Integridad de los datos: transacciones, backup y recuperación (I)

Administración de Bases de Datos

Curso 2018-2019

Jesús Correas, Mercedes G. Merayo, Yolanda García

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

Sobre este tema

- Estas transparencias están basadas en los siguientes documentos:
  - J. Watson OCA Oracle Database 11g: Administration I Exam Guide (Exam 1Z0-052). Oracle Press - McGraw-Hill, 2008.
  - S. R. Alapati Expert Oracle Database 11g Administration. Apress, 2009.
  - Oracle Database Concepts, 11g Release 2 (11.2) E40540-04. Oracle, 2015.
  - Oracle Database Administrator's Guide, 11g Release 2 (11.2) E25494-07. Oracle, 2015.
  - Oracle Database Backup and Recovery User's Guide, 11g Release 2 (11.2) E10642-08. Oracle, 2015.

Integridad de los datos

- Una de las principales misiones de una BD (y del DBA) es garantizar que no se pierden datos del sistema, por el motivo que sea:
  - Modificaciones concurrentes de los datos;
  - Cortes en el suministro de corriente;
  - Fallos de hardware del servidor, la red, los ordenadores de usuarios;
  - Fallo de la instancia de Oracle, etc.
- También deben considerarse otros tipos de fallos que implican pérdida de datos:
  - Errores de los usuarios;
  - Errores de programación;
  - Errores de administración.
- El mecanismo principal de cualquier BD para preservar la integridad de los datos es el uso de **transacciones.**
- También se utiliza el mecanismo de **copia de seguridad** (**backup**) para preservar el estado de la BD en un instante determinado.

ACID

- El sistema de transacciones de cualquier BD relacional debe cumplir cuatro propiedades denominadas **ACID**:
  - **A de atomicidad:** En una transacción, todas las operaciones se terminan, o bien no se realiza ninguna.
  - **C de consistencia:** Una transacción debe **preservar la consistencia** de la BD.
    - ⋆ En particular, una consulta debe ser consistente con el estado de la base de datos **en el instante de inicio** de la ejecución de la consulta.
  - **I de aislamiento:** Una transacción no completada (con `COMMIT`) es invisible al resto del mundo.
  - **D de durabilidad:** Cuando se completa una transacción con `COMMIT`, entonces es imposible que la base de datos la pierda.
- Oracle garantiza estas propiedades a través de los segmentos de *undo* (ACI) y los ficheros *redo log* (D).
- Normalmente las prop. ACID se aplican a las **transacciones** de la BD, pero también es aplicable a la integridad de los datos en general.

Control de Transacciones: **COMMIT** y **ROLLBACK**

- Para ello existe un conjunto de instrucciones SQL que permiten implementar el control de las transacciones.
- **No existe una sentencia específica de inicio de transacción:**
  - Se inicia cuando se ejecuta una sentencia DML o DDL.
  - O cuando se ejecuta una sentencia **SET TRANSACTION.**
- Hay dos sentencias básicas para **finalizar una transacción:** **COMMIT** y **ROLLBACK**.
  - **COMMIT confirma** las modificaciones realizadas en la BD y los hace permanentes y visibles a las demás sesiones activas.
  - **ROLLBACK cancela todos los cambios** que se hayan realizado desde el inicio de la transacción.
- Además, cualquier sentencia DDL **termina implícitamente cualquier transacción** (confirmando las sentencias DML anteriores, como un `COMMIT`).
- Cuando termina una transacción, la siguiente instrucción SQL ejecutable inicia automáticamente la siguiente transacción.

Ejemplo

- **Ejemplo2.sql: ¿Cuál es el estado de la BD después de ejecutar las siguientes sentencias?**

```
CREATE TABLE empl (
  NIF VARCHAR2(9) PRIMARY KEY,
  NOMBRE VARCHAR2(20),
  SALARIO NUMBER(6,2)
);
INSERT INTO empl VALUES ('10A','Jorge Perez',3000.11);
ROLLBACK;
INSERT INTO empl VALUES ('30C','Javier Sala',2000.22);
INSERT INTO empl VALUES ('30C','Soledad Lopez',2000.33);
INSERT INTO empl VALUES ('40D','Sonia Moldes',1800.44);
INSERT INTO empl VALUES ('50E','Antonio Lopez',1800.44);
COMMIT;
INSERT INTO empl VALUES ('70C','Soledad Martin',2000.33);
```

- **¿Cuál es el estado de la BD visible desde otras sesiones?**

Control de Transacciones: **SAVEPOINT**

- Además de las sentencias **COMMIT** y **ROLLBACK**, se pueden fijar **puntos intermedios:**
  - La instrucción ***SAVEPOINT identificador*** establece un punto en una transacción hasta el que se puede cancelar parcialmente.
  - Para hacerlo se utiliza la sentencia:

    ROLLBACK TO SAVEPOINT identificador.

