

Índice

- 1.1. Conceptos básicos de la programación cliente/servidor
- 1.2. Capas de abstracción de una base de datos
- 1.3. Programación de restricciones en la definición de tablas
- 1.4. Generación de números secuenciales
- 1.5. Extensiones procedurales a SQL
- 1.6. Cursores
- 1.7. Procedimientos y funciones almacenados
- 1.8. Disparadores
- 1.9. Manejo de excepciones
- 1.10. El diccionario de datos

Bibliografía

- Database: Principles, Programming, and Performance, 2ª Edición
P. O'Neil y E. O'Neil. Morgan Kaufmann, 2000
Capítulos 4, 5 y 7



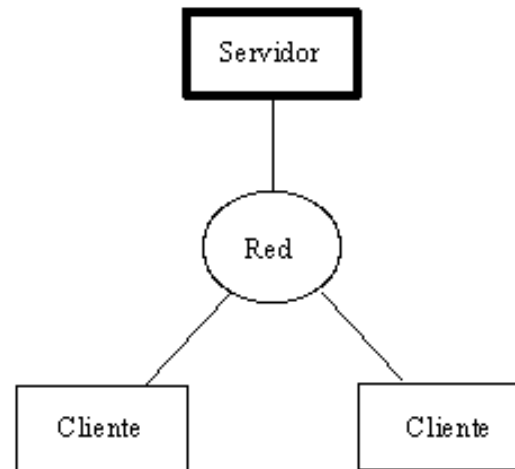
1.1. Conceptos básicos de la programación cliente/servidor

Programación del servidor: PL/SQL

Aplicación se divide en:

Cliente: parte cercana al usuario, recogida y presentación de la información

Servidor: parte cercana a los datos, ejecuta el gestor de bases de datos



Ventajas de la arquitectura cliente/servidor:

- Delegación de responsabilidades
- Independencia de la situación física de los datos
- Servidor explota características potentes de HW+SO (conurrencia, memoria, ...)
- Optimización y ampliación independientes de las partes cliente y servidor
- Utilización de la red: mejor acceso concurrente y disminución del tráfico

1.1. Conceptos básicos de la programación cliente/servidor

Programación del servidor: PL/SQL

1. Diseño de la Base de datos

- Diseño conceptual - Diseño lógico - Normalización - Diseño físico ...
- Es una tarea de **modelado de datos**, no de procesos.

2. Diseño de la aplicación

- Es un proceso interactivo, puede implicar rediseñar la BD
- Hay que utilizar las herramientas proporcionadas por el servidor de BD
 - Restricciones de integridad
 - Procedimientos almacenados
 - Disparadores
 - ...

3. Ajuste de la aplicación

- Ajuste y optimización SQL
- Ajuste del diseño de la aplicación



¿Por qué programar el servidor?

Programación del servidor: PL/SQL

BANCOS | Según FACUA

Un error 'masivo' del BBVA provoca el cobro duplicado a clientes de varios bancos




Efe | Sevilla




Actualizado martes 28/05/2013 13:25 horas

La FACUA-Consumidores en Acción ha alertado que un "error masivo" del BBVA ha provocado en los últimos días que ciertos usuarios de diversas compañías y distintos sectores, como suministros de agua, electricidad y telecomunicaciones, hayan sufrido el cobro por duplicado de recibos.

DELEGACIÓN DE RESPONSABILIDADES

Cesta > Identificación > Dirección de envío > Datos comprador > Verificación > Confirmación

| Cesta de la compra | | | |
|--|----------|---------|--|
| Artículos con entrega por agencia de transporte | Unidades | Precio | Total |
|  | 1 | 169 € | 169 € BORRAR |
|  | 1 | 130 € | 130 € BORRAR |
|  | 1 | 1.200 € | 1.200 € BORRAR |
| Actualizar cantidad | | | Total 1.499 € (No incluye gastos de envío) |



Inicio Sesiones Planes Grupos   

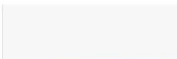

crear sesión
tus sesiones
tu historial
ranking
player

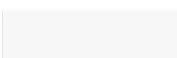

Tus sesiones

programadas realizadas caducadas favoritas

Tienes 21 sesiones para estos criterios de búsqueda.

