Administración y Programación de Bases de Datos

Lenguaje PL/SQL [a]

Profesor: Gilberto Gutiérrez R.

Departamento de Ciencias de la Computación y Tecnologías de la Información

Facultad de Ciencias Empresariales

Universidad del Bío-Bío

[a] Algunas slides están basadas en el tutorial PL/SQL localizado en el sitio *http://www.devjoker.com/gru/tutorial-PL-SQL/PLSQ/Tutorial-PL-SQL.aspx*

sqlplus

```
[ggutierr@oracle ~]$ rl_sqlplus

SQL*Plus: Release 11.2.0.1.0 Production on Sat Aug 10 22:47:35 2013

Copyright (c) 1982, 2009, Oracle.  All rights reserved.

Enter user-name: ggutierr
Enter password:

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

sqlplus (Help)

```
SQL> help index

Enter Help [topic] for help.

 @                COPY           PAUSE                      SHUTDOWN
 @@               DEFINE         PRINT                      SPOOL
 /                DEL            PROMPT                     SQLPLUS
 ACCEPT           DESCRIBE       QUIT                       START
 APPEND           DISCONNECT     RECOVER                    STARTUP
 ARCHIVE LOG      EDIT           REMARK                     STORE
 ATTRIBUTE        EXECUTE        REPFOOTER                  TIMING
 BREAK            EXIT           REPHEADER                  TTITLE
 BTITLE           GET            RESERVED WORDS (SQL)       UNDEFINE
 CHANGE           HELP           RESERVED WORDS (PL/SQL)    VARIABLE
 CLEAR            HOST           RUN                        WHENEVER OSERROR
 COLUMN           INPUT          SAVE                       WHENEVER SQLERROR
 COMPUTE          LIST           SET                        XQUERY
 CONNECT          PASSWORD       SHOW

SQL>
```

sqlplus (Edición)

```
SQL> !vi mul.pl

declare
 i number;
 j number;
begin
  i :=1;
  while (i <= 10) loop
   dbms_output.put_line('------------------' );
   dbms_output.put_line('Tabla del '  || i );
   dbms_output.put_line('------------------' );
   j := 1;
   while (j <= 10) loop
     dbms_output.put_line(i || ' x ' ||  j || ' = ' || i * j );
     j := j +1 ;
   end loop;
   i := i +1;
 end loop;
end;
```

sqlplus (Ejecución)

```
SQL> set serverout on
SQL> get mul.pl
  1  declare
  2   i number;
  3   j number;
  4  begin
  5    i :=1;
  6    while (i <= 10) loop
  7     dbms_output.put_line('------------------' );
  8     dbms_output.put_line('Tabla del '  || i );
  9     dbms_output.put_line('------------------' );
 10     j := 1;
 11     while (j <= 10) loop
 12       dbms_output.put_line(i || ' x ' ||  j || ' = ' || i * j );
 13       j := j +1 ;
 14     end loop;
 15     i := i +1;
 16   end loop;
 17* end;
SQL>run
```

sqlplus (Ejecución)

```
------------------
Tabla del 1
------------------
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
1 x 10 = 10
------------------
Tabla del 2
------------------
2 x 1 = 2
.........
```

Introducción

Manejo descentralizado de reglas de negocio

Ejemplo: "*El porcentaje de descuento por volumen de venta debe ajustarse a la siguiente tabla*"

- entre 0 y 999, 0 % de descuento
- entre 1 millón y 3 millones, 5% de descuento
- más de 3 millones, 8% de descuento

Indique los problemas que se pueden producir al manejar esta regla de negocio de manera descentralizada

Introducción

- Ventajas de PL/SQL sobre SQL inmerso (embebido)
- PL/SQL se administra centralmente dentro de la base de datos Oracle.
- PL/SQL se comunica en forma nativa con objetos de la base de datos
- PL/SQL fácil de modularizar y manejar.

Programas en PL/SQL

- Modularidad. Reusabilidad, esconder la complejidad
  - Procedimientos. Aceptan y retornan cero o más valores.
  - Funciones. Aceptan y retornan un valor.
  - Triggers. Sentencias asociadas a una tabla
  - Paquetes. Colección de procedimientos y funciones (definición y cuerpo)

Bloque PL/SQL

Un bloque se compone de tres secciones:

1. Declaración de variables.(tipos Oracle, tipo exclusivo PL/SQL)
2. Sección ejecutable. Instrucciones a ejecutar dentro del bloque
3. Sección de manejo de excepciones. Se definen los manejadores de errores que soportará el bloque.

Dos tipos de bloques:

- Con nombre
- Sin nombre

Bloque PL/SQL

Ejemplo (bloque sin nombre):

```
DECLARE
        FechaSistema DATE;
BEGIN
        SELECT SYSDATE
        INTO FechaSistema
        FROM DUAL;
        dbms_output.put_line('Fecha del sistema: ' || FechaSistema);
EXCEPTION
        WHEN OTHERS THEN
        dbms_output.put_line('Se ha producido un error');
END;
```

La sección ejecutable comienza con la palabra BEGIN y termina con la palabra END, o con la palabra EXCEPTION
La sección EXCEPTION es opcional.

Bloque PL/SQL

Ejemplo (bloque con nombre)

```
FUNCTION fn_Obtener_Precio(p_producto VARCHAR2)
RETURN NUMBER
IS
  result NUMBER;
BEGIN
  SELECT PRECIO INTO result
  FROM PRECIOS_PRODUCTOS
  WHERE CO_PRODUCTO = p_producto;
  return(result);
EXCEPTION
WHEN NO_DATA_FOUND THEN
  return 0;
END ;
```

Estructuras de control de flujo

- IF

