BASES DE DATOS

2º CURSO E.U.I. / F.I.

Práctica 4: Estudio del SGBD ORACLE8[1]
Gestión de transacciones

22 DE MAYO DE 2000

[1] Se recomienda que para la realización de esta práctica se revisen los conceptos estudiados en el tema IV (capítulo V del libro de texto).

1. Gestión de transacciones en el SGBD ORACLE8

Concepto de transacción

Una transacción es una secuencia de instrucciones de manipulación de la base de datos que constituye una unidad lógica de ejecución. Una transacción debe satisfacer las propiedades de atomicidad, consistencia, aislamiento y persistencia (ver capítulo V del libro de texto, apartado 5.3.1). El mantenimiento de cada una de estas propiedades es responsabilidad de un módulo del SGBD, la atomicidad y la persistencia son responsabilidad del módulo de recuperación, el aislamiento es responsabilidad del módulo de control de la concurrencia y la consistencia es responsabilidad del módulo de comprobación de la integridad y del módulo de recuperación.

Gestión de transacciones en SQL92

El lenguaje SQL92 ofrece las siguientes instrucciones para el manejo de transacciones:

- no existe una instrucción para definir el inicio de una transacción.
- finalización de una transacción: COMMIT [WORK] (transacción confirmada parcialmente).
- anulación de una transacción: ROLLBACK [WORK] (transacción anulada).

Es importante recordar que la finalización (confirmación parcial) de una transacción no significa su confirmación definitiva, ya que ésta puede estar condicionada a la comprobación posterior de restricciones de integridad. Para velar por el mantenimiento de las buenas propiedades de las transacciones el SGBD debe asegurar que los cambios realizados por una transacción confirmada (de forma definitiva) quedan grabados permanentemente en la base de datos y que los cambios realizados por una transacción anulada que ya hayan sido grabados son deshechos.

Una de las propiedades importantes de las transacciones es la *consistencia*, es decir la transacción debe conducir la base de datos a un estado consistente en el que se cumplan todas las restricciones de integridad, por ello es importante conocer la estrategia seguida por el sistema para realizar esta comprobación.

La existencia del concepto de transacción y su carácter de unidad lógica de ejecución, plantea dos posibles estrategias para la comprobación de la integridad. Una restricción se puede comprobar después de la ejecución de cada instrucción SQL relevante para la restricción, modo ***inmediato*** (IMMEDIATE) o después de la finalización de cualquier transacción que incluya una instrucción SQL relevante para la restricción, modo ***diferido*** (DEFERRED).

El modo de una restricción se puede cambiar localmente para una transacción si la restricción se ha definido como ***diferible*** (DEFERRABLE); una restricción es diferible si tiene la propiedad de poder diferirse su comprobación al final de la transacción.

El modo de comprobación de una restricción se define en el esquema de la base de datos con la cláusula *[cuando_comprobar]* :

cuando_comprobar:=

[[NOT] DEFERRABLE] [INITIALLY {IMMEDIATE | DEFERRED}]

La semántica de cada una de las versiones de la cláusula *cuando_comprobar* es la siguiente:

- si no se utiliza esta cláusula la restricción se define como no diferible y con modo inmediato.
- la versión DEFERRABLE INITIALLY IMMEDIATE (resp. DEFERRED) define una restricción como diferible y con modo por defecto inmediato (resp. diferido).
- la versión NOT DEFERRABLE INITIALLY IMMEDIATE coincide con los valores por defecto.
- la versión NOT DEFERRABLE INITIALLY DEFERRED está prohibida.
- la versión DEFERRABLE (resp. NOT DEFERABLE) define una restricción como diferible (resp. no diferible) y con modo por defecto inmediato.
- la versión INITIALLY IMMEDIATE define una restricción como no diferible y con modo inmediato.
- la versión INITIALLY DEFERRED define una restricción como diferible y con modo por defecto diferido.

La instrucción SQL que permite cambiar, localmente en una transacción, el modo de una restricción definida como diferible, es:

SET CONSTRAINT {coma*lista_nombre_restricción* | ALL}
{IMMEDIATE | DEFERRED}.

Cada restricción especificada en la lista debe ser diferible y la opción ALL hace referencia a todas las restricciones diferibles del esquema de la base de datos.

El alcance del cambio producido por la instrucción SET CONSTRAINT es la transacción en la que se incluye o el fragmento de transacción hasta la siguiente aparición de la misma instrucción.

Si se incluye la instrucción en medio de la transacción con la opción IMMEDIATE, las restricciones afectadas por la instrucción son comprobadas cuando se ejecuta ésta, si alguna de estas restricciones falla, la instrucción SET falla y el modo de las restricciones permanece sin modificar.