**Sesión en Flat - 58512** 
01:00:00,0
Llano Indoor
25/03/2013 23:59

**Sesión en Flat - 68115** 
Generada mediante fichero csv
01:20:00,0
Llano Indoor
25/03/2013 23:59

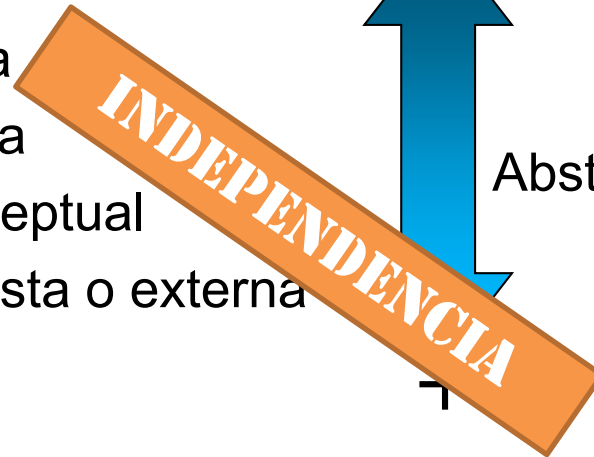
**Sesión en samano-castro-granja** 
desde el pueblo de samano por la general de castro hasta guriezo subimos la granja y volvemos a samano. el calentamiento del verano
Generada mediante fichero gpx
01:00:00,0
Llano Indoor
25/03/2013 23:59



