PRÁCTICA 7: BLOQUEOS Y TRANSACCIONES

Javier Guzmán Muñoz y Jorge Villarrubia Elvira

APARTADO 1: BLOQUEOS (SELECT)

1. Creamos la tabla “cuenta” desde SQLDeveloper e insertamos las tuplas indicadas. Hacemos commit.

```
CREATE TABLE cuentas (
numero number primary key,
 saldo number not null
 );
INSERT INTO cuentas VALUES (123, 400);
INSERT INTO cuentas VALUES (456, 300);
COMMIT;
```

2. Abrimos dos sesiones de SQLPlus desde consola.

Obtenemos la siguiente salida por consola (es la misma para las dos consolas):

```
ENTORNO ORACLE 11G client
c:\hlocal>sqlplus
SQL*Plus: Release 11.2.0.1.0 Production on Vie Dic 14 13:29:52 2018
Copyright (c) 1982, 2010, Oracle.  All rights reserved.

Introduzca el nombre de usuario: DG009@BDd/DG009PWD
Conectado a:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit
Production
With the Partitioning, OLAP, Data Mining and Real Application Testing
options
```

3. Deshabilitamos la autoconfirmación en ambas sesiones.

```
SQL> SET AUTOCOMMIT OFF;
```

4. Desde la sesión 1 aumentamos el saldo de la cuenta 123 en 100 euros.

```
SQL> UPDATE cuentas SET saldo = saldo + 100 where numero = 123;

1 fila actualizada.
```

5. Desde la sesión 2 consultamos el saldo de la cuenta 123.

Obtenemos el siguiente resultado:

```
SQL> select saldo from cuentas where numero = 123;


     SALDO
----------
       400
```

Como aún no hemos realizado commit en la sesión 1 el valor del saldo está aún sin actualizar.

6. Desde la sesión 1 confirmamos los datos:

```
SQL> commit;

Confirmaci¾n terminada.
```

7. Desde la sesión 2 consultamos de nuevo el valor del saldo de la cuenta 123

```
SQL> select saldo from cuentas where numero = 123;


     SALDO
----------
       500
```

Comprobamos que después de comprometer los datos desde la sesión 1 aparecen actualizados en la otra sesión, como era de esperar.

APARTADO 2: BLOQUEOS (UPDATE)

1. Deshabilitamnos la autoconfirmación en la sesión 2:

```
SQL> SET AUTOCOMMIT OFF;
```

2. Incrementamos en 100 euros el saldo de la cuenta 123 desde la sesión 1.

```
SQL> UPDATE cuentas SET saldo = saldo + 100 where numero = 123;


1 fila actualizada.
```

3. Desde la sesión 2 intentamos aumentar en 200 euros el saldo de la cuenta 123.

```
SQL> UPDATE cuentas SET saldo = saldo + 200 where numero = 123;
```

La segunda sesión se queda en espera y no ejecuta nada, ya que está esperando una confirmación desde la otra sesión. Al estar el nivel de aislamiento de Oracle por defecto en ReadCommited, no se nos permite la lectura sucia y por tanto no podemos consultar un valor que está siendo modificado.

4. Desde la sesión 1 confirmamos la actualización de los datos:

```
SQL> commit;

Confirmaci¾n terminada.
```

Observamos que la sesión 2 deja de estar en espera y nos confirma la actualización.

```
1 fila actualizada.
```

5. Desde la sesión 1 consultamos el saldo de la cuenta 123

```
SQL> select saldo from cuentas where numero = 123;


     SALDO
----------
       600
```

Solo ha comprometido el primer cambio ya que desde la sesión 2 no hemos hecho commit.

6. Hacemos commit en la sesión 2:

```
SQL> commit;

Confirmaci¾n terminada.
```

7. Volvemos a consultar el saldo de la cuenta 123 desde la sesión 1

```
SQL> select saldo from cuentas where numero = 123;

     SALDO
----------
       800
```

Ahora en la sesión 1 sí nos aparecen los datos actualizados.

APARTADO 3: BLOQUEOS (DEADLOCK)

Al hacer los updates cruzados ambas sesiones se quedan a la espera de un commit en la otra, se ha producido un interbloqueo y Oracle detecta como error la consulta en una de las dos sesiones, en este caso es la 1.

En la sesión 1 tendriamos:

```
SQL> update cuentas set saldo = saldo + 100 where numero = 123;


1 fila actualizada.


SQL> update cuentas set saldo = saldo + 300 where numero = 456;

update cuentas set saldo = saldo + 300 where numero = 456
       *

ERROR en lÝnea 1:

ORA-00060: detectado interbloqueo mientras se esperaba un recurso
```

En la sesión 2 tenemos:

```
SQL> update cuentas set saldo = saldo + 200 where numero = 456;


1 fila actualizada.


SQL> update cuentas set saldo = saldo + 400 where numero = 123;
```

APARTADO 4: NIVELES DE AISLAMIENTO

Como por defecto tenemos read commited como nivel de aislamiento, hasta ahora no nos estaba haciendo lectura sucia, pues cuando actualizamos algo en una transacción sin hacer commit y lo consultamos desde otra no nos muestra el resultado actualizado. Sin embargo, sí que permite lecturas fantasma y no repetibles.

