Práctica 6: Disparadores e índices

Bases de datos

Objetivos

- Programación con PL/SQL avanzado: disparadores (*triggers*).
- Creación y uso de índices para mejorar el rendimiento.

Enunciado

Considérense las siguientes tablas de una base de datos de una pizzería por internet:

pedidos(código, fecha, importe, cliente, notas)

- Almacena cada pedido que los clientes hacen por la web.

- Tipos: **char(6)**, **char(10)**, **number(6,2)**, **char(20)**, **char(1024)**

contiene(pedido, plato, precio, unidades)

- Almacena un código de pedido, cada plato en cada pedido con su precio individual y el número de platos en total. El precio se entiende por cada unidad.

- Tipos: **char(6)**, **char(20)**, **number(6,2)**, **number(2,0)**

auditoría(operación, tabla, fecha, hora)

- Almacena el tipo de operación sobre cada tabla, indicando la fecha y hora en que se realizó.

- Tipos: **char(6)**, **char(50)**, **char(10)**, **char(8)**

- *Sin clave primaria*. ¿Por qué?

Como resultado de esta práctica se debe subir al CV un documento PDF con las instrucciones usadas, disparadores y resultados de las ejecuciones para cada uno de los apartados siguientes:

Apartado 1. Disparador por tabla

a) Crea las tablas anteriores. Más tarde las rellenarás con los datos que veas necesarios para comprobar el resto de apartados.

Solución:

```
DROP TABLE auditoría;
DROP TABLE contiene;
DROP TABLE pedidos;

CREATE TABLE pedidos(código char(6) primary key, fecha char(10),
importe number(6,2), cliente char(20), notas char(1024));
CREATE TABLE contiene(pedido char(6), plato char(20), precio
number(6,2), unidades number(2,0), primary key (pedido, plato));
CREATE TABLE auditoría(operación char(6), tabla char(50), fecha
char(10), hora char(8));
```

b) Crea y comprueba el funcionamiento de un disparador denominado **trigger_pedidos** sobre la tabla **pedidos** de manera que se auditen los cambios producidos por inserciones, borrados y actualizaciones: se incluirá una fila en la tabla **auditoría** con el tipo de operación realizada (**INSERT**, **UPDATE** o **DELETE**), el nombre de la tabla (**pedidos**), la fecha y la hora. Para conseguir estos dos últimos datos se usan las funciones **to_char(sysdate,'dd/mm/yyyy')** y **to_char(sysdate,'hh:mi:ss')** respectivamente. Este disparador se ejecutará *después* de la actualización (**AFTER INSERT OR DELETE OR UPDATE**) y para la tabla global (no por cada fila). Para determinar cuál es la operación en el cuerpo del disparador se usan las comprobaciones **IF INSERTING THEN** o **IF DELETING THEN** o **IF UPDATING THEN**. Un ejemplo de su resultado podría ser:

```
OPERACIÓN TABLA                FECHA      HORA
--------- -------------------- ---------- --------
```

```
INSERT    pedidos              06/01/2015 04:27:06
UPDATE    pedidos              06/01/2015 04:28:30
DELETE    pedidos              06/01/2015 04:29:16
```

Solución:

```
CREATE OR REPLACE TRIGGER trigger_pedidos
AFTER INSERT OR DELETE OR UPDATE ON pedidos
BEGIN
  IF INSERTING THEN
    INSERT INTO auditoría VALUES ('INSERT', 'pedidos',
     to_char(sysdate,'dd/mm/yyyy'), to_char(sysdate,'hh:mi:ss'));
  ELSIF DELETING THEN
    INSERT INTO auditoría VALUES ('DELETE', 'pedidos',
     to_char(sysdate,'dd/mm/yyyy'), to_char(sysdate,'hh:mi:ss'));
  ELSIF UPDATING THEN
    INSERT INTO auditoría VALUES ('UPDATE', 'pedidos',
     to_char(sysdate,'dd/mm/yyyy'), to_char(sysdate,'hh:mi:ss'));
  END IF;
END trigger_pedidos;

INSERT INTO pedidos VALUES ('P1','1-1-1',1,'C1','N1');
UPDATE pedidos SET importe=2;
DELETE FROM pedidos;
```

Apartado 2. Disparador por fila

Para automatizar los pedidos, crea un disparador llamado **trigger_contiene** que se asocie a todas las operaciones posibles de actualización (**INSERT**, **DELETE** y **UPDATE**) sobre la tabla **contiene**, que opere *después* de la modificación y por cada fila (**FOR EACH ROW**). Al insertar una nueva fila en esta tabla, se deberá incrementar el valor del campo **importe** de la tabla **pedidos** con el nuevo valor de la tabla **contiene** (**:NEW.precio**) multiplicado por el número de unidades (**:NEW.unidades**). Si se produce la eliminación (**DELETE**) o modificación (**UPDATE**) de una fila, el importe se debe ajustar según la modificación introducida (restando al antiguo valor **:OLD.precio** el nuevo, o restando el antiguo y sumando el nuevo, respectivamente).