```
IF (expresion) THEN
    Instrucciones
ELSIF (expresion) THEN
    Instrucciones
ELSE
    Instrucciones
END IF;
```

- GOTO

```
BEGIN
 IF (flag = 1) THEN
    GOTO paso2;
 END IF;
<<paso1>>
      dbms_output.put_line('Ejecucion de paso 1');
<<paso2>>
      dbms_output.put_line('Ejecucion de paso 2');
END;
```

Estructuras de control de flujo

- LOOP

```
LOOP
 Instrucciones
 IF (expresion) THE
        Instrucciones
        EXIT;
 END IF;
END LOOP;
```

- WHILE

```
WHILE (expresion) LOOP
   Instrucciones
END LOOP;
```

- FOR

```
FOR contador IN [REVERSE] inicio..final LOOP
  Instrucciones
END LOOP;
```

Tipos de datos

- Tipos de la base de datos
  - `Number(tamaño[,precisión])`
  - `CHAR(tamaño), VARCHAR(tamaño)`
  - `DATE`
  - `LONG` almacena bloques de texto (hasta 2GB)
  - `LONG RAW`. Datos multimedia (binario, sonido imágenes)
  - `ROWID` . Identificador único para cada fila.
  - ...

Tipos de datos

- Tipos de datos propios de PL/SQL
  - `DEC, DECIMAL, REAL, DOUBLE_PRECISION, INTEGER, ...`
  - `BOOLEAN`
  - `TABLE, RECORD`. TABLE similar a un Arreglo y RECORD tipos de datos compuestos
  - ...

```
DECLARE
 empladoNombre empleado.nombre%TYPE
```

La variable empleadoNombre tendrá el tipo de dato declarado para la columna nombre en la tabla empleado

Tipos de datos

```
DECLARE
empleado empleado%ROWTYPE
```

Se crea un RECORD idéntico a una tupla de la table empleado (con los mismos atributos y mismos tipos de datos).

```
DECLARE
 numerodeMeses CONSTANT NATURAL := 12;
```

Operadores

Descripción	Simbolo
Asignación	:=
Aritméticos	+, -,*, /, ** (exponente)
Relacionales	= , <>, < , . . .
Booleanos	AND, NOT, OR
Concatenación	||

Interacción con la base de datos (INSERT)

```
SQL> desc amigos;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 ID                                                 NUMBER(38)
 NOMBRE                                             VARCHAR2(20)
 CEL                                                NUMBER(38)
 FNAC                                               DATE
 SEXO                                               CHAR(1)
 ----------------------------------------- -------- ----------------
```

```
declare
  m amigos%ROWTYPE;
begin
  m.id:=1040;
  m.nombre := 'Ernesto ';
  m.cel := 98989898;
  m.fnac := date '1978-05-12';
  m.sexo := 'M';
 insert into amigos values(m.id,m.nombre,m.cel,m.fnac,m.sexo);
end;
```

Interacción con la base de datos (INSERT)

```
SQL> select * from amigos;
        ID NOMBRE                          CEL FNAC      S
---------- -------------------- ---------- --------- -
      1020 Juan                   75488878 25-DEC-98 M
      1030 Veronica               88866776 10-DEC-80 F
SQL> get insert.sql
SQL> run
SQL> select * from amigos;

        ID NOMBRE                          CEL FNAC      S
---------- -------------------- ---------- --------- -
      1020 Juan                   75488878 25-DEC-98 M
      1030 Veronica               88866776 10-DEC-80 F
      1040 Ernesto                98989898 12-MAY-78 M
```

Interacción con la base de datos (UPDATE)

```
DECLARE
  mid  amigos.ID%TYPE;
BEGIN
  mid := 1040;
  update  amigos
   set NOMBRE = 'Juan Ernesto'
   where ID = mid;
END;
```

Interacción con la base de datos (DELETE)

```
DECLARE
  mid  amigos.ID%TYPE;
BEGIN
  mid := 1040;
   delete from amigos
   where ID = mid;
END;
```

Interacción con la base de datos (CURSOR)

Cursores: Controladores de áreas de memoria que almacenan los resultados de una instrucción SELECT

- Implícitos
- Explícitos

Interacción con la base de datos (CURSOR)

- Cusor implícitos

```
DECLARE
    m  amigos%ROWTYPE;
BEGIN
 select *
  INTO m
  from amigos
  where ID =1020;
 dbms_output.put_line('Mi amigo se llama '  || m.nombre );
END;
```

Interacción con la base de datos (CURSOR)

- Cusor explicitos: SELECT recupera cero, o una o más tuplas

Se requiere:

- Declarar el cursor.
- Abrir el cursor (OPEN).
- Leer los datos (FETCH).
- Cerrar el cursor y liberar los recursos (CLOSE).

Interacción con la base de datos (CURSOR)

Declaración del cursor: (dos formas)

```
DECLARE
 ...
CURSOR nombreCursor IS
  instruccion SELECT
```

```
DECLARE
 ...
 CURSOR nombreCursor(param1 tipo1, ..., paramN tipoN) IS
  instruccion SELECT
```

Interacción con la base de datos (CURSOR)

Abrir el cursor

```
OPEN nombre_cursor;
OPEN nombre_cursor(valor1, valor2, ..., valorN);
```

Recuperar los datos del Buffer

```
FETCH nombre_cursor INTO lista_variables;
FETCH nombre_cursor INTO registro_PL/SQL;
```

Cerrar el cursor

