BASES DE DATOS Fecha Entrega 30-05-2014

Profesor: Héctor Gómez Gauchía

Práctica 4: USO de Bloques PL/SQL

Desarrollo de la práctica

NOTA: Se valorará la claridad del código. Para ello ponga comentarios en cada consulta y en los pasos que da en los procedimientos. Se puede hacer con /* ..comentario..*/ , o en línea completa empezando con '--'.

APARTADO 1.-

Para familiarizarse con los bloques PL/SQL ejecute el ejemplo que se pide:

a) – Ejecuta el `Ejeprac2.sql` para restaurar la BDejemplo.

b) – Crear un procedimiento en la BD y ejecutarlo:

- Abrir en el sql-Developer el fichero ejemplo: `cursor3.sql`
- Ejecutar el fichero:

Este fichero crea un procedimiento dentro de Oracle llamado cursor3 (NO lo ejecuta)

- *Consultar el procedimiento creado*:
  - Bajo la conexión BD, en carpeta Procedimientos: aparecen todos los procedimientos creados:

ábrelo -> Se abre en un editor diferente: permite compilar en os modos, ejecutar, debug

- En ese editor: Ejecutarlo con icono Triángulo: aparece *un cuadro con varios datos a introducir*.

En este caso no hay datos, pulsar aceptar. Se ejecuta y da resultado en ventana LOG, abajo.

Ejecutará el procedimiento entero: `begin cursor3; end;`

c) – Para ver con instrucción SQL qué objetos (procedimientos en este caso) tenemos creados ejecuta:

```
select object_type, object_name
from user_objects
where object_type ='PROCEDURE';
```

d) – Comprueba qué se ha modificado en la BDejemplo con consultas a la tabla correspondiente.

Entregar: Fichero prac41.docx con el resultado de lo ejecutado en cada paso. Explica qué ha hecho el cursor3.sql paso a paso, tanto en lo que da de salida como en la BDEjemplo.

APARTADO 2.-

Se pide hacer un procedimiento PL/SQL para tu BD propia que haga una actualización usando un *cursor*. Debe incluir al menos tres condiciones diferentes sobre la actualización, del mismo tipo que en cursor3 hay para los DNI terminados en 2 o en 4. Comprueba que funciona ejecutando algún ejemplo.

Entregar: Ficheros prac42.sql con las instrucciones necesarias en sql.

Fichero prac42.docx : describe lo que hace tu procedimiento y resultado de la ejecución

APARTADO 3.-

Para familiarizarse con los triggers ejecute el ejemplo que se proporciona:

a) – **ejecuta el** `Ejeprac2.sql` para restaurar la BDejemplo de nuevo.

b) – **Para crear el disparador o trigger en el sql-Developer**:

b.1) Antes, debes crear una secuencia que usa el trigger:

→ **Pero** falta el privilegio "create sequence". Se entra en el ADMINUSER
- Dentro escribe la instrucción:

```
GRANT create sequence TO tuUsuario;
```

- **Sales** del ADMINUSER

→ **Ahora** ya podemos crear la secuencia:

a).- Vamos a carpeta "Secuencias" en tu conexión
y botón dcho. Escogemos nueva secuencia, no hace falta ningún argumento
solo el nombre : `cuentapremio`

b).- También existe la sentencia:

```
create sequence cuentapremio;
```

b.2) Abre en el sql-Developer el fichero ejemplo: `trigger3.sql`

b.3) Comprobar que ha sido creado, en la carpeta Disparadores (refresca si no aparece)
o con sentencia:

```
select object_type, object_name
from user_objects
where object_type ='TRIGGER';
```

b.4) Abrirlo desde el menú de Disparadores: se abre un editor donde se puede editar y compilar (ruedas), pero no se puede ejecutar como en el procedimiento. Hay que provocar el disparo (siguiente apartado).
→ Para comprobar si ha dado errores: una ventana abajo da mensajes de error.

c) Sigue estos pasos:

-- Provoca el disparo del trigger3 ejecuta:

```
update invierte
set cantidad = 1000001
where dni = '00000003';
```

-- Comprueba si se ha disparado: haz consulta sql adecuada

-- Ejecuta ahora esta instrucción: (no debería dispararlo)

```
update invierte
 set cantidad = 100000
where dni = '00000003';
```

-- Comprueba si se ha disparado: haz consulta sql adecuada

-- Modifica el procedimiento cursor3 para que se dispare trigger3 cuando se ejecuta cursosr3.
-- Comprueba si se ha disparado: haz consulta sql adecuada

Entregar: Fichero prac43.docx: -Explica qué hace el trigger. – Muestra el resultado de ejecutar cada uno de los pasos

APARTADO 4.-

a) – Se pide hacer un *trigger* para tu BD propia (de la prac2) que inserte una fila en una tabla llamada TRAZA *antes* de que se inserte o modifique una fila en una de sus tablas. Se ejecutará cuando cumpla una condición sobre algún valor de la fila afectada de dicha tabla. TRAZA debe tener un atributo con el valor de la clave de la fila afectada, otro atributo que sea el nombre de la tabla afectada y otro con la fecha del sistema (sysdate).
b) – Ejecuta varios ejemplos que activen el trigger y compruebe con consultas los resultados en sus tablas y en TRAZA.

