Práctica 5: Disparadores e índices

Bases de datos

Objetivos

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

Enunciado

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

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

- Esta tabla almacena cada pedido que los clientes hacen por la web.

- Tipos: **char(6),char(10),number(6,2),char(20),char(1024),char(1) ∈ {'S','N'}**

contiene(pedido, plato, precio, unidades)

- Esta tabla 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)

- Esta tabla 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*. Responde el porqué.**

Respuesta: Es posible que se produzca una misma operación sobre la misma tabla en el mismo momento, por lo que habría duplicados. Una clave primaria sobre los cuatro atributos impediría estos duplicados.

Añade las restricciones de integridad que consideréis oportunas (integridad referencial, limitación de valores...)

Como resultado de esta práctica se debe subir al CV un documento PDF con las respuestas a las preguntas planteadas, 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) NOT NULL,
  importe NUMBER(6,2) NOT NULL,
  cliente CHAR(20) NOT NULL,
  notas CHAR(1024),
  especial CHAR(1) CHECK (especial IN ('S', 'N')));

CREATE TABLE contiene(
```

```
  pedido CHAR(6) REFERENCES pedidos(código) ON DELETE CASCADE,
  plato CHAR(20) NOT NULL,
  precio NUMBER(6,2) NOT NULL,
  unidades NUMBER(2,0) NOT NULL,
  PRIMARY KEY (pedido, plato),
  CHECK (precio >= 0 AND unidades > 0));

CREATE TABLE auditoría(
  operación CHAR(6) CHECK (operación IN ('INSERT', 'UPDATE', 'DELETE')),
  tabla CHAR(50) NOT NULL,
  fecha CHAR(10) NOT NULL,
  hora CHAR(8) NOT NULL);
```

b) Crea y comprueba el funcionamiento de un disparador denominado **tr_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 actuales. 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). Comprueba el resultado de la ejecución del disparador con varias instrucciones de prueba. Un ejemplo de su resultado podría ser:

```
OPERACIÓN TABLA                FECHA      HORA
--------- -------------------- ---------- --------
INSERT    pedidos              06/11/2021 04:27:06
UPDATE    pedidos              06/11/2021 04:28:30
DELETE    pedidos              06/11/2021 04:29:16
```

Solución:

```
CREATE OR REPLACE TRIGGER trigger_pedidos
AFTER INSERT OR DELETE OR UPDATE ON pedidos
DECLARE
v_op CHAR(6);
BEGIN
  IF    INSERTING THEN v_op := 'INSERT';
  ELSIF DELETING  THEN v_op := 'DELETE';
  ELSE                 v_op := 'UPDATE';
  END IF;
  INSERT INTO auditoría VALUES(
    v_op,
    'pedidos',
    TO_CHAR(SYSDATE,'dd/mm/yyyy'),
    TO_CHAR(SYSDATE,'hh:mi:ss'));
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 **tr_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 (**AFTER**) 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 a donde corresponda, respectivamente). Ten en cuenta que también puede cambiar el código de pedido al modificar con **UPDATE**. Comprueba los resultados de la ejecución del disparador con varias instrucciones de prueba.

Solución:

```
CREATE OR REPLACE TRIGGER tr_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
    IF :NEW.pedido = :OLD.pedido THEN
      UPDATE pedidos SET
        importe = importe - :OLD.precio * :OLD.unidades
                          + :NEW.precio * :NEW.unidades
        WHERE código = :NEW.pedido;
    ELSE
      UPDATE pedidos SET
        importe = importe - :OLD.precio * :OLD.unidades
        WHERE código = :OLD.pedido;
      UPDATE pedidos SET
        importe = importe + :NEW.precio * :NEW.unidades
        WHERE código = :NEW.pedido;
    END IF;
  END IF;
END;
/
SHOW ERRORS;

INSERT INTO pedidos VALUES('1','2021/01/01',0,'C1','N1','N');
INSERT INTO pedidos VALUES('2','2021/02/02',0,'C2','N2','N');

INSERT INTO contiene VALUES('1','P1',10,2);

UPDATE contiene SET pedido='2', precio=15 WHERE pedido='1';
```

Apartado 3. Disparador ¿mutante?