```
CLOSE nombre_cursor;
```

Interacción con la base de datos (CURSOR)

Ejemplos:

```
DECLARE
-- Un ejemplo de cursor explicito
  CURSOR  amigas IS
  SELECT Nombre FROM amigos
  WHERE sexo='F';
  elNombre amigos.Nombre%TYPE;
 BEGIN
   OPEN amigas;
   FETCH amigas INTO elNombre;
   dbms_output.put_line( ' --->  ' || elNombre );
   CLOSE amigas;
end;
```

Interacción con la base de datos (CURSOR)

Ejemplos (cursor con parámetros)

```
DECLARE
-- Un ejemplo de cursor explicito
  CURSOR mh (elSexo amigos.Sexo%TYPE)  IS
  SELECT Nombre FROM amigos
  WHERE sexo=elSexo;
  elNombre amigos.Nombre%TYPE;
 BEGIN
--Un hombre
   OPEN mh('M');
   FETCH mh INTO elNombre;
   dbms_output.put_line( ' --->  ' || elNombre );
   CLOSE mh;
--- Una mujer
   OPEN mh('F');
   FETCH mh INTO elNombre;
   dbms_output.put_line( ' --->  ' || elNombre );
   CLOSE mh;
end;
```

Interacción con la base de datos (CURSOR)

Ejemplos: (Usando registro)

```
DECLARE
-- Un ejemplo de cursor explicito
  CURSOR  misAmigos IS
  SELECT * FROM amigos;
  elAmigo  amigos%ROWTYPE;
 BEGIN
   OPEN misAmigos;
   FETCH misAmigos INTO elAmigo;
   dbms_output.put_line('Mi amigo se llama  --->  ' || elAmigo.nombre  );
   CLOSE misAmigos;
end;
```

Interacción con la base de datos (CURSOR)

Atributos de los cursores:

Atributo	Antes de abrir	Al abrir	Durante la recu-peración	Al final de la recu-peración	después de cerrar
%NOTFOUND	ORA-1001	NULL	FALSE	TRUE	ORA-1001
%FOUND	ORA-1001	NULL	TRUE	FALSE	ORA-1001
%ISOPEN	FALSE	TRUE	TRUE	TRUE	FALSE
%ROWCOUNT	ORA-1001	0	*	**	ORA-1001

* Número de tuplas que ha recuperado hasta el momento, ** Número de total de tuplas

- Cuando un cursor está cerrado, no se puede leer.
- Cuando se cierra el cursor, es ilegal tratar de usarlo.
- Es ilegal tratar de cerrar un cursor que ya está cerrado o no ha sido abierto

Interacción con la base de datos (CURSOR)

Ejemplos: (Atributos Cursor)

```
DECLARE
-- Un ejemplo de cursor explicito
  CURSOR  misAmigos IS
  SELECT * FROM amigos;
  elAmigo  amigos%ROWTYPE;
 BEGIN
   OPEN misAmigos;
   LOOP
      FETCH misAmigos INTO elAmigo;
      EXIT WHEN misAmigos%NOTFOUND;
      dbms_output.put_line('Mi amigo se llama  ' || elAmigo.nombre);
      dbms_output.put_line('Tuplas recuperadas ' || misAmigos%ROWCOUNT);
   END LOOP;
   dbms_output.put_line('Tuplas  totales ' || misAmigos%ROWCOUNT);
   CLOSE misAmigos;
END;
```

Interacción con la base de datos (CURSOR)

Ejemplos: (Usando FOR)

```
DECLARE
  CURSOR  misAmigos IS
  SELECT * FROM amigos;
  elAmigo  amigos%ROWTYPE;
 BEGIN
   FOR elAmigo in misAmigos LOOP
    dbms_output.put_line('Mi amigo se llama  --->  ' || elAmigo.nombre  );
   END LOOP;
 END;
```

No se necesita

- Abrir el cursor
- Hacer FETCH
- Cerrar el cursor

Interacción con la base de datos (CURSOR)

Ejemplos: (Usando FOR)

```
DECLARE
  elAmigo  amigos%ROWTYPE;
 BEGIN
   FOR elAmigo in (SELECT * FROM amigos)  LOOP
    dbms_output.put_line('Mi amigo se llama  --->  ' || elAmigo.nombre  );
   END LOOP;
 END;
```

Interacción con la base de datos (CURSOR)

Ejemplos: (Cursor UPDATE)

```
DECLARE
-- Un ejemplo de actualizacion
  CURSOR  misAmigos IS
  SELECT * FROM amigos
  FOR UPDATE;
  elAmigo  amigos%ROWTYPE;
 BEGIN
   OPEN misAmigos;
    FETCH misAmigos INTO elAmigo;
    WHILE misAmigos%FOUND
    LOOP
      UPDATE amigos
      SET nombre = elAmigo.nombre ||'.'
      WHERE CURRENT OF misAmigos;
      FETCH misAmigos INTO elAmigo;
    END LOOP;
   CLOSE misAmigos;
end;
```

Manejo de Excepciones

- En PLSQL una advertencia o condición de error es llamada una excepción.
- Las excepciones se controlan dentro de su propio bloque

Estructura del bloque con excepciones

