Entrega práctica 6: Restricciones e índices.

Grupo: DG04

Autor: Manuel Ortega Salvador.
Autor: Francisco Javier Blázquez Martínez.

Apartado 1: Bloqueos (select)

1.1.- Creación de tablas e insercción de datos:

```
CREATE TABLE cuentas
(
   numero  number primary key,
   saldo      number not null
);

INSERT INTO cuentas VALUES (123, 400);
INSERT INTO cuentas VALUES (456, 300);
COMMIT;

-- Respuesta del servidor:
Table CUENTAS creado.
1 fila insertadas.
1 fila insertadas.
Confirmación terminada.
```

1.2.- Abrir sesiones de sqlplus:

Ejecutamos los siguientes comandos desde la consola de oracle 11g:
C:\hlocal> sqlplus DG04/DG04PWD@BDd (En la terminal T1)
C:\hlocal> sqlplus DG04/DG04PWD@BDd (En la terminal T2)

1.3.- Prueba de concurrencia en las sentencias select:

Se muestran las instrucciones ejecutadas por ambas terminales secuencialmente. T1 indica que esta fue ejecutada en la terminal 1, T2 indica lo proprio para la terminal 2.

```
T1> SET AUTOCOMMIT OFF;
T2> SET AUTOCOMMIT OFF;
T1> update cuentas set saldo=saldo+100 where numero=123
T2> select saldo from cuentas where numero=123;
      SALDO
      ----------
      400                -- Sin modificaciones
T1>COMMIT;
T2> select saldo from cuentas where numero=123;
      SALDO
      ----------
      500                -- Ahora se aprecian los cambios
```

Lo que sucede es que en Oracle vamos a tener la opción de aislamiento por defecto de ‘read commited’, entonces, en las consultas se garantiza la integridad de los datos leídos pues solo se muestran tras ser comprometidos pero no se garantiza la consistencia con lecturas anteriores. Esto es, no se puede dar el fenómeno de lectura sucia pero si se puede dar el de lectura no repetible. Podemos observar modificaciones en consultas aun sin modificar nosotros los datos.

Apartado 2: Bloqueos (update)

Para este apartado partimos de tener ya nuestras tablas creadas y dos sesiones de sqlplus abiertas. Se muestra la ejecución de instrucciones en su orden e indicando la terminal desde la que se ejecutaron

T1> SET AUTOCOMMIT OFF
T2> SET AUTOCOMMIT OFF
T1> update cuentas set saldo=saldo+100 where numero=123
T2> update cuentas set saldo=saldo+200 where numero=123

//Llegados a este punto la ejecución del update de T2 queda en espera, hay un bloqueo a nivel de //fila por el update que se hizo en T1 sobre la cuenta naranja 123 de ING Direct (automático en Oracle).

T1> COMMIT;

//Confirmamos los cambios de T1 y automáticamente el update de T2 se ejecuta (aunque no se //compromete de momento.

T1> select saldo from cuentas where numero=123;
SALDO
---
600 -- Se muestran solo los cambios de T1

//Se muestran los 100€ de incremento de T1 pero no los 200€ de incremento de T2 aunque su //sentencia de update ya haya terminado (el nivel de aislamiento es read commited). La instrucción //que estaba parada en T2 se puede ejecutar precisamente porque comprometer los cambios en T1 //libera el bloqueo sobre esa tupla.

T2> COMMIT;
T1> select saldo from cuentas where numero=123;
SALDO
---
800 -- Se suma el incremento en T2

Apartado 3: Bloqueos (deadlock)

T1> SET AUTOCOMMIT OFF
T2> SET AUTOCOMMIT OFF

T1> update cuentas set saldo=saldo+100 where cuenta=123 --1 fila actualizada.
T2> update cuentas set saldo=saldo+200 where cuenta=456 --1 fila actualizada.

T1> update cuentas set saldo=saldo+300 where numero=456
//La terminal de T1 queda en espera a que T2 comprometa los cambios. Hay un bloqueo a
// nivel de fila por el update a la cuenta 456 en T2.

T2> update cuentas set saldo=saldo+400 where numero=123
//Sucede ahora lo mismo pero a la inversa, T2 tiene que esperar a que T1 comprometa cambios que
//a su vez espera a que T1 termine. Se da un interbloqueo. Oracle lo detecta automáticamente.

T1> ERROR en lÝnea 1:
ORA-00060: detectado interbloqueo mientras se esperaba un recurso

//Cancela el segundo update (en la cuenta 456) de T1 para evitar el interbloqueo pero T2 sigue
//esperando a que T1 comprometa los cambios de su update en la cuenta 123.

Apartado 4: Niveles de aislamiento

T1:> SET AUTOCOMMIT OFF
T2:> SET AUTOCOMMIT OFF

//4.1.- NIVEL DE AISLAMIENTO SERIALIZABLE:

T1:> ALTER SESSION SET ISOLATION_LEVEL = SERIALIZABLE -- Sesi¾n modificada.
T1:> SELECT SUM(saldo) FROM cuentas -- (400 + 300)
SUM(SALDO)

700
T2:> UPDATE cuentas SET saldo=saldo +100 --2 filas actualizadas.
T2:> COMMIT --Confirmaci¾n terminada.
T1:> SELECT SUM(saldo) FROM cuentas
SUM(SALDO)

700

//En este nivel de aislamiento (más restrictivo) no se permite a otras transacciones modificar los
//datos que manejamos en la nuestra propia. Esto es, no dejamos que aunque la transacción 2 haga
//confirmación de sus cambios se reflejen en nuestra transacción. Ambas son independientes.