Entregar: Ficheros prac44a.sql con las instrucciones necesarias en sql.
Fichero prac44b.docx con el resultado de hacer lo pedido

APARTADO 5.-

Básandote en el ejemplo de ejecución de Oracle visto en clase, vas a hacer dos Ejemplos de ejecución. Para cada ejemplo, ejecuta en dos sesiones concurrentes, las mismas operaciones de consulta y actualización usando la ***BDejemplo***. El objetivo es provocar dos comportamientos distintos cuando usas los dos niveles de aislamiento : `Read Commited` y `Serializable`
- Ejemplo 1: las dos sesiones con `Read Commited`
- Ejemplo 2: : las dos sesiones con `Serializable`

Para entregar: Haz una tabla con los pasos seguidos en el tiempo, y dos columnas, una para cada sesión, indicando cada paso dado y resultado obtenido. Las respuestas a las preguntas las contestas dentro de la misma tabla. Separa los dos ejemplos en dos tablas:

======= **EJEMPLO 1** ===========

PASOS: Síguelos en el **orden descrito**:
- Abre dos conexiones (en el enunciado: Sesión 1 es la 1ª que abriste)
  - abriendo **sql Developer** dos veces y conectándote con tu usuario
  - después abre un editor: menú Herramientas + Hoja de Trabajo SQL

- en la Sesión 1:
  `SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`
  `set autocommit off;` (siempre puedes comprobarlo con `show autocommit`)
- en la Sesión 2:
  `SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`
  `set autocommit off;`

- en la Sesión 1:
  Una consulta de una tabla (por ejemplo COMPRAS) **que solo devuelva una fila**
  Actualiza la misma fila y el mismo atributo
  Consulta esa fila y atributo: ¿Qué valor ves?

- en la Sesión 2:
  - Consulta esa fila y atributo: ves lo mismo?
  - Actualiza la misma fila y el mismo atributo con otro valor
    -Que sucede?
    Oracle no debería responder : Porqué?

- en la Sesión 1:
  - Escribe un commit

- Consulta esa fila y atributo: que ves?, de donde ha salido?
- Se ha perdido la actualización de sesión 1?
- Cómo solucionarlo? Ver la ejecución del ejemplo 2 a continuación

======= **EJEMPLO 2** ==========

→ Repetir los mismos pasos , teniendo en ambas sesiones

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

- Se ha perdido algo ahora?
- Cómo lo ha solucionado Oracle?

Entregar: Ficheros prac45.sql con la tabla de dos columnas y las respuestas, en texto, a las preguntas dentro de la tabla.

APARTADO 6.-

a) (usa read commited)

Provocar un interbloqueo (deadlock), comprobar resultado haciendo consultas y Arreglarlo

Trabajando sobre la BDEjemplo vamos a provocar el deadlock. Asumiendo que tienes que hacer esas operaciones: ¿Cómo lo evitarías? Demuéstralo ejecutando de nuevo los pasos.

Abre un sqldeveloper (Sesión 1):

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
set autocommit off;
```

Abre otro sqldeveloper (Sesión 2):

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
set autocommit off;
```

Sesión 1:
- actualiza una única fila X de una tabla T
- actualiza la fila X de la tabla T con otro valor

Sesión 2:
- actualiza una única fila Y de una tabla T
- actualiza la fila Y de la tabla T con otro valor

Sesión 1:
- actualiza la fila Y de la tabla T con otro valor (qué pasa?, sigue el siguiente paso)

Sesión 2:
- actualiza la fila X de la tabla T con otro valor

Sesión 1:
- commit;
- consulta los valores de X e Y

Sesión 2:
- commit;
- consulta los valores de X e Y

b) (usa serializable)

Provocar un interbloqueo (deadlock), comprobar resultado haciendo consultas y Arreglarlo.

Repite los mismos pasos con el nivel de aislamiento “serializable”. Responde a lo mismo que en el partado anterior.

c) Indica claramente la diferencia de comportamiento entre a) y b)

Entregar: Ficheros prac46.sql con la tabla de dos columnas y las respuestas, en texto, a las preguntas dentro de la tabla.

APARTADO 7.- (EXTRA para nota)

Los bloqueos explícitos de tablas (LOCK TABLE) obligan a no tener en cuenta los implícitos de Oracle. Normalmente con los implícitos es suficiente.

Hay varios modos, aunque los más usuales son los vistos en clase. Aquí hay un resumen:

ROW SHARE Permite acceso concurrente. No permite hace lock exclusivo.
(es igual que el antigüo SHARE UPDATE)

ROW EXCLUSIVE igual que ROW SHARE pero tampoco permite lock share mode
lo asigna automáticamente en las instrucciones: update, insert, or delete

SHARE permite consultas y prohibe actualizaciones

SHARE ROW EXCLUSIVE bloquea la tabla, permite consultas a filas. No permite otros bloqueos SHARE o para actualizar

EXCLUSIVE permite consultas pero prohibe cualquier otra cosa.

Se pide:
a) Busca en la Web cómo se usan
b) haz ejemplos en el sqldeveloper que demuestren los resultados de sus efectos,
c) Explica los efectos textualmente.

Entregar: Ficheros prac47.sql con las ejecuciones y las respuestas, en texto, a las preguntas.