modificaciones salvo que, en este caso, al no poder ejecutarse de forma real las transacciones (los ficheros están siendo leídos para su volcado) no se borrarían los registros ya procesados y la tabla se nos desbordaría. La solución a este problema es ir grabando en un fichero diferente cada *llenado completo* de la tabla de modificaciones e ir numerándolos en secuencia, de manera que cuando la copia haya finalizado, el sistema los vaya recorriendo ordenadamente y ejecutando su contenido para poner al día la BD. Cuando un sistema va guardando en ficheros los sucesivos llenados de su tabla de modificaciones se dice que se encuentra en **modo archivado** o **archivelog mode** y éste debe estar habilitado forzosamente cuando se quiere realizar una copia de seguridad en caliente. En la Figura 4.9 se muestra cómo se almacena el buffer actual de redo log sin la opción *archivado* y con ella.

En el primer caso, el buffer se reutiliza constantemente, ya que los cambios producidos por las transacciones finalizadas se van volcando en los datafiles, mientras que en el segundo caso, han de grabarse las sucesivas *versiones* de la tabla de modificaciones en ficheros debido a que los datafiles están siendo copiados y, por tanto, innaccesibles por el momento.

⁴Solo disponible en la versión 7.

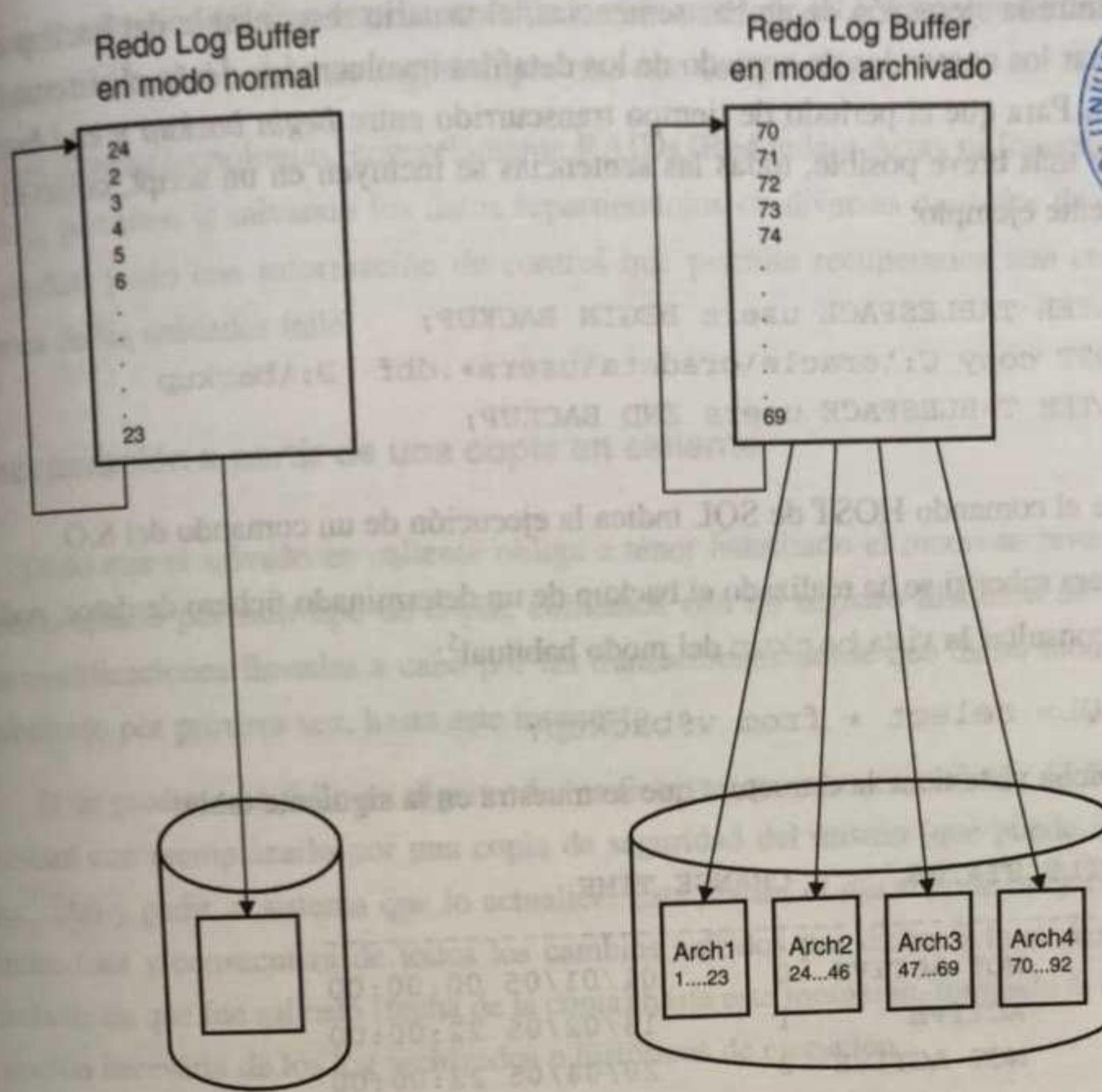


FIGURA 4.9 Volcado de la tabla de modificaciones sin y con modo archivado.

Los backups en caliente han de realizarse tablespace a tablespace por lo que, de manera simultánea, solo se encontrarán inhabilitados los datafiles pertenecientes al que se encuentre siendo copiado en cada instante.

Las sentencias SQL que permiten realizar backups on-line son las siguientes:

1. `ALTER TABLESPACE <tablespace-id> BEGIN BACKUP;`
que indica comienzo de backup on-line sobre un tablespace particular.
2. `ALTER TABLESPACE <tablespace-id> END BACKUP;`
que indica al sistema que la copia ha finalizado.

Entre la ejecución de ambas sentencias, el usuario responsable del backup debe ejecutar los comandos de copiado de los datafiles involucrados desde el sistema operativo. Para que el periodo de tiempo transcurrido entre *begin backup* y *end backup* sea lo más breve posible, todas las sentencias se incluyen en un script, como el del siguiente ejemplo:

```
ALTER TABLESPACE users BEGIN BACKUP;