Solución:

```
CREATE OR REPLACE TRIGGER trigger_contiene
AFTER INSERT OR UPDATE OR DELETE ON contiene
FOR EACH ROW
BEGIN
  IF INSERTING THEN
    UPDATE pedidos SET
      importe = importe + :NEW.precio * :NEW.unidades
    WHERE código = :NEW.pedido;
  ELSIF DELETING THEN
    UPDATE pedidos SET
      importe = importe - :OLD.precio * :OLD.unidades
    WHERE código = :OLD.pedido;
  ELSIF UPDATING THEN
    UPDATE pedidos SET
      importe = importe - :OLD.precio * :OLD.unidades
                        + :NEW.precio * :NEW.unidades
    WHERE código = :NEW.pedido;
  END IF;
END;
/
SHOW ERRORS;

INSERT INTO contiene VALUES
```

Apartado 3. Creación y uso de índices

a) Activa la temporización para obtener el tiempo de ejecución de cada una de las siguientes consultas con **`SET TIMING ON`** (anota también estos tiempos en el pdf a entregar). Crea un índice denominado **`index_pedidos`** y que admita duplicados sobre el campo **`cliente`** de la tabla **`pedidos`**. **Ayuda:** consúltese **`CREATE INDEX`**.

Solución:

```
CREATE INDEX index_pedidos ON pedidos(cliente);
```

b) Rellena la tabla pedidos automáticamente con un bucle **`FOR`** con tuplas de la forma: **`(`*`I`*`,'06/01/2015',10.0,'C`*`I`*`',' ')`** donde ***`I`*** es el índice que recorre el bucle desde 1 hasta 300.000. Para concatenar cadenas, usa el *pipe* doble (**`||`**). La coerción de tipos se hace automáticamente.

Solución:

Si el espacio de tablas se queda pequeño, es necesario modificarlo desde ADMINUSER con:

```
ALTER DATABASE DATAFILE 'D:\oracle\DG XX _TABLESPACE' AUTOEXTEND ON
MAXSIZE 1000M;
ALTER DATABASE DATAFILE 'D:\oracle\DG XX _TABLESPACE' RESIZE 1000M;
```

donde ***`XX`*** es el identificador de grupo en el laboratorio (**`01`** ...).

```
DECLARE
  I INT;
BEGIN
  FOR I IN 1..300000 LOOP
    INSERT INTO pedidos VALUES(I,'06/01/2015',10.0,'C'||I,' ');
  END LOOP;
END;
```

c) Mostrar con una instrucción **`SELECT`** los valores de todos los campos de la tabla **`pedidos`** para el cliente con código **`'C300000'`**. Eliminar el índice (usa la instrucción **`DROP INDEX`**) y repetir la consulta. Comparar los resultados de tiempo.

Solución:

```
SET TIMING ON;

SELECT * FROM pedidos WHERE cliente='C300000';

DROP INDEX index_pedidos;

SELECT * FROM pedidos;
```

d) Mostrar con una instrucción **`SELECT`** los valores de todos los campos de la tabla **`pedidos`** para el pedido **`300000`**. Elimina la clave primaria (con la instrucción ALTER TABLE pedidos **`DROP PRIMARY KEY`**). Vuelve a ejecutar la consulta. ¿Qué ocurre?

Solución:

```
SELECT * FROM pedidos WHERE código=300000;

ALTER TABLE pedidos DROP PRIMARY KEY;
```

El tiempo aumenta de forma parecida al apartado anterior. Se comprueba que la clave primaria tiene asociado un índice.

e) Intenta crear un índice sobre una vista. ¿Qué ocurre? ¿Y si la vista es materializada? **Ayuda:** consúltese el concepto de vista materializada en las transparencias del tema 3.

Solución:

```
CREATE VIEW vista_pedidos AS SELECT * FROM pedidos;
CREATE UNIQUE INDEX index_vista_pedidos ON vista_pedidos(cliente);
ERROR en línea 1:
ORA-01702: una vista no es apropiada aquí
```

El problema es que no se puede crear un índice sobre un fichero de datos que no existe; hay que recordar que una vista no es otra cosa que una consulta almacenada en la base de datos. Su resultado se calcula dinámicamente y por ello no tiene sentido crear un índice sobre ella. Sin embargo, sí es posible crear índices sobre vistas materializadas, dado que en este caso sí se almacena el resultado de la consulta, que se actualizará periódicamente. Por lo tanto, podemos crear primero la vista materializada y después el índice de la siguiente forma**:**

```
DROP VIEW vista_pedidos;

CREATE MATERIALIZED VIEW vista_pedidos AS
SELECT * FROM pedidos;

CREATE UNIQUE INDEX index_vista_pedidos ON vista_pedidos(cliente);
```