Gestión de transacciones en ORACLE8

En el uso interactivo del sistema ORACLE8 ***una transacción se inicia*** con la primera instrucción SQL[2] ejecutada por el usuario desde que finalizó la última transacción o desde el inicio de la sesión y ***termina*** cuando el usuario la ***finaliza explícitamente*** con la sentencia COMMIT [WORK] (transacción confirmada parcialmente) o la ***anula explícitamente*** con la sentencia ROLLBACK [WORK] (transacción anulada) o bien cuando el sistema la ***finaliza implícitamente*** debido al cierre de la sesión (transacción confirmada parcialmente) o la ***anula implícitamente*** debido a la ocurrencia de un error (transacción anulada). Es decir, en ORACLE8 la operación que inicia una transacción es implícita y la operación que termina una transacción puede ser explícita o implícita.

Respecto a la estrategia de comprobación de las restricciones en ORACLE8, se sigue la propuesta del SQL estándar con el siguiente comportamiento respecto a la violación de una restricción: “si durante la ejecución de una transacción se viola una restricción con modo inmediato el sistema deshace el efecto de la operación SQL que ha causado la violación de la restricción (**statement rollback**) y la transacción puede continuar; si al finalizar una transacción se viola una restricción con modo diferido el sistema anula la transacción y deshace su efecto global (**transaction rollback**)”.

[2] Se supone que sólo se utilizan sentencias de DML.

Ejercicios

Considera la base de datos que has diseñado en la práctica anterior y cuyo esquema se encuentra en el anexo para la gestión de una biblioteca y realiza los siguientes ejercicios.

a) En el sistema ORACLE8 no existe la directriz *actualización en cascada* para la restauración de la integridad referencial. ¿Cómo podrías modificar el código de un socio que ha tenido préstamos, sin violar la integridad referencial de la relación de préstamos históricos? (*necesidad del concepto de transacción*).

b) Diseña una transacción que realice inserciones de tuplas sobre la relación de socios según el siguiente esquema de transacción:

INSERT INTO socios VALUES <s1, ..., ..., ..., 0>,
INSERT INTO socios VALUES <s2, ..., ..., ..., 0>,
INSERT INTO socios VALUES <s1, ..., ..., ..., 0>,
INSERT INTO socios VALUES <s3, ..., ..., ..., 0>,

(s1, s2 y s3 representan códigos de socio que no hayas usado hasta el momento de ejecutar la transacción y los puntos suspensivos representan valores cualesquiera para los atributos nombre, dirección y teléfono).
Esta transacción viola la restricción de clave primaria para la relación *socio*.
Ejecuta dos instancias del anterior esquema de transacción, una con la restricción de clave primaria para la relación *socio* en modo inmediato y otra con la restricción en modo diferido[3]. ¿Qué diferencias observas en ambos casos? (*atomicidad y consistencia*).

c) Inicia desde tu PC dos sesiones distintas sobre la misma base de datos y en cada sesión realiza las siguientes transacciones (los $\mathbf{t_i}$ indican el orden en que se deben realizar las operaciones):

Sesión 1	**Sesión 2**
$\mathbf{t_0}$ "consulta el número total de socios"	
	$\mathbf{t_1}$ "consulta el número total de socios"
$\mathbf{t_2}$ "inserta un nuevo socio"	
$\mathbf{t_3}$ "consulta el número total de socios"	
	$\mathbf{t_4}$ "consulta el número total de socios"
$\mathbf{t_5}$ "finaliza la transacción"	
	$\mathbf{t_6}$ "consulta el número total de socios"
	$\mathbf{t_7}$ "inserta un nuevo socio"
	$\mathbf{t_8}$ "consulta el número total de socios"
	$\mathbf{t_9}$ "anula la transacción"

Terminadas las dos transacciones, consulta el número total de socios en las dos sesiones. ¿Cómo se interpretan los resultados de las operaciones de consulta de las dos transacciones? ¿y el resultado de la consulta final? (*aislamiento y persistencia*).
¿Con las observaciones realizadas puedes afirmar que el sistema mantiene la propiedad de *persistencia* de una transacción?

[3] Para cambiar localmente en una transacción el modo de una restricción, ésta debe haberse definido como diferible.

