FICHEROS y BB.DD.

Práctica 2 – Sesión 3

Elementos Avanzados

uc3m | Universidad Carlos III de Madrid

Prácticas de la asignatura: hoja de ruta

sesión 1
- Modelado Relacional (*esquema relacional*)
- Implementación: entorno SQL+ (consola interacción)
- Estática Relacional: creación de tablas en SQL (LDD)

sesión 2
- Dinámica Relacional
  - consultas básicas en SQL y gestión transaccional
  - carga de datos (ejecución de scripts + volcado)
- del álgebra relacional al SQL

sesión 3
- Mecanismos de SQL avanzados
  - vistas y disparadores

examen de prácticas

sesión 4
- Diseño Físico
  - Parametrización de la base
  - Organizaciones base y auxiliares
  - Hints

práctica 1

práctica 2

práctica 3

Resolución de la Práctica 3

- Pasos a seguir:

1. Diseño Externo
    1. Identificación de Vistas
    2. Diseño de la sub-consulta
    3. Implementación
    4. **Pruebas**: sintaxis, validez de resultado (consulta), operatividad (debe realizar las operaciones permisibles)
2. Completitud Semántica
    1. Identificación de Necesidades
    2. Diseño de Soluciones (disparadores, procedimientos,...)
    3. Implementación
    4. **Pruebas**: sintaxis, ejecución (casos de prueba)
3. Documentación

Planteamiento del Problema (ejemplo 1)

- Se tienen dos tablas recogiendo información de contratos y sus claúsulas.
- num_claúsulas debe estar actualizada siempre (count clausulas).
- La fecha de una clausula siempre es igual o anterior a la de su contrato.
- Las clausulas no pueden eliminarse de la base, si bien el usuario sí puede borrarlas (al consultar, ya no aparecerán la borradas). La claúsula permanecerá sin borrarse, anotando además la fecha e identidad de usuario que la 'eliminó'.

```
CREATE TABLE contratos(
    referencia        VARCHAR2(25) PRIMARY KEY,
    fecha_firma       DATE DEFAULT SYSDATE,
    num_clausulas     NUMBER(3) DEFAULT 0 );

CREATE TABLE clausulas(
    referencia        VARCHAR2(25),
    n_orden           NUMBER(3),
    fecha_cl          DATE DEFAULT SYSDATE,
    CONSTRAINT PK_clausul PRIMARY KEY(referencia,n_orden),
    CONSTRAINT FK_clausul FOREIGN KEY (referencia)
       REFERENCES contratos(referencia) ON DELETE CASCADE);
```

Planteamiento del Problema (ej. 1/7)

- *num_claúsulas debe estar actualizada siempre (count clausulas).*

```
CREATE TABLE contratos_ALL(
    referencia        VARCHAR2(25) PRIMARY KEY,
    fechafirma        DATE DEFAULT SYSDATE );

CREATE TABLE clausulas_ALL(
    referencia        VARCHAR2(25),
    n_orden           NUMBER(3),
    fecha_cl          DATE DEFAULT SYSDATE,
    CONSTRAINT PK_clausul PRIMARY KEY(referencia,n_orden),
    CONSTRAINT FK_clausul FOREIGN KEY (referencia)
       REFERENCES contratos_ALL(referencia) ON DELETE CASCADE);

CREATE VIEW contratos AS (
       SELECT referencia, fechafirma, COUNT('X') num_claúsulas
              FROM contratos_ALL NATURAL JOIN clausulas_ALL
              GROUP BY (referencia, fechafirma)
       );
```

Planteamiento del Problema (ej. 2/7)

- *Operatividad de la vista: ¿se puede borrar? Sí. ¿se puede modificar? Sí.*

Puede ser mantenidas por la RI

- *¿se puede insertar? …*

```
CREATE TRIGGER ins_contratos
   INSTEAD OF INSERT ON contratos
   FOR EACH ROW
BEGIN
   INSERT INTO contratos_ALL
          VALUES(:NEW.referencia, :NEW.fechafirma)
END;
```

```
Disparador creado con errores de compilación.
SQL>
```

  - *¿Cómo localizar el fallo? …*

LDD – Depuración de Disparador

1.- Para saber los disparadores creados, consultar el catálogo

```
Select TRIGGER_TYPE, TRIGGERING_EVENT,
  BASE_OBJECT_TYPE, TABLE_NAME, WHEN_CLAUSE,
  STATUS, ACTION_TYPE, DESCRIPTION, TRIGGER_BODY
  from user_triggers where trigger_name='...';
```

2.- Si al crear un disparador existen errores de compilación.