1.2 Capas de abstracción de una BBDD

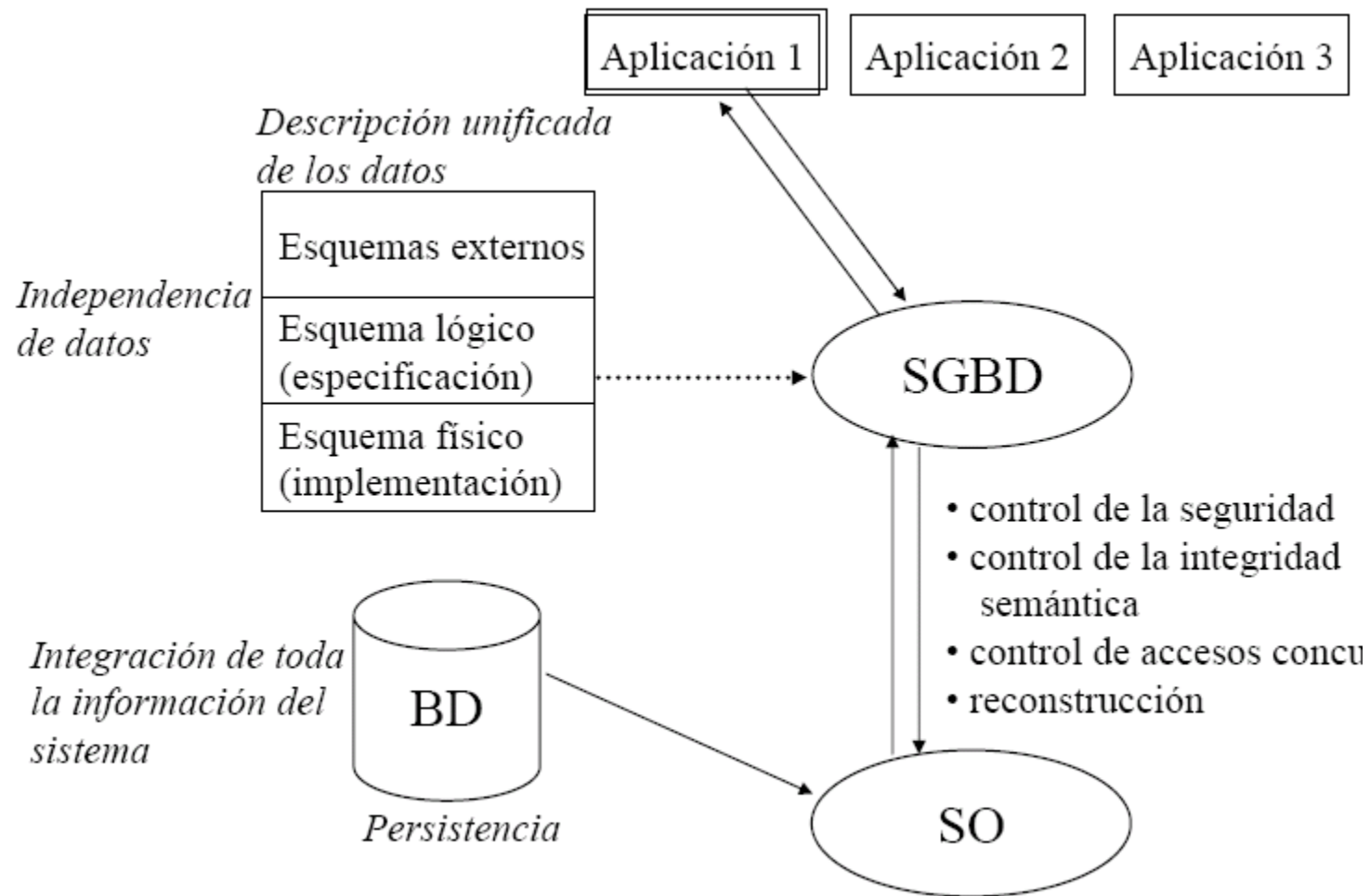
Programación del servidor: PL/SQL

- Capa física
- Capa lógica
- Capa conceptual
- Capa de vista o externa



1.2 Capas de abstracción: independencia de datos

Programación del servidor: PL/SQL



1.3. Programación de restricciones en la definición de tablas

Programación del servidor: PL/SQL

CREATE TABLE:

- Atributos no nulos
`... marca VARCHAR(15) NOT NULL ...`
- Atributos de valor único
`... num_bastidor VARCHAR(9) UNIQUE ...`
- Valor por defecto
tiene que ser la primera restricción y no lleva clausula `constraint`
`... estado_civil VARCHAR(3) DEFAULT 'sol' ...`
- Condiciones sobre los valores de los atributos
 - restricción de columna:
`... est_civ VARCHAR(3) CHECK (est_civ IN ('sol','cas','viu','div')) ...`
 - restricción de tablas:
`... CHECK (fecha_reparacion>=fecha_entrada) ...`
- Clave primaria
 - NOT NULL y UNIQUE
 - restricción de columna: `... DNI VARCHAR(9) PRIMARY KEY ...`
 - restricción de tabla: `... PRIMARY KEY (DNI,mat) ...`
 - IDENTITY (Oracle 12c)



1.3. Programación de restricciones en la definición de tablas

Programación del servidor: PL/SQL

CREATE TABLE:

- Claves externas
 - restricción de columna: `... mat VARCHAR(8) REFERENCES coches ...`
 - restricción de tabla: `... FOREIGN KEY mat REFERENCES coches ...`
- Integridad referencial
 - `... FOREIGN KEY atributo REFERENCES tabla(atributo) ON DELETE opción`
 - `... FOREIGN KEY atributo REFERENCES tabla(atributo) ON UPDATE opción`
 - Opciones:
 - Restringir: opción por defecto
 - Cascada: `CASCADE`
 - Hacer nulo: `SET NULL`
 - Valor por defecto: `DEFAULT valor`
- Integridad de identidad
 - Columnas `IDENTITY` (a partir de Oracle 12c)
 - Objetos contadores+disparadores



1.4. Generación de números secuenciales

Programación del servidor: PL/SQL

- Útil para dar valores únicos a llaves primarias
- Evita la serialización programada y mejora la concurrencia

```
CREATE SEQUENCE nombre_secuencia
  [INCREMENT BY numero] [START WITH numero]
  [MAXVALUE numero | NOMAXVALUE ] [MINVALUE numero | NOMINVALUE]
  [CYCLE | NOCYCLE ] [CACHE numero | NOCACHE ] [ORDER | NOORDER]
```

```
CREATE SEQUENCE misecuencia
  INCREMENT BY 1
  START WITH 1
  NOMAXVALUE NOCYCLE CACHE 10;
```

- Manipulación de la secuencia con sus pseudocolumnas **NEXTVAL** y **CURRVAL**

```
INSERT INTO mitabla (at1, ...) VALUES (misecuencia.NEXTVAL, ...);
```

```
UPDATE mitabla                                SELECT misecuencia.currval
SET at1=misecuencia.CURRVAL                    FROM dual;
WHERE ...;
```