Anexo: Base de datos de la biblioteca (Práctica 3 sesión 2)

Solución con relación *Préstamo* que almacena sólo el préstamo histórico

```
create table socio(
scod char(5) constraint cp_socio primary key deferrable,
nombre varchar2(60),
direccion varchar2(50),
tel varchar2(20),
libros number(3) default 0);

create table libro(
lcod char(5) constraint cp_libro primary key  deferrable,
titulo varchar(100),
tematica varchar(15) constraint tematica_nulo not null deferrable,
scod char(5) constraint ca_libro_socio references socio deferrable,
fecha_pre date,
constraint c_tematica check (tematica in ('física','electricidad','mecánica','óptica')) deferrable,
constraint c_prestamo check ((scod is null and fecha_pre is null) or
                             (scod is not null and fecha_pre is not null)) deferrable);

create table autores(
lcod char(5) constraint ca_autor_libro references libro(lcod)  deferrable,
autor varchar2(40) constraint autor_nulo not null deferrable,
constraint cp_autores primary key(lcod, autor) deferrable);

create table prestamo(
scod char(5) constraint ca_pre_socio references socio(scod) deferrable,
lcod char(5) constraint ca_pre_libro references libro(lcod) deferrable,
fecha_pre date constraint fecha_pre_nulo not null deferrable,
fecha_dev date constraint fecha_dev_nulo not null deferrable,
constraint cp_prestamo primary key(lcod,fecha_pre) deferrable,
constraint fechas check (fecha_dev>fecha_pre) deferrable);

create or replace trigger total_libros
after insert or delete or update of scod on libro
for each row

begin
if inserting and not(:new.scod is null) then
       update socio set libros=libros+1 where scod= :new.scod;
else
       if deleting and not(:old.scod is null) then
              update socio set libros=libros-1 where scod= :old.scod;
       else
           if (:old.scod is null) and not(:new.scod is null) then
              -- prestamo de un libro
              update socio set libros=libros+1 where scod= :new.scod;
           else
              if not(:old.scod is null) and :new.scod is null then
                 -- devolución de libro
                 update socio set libros=libros-1 where scod= :old.scod;
              else
                   if (:old.scod <> :new.scod) and  not(:old.scod is null) and
                     not(:new.scod is null) then
                      -- cambio de libro prestado
                     update socio set libros=libros+1 where scod= :new.scod;
                     update socio set libros=libros-1 where scod= :old.scod;
                   end if;
             end if;
           end if;
       end if;
end if;

end;
```

Solución con relación *Préstamo* que almacena el préstamo actual y el histórico

```
create table socio(
       scod char(5) constraint cp_socio primary key deferrable,
       nombre varchar2(60),
       direccion varchar2(50),
       tel varchar2(20),
       libros number(3) default 0 not null deferrable );

create table libro(
       lcod char(5) constraint cp_libro primary key  deferrable,
       titulo varchar(100),
       tematica varchar(15) constraint tematica_nulo not null deferrable,
       constraint c_tematica check (tematica in
              ('física','electricidad','mecánica','óptica')) deferrable);

create table autores(
       lcod char(5) constraint ca_autor_libro references libro(lcod)  deferrable,
       autor varchar2(40) constraint autor_nulo not null deferrable,
       constraint cp_autores primary key(lcod, autor) deferrable);

create table prestamo(
       scod char(5) constraint ca_pre_socio references socio(scod) deferrable,
       lcod char(5) constraint ca_pre_libro references libro(lcod) deferrable,
       fecha_pre date,
       fecha_dev date,
       constraint cp_prestamo primary key(lcod,fecha_pre) deferrable,
       constraint fechas check ((fecha_dev is null) or (fecha_dev>fecha_pre)) deferrable);

create or replace trigger prestamo_total_libros
after insert on prestamo
for each row when (new.fecha_dev is null)

begin
update socio set libros=libros+1 where scod= :new.scod;
end;

create or replace trigger devolucion_total_libros
after update of fecha_dev on prestamo
for each row when (old.fecha_dev is null and new.fecha_dev is not null)

begin
update socio set libros=libros-1 where scod= :old.scod;
end;

create or replace trigger cambio_prestamo_total_libros
after update of scod on prestamo
for each row when (old.scod<>new.scod and new.fecha_dev is null)

begin
update socio set libros=libros-1 where scod= :old.scod;
update socio set libros=libros+1 where scod= :new.scod;
end;

create or replace trigger canc_pres_hist_total_libros
after update of fecha_dev on prestamo
for each row when (old.fecha_dev is not null and new.fecha_dev is null)

begin
update socio set libros=libros+1 where scod= :old.scod;
end;

create or replace trigger canc_pres_actual_total_libros
after delete on prestamo
for each row when (old.fecha_dev is null)

begin
update socio set libros=libros-1 where scod= :old.scod;
end;
```