Consultar las tablas del catálogo a través del SQL+

```
Show errors trigger <name_trigger>;
```

3.- Para depurar: poner trazas con mensajes por pantalla

```
SET SERVEROUTPUT ON
DBMS_OUTPUT.PUT_LINE('v_fmax: ' || :new.F_INI);
```

Planteamiento del Problema (ej. 3/7)

- La fecha de una clausula siempre es igual o anterior a la de su contrato.

```
CREATE TRIGGER CHK_fecha_clausulas
BEFORE INSERT OR UPDATE OF fecha_cl ON clausulas_ALL
FOR EACH ROW

DECLARE fechamala EXCEPTION; fecha DATE;
BEGIN
  SELECT fechafirma INTO fecha
     FROM contratos_ALL
     WHERE referencia=:NEW.referencia;
  IF :NEW.fecha_cl > fecha THEN RAISE fechamala;
  END IF;
EXCEPTION
  WHEN fechamala THEN
       RAISE_APPLICATION_ERROR (-20000,'Wrong DATE!');
END;
```

* -20000 a -20999

Planteamiento del Problema (ej. 4/7)

- Las clausulas no pueden eliminarse de la base, si bien el usuario sí puede borrarlas (al consultar, ya no aparecerán las borradas). La claúsula permanecerá sin borrarse, anotando además la fecha e identidad de usuario que la 'eliminó'.

```
ALTER TABLE clausulas_ALL ADD(
    usuario          VARCHAR2(25),
    fecha_dlt        DATE );

CREATE VIEW clausulas AS (
 SELECT referencia, n_orden, fecha_cl FROM clausulas_ALL
       WHERE fecha_dlt is NULL );
```

- *Operatividad de la vista: ¿se puede modificar? Sí. ¿se puede insertar? Sí.*
- *¿se puede borrar? Sí, pero…*

```
CREATE TRIGGER no_borra_clausul
  INSTEAD OF DELETE ON clausulas
BEGIN
  UPDATE clausulas_ALL set usuario=USER, fecha_dlt=SYSTATE
    WHERE referencia=:OLD.referencia AND n_orden=:OLD.n_orden;
END;
```

Planteamiento del Problema (ej. 5/7)

- Si se borra un contrato, se borran sus clausulas → ¡SÍ!
- Si se modifica la referencia de un contrato, se propaga el cambio → aún no…

```
CREATE TRIGGER UC_clausulas_disp
AFTER UPDATE OF referencia ON contratos_ALL
FOR EACH ROW
BEGIN
  UPDATE clausulas_ALL set referencia = :NEW.referencia
      WHERE referencia = :OLD.referencia;
END;
```

```
SQL> UPDATE contratos_ALL set referencia=0 where referencia=1;

1 fila modificada

SQL> UPDATE contratos_ALL set referencia = referencia+1;

ORA-04091: table BD_TABLE_NAME is mutating,
...
```

LDD – Beware the Mutating table ERROR

```
ORA-04091: table BD_TABLE_NAME is mutating,
trigger/function may not see it
ORA-06512: at "BD_XX.TRIGGER_NAME", line 5
ORA-04088: error during execution of trigger
'BD_XX.TRIGGER_NAME'
```

- No es un error de compilación → **se produce en la ejecución**
- Una tabla mutante es aquella que está siendo actualizada por la sentencia del disparo, como consecuencia de una restricción referencial en cascada, o por otro disparador activo
- Surge con la granularidad FOR EACH ROW

LDD – Beware the Mutating table ERROR

Solución para evitar este error:

- Implementar la ECA en dos pasos, almacenando la descripción de los cambios en primera instancia (for each row), y ejecutando la acción correctora en la segunda (for each statement).
- Esta estrategia admite dos implementaciones:
  - GENERAL: crear un almacén intermedio, ya sea tabla temporal o estructuras en memoria pertenecientes a un paquete. Después, crear un disparador de fila que almacene los cambios, y otro de instrucción que realice el efecto global esperado.
  - ORACLE proporciona disparadores compuestos. En un disparador compuesto se puede incluir un cuerpo para cada temporalidad/granularidad y una sección declarativa global (donde se definirá el almacén intermedio).