Tras ejecutar las instrucciones de este apartado obtenemos lo siguiente en las consolas de ambas sesiones:

En la sesión 1:

```
SQL> ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE;


Sesi¾n modificada.


SQL> SELECT SUM(saldo) FROM cuentas;


SUM(SALDO)
----------
      1800
```

En la sesión 2:

```
SQL> UPDATE cuentas SET saldo=saldo +100;


2 filas actualizadas.


SQL> COMMIT;


Confirmaci¾n terminada.
```

En la sesión 1:

```
SQL> SELECT SUM(saldo) FROM cuentas;


SUM(SALDO)
----------
      1800
```

No detecta la actualización realizada en la sesión 2 porque como estamos en nivel de aislamiento serializable, el aislamiento es máximo y hasta que no se concluya la transacción T1 no se contemplarán cambios realizados por otras transacciones, ya que este nivel de aislamiento no permite lecturas no repetidas.

En la sesión 1:

```
SQL> ALTER SESSION SET ISOLATION_LEVEL = READ COMMITTED;


Sesi¾n modificada.
```

En la sesión 1:

```
SQL> SELECT SUM(saldo) FROM cuentas;


SUM(SALDO)
----------
      2000
```

Ahora ya nos muestra las actualizaciones hechas en T2 porque este modo de aislamiento sí que permite las lecturas no repetidas.

En la sesión 2:

```
SQL> UPDATE cuentas SET saldo=saldo +100;


2 filas actualizadas.


SQL> COMMIT;


Confirmaci¾n terminada.
```

En la sesión 1:

```
SQL> SELECT SUM(saldo) FROM cuentas;


SUM(SALDO)
----------
      2200
```

Como era de esperar, se contemplan las actualizaciones realizadas en T2 pues este nivel de aislamiento sí que lo permite.

APARTADO 5: TRANSACCIONES

1. Creación de tablas

```
CREATE TABLE butacas (  id number(8) primary key,
                        evento varchar(30),
                        fila varchar(10),
                        columna varchar(10)) ;
CREATE TABLE reservas(  id number(8) primary key,
                        evento varchar(30),
                        fila varchar(10),
                        columna varchar(10)) ;
```

Salida por consola:

```
Table BUTACAS creado.
Table RESERVAS creado.
```

```
CREATE SEQUENCE Seq_Butacas INCREMENT BY 1 START WITH 1 NOMAXVALUE;
CREATE SEQUENCE Seq_Reservas INCREMENT BY 1 START WITH 1 NOMAXVALUE;
```

Salida por consola:

```
Sequence SEQ_BUTACAS creado.

Sequence SEQ_RESERVAS creado.
```

2. Inserción de valores

```
INSERT INTO butacas VALUES (Seq_Butacas.NEXTVAL,'Circo','1','1');

INSERT INTO butacas VALUES (Seq_Butacas.NEXTVAL,'Circo','1','2');

INSERT INTO butacas VALUES (Seq_Butacas.NEXTVAL,'Circo','1','3');

COMMIT;
```

Salida por consola:

```
1 fila insertadas.

1 fila insertadas.

1 fila insertadas.

Confirmación terminada.
```

3. Probamos el scropt dado realizando los cambiones necesarios:

Hemos tenido que cambiar el tamaño de la variable v_error a 5 y cambiar las direcciones al directorio donde tenemos almacenados los scripts preguntar y no_preguntar.

Obtenemos el siguiente resultado:

```
INFO: Se intenta reservar.


Procedimiento PL/SQL terminado correctamente.


SCRIPT_COL
---------------------------
"C:\hlocal\preguntar.sql"
```

```
V_ERROR
--------------------------------------------------------------------------
----------
false

'¿Confirmar la reserva?'
s
INFO: Localidad reservada.

Procedimiento PL/SQL terminado correctamente.
Confirmación terminada.
```

4. Si volvemos a ejecutar el script obtenemos un error. No podemos volver a reservar la misma localidad.

```
Procedimiento PL/SQL terminado correctamente.
Confirmación terminada.
ERROR: La localidad ya está reservada.

Procedimiento PL/SQL terminado correctamente.

SCRIPT_COL
----------------------------
"C:\hlocal\no_preguntar.sql"


V_ERROR
--------------------------------------------------------------------------
----------
true

n
INFO: No se ha reservado la localidad.
```

```
Procedimiento PL/SQL terminado correctamente.

Confirmación terminada.
```

5. Si cambiamos la columna 1 por un 4, nos sale un mensaje de error, puesto que no es posible reservar un asiento no válido como es este.

```
ERROR: No existe esa localidad.


Procedimiento PL/SQL terminado correctamente.


SCRIPT_COL
----------------------------
"C:\hlocal\no_preguntar.sql"



V_ERROR
--------------------------------------------------------------------------------
----------
true


n
INFO: No se ha reservado la localidad.


Procedimiento PL/SQL terminado correctamente.

Confirmación terminada.
```

6, 7 y 8: Se ejecuta el script en la primera consola dejando en espera la confirmación.