HOST copy C:\oracle\oradata\users*.dbf D:\backup
ALTER TABLESPACE users END BACKUP;
```

donde el comando HOST de SQL indica la ejecución de un comando del S.O.

Para saber si se ha realizado el backup de un determinado fichero de datos, podemos consultar la vista backup del modo habitual⁵:

```
SQL> select * from v$backup;
```

Dicha vista tiene la estructura que se muestra en la siguiente tabla:

FILE	STATUS	CHANGE	TIME
1	NOT ACTIVE	0	01/01/05 00:00:00
2	ACTIVE	1	13/02/05 22:00:00
3	NOT ACTIVE	2	20/04/05 22:00:00

donde:

- FILE es el identificador del fichero de datos en cuestión.
- STATUS indica su estado actual. Puede tomar los valores ACTIVE (backup en ejecución) o NOT ACTIVE (backup terminado).
- CHANGE indica cuántos backups se han realizado hasta el momento.
- TIME indica fecha y hora de comienzo de la última copia de seguridad.

Lo normal es tener varios scripts para llevar a cabo volcados de distintas partes de la BD en diferentes días y horas. Adicionalmente, como norma general, deberán

⁵El prefijo v\$ lo llevan las vistas del catálogo que son dinámicas.

realizarse salvados off-line tras la creación o el borrado de tablespaces y de ficheros de datos, de control o de *redo log*, siempre que esto sea posible.

Las últimas tecnologías, concretamente RAIDs (Redundant Array of Independent Disks), permiten ir salvando los datos repartiéndolos en diversas unidades de disco separadas, junto con información de control que permite recuperarlos aun cuando alguna de las unidades falle.

Recuperación a partir de una copia en caliente

Dado que el salvado en caliente obliga a tener habilitado el modo archivado, si hemos optado por este tipo de copia, contamos con un registro histórico de todas las modificaciones llevadas a cabo por las transacciones desde que dicho modo fue habilitado por primera vez, hasta este momento.

Si se produce un fallo en alguno de los ficheros que almacena datos de la BD, bastará con reemplazarlo por una copia de seguridad del mismo (que puede no ser reciente) y pedir al sistema que lo actualice. Esta *puesta al día* supone la ejecución inmediata y consecutiva de todos los cambios sufridos por dicho fichero desde el instante en que fue salvado (fecha de la copia) hasta este momento, tomando la información necesaria de los log archivados o históricos de ejecución.