- Sin embargo, **los SAVEPOINT intermedios no finalizan la transacción:** solo deshacen algunos de los cambios realizados.
- Se pueden utilizar **varios SAVEPOINT** en una misma transacción:
  - Cuando se retrocede a un `SAVEPOINT`, se eliminan todos los `SAVEPOINT` posteriores, pero no los anteriores.
  - Se puede **desplazar** un `SAVEPOINT` utilizando el mismo identificador en distintos puntos: se retrocede al último punto ejecutado con ese identificador.

Otro ejemplo

- **Ejemplo3.sql: ¿Cuál es el estado de la BD después de ejecutar las siguientes sentencias?**

```
SET TRANSACTION NAME 'sal_update';
UPDATE empl SET salario = 7000 WHERE NIF= '40D';

SAVEPOINT after_salario;
UPDATE empl SET salario = salario + 100 WHERE NIF= '40D';

ROLLBACK TO SAVEPOINT after_salario;
UPDATE empl SET salario = salario + 250 WHERE NIF= '40D';
COMMIT;
```

Transacciones en Oracle: **SELECT**

- Vamos a ver cómo funciona el procesamiento de cada una de las sentencias DML en una transacción.
- **SELECT:**
  - La ejecución de una sentencia `SELECT` en el proceso servidor comprueba si los datos a recuperar están en el buffer de la SGA
    - ⋆ Si no están, los recupera del disco.
  - A continuación procesa la consulta y utiliza la PGA si es necesario.
  - Una vez terminada la consulta, se devuelve el resultado al proceso del usuario.
  - Si durante el proceso se detecta que algún bloque ha cambiado, **se utiliza la información que está en el segmento de undo**
  - De esta forma se garantiza la **consistencia**.

Transacciones en Oracle: **UPDATE**

- Las sentencias de modificación son más complejas. Lo vemos con **UPDATE.** Los pasos que se realizan son:
  1. Se traen los bloques afectados al buffer de datos de la SGA (si no están en la SGA).

     Además, **se bloquean todas las filas y entradas de índices** que van a verse afectados por la operación.

  2. Se crea un **bloque vacío en un segmento de undo.**
  3. Se **genera la información de redo.** Contiene los cambios que se realizan sobre la BD en forma de **vectores de cambio**:
     - ★ Para el bloque de la tabla/índice que se ha modificado: almacena el id de la fila (*rowid*) y el nuevo valor.
     - ★ para el bloque de *undo*: el *rowid* y **el valor antiguo.**
  4. Se ejecuta el cambio **en los dos bloques en la SGA:**
     - ★ Bloque de la tabla/índice con el nuevo valor.
     - ★ Bloque de *undo* **con el valor antiguo.**

Transacciones en Oracle: **UPDATE**, **INSERT**, **DELETE**

- **UPDATE** (continuación):
  - En cuanto se genera el bloque de *undo*, cualquier consulta de otra sesión sobre esos datos **se redirige al bloque de undo** hasta que se ejecuta la sentencia `COMMIT`.
  - El uso del segmento de *undo* garantiza la **atomicidad**, **consistencia** y **aislamiento**.
- **INSERT** y **DELETE** son similares a `UPDATE`.
  - Difieren en la **información de undo que se debe generar:**
  - **INSERT:** solo hay que almacenar el rowid de la fila que se inserta.
  - **DELETE:** hay que almacenar los datos de toda la fila.

Cuando se ejecutan estas tres sentencias **No se realiza ninguna escritura sobre disco:** esto lo hacen los procesos `DBWn` y `LGWR` de forma independiente.