1.4. Generación de números secuenciales: Identity (Oracle 12c)

Programación del servidor: PL/SQL

```
create table identity_tst_tab (  
  2  id number(10) GENERATED AS IDENTITY,  
  3  name varchar(15) );
```

Table created.

```
SQL:labpa> desc IDENTITY_TST_TAB
```

| Name | Null? | Type |
|-------|----------|--------------|
| ----- | ----- | ----- |
| ID | NOT NULL | NUMBER(10) |
| NAME | | VARCHAR2(15) |

```
SQL:labpa> insert into identity_tst_tab (name) values ('abbas');
```

1 row created.

```
SQL:labpa> insert into IDENTITY_TST_TAB (name) values ('fazal');
```

1 row created.

```
SQL:labpa> select * from identity_tst_tab;
```

| ID | NAME |
|-------|-------|
| ----- | ----- |
| 1 | abbas |
| 2 | fazal |

```
SQL:labpa> insert into identity_tst_tab values (100,'xyz');
```

```
insert into identity_tst_tab values (100,'xyz')
```

*

ERROR at line 1:

ORA-32795: cannot insert into a generated always identity column



- Crea una base de datos con tres tablas:
- ALUM (cod, dni, nombre, apellidos, email, movil, direccion-postal)
- AS (cod, nombre, titulacion, fecha, ct, cp)
- ALAS (cod, codal, codas, nota)
 - Todos los campos clave deben crearse con secuencias
 - ct y cp deben tomar un valor comprendido entre 3 y 7.5
 - Notación:
 - clave principal
 - *clave externa*

1.5. Extensiones procedurales a SQL

Programación del servidor: PL/SQL

- Convierten a SQL en un lenguaje procedimental tradicional
- Están recogidas en el estándar SQL-99 (SQL-99/PSM), en Oracle: PL/SQL
- Organización en bloques

```
DECLARE
    definición de constantes, variables, cursores y excepciones de usuario
BEGIN
    sentencias PL/SQL
    [EXCEPTION
        manejo de excepciones]
END;
/
```

- Declaración de variables

```
nombre_variable [CONSTANT] {tipo_base | tipo_ancla} [NOT NULL] [:= expr]
```

- Tipo base: **DATE**, **VARCHAR**, **NUMBER**, **BOOLEAN** (**TRUE** y **FALSE**)
- Tipo ancla: {nombre_tabla | nombre_cursor}%**ROWTYPE** ó nombre_tabla.nombre_atributo%**TYPE**



1.5. Extensiones procedurales a SQL

Programación del servidor: PL/SQL

- Modificación de la sentencia SELECT para asignación de variables

```
SELECT ...  
INTO lista_variables  
FROM ...;
```

```
DECLARE  
    elnombre mecanico.nombre%type;  
    elpuesto mecanico.puesto%type;  
BEGIN  
    SELECT nombre,puesto  
    INTO elnombre,elpuesto  
    FROM mecanico  
    WHERE dni='11111111A' ;  
END;  
/
```

```
DECLARE  
    elnombre mecanico.nombre%type;  
    elpuesto mecanico.puesto%type;  
BEGIN  
    SELECT nombre,puesto  
    INTO elnombre,elpuesto  
    FROM mecanico;  
END;  
/
```

Excepción **TOO_MANY_ROWS** !!



1.5. Extensiones procedurales a SQL

Programación del servidor: PL/SQL

Estructuras de control

- Bucles

```
[<<etiqueta>>] LOOP sentencias END LOOP;
```

```
[<<etiqueta>>] WHILE condición LOOP sentencias END LOOP;
```

```
[<<etiqueta>>]
```

```
FOR contador IN [REVERSE] valor_inicio .. valor_fin LOOP sentencias END LOOP;
```

```
FOR variable IN nombre_cursor LOOP sentencias END LOOP;
```

```
FOR variable IN subconsulta_SELECT LOOP sentencias END LOOP;
```

- Ruptura de bucle

```
EXIT [etiqueta] [WHEN condicion];
```

- Condicional

```
IF condicion THEN sentencias
```

```
[ELSIF condicion THEN sentencias]
```

```
[ELSE sentencias]
```

```
END IF;
```