```
DECLARE
 -- Declaraciones
BEGIN
  -- Ejecucion
EXCEPTION
WHEN NO_DATA_FOUND THEN
  -- Se ejecuta cuando ocurre una excepcion de tipo NO_DATA_FOUND
WHEN ZERO_DIVIDE THEN
  -- Se ejecuta cuando ocurre una excepcion de tipo ZERO_DIVIDE
WHEN OTHERS THEN
  -- Se ejecuta cuando ocurre una excepcion de un tipo no tratado
  -- en los bloques anteriores
END;
```

Manejo de Excepciones

Excepciones predefinidas.

Excepción	Significado	SQLCODE
ACCESS_INTO_NULL	Se intenta asignar valores a los atributos de un objeto no inicializado	-6530
STORAGE_ERROR	No hay memoria esta se encuentra corrupta	-6500
SUBSCRIPT_BEYOND_COUNT	Se trata de referenciar un elemento de un arreglo indexado que se encuentra en una posición más grande que el número real de elementos de la colección	-6533
...	...	...

Manejo de Excepciones

Excepciones definidas por el programador

```
DECLARE
 -- Declaraciones

MyExcepcion EXCEPTION;
BEGIN
  -- Ejecucion
EXCEPTION
  -- Excepcion
END;
```

Una excepción declarada en un bloque es local a ese bloque y global a todos los sub-bloques que comprende.

Manejo de Excepciones

Excepciones definidas por el programador

```
  DECLARE
 -- Declaramos una excepcion identificada por VALOR_NEGATIVO
VALOR_NEGATIVO EXCEPTION;
valor NUMBER;
BEGIN
  -- Ejecucion
valor := -1;
IF valor < 0 THEN
 RAISE VALOR_NEGATIVO;
END IF;

EXCEPTION
  -- Excepcion
WHEN VALOR_NEGATIVO THEN
 dbms_output.put_line('El valor no puede ser negativo');
END;
```

Manejo de Excepciones

SQLCODE y SQLERRM