Transacciones en Oracle: **ROLLBACK**

- **ROLLBACK:** Se realiza cuando la sesión activa **ejecuta una sentencia ROLLBACK,** pero también en otros casos:
  - **problemas con la sesión** que ha iniciado una transacción: error de red, fallo del programa del cliente, etc. El proceso `PMON` realiza un rollback automático.
  - **Reinicio del servidor:** el proceso `SMON` realiza un rollback automático de **todas** las transacciones activas.
- En todos los casos **el funcionamiento es el mismo:**
  - Si se ejecutó un `UPDATE`, se genera un nuevo `UPDATE` utilizando la información del bloque correspondiente del segmento de *undo*.
  - Si se ejecutó un `INSERT`, se genera una sentencia `DELETE` para eliminar la fila (*rowid* está en el segmento de *undo*).
  - Si se ejecutó un `DELETE`, se genera una sentencia `INSERT` a partir de la información del bloque de *undo*.
- Por último, **se desbloquean las filas afectadas.**
- Se garantiza la **atomicidad:** se descartan **todas las operaciones.**

Transacciones en Oracle: **COMMIT**

- Cuando se ejecuta una sentencia **COMMIT,** se hace lo siguiente:
  1. **El proceso LGWR escribe el buffer de redo log en disco,** y
  2. se marca la transacción como completada y **se desbloquean las filas afectadas.**
- **No se hace nada más: no se escribe sobre los datafiles.**
- Los bloques de los segmentos modificados se escriben en los datafiles **cuando los procesos DBWn lo crean conveniente.**
- Esto aumenta mucho la eficiencia del sistema.
- Y por otra parte la propiedad de **durabilidad** se cumple:
  - en caso de daños o fallos en el sistema, se puede recuperar el estado a partir de una copia de seguridad y los ficheros de *redo log*.
- El *redo log* contiene **todos los segmentos:** de datos y de *undo* (de las transacciones no completadas).

Transacciones en Oracle: "Auto-**COMMIT**"

- En determinadas circunstancias la sentencia `COMMIT` está implícita:
  - En las sentencias DDL.
  - En las sentencias DCL `GRANT` y `REVOKE`.
  - Cuando se sale con `EXIT` de `SQL*Plus` (si se cierra el programa de otra forma, se hace un `ROLLBACK`).
- Esto quiere decir que se completan **todas las sentencias DML anteriores,** aunque no se utilice `COMMIT` explícitamente.

Monitorización y resolución de conflictos de bloqueo

- Existe un **conflicto** cuando dos sesiones de acceso a Oracle intentan modificar la misma fila a la vez.
- La **propiedad de aislamiento** de ACID exige que dos transacciones no completadas no puedan afectarse mutuamente.
- La base de datos debe **serializar** los accesos concurrentes, mediante mecanismos de **bloqueo de filas o tablas.**
- Oracle intenta bloquear la menor cantidad de datos necesaria para obtener el máximo nivel de concurrencia.
  - Un bloqueo **restringe el acceso a estos elementos para determinadas operaciones.**

Monitorización y resolución de conflictos de bloqueo

- De forma simplificada, hay **dos tipos de bloqueo:**
  - **Bloqueos compartidos:** los producen las sentencias de consulta (**`SELECT`**). Otras sesiones pueden modificar los datos bloqueados, pero **no son visibles** desde el `SELECT`.
  - **Bloqueos exclusivos:** los producen las sentencias DDL y de modificación (**`INSERT`**, **`UPDATE`**, **`DELETE`**). Otras sesiones no pueden modificar los datos hasta que no se libera el bloqueo.
  - Aunque son diferentes, consideramos **bloqueos exclusivos** los generados por sentencias **`SELECT...FOR UPDATE.`**
- En un bloqueo exclusivo, otras sesiones pueden consultar el contenido de las tablas **sin los cambios de transacciones no confirmadas.**
  - utilizan la información de *undo* para mostrar la información **anterior a la transacción.**

Conflictos de bloqueo: Comportamiento ante el bloqueo

1. **Sentencias DDL:** Si no pueden obtener el bloqueo sobre un objeto, entonces **terminan inmediatamente con una condición de error.**
2. **Sentencias DML de lectura:** realizan bloqueos compartidos y **no se ven afectadas por estos bloqueos**, pero utilizan la información **anterior a la transacción:**
   - Para ello, el SGBD debe mantener **la información anterior a cualquier transacción no terminada.**
   - Si el SGBD permite consultar información no confirmada con `COMMIT`, se dice que permite realizar *lectura sucia.*
   - Oracle no permite realizar *lectura sucia.*
3. **Sentencias DML de modificación de datos:** si no pueden obtener el bloqueo, **esperan en una cola ordenada cronológicamente.**

- Esta espera se denomina **contención por bloqueo.**
  - La espera puede ser muy larga si hay muchas transacciones con muchas operaciones cada una.