Recursos Rel. – Tablas Temporales

La definición de las tablas temporales se guarda en el catálogo, pero sus datos sólo son persistentes por sesión o por transacción, según se defina (por defecto, se eliminan al finalizar la transacción).

```
CREATE GLOBAL TEMPORARY TABLE t_produccion1
ON COMMIT PRESERVE ROW
AS (SELECT titulo, nacionalidad FROM produccion)
   WITH NO DATA;
```

esta coletilla no la permiten todos los SGBD;
si es el caso del tuyo, puedes probar: "... WHERE 1=0;

```
CREATE GLOBAL TEMPORARY TABLE t_produccion2
(title, nationality)
ON COMMIT DELETE ROW
AS SELECT titulo, nacionalidad FROM produccion;
```

* Pregunta: *¿qué diferencias encuentras entre la creación de ambas tablas?*

Recursos Relacionales – cursores (I)

- Objeto local (declarado): `CURSOR nombre IS (SELECT …)`
- Objeto global (creado): `CREATE CURSOR nombre IS (SELECT …)`
- Ejemplo:

```
DECLARE
  l_total INTEGER := 10000;
  CURSOR employee_id_cur IS
    SELECT employee_id FROM plch_employees ORDER BY salary ASC;
    l_employee_id employee_id_cur%ROWTYPE;
BEGIN
  OPEN employee_id_cur;
  LOOP
   FETCH employee_id_cur INTO l_employee_id;
   EXIT WHEN employee_id_cur%NOTFOUND;
   assign_bonus (l_employee_id, l_total);
   EXIT WHEN l_total <= 0;
  END LOOP;
  CLOSE employees_cur;
END;
```

http://www.oracle.com/technetwork/issue-archive/2013/13-mar/o23plsql-1906474.html

Recursos Relacionales – cursores (II)

- Ejemplo de cursor con parámetro:

```
CREATE OR REPLACE FUNCTION GetSalary IS

cur_sal NUMBER;
CURSOR cur_salary(emp_id IN NUMBER) IS
  SELECT salary
  FROM employee
  WHERE employee_id = emp_id;

BEGIN
  OPEN cur_salary(138);
  FETCH cur_salary IN cur_sal;
  IF cur_salary%NOTFOUND THEN
    cur_sal := 100000;
  END IF;
  CLOSE cur_salary;

END;
```

Recursos Relacionales – cursores (III)

- Ejemplo de cursor implícito:

```
CREATE OR REPLACE FUNCTION GetSalary IS

cur_sal NUMBER;

BEGIN

  FOR fila IN
      (SELECT salary FROM employee WHERE employee_id = 138)
  LOOP
     IF fila.atributo = 8 THEN cur_sal := 100000; END IF;
  END LOOP;

END;
```

Planteamiento del Problema (ej. 6/7)

ERROR DE TABLA MUTANTE: **Solución general**

```
CREATE GLOBAL TEMPORARY TABLE tmp_contratos
(oldref VARCHAR2(25), newref VARCHAR2(25) );

CREATE TRIGGER UC_clausulas_disp1
BEFORE UPDATE OF referencia ON contratos_ALL
FOR EACH ROW
BEGIN
   INSERT INTO tmp_contratos
       VALUES (:OLD.referencia, :NEW.referencia);
END;

CREATE TRIGGER UC_clausulas_disp2
AFTER UPDATE OF referencia ON contratos_ALL
BEGIN
   FOR fila IN (SELECT * FROM tmp_contratos) LOOP
       UPDATE clausulas_ALL SET referencia = fila.newref
              WHERE referencia = fila.oldref;
   END LOOP;
END;
```