La sentencia SQL que permite *poner al día* ficheros obsoletos es:

SQL> recover tablespace <tb_id>;

si se trata de recuperar todos los datafiles de un tablespace,

SQL> recover datafile '/disk1/data/data02.dbf';

si se trata de recuperar un fichero aislado, o

SQL> recover database;

si se trata de recuperar la BD entera a partir de una copia completa.

4.3.1.2. Copia Lógica: Exportación de la BD

La utilidad *export* de Oracle lee la base de datos, incluido el catálogo, y escribe el resultado en un archivo binario llamado *archivo de volcado para exportación*,

cuya extensión suele ser .dmp. Esta operación puede realizarse para toda la base de datos, para usuarios específicos o para tablas específicas, tratándose, por tanto, de una copia lógica del contenido de la BD. La ventaja más interesante de este tipo de copia con respecto a las anteriores, es que el fichero obtenido no es sensible a la versión de Oracle empleada para su recuperación, lo que permite transferir datos entre diferentes BD con relativa sencillez. Esto se debe a que el formato interno de los ficheros salvados con export es universal para sistemas Oracle.

Durante las exportaciones, se puede elegir entre exportar o no la información del diccionario asociada con las tablas, como los privilegios, los índices y las restricciones de integridad. El archivo en el que se escribe la exportación contiene toda la información necesaria para permitir recuperar por completo y con todas sus características los objetos elegidos.

Las exportaciones que guardan toda la base de datos se denominan **completas** y aquellas que guardan solo las tablas modificadas desde la última copia reciben el nombre de **incrementales**.

La utilidad de Oracle para realizar exportaciones es **export** y con **import** se recupera el contenido de las tablas exportadas que, de no existir, se crearían de nuevo.

En la Tabla 4.6 se muestran las opciones más comunes que la utilidad **export** soporta.

Como puede observarse, algunos de los parámetros especificados pueden entrar en conflicto entre sí, como por ejemplo **FULL=Y** y **OWNER=opc**, ya que el parámetro **FULL** solicita exportación completa mientras que **OWNER** especifica la exportación del esquema de uno o varios usuarios.

Un ejemplo de su utilización es el siguiente:

```
exp system/manager file=exportado.dmp compress=y
indexes=y
owner=(x234212, x453456);
```

que exporta al fichero **exportado.dmp** todos los objetos de los esquemas de los usuarios **x234212** y **x453456** con sus correspondientes índices y en modo compresión.

TABLA 4.6 Opciones de la utilidad *export*.

Cláusula	Descripción
USERID	Nombre de usuario/Contraseña de la cuenta desde la que se ejecuta la exportación.
FILE	Nombre del archivo en el que se copiarán los datos.
COMPRESS=y/n	Indica si deben comprimirse o no los segmentos fragmentados. Por defecto toma el valor <i>y</i> .
GRANTS=y/n	Indica si deben exportarse o no los privilegios sobre los objetos copiados. Por defecto toma el valor <i>y</i> .
INDEXES=y/n	Indica si deben exportarse o no los índices asociados a las tablas copiadas. Por defecto toma el valor <i>y</i> .
CONSTRAINTS=y/n	Indica si deben exportarse o no las restricciones sobre las tablas. Su valor por defecto es <i>y</i> .
FULL=y/n	Indica si la copia va a ser completa. Su valor por defecto es <i>n</i> .
OWNER=propietario	Se exportarán todos los objetos de la cuenta de ese usuario.
TABLES	Lista de las tablas que se van a exportar.

4.3.1.3. Importación de la BD

La utilidad *import* sirve para leer un archivo de volcado de una exportación y recuperar los objetos almacenados en él. Puede utilizarse para extraer de forma selectiva objetos o esquemas de usuario. En la Tabla 4.7 se muestran las opciones más comunes de la utilidad *import* de Oracle.

Como en el caso de *export*, algunos de los parámetros especificados en la Tabla 4.7 son incompatibles entre sí. El propio sistema detecta esta situación y lo comunica dando el correspondiente mensaje de error.

♦ *Un ejemplo de su utilización es el siguiente:*

```
imp system/manager file=cop_segur.dmp tables=(emp,ctas)
touser=opc indexes=y;
```

TABLA 4.7 Opciones de la utilidad *import*.