Resolución de conflictos de bloqueo

- Se puede **controlar el tiempo de espera** en casos de contención utilizando una sentencia `SELECT` especial:

```
SELECT ... FOR UPDATE NOWAIT
SELECT ... FOR UPDATE WAIT n
```

- Esta sentencia es de lectura pero **bloquea como si se fueran a modificar datos.**
- Esta sentencia termina inmediatamente (o después de `n` segundos) si las filas seleccionadas están bloqueadas por otra sesión.
- Cierto nivel de contención es normal en cualquier base de datos, pero **se puede deteriorar el rendimiento del sistema por errores de diseño o de programación de las aplicaciones.**

Contención por bloqueos

Problemas de diseño/programación que incrementan el nivel de contención:

- **Transacciones demasiado largas.** Ejemplo: si entre el bloqueo de una tabla y la sentencia `COMMIT` se espera una acción del usuario (que está tomando café).
- **Procesos batch.** Ejemplo: el cierre mensual de la contabilidad: *conceptualmente* es una única transacción, pero en la práctica puede suponer el bloqueo de miles de filas de muchas tablas durante horas.
- **Aplicaciones externas.** Otras aplicaciones que utilizan Oracle pueden incluir niveles de contención muy altos.
- **Bloqueos para realizar consultas repetidas.** Algunas aplicaciones bloquean filas de la tabla para realizar varias consultas *consistentes* sobre una tabla. Se puede resolver con

  ```
  SET TRANSACTION READ ONLY
  ```

  No se bloquea ninguna fila pero garantiza que el estado de las tablas no está afectado por otras sesiones.

Detección y resolución de contención por bloqueo

- Se pueden detectar los bloqueos utilizando las herramientas de Oracle.
- **Primero vamos a forzar un bloqueo:** abrimos dos sesiones de `SQL*Plus` o `SQL Developer` y ejecutamos la siguiente línea en las dos sesiones:

```
SELECT * FROM mitabla FOR UPDATE;
```

- La segunda sesión se queda esperando a que la primera termine (con `COMMIT` o `ROLLBACK`).
- **A continuación podemos utilizar Enterprise Manager:**
  - En la pestaña **Performance**, enlace **Instance Locks.**
- Si terminamos la primera transacción con `COMMIT` y refrescamos Enterprise Manager, podemos ver cómo desaparece el bloqueo.

Detección y resolución de contención por bloqueo

- El administrador no puede hacer mucho con los bloqueos:
  - comunicarlo a los desarrolladores de la aplicación.
  - En caso necesario, interrumpir la sesión que está bloqueando a otras, con *Enterprise Manager* o bien con:

```
ALTER SYSTEM KILL SESSION num;
```

- Un tipo especial de bloqueo es el interbloqueo (**deadlock**)
  - Se produce cuando dos transacciones se están esperando mutuamente.
  - Oracle lo detecta **y cancela automáticamente una de las transacciones lanzando una excepción.**
  - Se pueden consultar los interbloqueos en el log de alertas de Oracle (en el directorio indicado en "Diag Trace" de la vista `V$DIAG_INFO`).

Control de Transacciones: **SET TRANSACTION**

- En una transacción, las filas de las tablas afectadas se van bloqueando **según se ejecuta la transacción:**
  - Si otras sesiones terminan transacciones con `COMMIT`, los cambios serán visibles **si no se han bloqueado antes las filas afectadas.**
- Esto puede ocasionar **inconsistencias en las transacciones.**
- Para evitarlo, se utiliza la sentencia **`SET TRANSACTION:`**

  `SET TRANSACTION [READ ONLY | READ WRITE] NAME` ***`nombre`***
  - **inicia una transacción explícitamente.**
  - Si es **`READ ONLY`**, **Todas las sentencias** que se incluyen en la transacción acceden a los datos en el estado en el que están en la BD **en el instante de inicio de la transacción: no ven los cambios realizados** por otros usuarios durante la ejecución de la transacción.
- La opción **`READ WRITE`** es la opción por defecto: fija el inicio de una transacción que modifica datos.
- La opción **`READ ONLY`** inicia una transacción **que no modifica datos.**
  - Se utiliza para que todas las consultas sean **consistentes con un estado de la BD:** el del inicio de la transacción.