Se abre una segunda consola y se ejecuta el script reservando la misma butaca pero confirmando la reserva en este caso.

Esto es lo que obtenemos al confirmar la reserva desde la segunda consola.

```
INFO: Se intenta reservar.


Procedimiento PL/SQL terminado correctamente.


SCRIPT_COL
----------------------------
"C:\hlocal\preguntar.sql"



V_ERROR
--------------------------------------------------------------------------------
----------
false

'¿Confirmar la reserva?'
s
INFO: Localidad reservada.


Procedimiento PL/SQL terminado correctamente.

Confirmación terminada.
```

Al confirmar la reserva del punto 6 también nos deja confirmarla aunque ya la habíamos reservado desde otra consola.

Salida por **consola**:

```
INFO: Se intenta reservar.


Procedimiento PL/SQL terminado correctamente.


SCRIPT_COL
----------------------------
"C:\hlocal\preguntar.sql"



V_ERROR
--------------------------------------------------------------------------------
----------
false


'¿Confirmar la reserva?'
s
INFO: Localidad reservada.


Procedimiento PL/SQL terminado correctamente.
Confirmación terminada.
```

Para reseolverlo modificamos el script añadiendo de nuevo las **comprobaciones sobre la existencia y disponibilidad** de las butacas (realmente la existencia no será necesaria para resolver este error pero sí si se hiciese un delete de ese dato desde otra consola) tras mostrar la ventana de diálogo en la que se pide confirmar. Para probarlo usamos la fila 1 y columna 3 que aún no estaba reservada.

Salida por consola 1 (Desde la que dejamos en espera la confirmación):

```
Procedimiento PL/SQL terminado correctamente.

Confirmación terminada.

INFO: Se intenta reservar.


Procedimiento PL/SQL terminado correctamente.


SCRIPT_COL

----------------------------

"C:\hlocal\preguntar.sql"



V_ERROR

--------------------------------------------------------------------------
----------

false


'¿Confirmar la reserva?'

s

ERROR: La localidad ya está reservada.


Procedimiento PL/SQL terminado correctamente.

INFO: No se ha reservado la localidad.


Procedimiento PL/SQL terminado correctamente.

Confirmación terminada.
```

Salida por consola 2 (Desde la que confirmamos):

```
INFO: Se intenta reservar.


Procedimiento PL/SQL terminado correctamente.


SCRIPT_COL
----------------------------
"C:\hlocal\preguntar.sql"



V_ERROR
--------------------------------------------------------------------------
----------
false

'¿Confirmar la reserva?'
s
Procedimiento PL/SQL terminado correctamente.
INFO: Localidad reservada.

Procedimiento PL/SQL terminado correctamente.
Confirmación terminada.
```

SCRIPT MODIFICADO: (En rojo las modificaciones realizadas)

```
--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET serveroutput ON size 1000000;
set define on;
SET echo OFF;
SET verify OFF;
def v_evento='Circo';
def v_fila='1';
def v_columna='3';
```

```
variable v_error char(5)

/

declare

  v_existe varchar(20) default null;

begin

  select count(*) into v_existe from butacas where evento='&v_evento' and fila='&v_fila' and
columna='&v_columna';

  if v_existe<>'0' then

    select count(*) into v_existe from reservas where evento='&v_evento' and fila='&v_fila' and
columna='&v_columna';

    if v_existe='0' then

      dbms_output.put_line('INFO: Se intenta reservar.');

      :v_error:='false';

    else

      dbms_output.put_line('ERROR: La localidad ya está reservada.');

      :v_error:='true';

    end if;

  else

    dbms_output.put_line('ERROR: No existe esa localidad.');

    :v_error:='true';

  end if;

end;

/

col SCRIPT_COL new_val SCRIPT

select decode(:v_error,'false','"C:\hlocal\preguntar.sql"','"C:\hlocal\no_preguntar.sql"') as
SCRIPT_COL from dual;

print v_error

--prompt 'Valor script: '&SCRIPT

@ &SCRIPT

prompt &v_confirmar
```

```
declare
 v_existe varchar(20) default null;
begin
 select count(*) into v_existe from butacas where evento='&v_evento' and fila='&v_fila' and
columna='&v_columna';
 if v_existe<>'0' then
  select count(*) into v_existe from reservas where evento='&v_evento' and fila='&v_fila' and
columna='&v_columna';
  if v_existe='0' then
   :v_error:='false';
  else
   dbms_output.put_line('ERROR: La localidad ya está reservada.');
   :v_error:='true';
  end if;
 else
  dbms_output.put_line('ERROR: No existe esa localidad.');
  :v_error:='true';
 end if;
end;

/
begin
 if '&v_confirmar'='s' and :v_error='false' then
  insert into reservas values (Seq_Reservas.NEXTVAL,'&v_evento','&v_fila','&v_columna');
  dbms_output.put_line('INFO: Localidad reservada.');
 else
  dbms_output.put_line('INFO: No se ha reservado la localidad.');
 end if;
end;
/
COMMIT;
```