```
...
EXCEPTION
WHEN OTHERS THEN
  ...
  DBMS_OUTPUT.put_line('Error:'|| SQLCODE);
  DBMS_OUTPUT.put_line(SQLERRM);
END;
```

Transacciones

Una transacción es un programa en ejecución que constituye una unidad lógica del procesamiento de una base de datos.

- Incluye una o más operaciones de acceso a la base de datos (inserción, lectura o actualización)
- Las operaciones de una transacción pueden estar incrustadas en una aplicación o pueden espcificarse interactivamente
- Un programa puede contener más de una transacción.

Propiedades deseables de las transacciones

- Atomicidad
- Conservación de la consistencia de la Base de Datos
- Aislamiento - Como si estuviera ejecutándose aisladamente
- Durabilidad - Los cambios realizados por una transacción deben persistir en la base de datos

Ejemplos de transacciones

(a)

T_1
read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

(b)

T_2
read_item(X);
$X := X + M$;
write_item(X);

Problemas con la concurrencia

Problema por pérdida de actualización

T_1	T_2
read_item(X);	
$X := X - N$;	
	read_item(X);
	$X := X + M$;
write_item(X);	
read_item(Y);	
	write_item(X) ;
$Y := Y + N$;	
write_item(Y);	

El elemento X tiene un valor incorrecto porque se ha perdido su actualización realizada por T_1

Problemas con la concurrencia (cont.)

Problema de la actualización temporal o lectura sucia

T_1	T_2
read_item(X);	
$X := X - N$;	
write_item(X);	
	read_item(X);
	$X := X + M$;
	write_item(X);
read_item(Y);	

La transacción T_1 falla y debe cambiar el valor de X por el valor antiguo; mientras T_2 ha leído el valor temporal incorrecto de X

Problemas con la concurrencia (cont.)

El problema de la suma incorrecta

T_1	T_3
	$sum := 0;$
	read_item(A);
	$sum := sum + A;$
	$\vdots$
read_item(X);	
$X := X - N;$	
write_item(X);	
	read_item(X);
	$sum := sum + X;$
	read_item(Y);
	$sum := sum + Y;$
read_item(Y);	
$Y := Y + N;$	
write_item(Y);	

T_3 lee X después de restarse N y lee Y antes de añadirse a N; el resultado es erróneo

Un **bloqueo** es una variable asociada a un elemento de datos

- Describe el estado del elemento de datos respecto a las posibles operaciones que se le pueden aplicar.
- Generalmente hay un bloqueo por cada elemento de la base de datos
- Se utilizan como medios para sincronizar el acceso de las transacciones concurrentes a los elementos de la base de datos.

Tipos de bloqueo:

- Bloques binarios
- Bloques compartidos/exclusivos

Bloqueos binarios

- Puede tener sólo dos estados: bloqueado o desbloqueado (1 ó 0)
- Si el valor del bloqueo sobre X es 1, entonces el elemento no podrá ser accedido por una operación de bases de datos de una transacción

Cada transacción debe cumplir las siguientes reglas:

1. T debe emitir una operación **bloquear_elemento(X)** antes de ejecutar **read_item** o **write_item**
2. T debe emitir una operación **desbloquear_elemento(X)** después de haberse completado todas las operaciones **read_item** y **write_item**
3. T no emitirá una operación **bloquear_elemento(X)** si ya posee el bloqueo de X
4. T no emitirá una operación **desbloquear_elemento(X)** a menos que ya posea el bloqueo de X

Bloqueos compartido/exclusivos (lectura/escritu

- Permite que varias transacciones tengan acceso al item X si ellas ecceden a X para leer
- Si una transacción va a escribir un elemento X, entonces debe tener acceso exclusivo a X
- Existen tres operaciones de bloqueo
  - **bloquear_lectura(X)**
  - **bloquear_escritura(X)**
  - **desbloquear(X)**

Reglas de Bloqueos compartido/exclusivos

1. T debe emitir la operación **bloquear_lectura(X)** o **bloquear_escritura(X)** antes de que se ejecute cualquier operación **read_item(X)**
2. T debe emitir la operación **bloquear_escritura(X)** antes de que se ejecute cualquier operación **write_item(X)**
3. T debe emitir la operación **desbloquear(X)** una vez que se hayan completado todas las operaciones **read_item(X)** y **write_item(X)** de T.
4. T no emitirá una operación **bloquear_lectura(X)** si ya posee un bloqueo de lectura o de escritura para X
5. T no emitirá una operación **bloquear_escritura(X)** si ya posee un bloqueo de lectura o de escritura para X
6. T no emitirá una operación **desbloquear(X)** a menos que ya posea un bloqueo de lectura o de escritura sobre X.

Recuperación después de falla

EL DBMS es el responsable de que todas las transacciones se completen satisfactoriamente y que su efecto grabe de manera permanente o de que la transacción no afecte a la base de datos en caso de fallas.

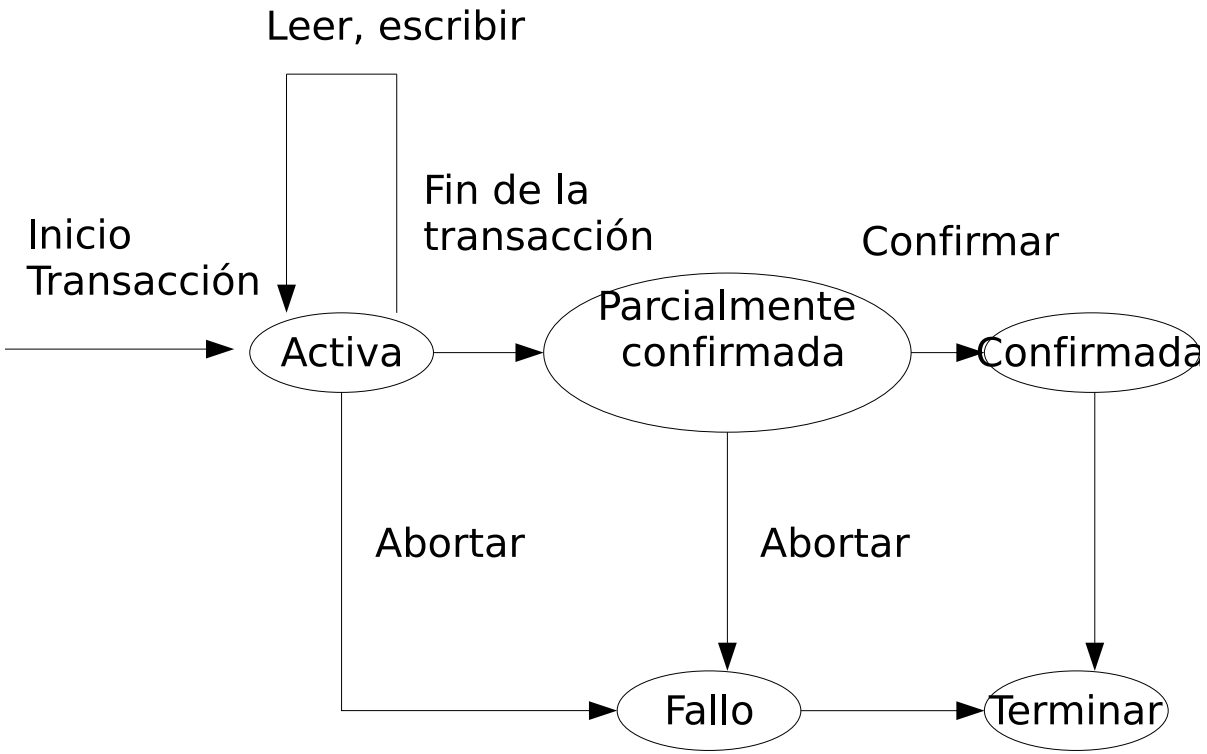
Tipos de fallas:

1. Caída del sistema
2. Error de la transacción o del sistema
3. Condiciones de excepción detectados por la transacción
4. Control de la concurrencia
5. Falla del disco
6. Problemas físicos y catástrofe (alimentación de energía por ejemplo)

Estados de una transacción

- BEGIN_TRANSACTION - Marca el inicio de la transacción
- READ o WRITE
- END_TRANSACTION - Read y Write han terminado y marca el final de la ejecución de la transacción.
- COMMIT_TRANSACTION - Finalización satisfactoria. Los cambios no se desecharán
- ROLLBACK (o ABORT) - No ha terminado satisfactoriamente (deshacer los cambios).

Estado de una transacción (cont.)



Transacciones PL/SQL

COMMIT / ROLLBACK

```
  DECLARE
  ...
BEGIN
     UPDATE CUENTAS SET SALDO = SALDO - 1000
     WHERE CUENTA = 1020;
     UPDATE CUENTAS SET SALDO = SALDO + 1000
     WHERE CUENTA = 1030;
     ...
     COMMIT;
EXCEPTION
WHEN OTHERS THEN
     dbms_output.put_line('Error'||SQLERRM);
     dbms_output.put_line('La transaccion se aborto');
     ROLLBACK;
END;
```

Transacciones

- COMMIT Compromete los cambios
- ROLLBACK Deshace los cambios
- SAVEPOINT Indica a ROLLBACK hasta donde deshacer

```
DECLARE
  emp_id emp.empno%TYPE;
BEGIN
  UPDATE emp SET ... WHERE empno = emp_id;
  DELETE FROM emp WHERE ...
  SAVEPOINT do_insert;
  INSERT INTO emp VALUES (emp_id, ...);
EXCEPTION
 WHEN DUP_VAL_ON_INDEX THEN
   ROLLBACK TO do_insert;
END;
```

Antes de ejecutar INSERT, UPDATE, o DELETE, ORACLE hace un SAVEPOINT (no disponible para el programador)

Transacciones

SET TRANSACTION

