1. Uso de transacciones para concurrencia.

En una entidad bancaria se utiliza Oracle para el sistema de gestión de cuentas bancarias. Se ha informado al departamento de informática de un problema que ocurre de forma intermitente: las operaciones de traspasos de fondos entre cuentas bancarias generan algunas veces un mensaje de error indicando que el saldo de la cuenta no coincide con la suma de los importes de los movimientos realizados. Sin embargo, cuando se realizan las comprobaciones de balance al final de cada día, no se detecta ningún error.

Las tablas y el procedimiento almacenado que realiza los traspasos está contenido en el fichero `EJERCICIO-transacciones-backup.sql`. (este código es una simplificación de una operación de traspaso real; no se incluyen comprobaciones que se deberían realizar –p. ej., que hay saldo suficiente en la cuenta de origen).

Nos han informado de que este error se produce especialmente en aquellas cuentas que tienen gran número de traspasos resultado de distintas operaciones. Los programadores han estudiado el código y no han visto ningún problema en la transacción incluida en el procedimiento, y por otra parte en las pruebas unitarias no se ha conseguido reproducir ningún error, por lo que se sospecha de que pueden estar produciéndose **interferencias entre operaciones de traspaso que se ejecutan concurrentemente**.

Contesta a las siguientes preguntas:

a. ¿Qué está ocurriendo?

NOTA: Para este ejercicio **no ejecutes el procedimiento PL/SQL directamente** para intentar reproducir el error: solo sucede cuando se ejecutan concurrentemente varias operaciones de traspaso, y es realmente difícil reproducirlo con ejecuciones manuales. Es mejor razonar en el código fuente cómo se pueden intercalar las sentencias para que se produzca un resultado incorrecto y, si es necesario, ejecutar las sentencias una por una utilizando dos sesiones distintas de SQL*Plus o SQLDeveloper para ver qué es lo que está ocurriendo en un ejemplo sencillo.

b. ¿Cómo se podría corregir el problema **de forma sencilla**?

c. ¿Se podría permitir un nivel de concurrencia más alto en el apartado (b)?

d. ¿Cómo arreglarías el problema si lo único que puedes hacer es **modificar (o comentar) las sentencias SQL `SELECT` existentes**, pero no añadir ni cambiar de sitio ninguna sentencia?

2. Bloqueos y contención por bloqueo.

2.1. Tipos de bloqueo.

En Oracle hay dos tipos de bloqueo: bloqueo compartido y bloqueo exclusivo.

Los bloqueos exclusivos se toman sobre filas o la tabla completa: para modificar los datos de una fila con sentencias DML en el primer caso; para modificar la estructura

de una tabla con sentencias DDL en el segundo. Cuando una sesión tiene el bloqueo exclusivo sobre una fila (o tabla), ninguna otra sesión puede tomar el bloqueo exclusivo sobre la misma fila (o cualquier fila de la tabla en el segundo caso).

Los bloqueos compartidos se toman solamente sobre tablas, e impiden que otra sesión pueda tomar un bloqueo exclusivo sobre la tabla. Vamos a ver una situación en la que una sentencia `SELECT` puede bloquear determinadas operaciones.

Ejercicio 2.1: (bloqueo compartido) Crea una tabla con dos columnas, C1 y C2, de tipo `integer` e inserta en ella 2 filas:

```
create table mitabla (c1 integer, c2 integer);

insert into mitabla values (10,0);

insert into mitabla values (11,0);
commit;
```

Vamos a **bloquear la tabla** con un bloqueo compartido producido por una consulta SQL de modificación de datos y vamos a comprobar que efectivamente está bloqueada, intentando modificar la estructura de la tabla mientras se ejecuta la consulta SQL:

Utiliza dos sesiones simultáneas de SQL*Plus o SQLDeveloper. En la primera sesión ejecuta una consulta de modificación de datos. Por ejemplo:

```
update mitabla set c2 = 1 where c1 = 10;
```

Antes de confirmarla con `commit`, ejecuta en la segunda sesión una sentencia DDL para eliminar la columna C2 de la tabla:

```
alter table drop column C2;
```

Anota en la memoria del ejercicio lo que ocurre en la segunda sesión.

¿Qué ocurre si en lugar de una sentencia DDL ejecutas una sentencia DML?: utiliza en la segunda sesión una sentencia `UPDATE` para modificar otra fila diferente a la modificada en la sentencia anterior y vuelve a repetir el ejercicio. Por ejemplo ejecuta en la segunda sesión:

```
update mitabla set c2 = 1 where c1 = 11;
```

Anota en la memoria del ejercicio lo que ocurre en este caso.

2.2. Contención por bloqueo.

Ejercicio 2.2: Prepara dos consultas `SELECT … FOR UPDATE` en sesiones distintas de tal forma que la primera bloquee a la segunda. Consulta en *Enterprise Manager* el bloqueo que se ha producido: entra en la pestaña *Performance* y ahí en el enlace *Instance Locks*. En el bloqueo que se muestra en la pantalla, utiliza el botón correspondiente para cancelar la sesión que está bloqueando y anota en la memoria del ejercicio el mensaje que aparece en la sesión cancelada.

3. Interbloqueos (*Deadlocks*).

3.1. Interbloqueo sencillo entre dos transacciones.

Ejercicio 3.1a: Considera la tabla y las filas que has creado para el ejercicio 2.1 y abre dos sesiones de SQL*Plus o SQL Developer. Indica la secuencia de sentencias SQL que operen sobre esa tabla y lleven a un interbloqueo entre ambas sesiones. Muestra el comportamiento de las sesiones ante el interbloqueo.