Recursos Relacionales - tipos de datos

- Referencias a tipos de datos:

tablename%rowtype
attribute%type
IS TABLE OF *tipodatos*

-Ejemplos:

```
DECLARE
  l_employee employees%ROWTYPE;
BEGIN
  SELECT * INTO l_employee FROM employees WHERE employee_id = 138;
  DBMS_OUTPUT.put_line ( l_employee.last_name);
END;
```

```
DECLARE
 l_last_name employees.last_name%TYPE;
 l_department_name departments.department_name%TYPE;
BEGIN
 SELECT last_name, department_name INTO l_last_name, l_department_name
 FROM employees e, departments d
 WHERE e.department_id=d.department_id AND e.employee_id=138;
 DBMS_OUTPUT.put_line ( l_last_name || ' in ' || l_department_name);
END;
```

http://www.oracle.com/technetwork/issue-archive/2013/13-mar/o23plsql-1906474.html

Planteamiento del Problema (ej. 7/7)

ERROR DE TABLA MUTANTE: Solución ORACLE

```
CREATE OR REPLACE TRIGGER UC_clausulas
FOR UPDATE OF referencia ON contratos_ALL
COMPOUND TRIGGER
DECLARE
   TYPE fila IS RECORD (oldref VARCHAR2(25), newref VARCHAR2(25));
   TYPE TmpTab IS TABLE OF fila INDEX BY BINARY_INTEGER;
   tablaux TmpTab;
   j integer :=1

  BEFORE EACH ROW IS
   BEGIN
      tablaux(j).oldref := :OLD.referencia;
      tablaux(j).newref := :NEW.referencia;
      j:=j+1;
  END BEFORE EACH ROW;

  AFTER STATEMENT IS
   BEGIN
      FOR i IN 1 .. tablaux.COUNT LOOP
          UPDATE clausulas_ALL SET referencia = tablaux(i).newref
                 WHERE referencia = tablaux(i).oldref ;
      END LOOP;
  END AFTER STATEMENT;

END UC_clausulas;
```

Recursos Rel. – Paquetes Nativos

- Utilidades (bibliotecas) para desarrollar sistemas complejos.
- *Oracle's supplied Packages*: extensa colección (237 paq. en 11g) de utilidades y herramientas para el programador en SQL. Ejemplos:
  - `DBMS_output:` I/O básica para fichero (UTL_file) o interfaz estándar
  - `DBMS_metadata:` simplifica el manejo del catálogo relacional (data dictionary)
  - `DBMS_alert:` envía señales por socket; evita hacer polling (sondeo)
  - `DBMS_crypto:` #%ft9$s88_"!
  - `DBMS_jobs:` trabajos periódicos (eventos temporales)
  - `DBMS_utility:` cajón de sastre (hora, versión, hash, table_to_comma,…)
  - `DBMS_random:` valores aleatorios (números, string,…)
  - `DBMS_monitor:` permite controlar trazas (DBMS_trace) y estadísticas
  - `DBMS_LOB:` permite manejar campos lob, blob, clob, …
  - `DBMS_FGA:` permite aplicar politicas de auditoría de grano fino
  - `SDO_*:` paquetes de Oracle Spatial (SDO_GEOR, SDO_TUNE, …)
  - `OWA_*:` paquetes de Oracle Web Applications
  - `DBMS_XML*:` paquetes para manejo de XML
  - …

http://docs.oracle.com/cd/B28359_01/appdev.111/b28419/intro.htm

Recursos Relacionales – SQL dinámico

- Programación dinámica: generar código en tiempo de ejecución.
- En SQL se hace mediante la instrucción EXECUTE IMMEDIATE.
- Ejemplo:

```
CREATE OR REPLACE PROCEDURE show_values ( table_in IN VARCHAR2,
                                          column_in IN VARCHAR2,
                                          where_in IN VARCHAR2 ) IS
TYPE  values_t  IS TABLE OF NUMBER;
      l_values  values_t;
      instruct  LONG;
BEGIN
  instruct :=  'SELECT ' || column_in || ' FROM ' || table_in
               || ' WHERE ' || where_in ;
  EXECUTE IMMEDIATE instruct
     BULK COLLECT INTO l_values;
  FOR indx IN 1 .. l_values.COUNT  LOOP
     DBMS_OUTPUT.put_line (l_values (indx));
  END LOOP;
END;
```