Cláusula	Descripción
USERID	Nombre de usuario/Contraseña de la cuenta desde la que se ejecuta la importación.
FILE	Nombre del archivo del que se leerán los datos.
SHOW	Indica que debe mostrarse el contenido del archivo en vez de ejecutarse.
GRANTS=y/n	Indica si deben importarse o no los privilegios sobre los objetos copiados. Por defecto toma el valor <i>y</i> .
INDEXES=y/n	Indica si deben importarse o no los índices asociados a las tablas copiadas. Por defecto toma el valor <i>y</i> .
FULL=y/n	Indica si la importación va a ser completa. Su valor por defecto es <i>n</i> .
FROMUSER=usuario	Se importarán todos los objetos de la cuenta de ese usuario.
TOUSER=usuario	Lista de los usuarios sobre cuyos objetos se van a importar los datos leídos.
TABLES	Lista de las tablas que se van a importar.

que importa desde el fichero `cop_segur.dmp` las tablas `emp` y `ctas` asignándolas como objetos al esquema del usuario `opc` y creando los correspondientes índices.

4.3.1.4. Criterios de selección

Como Oracle ofrece diferentes mecanismos para realizar la copia de seguridad de la BD, deberá elegirse uno de ellos como método *primario* y alguno de los restantes como alternativa a posibles fallos en el primero. En el proceso de selección del método de copia de seguridad deben tenerse en cuenta las características y cobertura de cada uno de ellos, que se resumen brevemente en la Tabla 4.8.

Las copias de seguridad en frío son las menos flexibles de llevar a cabo, ya que exigen que la base de datos esté detenida. Además, al ser copias físicas del estado de la base de datos en un instante determinado, no permiten la recuperación selectiva

TABLA 4.8 Resumen de las características de los métodos de copia de seguridad.

Método	Tipo	Características
Copia en frío	Físico	Permite recuperar la BD al estado en el que se encontraba en el instante en que se realizó la copia. Es sensible a la versión.
Copia en caliente	Físico	Permite recuperar la base de datos a su estado actual, ya que exige el archivado de todas las transacciones realizadas hasta el momento. Es sensible a la versión.
Exportación	Lógico	Permite recuperar cualquier objeto de la BD al estado en que se encontraba al realizar la exportación. No es sensible a la versión.

de objetos a partir de ellas. En general, deben usarse como recurso en caso de que falle otro mecanismo de copia de seguridad primario. De los dos métodos restantes, la elección depende de si la base de datos está orientada o no a las transacciones. En caso afirmativo, resulta más aconsejable realizar copias de seguridad en caliente *on-line*, ya que se minimiza la cantidad de datos perdidos en caso de fallo. De haberse utilizado la exportación, habría que conformarse con recuperar los datos que se guardaron en el último volcado.

No obstante, hay otros criterios a tener en cuenta, como el volumen de los datos o la cantidad de transacciones que se realicen. Si la base de datos es pequeña y el volumen de transacciones es bajo, servirán tanto las copias de seguridad en frío como las exportaciones. Es más, si el interés de la base de datos se centra en una o dos tablas, lo más conveniente es utilizar la exportación de los objetos en cuestión. Para bases de datos grandes y con bajo volumen de transacciones, la copia de seguridad en frío resulta el método más adecuado, ya que una recuperación con import puede resultar prohibitiva en tiempo.

Como resumen, independientemente del método elegido, una estrategia razonable de copia de seguridad debería incluir una exportación y una copia de seguridad física.

ya que cada uno de estos métodos cubre distintas visiones y elementos de la base de datos considerada.

Aunque a lo largo de esta sección hemos comentado los mecanismos más utilizados de copia de seguridad en Oracle, existen otras alternativas, que dejamos a la curiosidad del lector [6].

PROBLEMAS PROPUESTOS

- 4.1** Escribir en SQL los siguientes grupos de órdenes:

— Grupo 1 —

- Crear un usuario llamado *pepe* con password *pepeluis*.
- Crear un rol llamado *alumno*.
- Conceder a *alumno* los privilegios de crear tablas, definir vistas, definir roles y crear índices.
- Conceder a *pepe* el privilegio de consulta e inserción sobre la tabla PIEZAS.
- Conceder a *pepe* el rol *alumno*.

— Grupo 2 —

- Crear un rol llamado *matricula* y concederle el privilegio de consulta y actualización sobre la tabla datos_al.
- Conceder el rol *matricula* a todos los usuarios que pueden conectarse.
- Permitir al usuario X4545454 hacer copias de seguridad y consultar cualquier tabla del sistema.