```
DECLARE
  daily_sales REAL;
  weekly_sales REAL;
  monthly_sales REAL;
BEGIN
  COMMIT; -- ends previous transaction
  SET TRANSACTION READ ONLY NAME 'Calculate sales figures';
  SELECT SUM(amt) INTO daily_sales FROM sales
    WHERE dte = SYSDATE;
  SELECT SUM(amt) INTO weekly_sales FROM sales
    WHERE dte > SYSDATE - 7;
  SELECT SUM(amt) INTO monthly_sales FROM sales
   WHERE dte > SYSDATE - 30;
  COMMIT; -- ends read-only transaction
END;
```

Procedimientos/funciones almacenados

- Se almacenan “junto” a las tablas
- Son parte integrante de la base de datos
- Tienen nombre, se pueden invocar mediante su nombre
- Pueden recibir parámetros y devolver un valor.

Procedimiento Almacenados

```
CREATE [OR REPLACE] PROCEDURE <procedure_name> [
 (<param1> [IN|OUT|IN OUT] <type>,
  <param2> [IN|OUT|IN OUT] <type>, ...)]
IS
  -- Declaracion de variables locales
BEGIN
  -- Sentencias
[EXCEPTION]
  -- Sentencias control de excepcion
END [<procedure_name>];
```

Procedimiento Almacenados

Ejemplo:

```
SQL> !nano miAmigo.sql

CREATE OR REPLACE PROCEDURE miAmigo(elId IN Integer) as
 elAmigo amigos%ROWTYPE;
 BEGIN
 SELECT * INTO elAmigo
 FROM AMIGOS
 WHERE ID = elId;
 dbms_output.put_line(elAmigo.nombre);
END miAmigo;
```

Procedimiento Almacenados

Ejecución del script que crea el procedimiento

```
SQL> start miAmigo
Procedure created.
```

Ejecución del procedimiento

```
SQL> execute miAmigo(1040);
Ernesto


PL/SQL procedure successfully completed.
```

También se puede ejecutar desde un bloque

```
SQL> begin miAmigo(1040); end;
  2  /
Ernesto
```

Procedimiento Almacenados

Lista todos los amigos

```
CREATE OR REPLACE PROCEDURE listaAmigos   AS
 CURSOR  listado IS
 SELECT * FROM AMIGOS;
 elAmigo amigos%ROWTYPE;
 BEGIN
  FOR elAmigo IN listado LOOP
    dbms_output.put_line(elAmigo.nombre);
  END LOOP;
 END;
/
```

Procedimientos Almacenados

actualiza el nombre de un amigo

```
CREATE OR REPLACE PROCEDURE actualiza(elId IN INTEGER, NuevoNombre IN VARCHAR)
 CURSOR elAmigo IS SELECT nombre FROM amigos where id=elId  FOR UPDATE;
 elnombre amigos.nombre%TYPE;  noexiste EXCEPTION;
 BEGIN
   OPEN elAmigo;
   FETCH  elAmigo INTO elnombre;
   IF (elAmigo%NOTFOUND) THEN
      raise noexiste;
   ELSE
      UPDATE AMIGOS SET NOMBRE = NuevoNombre WHERE ID =elID;
      COMMIT;
   END IF;
 EXCEPTION
   WHEN noexiste THEN
   dbms_output.put_line('Error --> el amigo ' || elid || ' no existe');
   ROLLBACK;
 END;
/
```

Funciones

Calcula la edad en años.

```
  -- Calcula la edad al dia de hoy
CREATE OR REPLACE FUNCTION edad(id_amigo number)
  RETURN NUMBER AS
  fnac_amigo DATE;
  now DATE;
BEGIN
  SELECT SYSDATE INTO now  FROM DUAL;
  SELECT FNAC  INTO fnac_amigo from amigos WHERE id_amigo = id;
  RETURN (now-fnac_amigo) / 365;
END;
/
```

Funciones

Como se ejecuta

```
SQL> select nombre, edad(id) from amigos;

NOMBRE                 EDAD(ID)
-------------------- ----------
Ernesto              37.9278739
Miguel               15.4977369
Rosita               33.9525314
Raul                 38.9306136
```

También puede ser asi:

```
SQL> begin dbms_output.put_line('edad -->' || edad(1040)); end;
  2  /
edad -->37.927880232115677321156773211567732115 62

PL/SQL procedure successfully completed.
```

Funciones

Listar procedimientos y funciones (SQLPLUS)

```
 SELECT DISTINCT name, type FROM user_source WHERE type='PROCEDURE' OR
type='FUNCTION';
```

Ejemplo

```
SQL> SELECT DISTINCT name, type  FROM user_source
     WHERE type='PROCEDURE' OR TYPE ='FUNCTION';
     NAME                           TYPE
------------------------------ ------------
SQUARE                         FUNCTION
MISPROC                        PROCEDURE
EDAD                           FUNCTION
LISTAAMIGOS                    PROCEDURE
MIAMIGO                        PROCEDURE
LISTA1                         PROCEDURE
CNOMBRE                        PROCEDURE
ACTUALIZA                      PROCEDURE
```

Funciones

Listar contenido de procedimientos y funciones (SQLPLUS)

```
SQL> SELECT text FROM user_source WHERE name='EDAD' ORDER BY line;