1.6. Cursores

Programación del servidor: PL/SQL

- Similar a la idea de fichero formado por una secuencia de registros
- Cursor: Nombre para un área de trabajo que almacena los datos devueltos por una consulta SELECT y permite el acceso a estos datos
- Ciclo de vida de un cursor:
 - Declaración del cursor
 - Declaración de variables para la descarga de los datos del cursor
 - Apertura del cursor
 - Bucle donde se descargan los datos
 - Cierre del cursor



- Declaración del cursor
 - Aparece en la parte DECLARE del bloque
 - Se especifica el nombre del cursor y la consulta que lleva asociada
 - Opcionalmente se especifican atributos que pueden ser modificados
 - Se bloquean esas tuplas para evitar que otros usuarios las modifiquen mientras se está leyendo el cursor

```
CURSOR nombre_cursor [(parametro IN tipo, ...)]  
IS sentencia_select  
[FOR UPDATE OF lista_atributos];
```

```
CURSOR coches_marca (marca_coche IN coche.marca%TYPE)  
IS SELECT mat  
      FROM coche  
      WHERE marca = marca_coche;
```

```
CURSOR sueldo_empleado (v_puesto IN mecanico.puesto%TYPE)  
IS SELECT sueldo  
      FROM mecanico  
      WHERE puesto=v_puesto  
FOR UPDATE OF sueldo;
```



- Declaración de variables para la descarga de los datos del cursor
 - Aparece en la parte DECLARE del bloque
 - Se suele utilizar los tipos ancla

```
nombre_variable {tipo | nombre_tabla.nombre_atributo%TYPE}  
nombre_registro nombre_tabla%ROWTYPE  
nombre_registro nombre_cursor%ROWTYPE
```

- Apertura del cursor
 - Se realiza dentro del cuerpo del bloque
 - Al abrir un cursor se ejecuta la consulta asociada y se activa el área de trabajo
 - El cursor apunta a la primera tupla devuelta por el SELECT asociado

```
OPEN nombre_cursor [(valor, ...)];
```

```
OPEN coches_marca('PEUGEOT');
```

1.6. Cursores

Programación del servidor: PL/SQL

- **Descarga de datos**
 - Se suele realizar dentro de un bucle que va recorriendo todo el cursor
 - Se descarga la tupla apuntada por el cursor
 - Después de la descarga el cursor apunta a la siguiente tupla

```
FETCH nombre_cursor  
INTO {lista_variables | nombre_registro} ...;
```

```
FETCH coches_marca  
INTO v_mat;
```

- **Cierre del cursor**
 - Deshabilita el cursor y elimina todos los recursos que tenga asociados

```
CLOSE nombre_cursor;  
  
CLOSE coches_marca;
```



1.6. Cursores

Programación del servidor: PL/SQL

- Atributos del cursor
 - Informan del estado en cada momento del cursor

`nomb_cursor%FOUND`: TRUE si último FETCH sobre `nomb_cursor` devolvió alguna tupla

`nomb_cursor%NOTFOUND`: TRUE si último FETCH sobre `nomb_cursor` NO devolvió ninguna tupla

`nomb_cursor%ISOPEN`: TRUE si `nomb_cursor` está abierto

`nomb_cursor%ROWCOUNT`: devuelve número de tuplas devueltas por FETCH sobre `nomb_cursor`

```
IF coches_marca%NOTFOUND THEN EXIT;
```



1.6. Cursores

Programación del servidor: PL/SQL

- Aumentar un 5% los sueldos de los mecánicos de chapa que ganen menos de 2000

```
DECLARE
    CURSOR mec_pue (m_puesto IN mecanico.puesto%TYPE)
        IS SELECT *
           FROM mecanico
          WHERE puesto = m_puesto
        FOR UPDATE OF sueldo;
    reg_mec mecanico%ROWTYPE;
BEGIN
    OPEN mec_pue('CHAPA');
    LOOP
        FETCH mec_pue INTO reg_mec;
        EXIT WHEN mec_pue%NOTFOUND;
        IF reg_mec.sueldo < 2000 THEN
            UPDATE mecanico
               SET sueldo = sueldo * 1.05
              WHERE dni = reg_mec.dni;
        END IF;
    END LOOP;
    CLOSE mec_pue;
END;
```