Escribe un disparador denominado **tr_especial** que actualice el campo **especial** de la tabla **pedidos** cada vez que se inserte (con una instrucción **INSERT**) o modifque un nuevo pedido (con una instrucción **UPDATE**). Al insertar consideramos que se proporcionan todos los valores salvo el del campo especial, que será responsabilidad de este disparador proporcionarle un valor. El pedido se considera especial (campo **especial='S'**) si el importe supera el importe promedio de todos los pedidos (incluido él mismo); en caso contrario no es especial ((campo **especial='N'**). Escribe cómo podría ser con granularidad **FOR EACH ROW**, inserta algunas filas y comprueba el resultado. Actualiza ahora el importe de una de las filas y describe lo que ocurre. Si encuentras problemas al implementarlo así, ¿podrías resolverlo de otro modo? (Ayuda: lee la transparencia sobre temporalidad de eventos).

Solución:

```
CREATE OR REPLACE TRIGGER tr_especial
BEFORE INSERT OR UPDATE ON pedidos
FOR EACH ROW
DECLARE
 v_promedio pedidos.importe%TYPE;
BEGIN
 SELECT (SUM(importe)+:NEW.importe)/COUNT(importe) INTO v_promedio
```

```
  FROM pedidos;
  IF :NEW.importe > v_promedio THEN
    :NEW.especial := 'S';
  ELSE
    :NEW.especial := 'N';
  END IF;
  UPDATE pedidos SET especial = CASE WHEN importe > v_promedio THEN 'S' ELSE 'N' END
WHERE código <> :NEW.código;
END;
```

Este disparador no funciona con una instrucción `UPDATE` por el problema de la mutación de la tabla sobre la que es sensible y pretende actualizar.

Una opción sería pensar en usar un disparador a nivel de sentencia en lugar de `FOR EACH ROW`:

```
CREATE OR REPLACE TRIGGER tr_especial
AFTER INSERT OR UPDATE ON pedidos
DECLARE
  v_promedio pedidos.importe%TYPE;
BEGIN
  SELECT AVG(importe) INTO v_promedio FROM pedidos;
  UPDATE pedidos SET especial = CASE WHEN importe > v_promedio THEN 'S' ELSE
'N' END;
END;
```

Sin embargo, con una actualización como la siguiente:

```
UPDATE pedidos SET importe=500 WHERE código='P005';
```

el problema que aparece es una ejecución en cascada del disparador: como dentro del disparador se llama a `UPDATE`, esto provoca una llamada al mismo disparador y así sucesivamente hasta que se alcance el nivel máximo de llamadas recursivas que se haya definido en Oracle.

Si se hace sensible solo al cambio de importe se podría evitar esta recursión:

```
CREATE OR REPLACE TRIGGER tr_especial
AFTER INSERT OR UPDATE ON pedidos OF importe
DECLARE
  v_promedio pedidos.importe%TYPE;
BEGIN
  SELECT AVG(importe) INTO v_promedio FROM pedidos;
  UPDATE pedidos SET especial = CASE WHEN importe > v_promedio THEN 'S' ELSE
'N' END;
END;
```

No obstante, sigue sin ser un disparador FOR EACH ROW.

Otra posible idea ahora sería desactivar el disparador justo antes de su instrucción `UPDATE`, como en:

```
CREATE OR REPLACE TRIGGER tr_especial
AFTER INSERT OR UPDATE ON pedidos
DECLARE
  v_promedio pedidos.importe%TYPE;
BEGIN
  SELECT AVG(importe) INTO v_promedio FROM pedidos;
  ALTER TRIGGER tr_especial DISABLE;
  UPDATE pedidos SET especial = CASE WHEN importe > v_promedio THEN 'S' ELSE
'N' END;
  ALTER TRIGGER tr_especial ENABLE;
END;
```

pero esto no es posible porque los disparadores no admiten comprometer transacciones (que es lo que ocurre cuando se emite un `ALTER TABLE ... DISABLE TRIGGER;`).