TEXT
--------------------------------------------------------------------------------
FUNCTION edad(id_amigo number)
  RETURN NUMBER AS
  fnac_amigo DATE;
  now DATE;
BEGIN
  SELECT SYSDATE INTO now  FROM DUAL;
  SELECT FNAC  INTO fnac_amigo from amigos WHERE id_amigo = id;
  RETURN (now-fnac_amigo) / 365;
END;

9 rows selected.
```

Procedimiento/Funciones

Ejecución por parte de otro usuario (SQLPLUS)

1. El Propietario del procedimiento o función debe dar permiso de ejecución:

```
GRANT EXECUTE ON actualiza TO luis
```

2. El usuario `luis`

```
SQL> EXECUTE ggutierr.actualiza(1040,'Ernesto.');
```

Triggers

Trigger: Sentencia que el sistema ejecuta automáticamente como un efecto secundario producto de una modificación en la base de datos (INSERT, DELETE, UPDATE).
Modelo: ECA — (Evento, Condición, Acción)

1. **Evento**—Acción sobre la BD que provoca que trigger se dispare (UPDATE, INSERT, DELETE, o eventos temporales)
2. **Condición**—Condiciones bajo las cuales se ejecuta la acción (if new.saldo <= 0)
3. **Acción**—Secuencia de instrucciones que se ejecutan una vez cumplido el evento y las condiciones. Normalmente una secuencia de sentencias SQL, pero también una transacción o un programa externo.

Triggers

```
CREATE [OR REPLACE] TRIGGER <nombre_del_trigger>
{BEFORE|AFTER}
{DELETE|INSERT|UPDATE [OF col1, col2, ..., colN]
[OR {DELETE|INSERT|UPDATE [OF col1, col2, ..., colN]...]}
ON <nombre_tabla>
[FOR EACH ROW [WHEN (<condicion>)]]
[DECLARE]
  ...
 BEGIN
  ...
[EXCEPTION]
 ...
END <nombre_trigger>;
```

Triggers

Dos tipos de triggers:

1. Nivel de fila: Se ejecuta por cada tupla de la tabla afectada por el evento.
2. Nivel de instrucción: Se ejecutan una sola vez a pesar de afectar a muchas tuplas

Opciones:

- BEFORE/AFTER: Indica si el disparador se ejecuta antes o después de la instrucción
- FOR EACH ROW: Trigger a nivel de fila
- WHEN: El trigger se activa solamente para aquellas tuplas que cumplen la condición

Triggers

variables de tupla NEW/OLD

- NEW Mantiene valores nuevos en los atributos de la tupla producidos por la acción de la instrucción SQL
- OLD Mantiene valores antiguos en los atributos de la tupla (antes de la acción de la instrucción SQL)

Estas variables permiten acceder a los valores de los atributos de las tuplas que se está insertando, eliminando o actualizando.

Triggers

Ejemplos: Suponemos las dos siguientes tablas:

```
SQL> desc empleados;
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
ID                                        NOT NULL NUMBER(38)
NOMBRE                                             VARCHAR2(20)
NRODEPTO                                           NUMBER(38)
SALARIO                                            NUMBER(38)
IDJEFE                                             NUMBER(38)

SQL> desc dpto;
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
NRO                                       NOT NULL NUMBER(38)
NOMBRE                                             CHAR(20)
IDDIRECTOR                                         NUMBER(38)
NROEMP                                             NUMBER(38)
```

El atributo **NROEMP** de la tabla **dpto** mantiene la cantidad de empleados en cada departamento.

Triggers

El atributo **NROEMP** de la tabla **dpto** se debe actualizar:

1. Cuando se inserta un empleado
2. Cuando se elimina un empleado
3. Cuando un empleado se cambia de depto

Crearemos un trigger por cada uno de estos eventos producidos en la tabla **empleados**

Triggers

Inserción de un empleado

```
 CREATE OR REPLACE TRIGGER InsertEmpleado
   AFTER INSERT ON empleados
   FOR EACH ROW
 BEGIN
   UPDATE dpto SET nroEmp = nroEMp + 1
   WHERE :NEW.nrodepto = nro;
END;
```

Triggers

Eliminación de un empleado

```
 CREATE OR REPLACE TRIGGER eliminaEmpleado
  AFTER DELETE ON empleados
  FOR EACH ROW
BEGIN
  UPDATE dpto SET nroEmp = nroEMp -1
  WHERE :OLD.nrodepto = nro;
END;
```

Triggers

Actualiza el dpto de un empleado

```
CREATE OR REPLACE TRIGGER updateEmpleado
  AFTER UPDATE OF nroDepto ON empleados
  FOR EACH ROW
BEGIN
  UPDATE dpto SET nroEmp = nroEMp +1
  WHERE :NEW.nrodepto = nro;
  UPDATE dpto SET nroEmp = nroEMp -1
  WHERE :OLD.nrodepto = nro;
END;
```

Triggers

Algunas restricciones:

1. No puede ejecutar COMMIT, ROLLBACK o SAVEPOINT. El trigger forma parte de una transacción.
2. No puede llamar a una función o procedimiento que ejecute COMMIT, ROLLBACK o SAVEPOINT.

Triggers

Otro ejemplo: un empleado no puede ganar más que su jefe

```
CREATE OR REPLACE TRIGGER masqueeljefe
  BEFORE  INSERT  ON empleados
  FOR EACH ROW
DECLARE
  salarioJefe INTEGER;
  elJefe VARCHAR(30);
BEGIN
 SELECT salario, nombre  INTO salarioJefe, elJefe FROM empleados
 WHERE :NEW.IdJefe = ID;
 IF(salarioJefe < :NEW.salario) THEN
  raise_application_error(-20000, ' < ' || :NEW.nombre || ' pretende ganar mas
 END  IF;
END;
/
```

Triggers

Procedimiento para insertar un empleado.