- 4.2** Comentar si son o no correctas las siguientes sentencias:

```

grant create any view on proveedores to chiquito;
revoke index, create view on piezas from X4321;
revoke all privileges from chiquito;
grant create table on proveedores to maria;
revoke select, create view on piezas from X4321;

```

4.3 Comentar la veracidad o falsedad de las siguientes afirmaciones:

- Es posible conceder un rol a otro rol.
- Cuando se anula un privilegio a un rol existente, dicho privilegio queda anulado para todos los usuarios a los que se concedió dicho rol con anterioridad.

4.4 Sean tres usuarios u_0, u_1, u_2 y una tabla T propiedad del usuario u_0 . Considerando el grafo de autorizaciones siguiente,

$$u_0 \rightarrow u_1 \rightarrow u_2$$

escribir las instrucciones SQL necesarias para reflejar la situación indicada en el mismo.

4.5 Sean tres usuarios u_0, u_1, u_2 . El usuario u_0 es propietario de la tabla $T(a, b, c)$. Supongamos que se tiene el siguiente grafo de autorizaciones en el que se concede el privilegio de actualizar la columna b de dicha tabla.

$$u_1 \leftarrow u_0 \rightarrow u_2$$

Escribir las instrucciones SQL necesarias para reflejar la situación indicada en el grafo.

Dado el esquema parcial de la vista que se presenta a continuación:

DBA_COL_GRANTS_MADE(grantor, column_name, table_name, owner, grantee), indicar las tuplas asociadas a las instrucciones efectuadas.

4.6 Elaborar la consulta que permite conocer los privilegios que tiene el usuario `tintin` concedidos de forma directa.

4.7 Elaborar la consulta que permite conocer los privilegios que tiene el rol `gestion`.

- 4.8** Elaborar la consulta que permite conocer la lista de privilegios que tiene el usuario filem\ 'on sea cual sea la procedencia de la concesión de los mismos.
- 4.9** Explicar las razones por las que es conveniente el uso de la técnica de modificación retardada en los SGBD.
- 4.10** ¿Para qué y cuándo se utilizan los procedimientos UNDO() y REDO()?
- 4.11** En la Tabla 4.9 se muestran tres transacciones que se llevan a cabo de forma concurrente. Si los valores iniciales para las variables son $A = 2, B = 4, W = 3, X = 6, Y = 8$:
1. Completar los campos de los registros log que se almacenarán durante la ejecución de las transacciones.
 2. Describir las actuaciones que debe llevar a cabo el gestor de recuperaciones con cada transacción si se produce un fallo en la línea señalada.

Nota: Hacer las hipótesis necesarias respecto al instante en que las transacciones ejecutan *commit*.

TABLA 4.9 Secuencia de ejecución de tres transacciones.

T_1	T_2	T_3	T_i	Estado	Dato	V. Ant	V.Nue
lee(A,r)							
r:= r*2	lee (W,n)						
escribe(A,r)	lee(X,t)						
	t:= t+n						
	escribe(X,t)	lee(A,s)					
	lee (Y,t)	s:= s*5					
	t:= t-n	escribe(B,s)					
lee(B,r)							
r:=r*5							
escribe (B,r)							
*****	fallo	*****					
	escribe(Y,t)						

- 4.12** ¿Qué mecanismos emplean los SGBD para garantizar la persistencia de las transacciones?

- 4.13 Indicar en qué consisten los mecanismos de copia de seguridad: exportación, copia en frío y copia en caliente, y señalar las principales diferencias entre ellos.
- 4.14 Explicar por qué es necesario tener habilitado el modo archivado cuando se hace una copia de seguridad en caliente.
- 4.15 Escribir un script que permita hacer una copia en caliente de los tablespaces users y system suponiendo que users tiene asociados dos datafiles y system solo uno.
- 4.16 Escribir la sentencia que permite exportar las tablas emp y depto del usuario scott sin incluir índices ni privilegios.
- 4.17 Escribir la sentencia que permite exportar todos los objetos del usuario user23 sin privilegios, sin índices y sin restricciones de integridad.
- 4.18 Escribir la sentencia que permite importar en el esquema de sys la tabla emp de scott.

de Concurrencia