Segmento de *undo* y tablespace de *undo*

- Oracle **genera automáticamente un segmento de undo por cada transacción iniciada.**
  - Permite deshacer transacciones no completadas mediante `ROLLBACK`.
  - Garantiza el **aislamiento:** permite ejecutar las transacciones como si se realizaran secuencialmente.
  - **Consistencia** (*read-consistency*): permite mostrar a una consulta el estado de la base de datos consistente con el **instante de inicio de la consulta.** Solo se garantiza *si la consulta tiene éxito (*).*
- Para ello, almacena la información anterior al inicio de cada transacción en el segmento de *undo*.
- Oracle maneja el tablespace de *undo* **exactamente igual que cualquier otro tablespace:** aplica las operaciones sobre los bloques y segmentos, información de *redo*, etc.

Segmento de *undo* y tablespace de *undo*

- Los segmentos de *undo* no son de tamaño fijo y **pueden crecer durante la ejecución de la transacción.**
- Los segmentos de *undo* se almacenan en un tablespace de *undo*.
- **¿Qué ocurre si la cantidad de información de undo es muy grande?**
- Oracle debe garantizar las propiedades ACID, pero sin exceder el espacio en disco disponible.
- Esto supone llegar a una **solución de compromiso** sobre qué información almacenar en el tablespace de *undo*:
  - La información de *undo* no se puede almacenar *indefinidamente.*
  - Si el tablespace de *undo* es autoextensible, es posible que **se llene el disco** en transacciones muy largas.
- Por estos motivos, **los segmentos de undo se suelen reutilizar** cuando ya no son necesarios.

Segmento de *undo* y tablespace de *undo*

- Un segmento de *undo* puede estar en tres estados:
  1. **Activo:** hasta que se completa la transacción con `COMMIT` o se deshace con `ROLLBACK`.
  2. **Expirado:** segmento de una transacción completada que Oracle **puede reutilizar para otra transacción.**
  3. **No expirado:** similar al anterior, pero Oracle puede utilizar la información del segmento para garantizar la consistencia de consultas muy largas que necesitan el estado anterior a la transacción.
- La diferencia entre (2) y (3) depende de un parámetro (`UNDO_RETENTION`) que indica el tiempo de expiración.
- Una consulta muy larga puede necesitar datos de un segmento expirado. Si ya se ha reutilizado, se produce un error *"Snapshot too old"* (La consistencia (*) no garantiza que las consultas terminen).

Dimensionamiento del tablespace de *undo*

- El administrador debe dimensionar el tablespace de *undo* para que
  - Haya espacio disponible para que **todas las transacciones se puedan realizar,** y
  - Haya suficiente información de *undo* para que **todas las consultas tengan éxito.**
- Si el tablespace está lleno y se intenta iniciar una nueva transacción:
  1. Si no hay segmentos expirados, reutiliza un segmento **no expirado aunque no haya vencido el tiempo de expiración.**
  2. Si todos los segmentos son activos, se produce un error.

  (el algoritmo es más complejo).
- Por tanto, si es necesario Oracle **elimina segmentos no expirados aunque eso produzca errores en consultas.**
- Pero hay casos en los que es preferible que se cancelen las transacciones:
  - En procesos de generación intensiva de informes (cierres de ejercicio).
  - Si se quieren utilizar consultas de *flashback*.
- Se puede configurar así al crear el tablespace:

```
CREATE UNDO TABLESPACE ... RETENTION GUARANTEE;
```

Fallos que pueden ocurrir en un sistema de BD

Hay varios tipos de fallos que pueden ocurrir en un sistema de base de datos:

1. Fallos en sentencias SQL.
2. Fallos del proceso de usuario.
3. Errores de usuario.
4. Fallos de dispositivos de almacenamiento.
5. Fallo de instancia de Oracle.

A continuación veremos cada uno de estos tipos de fallo.