//4.2.- NIVEL DE AISLAMIENTO READ COMMITED:

T1:> ALTER SESSION SET ISOLATION_LEVEL = READ COMMITTED --Sesi¾n modificada.
T1:> SELECT SUM(saldo) FROM cuentas -- (500 + 400)
SUM(SALDO)

900
T2:> UPDATE cuentas SET saldo=saldo +100 -- 2 filas actualizadas.
T2:> COMMIT -- Confirmaci¾n terminada.
T1:> SELECT SUM(saldo) FROM cuentas
SUM(SALDO)

1100

//En este nivel sin embargo (por defecto en oracle) dejamos que otras transacciones modifiquen
//los datos que manejamos si tenemos garantías de que son consistentes, esto es, han sido compro-
//metidos.

Apartado 5: Transacciones

5.1.- Creación de tablas y secuencias:

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

Tabla creada.

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

Tabla creada.

```
CREATE SEQUENCE Seq_Butacas  INCREMENT BY 1 START WITH 1 NOMAXVALUE;
```
Secuencia creada.

```
CREATE SEQUENCE Seq_Reservas INCREMENT BY 1 START WITH 1 NOMAXVALUE;
```
Secuencia creada.

5.2.- Insercción de valores:

```
INSERT INTO butacas VALUES (Seq_Butacas.NEXTVAL,'Circo','1','1');  -- 1 fila creada.
INSERT INTO butacas VALUES (Seq_Butacas.NEXTVAL,'Circo','1','2');  -- 1 fila creada.
INSERT INTO butacas VALUES (Seq_Butacas.NEXTVAL,'Circo','1','3');  -- 1 fila creada.

COMMIT;  -- Confirmaci¾n terminada.
```

5.3.- Reservar la butaca de la fila 1, columna 1 con el script:

```
SQL> @'C:\hlocal\Pr7\scripts\script.sql'

:V_ERROR
--------------------------------
INFO: Se intenta reservar.
Procedimiento PL/SQL terminado correctamente.

SCRIPT_COL
--------------------------------------
"C:\hlocal\Pr7\scripts\preguntar.sql"
V_ERROR
--------------------------------
false                                   <- Ningún tipo de error!
```

'┬┐ Confirmar la reserva?'
s

SCRIPT_COL

"C:\hlocal\Pr7\scripts\preguntar.sql"

INFO: Localidad reservada. <---------- !!!!

Procedimiento PL/SQL terminado correctamente.

Confirmaci¾n terminada. -- Por el commit del final del script

5.4.- Intentar reservar de nuevo la misma butaca:

SQL> @'C:\hlocal\Pr7\scripts\script.sql'

Confirmaci¾n terminada.

ERROR: La localidad ya est├í reservada. <---------- !!!

Procedimiento PL/SQL terminado correctamente.

SCRIPT_COL

"C:\hlocal\Pr7\scripts\no_preguntar.sql"

V_ERROR
---
true <- ---- Se obtiene un error!!!

n

SCRIPT_COL

"C:\hlocal\Pr7\scripts\no_preguntar.sql"

INFO: No se ha reservado la localidad.

Procedimiento PL/SQL terminado correctamente.
Confirmaci¾n terminada.

5.5.- Intentar reservar una butaca inexistente:

SQL> @'C:\hlocal\Pr7\scripts\script.sql'

Confirmaci¾n terminada.

ERROR: No existe esa localidad. <---------- !!!

Procedimiento PL/SQL terminado correctamente.
SCRIPT_COL

"C:\hlocal\Pr7\scripts\no_preguntar.sql"
V_ERROR
---
true

n
SCRIPT_COL

"C:\hlocal\Pr7\scripts\no_preguntar.sql"

INFO: No se ha reservado la localidad.

Procedimiento PL/SQL terminado correctamente.
Confirmaci¾n terminada.

5.6/7 .- Reservas concurrentes de una misma butaca:

//Cambiamos las variables de fila y columna del script para reservar el asiento fila 1 columna 2

T1> SQL> @'C:\hlocal\Pr7\scripts\script.sql'
T2> SQL> @'C:\hlocal\Pr7\scripts\script.sql'
T2> SQL> ‘Confirmar reserva?’ s
T1> SQL> ‘Confirmar reserva?’ s

//Todo parece terminar correctamente. Consultamos la tabla reservas para ver que ha sucedido.

SQL> select * from reservas;

ID EVENTO	FILA	COLUMNA
2 Circo	1	1
4 Circo	1	2
3 Circo	1	2

//Hemos comprometido la misma butaca a dos personas distintas!!

5.8.- Resolver el apartado anterior:

//Nuestro script lo primero que hace es ver que la butaca existe y ver que no está reservada, enton-
//ces la ofrece exigiendo una confirmación para reservarla. Es en este tiempo de espera de confir-
//mación cuando algún usuario puede reservarla. ¿Cómo saber si alguien la reserva antes? No
//hemos modificado el nivel de aislamiento, simplemente al ser read commited, si alguien reserva la
//butaca que nosostros queremos antes que nosotros (y confirma los cambios antes, llega al commit
//del final del script) entonces esos cambios serán visibles para nosotros.
//Añadimos entonces otra comprobación más que, tras confirmar que queremos esa butaca, vea si
//hemos sido demasiado lentos. Miramos si ya está presente en la tabla reservas la butaca. En caso
//contrario, la reservamos nosotros.

-- Llegados a este punto del script el usuario ya ha introducido una confirmación (s/n):

```
declare
  v_existe varchar(20) default null; -- Añadimos una variable para saber si está en reservas
begin
  select count(*) into v_existe from reservas where evento='&v_evento' and fila='&v_fila' and
columna='&v_columna';

  if '&v_confirmar'='s' and :v_error='false' and v_existe='0' then -- Comprobamos que sea 0

    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;
```

(Testeado en los laboratorios)