La opción que queda es usar un disparador de tipo `INSTEAD OF`, que reemplaza la acción a realizar: se puede cambiar un `UPDATE` por un `DELETE` y un `INSERT`, evitando así las llamadas recursivas. Sin

embargo, solo se puede aplicar a vistas actualizables. Por tanto, se podría modificar el esquema de la base de datos definiendo pedidos como vista y `t_pedidos` como la tabla de la que toma sus datos del siguiente modo:

```
DROP TABLE pedidos;
CREATE TABLE tabla_pedidos(código CHAR(6) PRIMARY KEY, fecha CHAR(10),
importe NUMBER(6,2) NOT NULL, cliente CHAR(20), notas CHAR(1024), especial
CHAR(1) CHECK (especial IN ('S', 'N')));
CREATE VIEW pedidos AS SELECT * FROM tabla_pedidos;

CREATE OR REPLACE TRIGGER tr_especial_update
INSTEAD OF UPDATE ON pedidos
FOR EACH ROW
DECLARE
  v_promedio pedidos.importe%TYPE;
BEGIN
  SELECT (SUM(importe)+:NEW.importe-:OLD.importe)/COUNT(importe)  INTO
v_promedio FROM pedidos;
  UPDATE pedidos SET especial = CASE WHEN importe > v_promedio THEN 'S' ELSE
'N' END;
  DELETE FROM pedidos WHERE código=:OLD.código;
  INSERT INTO pedidos VALUES (:NEW.código, :NEW.fecha, :NEW.importe,
:NEW.cliente, :NEW.notas, :NEW.especial);
END;
```

El trigger de inserción quedaría como al principio (sin que sea sensible a `UPDATE`):

```
CREATE OR REPLACE TRIGGER tr_especial
BEFORE INSERT ON pedidos
FOR EACH ROW
DECLARE
  v_promedio pedidos.importe%TYPE;
BEGIN
  SELECT (SUM(importe)+:NEW.importe)/COUNT(importe) INTO v_promedio
  FROM pedidos;
  IF :NEW.importe > v_promedio THEN
    :NEW.especial := 'S';
  ELSE
    :NEW.especial := 'N';
  END IF;
  UPDATE pedidos SET especial = CASE WHEN importe > v_promedio THEN 'S' ELSE
'N' END;
END;
```

Ejemplos de ejecución:

```
DELETE FROM pedidos;
INSERT INTO pedidos(código, fecha, importe, cliente) VALUES
('P007',to_date('01/01/2021'), 5000.00, 'Pepe');
UPDATE pedidos SET importe=1100 WHERE código='P007';
select * from pedidos;
```

Apartado 4. 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:** consulta **`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*, '06/01/2022', 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. Si el espacio de tablas se queda pequeño, es necesario modificarlo desde el usuario SYSTEM con:

```
ALTER DATABASE DATAFILE '<Ruta completa al fichero tablespace>' AUTOEXTEND ON
MAXSIZE 1000M;
ALTER DATABASE DATAFILE '<Ruta completa al fichero tablespace>' RESIZE 1000M;
```

Solución:

```
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:** consulta 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);
```

Apartado 5. Encuesta DESweb

Al igual que la encuesta Docentia, esta encuesta es anónima (así que critica todo lo que quieras, no habrá represalias :-)). Cada alumno debe rellenar su encuesta. Accede desde el CV, justo debajo de la imagen del Inspector Gadget en la pestaña General (este es el enlace). En el documento pdf debéis indicar como respuesta: Realizada o No realizada.