1. Fallos en sentencias SQL

- Ocurren cuando una sentencia SQL falla por algún motivo.
- El sistema de gestión de segmentos de *undo* deshace los cambios que se hubieran hecho en la sentencia SQL.
- Los programas de usuario deberían **tratar las excepciones** que se produzcan.
- Se pueden producir por diversas causas:
  - **Datos no válidos:** formato incorrecto, constraints, etc.
  - **Errores lógicos de la aplicación:** por ejemplo, *deadlocks*.
  - **Privilegios de usuario insuficientes.**
  - **Problemas de gestión de espacio en disco:** tablespace de datos o de *undo* lleno, cuota de disco sobrepasada, etc.
- Los primeros casos no son responsabilidad del administrador de la BD, pero debe notificarlos al responsable de la aplicación.
- Para el último caso se deben utilizar herramientas para monitorizar el almacenamiento en disco:
  - *Advisors*, monitor de diagnóstico ADDM, mecanismo de alertas.

2. Fallos del proceso de usuario

- El proceso que ejecuta el programa que accede a la base de datos puede fallar por diversos motivos:
  - El usuario apaga su ordenador sin desconectarse antes,
  - El programa tiene un fallo que para su ejecución, etc.
- Para estos casos, el proceso **PMON** comprueba periódicamente todos los procesos del servidor y **deshace las transacciones pendientes** de los que han perdido la conexión con el proceso cliente.
- Un caso particular es el de los **fallos de red:**
  - Demasiadas solicitudes de conexión concurrentes.
  - Fallos del hardware de red.
  - Fallos en la subred a la que se conecta el servidor (debería conectarse a **dos subredes distintas**).
- Debe notificar los errores para dimensionar o reparar los componentes que fallan.

3. Errores de usuario

- Son los más difíciles de manejar. Por ejemplo:
  - Una sentencia **`UPDATE`** lanzada por un usuario en la que se olvida la cláusula `WHERE`.
    - ★ Se actualizan millones de filas en lugar de una sola.
    - ★ Si se detecta antes de `COMMIT`, bien. Si no...
  - Un administrador que ejecuta **`DROP TABLE`** en la BD de producción pensando que está en la de pruebas...
    - ★ Las sentencias DDL **incluyen un `COMMIT` implícito.**
- Algunas herramientas pueden ayudarnos a arreglar estos errores:
  - **Consultas flashback:** utilizan el tablespace de *undo*. Depende del tiempo de expiración fijado para los segmentos de *undo*. Por ejemplo, el estado de una consulta 5 minutos atrás:

    ```
    SELECT ... AS OF TIMESTAMP(SYSDATE - 5/1440)
    ```

  - **flashback de `DROP` o `DATABASE`** :

    ```
    FLASHBACK TABLE ... TO BEFORE DROP
    ```

  - **Log Miner:** utiliza los ficheros *redo log* para recuperar el estado de la base de datos a un instante del pasado.

4. Fallos de dispositivos de almacenamiento

- Los discos fallan con cierta frecuencia, especialmente en servidores.
- Una base de datos tiene tres tipos de ficheros: controlfile, *redo log* y datafiles.
- Controlfile y los *redo log* online **siempre deben estar multiplexados en discos diferentes** (y con distintos controladores).
- Los datafiles no se pueden multiplexar en Oracle, pero se pueden recuperar de un backup (o usar RAID o BD replicadas).
- Para recuperar un datafile se debe hacer lo siguiente:
  1. **Restaurar** (*restore*) el datafile de un backup.
  2. **Recuperar** (*recover*) el datafile aplicando los cambios de los ficheros de *redo log*: **online** y **archivados.**
- Se debe hacer backup de:
  - El **controlfile.**
  - Los **datafiles.**
  - Los **ficheros de redo log,** en cuanto dejan de estar online.

5. Fallo de la instancia de Oracle

- Es una caída (*crash*) de la instancia: una parada de los procesos background.
- Debido a un corte de corriente eléctrica, un problema de hardware crítico, o un comando `SHUTDOWN ABORT`.
- La base de datos puede quedar **corrompida:**
  - Puede haber **transacciones completadas perdidas** y otras **trasacciones no completadas almacenadas** en los datafiles.
  - La BD se corrompe porque **la instancia trabaja sobre la memoria** y la actualización de los ficheros en disco (`DBWn`) se produce de forma independiente a las transacciones.
- Pero **el proceso `LGWR` trabaja casi en tiempo real escribiendo los ficheros de redo log en disco**.
- Por tanto, **el estado de la base de datos se puede recuperar de los ficheros de redo log.**

