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)

práctica 1

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

práctica 2

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 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 contracts(
    referenc          VARCHAR2(25) PRIMARY KEY,
    signature         DATE DEFAULT SYSDATE,
    num_clauses       NUMBER(3) DEFAULT 0 );

CREATE TABLE clauses(
    referenc          VARCHAR2(25),
    n_order           NUMBER(3),
    cl_date           DATE DEFAULT SYSDATE,
    CONSTRAINT PK_clause PRIMARY KEY (referenc,n_order),
    CONSTRAINT FK_clause FOREIGN KEY (referenc)
       REFERENCES contracts(referenc) ON DELETE CASCADE);
```

Planteamiento del Problema (ej. 1/7)

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

```
CREATE TABLE contracts_ALL(
    referenc          VARCHAR2(25) PRIMARY KEY,
    Signature         DATE DEFAULT SYSDATE );

CREATE TABLE clauses_ALL(
    referenc          VARCHAR2(25),
    n_order           NUMBER(3),
    cl_date           DATE DEFAULT SYSDATE,
    CONSTRAINT PK_clause PRIMARY KEY(referenc,n_order),
    CONSTRAINT FK_clause FOREIGN KEY (referenc)
       REFERENCES contracts_ALL ON DELETE CASCADE);

CREATE OR REPLACE VIEW contracts AS (
   SELECT referenc, signature, COUNT('X') AS num_clauses
       FROM contracts_ALL JOIN clauses_ALL USING(referenc)
            GROUP BY (referenc, signature)
       ) WITH READ ONLY;
```

Algunas pruebas...

```
INSERT INTO contracts_ALL(referenc) VALUES ('ref1');
INSERT INTO contracts_ALL(referenc) VALUES ('ref2');
INSERT INTO clauses_ALL(referenc,n_order) VALUES ('ref1',1);
INSERT INTO clauses_ALL(referenc,n_order) VALUES ('ref1',3);
INSERT INTO clauses_ALL(referenc,n_order) VALUES ('ref2',1);

CREATE VIEW conts AS ( SELECT referenc FROM contracts_ALL );
INSERT INTO conts(referenc) VALUES ('ref4');
UPDATE conts SET referenc='ref3' WHERE referenc='ref4';
DELETE FROM conts WHERE referenc='ref3';

SELECT * FROM contracts;
INSERT INTO contracts(referenc) VALUES ('ref3');

CREATE OR REPLACE VIEW contracts AS (
   SELECT referenc, signature, COUNT('X') AS num_clauses
       FROM contracts_ALL JOIN clauses_ALL USING(referenc)
            GROUP BY (referenc, signature)
       ) WITH CHECK OPTION;
```

Planteamiento del Problema (ej. 2/7)

- *Operatividad de la vista conts: ¿se puede borrar? ¿modificar? ¿insertar? → Sí*
- *Operatividad de la vista contracts: ¿se puede borrar? ¿modificar? ¿insertar?*

  → NO !

```
CREATE TRIGGER ins_contracts
    INSTEAD OF INSERT ON contracts
    FOR EACH ROW
BEGIN
    INSERT INTO contracts_ALL
           VALUES(:NEW.referenc, :NEW.signature)
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 OR REPLACE TRIGGER CHK_clause_date
BEFORE INSERT OR UPDATE OF cl_date ON clauses_ALL
FOR EACH ROW
DECLARE      signdate DATE;
             baddate EXCEPTION;
BEGIN
   SELECT signature INTO signdate
      FROM contracts_ALL WHERE referenc=:NEW.referenc;
   IF :NEW.cl_date > signdate
     THEN RAISE baddate;
   END IF;
EXCEPTION
  WHEN baddate THEN DBMS_OUTPUT.PUT_LINE('Wrong DATE!');
END CHK_clause_date;

SET SERVEROUTPUT ON
INSERT INTO clauses_ALL VALUES ('ref2',2,sysdate+1);
```

Planteamiento del Problema (ej. 3/7)

- *Prueba esta otra versión:*

```
CREATE OR REPLACE TRIGGER CHK_clause_date
BEFORE INSERT OR UPDATE OF cl_date ON clauses_ALL
FOR EACH ROW
DECLARE      signdate DATE;
             baddate EXCEPTION;
BEGIN
   SELECT signature INTO signdate
      FROM contracts_ALL WHERE referenc=:NEW.referenc;
   IF :NEW.cl_date > signdate
     THEN RAISE baddate;
   END IF;
EXCEPTION
  WHEN baddate
   THEN RAISE_APPLICATION_ERROR (-20001,'WRONG DATE!');
END CHK_clause_date;

INSERT INTO clauses_ALL VALUES ('ref2',2,sysdate+1);
```

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 clauses_ALL ADD(
    userid          VARCHAR2(25),
    dlt_date        DATE );

CREATE VIEW clauses AS (
    SELECT referenc, n_order, cl_date
       FROM clauses_ALL WHERE fecha_dlt is NULL );
```

- *Operatividad de la vista: ¿se puede modificar? Sí. ¿se puede insertar? Sí.*
- *¿se puede borrar? Sí, pero justamente eso no deberíamos…*

```
CREATE TRIGGER delete_clause
  INSTEAD OF DELETE ON clauses
BEGIN
  UPDATE clauses_ALL set userid=USER, dlt_date=SYSTATE
    WHERE referenc=:OLD.referenc AND n_order=:OLD.n_order;
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_clauses
AFTER UPDATE OF referenc ON contracts_ALL
FOR EACH ROW
BEGIN
  UPDATE clauses_ALL set referenc = :NEW.referenc
      WHERE referenc = :OLD.referenc;
END;
```

```
SQL> UPDATE contracts_ALL set referenc='ref4' where referenc='ref1';
1 row updated

SQL> UPDATE contracts_ALL set referenc = referenc||'...';

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

NOTA: este caso **NO** produce este error en nuestra **versión** actual de Oracle DB

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 DB 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_contracts
(oldref VARCHAR2(25), newref VARCHAR2(25) );

CREATE TRIGGER UC_clauses_disp1
BEFORE UPDATE OF referenc ON contracts_ALL
FOR EACH ROW
BEGIN
   INSERT INTO tmp_contracts
       VALUES (:OLD.referenc, :NEW.referenc);
END;

CREATE TRIGGER UC_clauses_disp2
AFTER OF referenc ON contracts_ALL
BEGIN
   FOR row IN (SELECT * FROM tmp_contracts) LOOP
       UPDATE clauses_ALL SET referenc = row.newref
              WHERE referenc = row.oldref);
   END LOOP;
END;
```

Recursos Relacionales - tipos de datos

- *tipos de datos:* **tablename%rowtype**
attribute%type
TYPE mylabel1 IS RECORD (col1, col2, …)
TYPE mylabel2 IS TABLE OF *tipodatos*

```
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_clauses
FOR UPDATE OF referenc ON contracts_ALL
COMPOUND TRIGGER
DECLARE
   TYPE myrow IS RECORD (oldref VARCHAR2(25), newref VARCHAR2(25));
   TYPE TmpTab IS TABLE OF myrow INDEX BY BINARY_INTEGER;
   tablaux TmpTab;

  BEFORE EACH ROW IS
   BEGIN
      tablaux(tablaux.COUNT+1).oldref := :OLD.referenc;
      tablaux(tablaux.COUNT).newref := :NEW.referenc;    -- Not +1
  END BEFORE EACH ROW;

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

END UC_clauses;
```

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;
```