```
CREATE OR REPLACE PROCEDURE  InsertaNuevoEmpleado(
 elId INTEGER, elNombre VARCHAR, elDepto INTEGER,
 elSalario INTEGER, elJefe INTEGER)   AS
 BEGIN
  INSERT INTO EMPLEADOS
  VALUES (elId, elNombre, elDepto, elSalario, elJefe);
  COMMIT;
 EXCEPTION
   WHEN OTHERS THEN
   dbms_output.put_line(SQLERRM);
   ROLLBACK;
 END;
/
```

Paquetes/Package

- Permite agrupar bloques PL/SQL relacionados dentro de una aplicación.
- Permite mantener ordenada las aplicaciones.
- Ventajas en los tiempos de ejecución.
- . . .

Paquetes/Package

Especificación de la Interfaz

```
CREATE [OR REPLACE] PACKAGE <nombrePaquete>
IS

  -- Declaraciones publicas
  --(tipos, registros, variables, constantes y cursores)

  -- Declaraciones de procedimientos y funciones publicas

  {[FUNCTION <nombreFuncion>(<Parametro> <tipoDedato>,...)
    RETURN <tipoDedato>;]}
  {[PROCEDURE <nombreProcedimiento>(<Parametro> <tipoDedato>, ...);]}

END <nombrePaquete>;
```

Paquetes/Package

Ejemplo:

```
CREATE OR REPLACE PACKAGE RectangleTools IS
 dx NUMBER;
 dy NUMBER;
 CURSOR elRec(elId  rectangulo.rid%type) IS
 SELECT * FROM RECTANGULO
 WHERE rID=elId;
 --calcula el area
 FUNCTION area(elrid INTEGER)
 RETURN NUMBER;
 --calcula el perimetro
 FUNCTION perimetro(elrid INTEGER)
 RETURN NUMBER;
 --verifica si un punto esta dentro de un rectangulo.
 FUNCTION puntoenRectangulo(x NUMBER, y NUMBER, elRid INTEGER)
 RETURN INTEGER;
END RectangleTools;
/
```

Paquetes/Package

Cuerpo del Paquete

```
CREATE [OR REPLACE] PACKAGE BODY <nombrePaquete> IS
 -- Declaraciones locales
 FUNCTION <nombreFuncion>(<Parametros> <tipoDeDatos>,...)
 RETURN <tipoDeDatos>   IS
   -- Variables locales de la funcion
 BEGIN
   ...
 END;

 PROCEDURE <nombreProcedimiento>(<Parametro> <tipoDeDatos>, ...)  IS
  -- Variables locales del procedimiento
 BEGIN
   ...
 END;

END <pkgName>;
```

Paquetes/Package

Ejemplo:

```
 CREATE OR REPLACE PACKAGE BODY RectangleTools IS
---el area
 FUNCTION area(elRid integer)
 RETURN NUMBER IS
 rec rectangulo%ROWTYPE;
BEGIN
 OPEN elRec(elRid);
 FETCH elRec into rec;
 dx := rec.XH - rec.XL;
 dy := rec.YH - rec.YL;
 CLOSE elRec;
 RETURN (dx * dy);
END area;
 ...
END RectangleTools;
/
```

Paquetes/Package

Como ejecutar una función/Procedimiento:

```
SQL> select rid, rectangleTools.Area(rid) as Area  from rectangulo;

       RID        AREA
---------- ----------
         1           4
         2          12
         3           2
         4          16

SQL>
```

Paquetes/Package

Como listar los paquetes:

```
SQL>  SELECT DISTINCT name, type FROM user_source WHERE type = 'PACKAGE';

NAME                           TYPE
------------------------------ ------------
RECTANGLETOOLS                 PACKAGE

SQL>
```

Paquetes/Package

Como listar el contenido de paquete

```
SQL>  SELECT text FROM user_source WHERE name= 'RECTANGLETOOLS';

TEXT
--------------------------------------------------------------------------------
PACKAGE RectangleTools IS
 --calcula el area
 FUNCTION area(elrid INTEGER)
 RETURN NUMBER;
 --calcula el perimetro
 FUNCTION perimetro(elrid INTEGER)
 RETURN NUMBER;
 --verifica si un punto esta dentro de un rectangulo.
 FUNCTION puntoenRectangulo(x NUMBER, y NUMBER, elRid INTEGER)
 RETURN INTEGER;
END RectangleTools;
 ...
```

Tipos de datos

```
 DECLARE

  TYPE PAIS IS RECORD
  (
    CO_PAIS     NUMBER  ,
    DESCRIPCION VARCHAR2(50),
    CONTINENTE  VARCHAR2(20)
  );
  miPAIS PAIS;
BEGIN
  miPAIS.CO_PAIS := 27;
  miPAIS.DESCRIPCION := 'ITALIA';
  miPAIS.CONTINENTE  := 'EUROPA';
END;
```

Arreglos

```
 DECLARE
    /* Declaramos el tipo VARRAY de cinco elementos VARCHAR2*/
    TYPE t_cadena IS VARRAY(5) OF VARCHAR2(50);
    /* Asignamos los valores con un constructor */
    v_lista t_cadena:= t_cadena('Aitor', 'Alicia', 'Pedro','','');
BEGIN
    v_lista(4) := 'Tita';
    v_lista(5) := 'Ainhoa';
END;
```