Recuperación de la instancia de Oracle

- Oracle detecta e intenta recuperar la instancia **automáticamente.**
- Se realiza cuando se ejecuta el comando `STARTUP`.
- La base de datos se puede abrir **solo si** se ha realizado la recuperación automática.
- Utiliza los ficheros de *redo log* para reconstruir **en memoria** el buffer de la base de datos tal y como estaba antes del fallo.
- Tiene dos fases:
  - Fase **roll forward:** realiza los cambios de *redo* a todos los bloques (de datos y de *undo*) de todas las transacciones (completadas o no).
  - Fase **rollback:** deshace las transacciones no completadas en el momento del fallo de la instancia.
- La clave **para poder recuperar el estado** del momento del fallo es que se escriba en el fichero de *redo log* **en el momento de completar cada transacción (COMMIT).**

Recuperación de la instancia de Oracle

- Por tanto, **si no hay fallos en los dispositivos de almacenamiento, siempre se puede recuperar la instancia de Oracle.**
- Pero la recuperación **puede ser un proceso muy lento:**
  - Si hay muchos cambios a recuperar de los ficheros de *redo log*.
- Para ello, Oracle utiliza el concepto de **posición de checkpoint** :

Un **checkpoint** es un instante (*system change number,* **SCN**) en el que se puede garantizar que todos los cambios sobre la base de datos están escritos en los datafiles.

- Al recuperar la instancia, solo hay que recuperar los cambios de los ficheros de *redo log* **desde el último checkpoint.**
- Se debe llegar a una **solución de equilibrio:**
  - Si los datafiles se actualizan con mucha frecuencia, **se deteriora el rendimiento de la BD** (en su funcionamiento habitual);
  - Si se hacen pocos *checkpoint*, **la recuperación es muy lenta.**

Recuperación de la instancia de Oracle

- *Mean time to recover* (**MTTR**): tiempo estimado que tardaría Oracle en recuperar la instancia al estado actual desde el último *checkpoint*.
- El ajuste manual para garantizar un MTTR adecuado es muy complejo.
- En las últimas versiones de Oracle se puede fijar un parámetro para que lo calcule automáticamente: **`FAST_START_MTTR_TARGET`.**
  - Si vale 0, maximiza el rendimiento de la BD;
  - Si no, es el tiempo (segs.) **que debe tardar** para recuperar la instancia.
- Además, se activa el denominado **checkpoint auto-tuning:** el ajuste automático de la actividad de `DBWn` para garantizar el MTTR.
  - Hace un estudio estadístico de la velocidad de I/O y uso de CPU del sistema.
  - En función de los resultados, **`DBWn` escribe en los datafiles bloques adicionales** para avanzar la posición de *checkpoint*: Son *checkpoint* incrementales.

Checkpoints, ficheros de *redo log*

- También se pueden hacer otros tipos de *checkpoint*:
  - **Totales:** Son muy costosos, normalmente al cerrar la base de datos de forma ordenada (`SHUTDOWN NORMAL|IMMEDIATE|TRANSACTIONAL`).
  - **Parciales:** Cuando se pone un datafile o tablespace en modo offline o modo backup, al borrar un segmento, etc.
- **Los ficheros de redo log se deben proteger** porque son **imprescindibles** para recuperar la instancia.
  - Debe haber **al menos dos** ficheros en cada grupo, en distintos discos, distintos controladores.
  - Los grupos de *redo log* pueden estar en diversos estados:
    - ★ **`CURRENT`:** es el grupo online, el que se está utilizando.
    - ★ **`ACTIVE`:** es necesario para recuperar la instancia porque tiene datos **posteriores a la posición de checkpoint.**
    - ★ **`INACTIVE`:** no es necesario para recuperar la instancia.

Base de datos en modo *archivelog*

- Los grupos de ficheros *redo log* se van reutilizando de forma circular: **Se sobrescriben cuando están inactivos.**
- Para no perder datos en caso de fallo en los discos, **se deben guardar todos los datos de redo log desde el último backup:**
- La BD debe estar en **modo archivelog:**
  - En este modo, los ficheros de *redo log* se **archivan automáticamente** sobre otros ficheros en disco en cuanto dejan de estar online.
    - ★ Estos ficheros de archivo deben tener un nombre único.
    - ★ Además, de estos ficheros se debe realizar copia de seguridad (backup), que junto con el backup de los datafiles permite recuperar la BD.
  - El archivo lo realizan otros procesos *background*: **`ARCn`.**
- Los archivos de *redo log* también se pueden **multiplexar** en distintos discos.