Ejercicio 3.1b: En el ejemplo visto en el apartado 1 (`EJERCICIO-transacciones-backup.sql`) el administrador de la BD detecta que en muy raras ocasiones se producen situaciones de interbloqueo. ¿Podrías determinar en qué caso se puede producir un interbloqueo entre dos traspasos ejecutando ese código? ¿Cómo se podría resolver?

3.2. Interbloqueo entre tres transacciones.

En clase hemos visto una forma sencilla de provocar un interbloqueo (*deadlock*) entre dos transacciones. Los interbloqueos pueden ser mucho más complejos en los que intervienen múltiples transacciones. Para resolver problemas complejos de interbloqueo, es fundamental obtener información sobre las sesiones que intervienen en el bloqueo y las consultas asociadas. Vamos a verlo en el siguiente ejercicio.

Ejercicio 3.2: Utiliza una tabla como la que se ha generado en el apartado 2 para provocar un **interbloqueo entre tres transacciones**. Para ello, abre tres sesiones con SQL*Plus y ejecuta sentencias `SELECT ... FOR UPDATE` para generar el interbloqueo. Una vez realizado el interbloqueo, Oracle automáticamente cancela una de las transacciones.

¿En qué estado quedan las otras dos transacciones? Explica el motivo por el que quedan en ese estado.

3.3. Consulta del fichero alert_orcl.log.

Para estudiar el interbloqueo vamos a consultar el log de alertas. Para ello, consulta la tabla `v$diag_info` para saber el directorio donde se guardan las trazas:

```
select name, value from v$diag_info where name = 'Diag Trace';
```

Ejercicio 3.3: Muestra el contenido del fichero `alert_orcl.log` que se encuentra en ese directorio y abre el **fichero de traza** que se indica en el mensaje de interbloqueo (está hacia el final del fichero). El fichero de traza contiene gran cantidad de información, incluyendo el volcado de la memoria del proceso cancelado. Esta información puede ser de gran utilidad para determinar las causas y plantear la solución de problemas de interbloqueo de las aplicaciones. Recorre este fichero para ver el tipo de información que se proporciona. En particular, puedes identificar la información relacionada con los procesos que intervienen en el interbloqueo

(*deadlock graph*), las sesiones y consultas SQL que se estaban ejecutando en cada una de ellas cuando se produjo el interbloqueo.

Incluye en la memoria de este ejercicio la información que consideres relevante de este fichero.

3.4. Consulta desde Enterprise Manager.

El *log* de alertas también lo puedes consultar desde Enterprise Manager: en la pestaña de inicio (Home) selecciona el enlace *Alert log contents*. Carga las últimas 50 entradas del log para mostrar la información del interbloqueo (el interbloqueo debería aparecer la primera).

4. El modo *archivelog*.

Los ficheros de *redo log* sirven, entre otras cosas, para recuperar sin perder datos el estado de la BD cuando hay un fallo de la instancia. Sin embargo, por sí solos no permiten recuperar el estado de la BD sin pérdidas cuando se restauran datafiles por errores de disco. Esto se debe a que los ficheros de *redo log* se reutilizan cíclicamente.

Para que la BD se pueda recuperar sin pérdida de datos incluso en el caso de errores de disco, **se deben archivar los ficheros de *redo log***. Con este fin, se debe poner la BD en modo *archivelog*. Vamos a hacerlo con el siguiente ejercicio.

Ejercicio 4: Para poner la BD en modo *archivelog* hay que parar la instancia de forma limpia (es decir, que no quede corrompida) y después arrancarla en un modo especial para activarlo. El proceso es más complejo de lo que se indica en este ejercicio porque, además de los pasos básicos, hay que decidir cuidadosamente si se va a multiplexar el archivo de *redo log,* su ubicación en los discos y el nombre de los ficheros de archivo. Además, después de cambiar el modo de la BD hay que hacer copias de seguridad inmediatamente. Sigue los siguientes pasos:

1. Para la instancia de forma limpia desde SQL*Plus (recuerda que debes utilizar el usuario SYS como SYSDBA):

```
SHUTDOWN IMMEDIATE;
```

2. Arranca de nuevo la instancia en modo `MOUNT`:

```
STARTUP MOUNT;
```

3. Modifica la BD para activar el modo *archivelog*:

```
ALTER DATABASE ARCHIVELOG;
```

4. Ahora ya se puede abrir la BD de nuevo:

```
ALTER DATABASE OPEN;
```

5. Ejecuta el siguiente comando para comprobar el modo archivelog:

```
archive log list;
```

Adjunta el resultado a la memoria del ejercicio. Por ejemplo, con este comando puedes comprobar que, como está activado el archivado de ficheros de *redo log*, la siguiente secuencia a archivar coincide con la actual. El comando `archive log` permite modificar las características del modo archivelog, aunque no lo veremos en este curso.

6. Ahora vamos a forzar el archivado del fichero *redo log* actual:

```
alter system archive log current;
```

7. Comprueba que se ha generado el archivo:

```
select name from v$archived_log;
```

Comprueba que se ha generado en el sistema de ficheros: muestra en un terminal el contenido del directorio mostrado en la consulta anterior (`ls -l`) y añádelo a la memoria del ejercicio.

8. Vamos a verificar que efectivamente se guarda el *redo log*. Ahora abre otra sesión con otro usuario y realiza operaciones de modificación de datos. Por ejemplo:

```
$ sqlplus usuario1/adbd@192.168.56.101/orcl
SQL> create table mitabla (c1 integer);
SQL> insert into mitabla
(select rownum from dual connect by level < 10000);
SQL> commit;
```

9. En la sesión del usuario SYS vuelve a archivar el fichero de *redo log* actual y comprueba el tamaño del nuevo archivo generado:

```
alter system archive log current;
select name from v$archived_log;
```

Añade a la memoria del ejercicio el resultado de esta consulta y un nuevo listado del directorio de archivos de redo log.