1.6. Cursores

Programación del servidor: PL/SQL

- Bucle FOR de cursor. Clausula WHERE CURRENT OF nombre_cursor.

```
DECLARE
    CURSOR mec_pue (m_puesto IN mecanico.puesto%TYPE)
    IS SELECT *
        FROM mecanico
        WHERE puesto = m_puesto
        FOR UPDATE OF sueldo;
reg_mec mecanico%ROWTYPE;
BEGIN
    FOR reg_mec IN mec_pue('CHAPA') LOOP
OPEN mec_pue('CHAPA');
LOOP
FETCH mec_pue INTO reg_mec;
EXIT WHEN mec_pue%NOTFOUND;
        IF reg_mec.sueldo < 2000 THEN
            UPDATE mecanico
            SET sueldo = sueldo * 1.05
            WHERE CURRENT OF mec_pue;
        END IF;
    END LOOP;
CLOSE mec_pue;
END;
```



1.7. Procedimientos almacenados

Programación del servidor: PL/SQL

- Conjunto de sentencias asociados a un nombre y que puede ser llamado
- Quedan **almacenados en la base de datos** y facilitan:
 - Desarrollo de aplicaciones (quedan centralizadas en el servidor de base de datos)
 - Integridad y seguridad
 - Optimización del tiempo de ejecución (código precompilado y compartido entre distintos usuarios y llamadas)
 - Ahorro de memoria (se carga una vez y se reutiliza por las diferentes llamadas)
- Sintaxis para la definición:

```
CREATE PROCEDURE nombre_procedimiento [(parametro {IN | OUT | IN OUT} tipo, ...)]  
  IS bloque_sentencias;
```

```
CREATE FUNCTION nombre_funcion [(parametro IN tipo, ...)] RETURN tipo  
  IS bloque_sentencias;
```

- Sintaxis para la llamada:
 - desde un bloque PL/SQL:
 - nombre_procedimiento [(valor | variable), ...]
 - variable := nombre_función [(valor, ...)]
 - desde el prompt de SQL*Plus:
 - `CALL nombre_procedimiento [(valor | variable), ...];`



1.7. Procedimientos almacenados

Programación del servidor: PL/SQL

Procedimiento para calcular la suma de los sueldos por puesto

```
CREATE PROCEDURE sumar_puesto (vpuesto IN mecanico.puesto%TYPE,
                               vsuma IN OUT NUMBER) IS
  CURSOR sueldos (cpuesto IN mecanico.puesto%TYPE) IS
    SELECT sueldo FROM mecanico WHERE puesto = cpuesto;
  csueldo mecanico.sueldo%TYPE;
BEGIN
  OPEN sueldos(vpuesto);
  vsuma:=0;
  LOOP
    FETCH sueldos INTO csueldo; EXIT WHEN sueldos%NOTFOUND;
    vsuma:=vsuma+csueldo;
  END LOOP;
  CLOSE sueldos;
END;
/
DECLARE
  total_nomina_chapa NUMBER;
BEGIN
  sumar_puesto('CHAPA',total_nomina_chapa);
END;
```



1.7. Procedimientos almacenados

Programación del servidor: PL/SQL

Función para calcular la suma de los sueldos por puesto

```
CREATE FUNCTION sumar_puesto (vpuesto IN mecanico.puesto%TYPE) RETURN NUMBER IS
    CURSOR sueldos (cpuesto IN mecanico.puesto%TYPE) IS
        SELECT sueldo FROM mecanico WHERE puesto = cpuesto;
    csueldo mecanico.sueldo%TYPE;
    vsuma NUMBER;
BEGIN
    OPEN sueldos(vpuesto);
    vsuma:=0;
    LOOP
        FETCH sueldos INTO csueldo;
        EXIT WHEN sueldos%NOTFOUND;
        vsuma:=vsuma+csueldo;
    END LOOP;
    CLOSE sueldos;
    RETURN vsuma;
END;
```

```
SQL> SELECT DISTINCT puesto,sumar_puesto(puesto) "Nomina puesto"
      FROM mecanico;
```



1.7. Procedimientos almacenados

Programación del servidor: PL/SQL

Función para calcular la suma de los sueldos por puesto (usando bucle FOR)

```
CREATE FUNCTION sumar_puesto (vpuesto IN mecanico.puesto%TYPE) RETURN NUMBER IS
    CURSOR sueldos (cpuesto IN mecanico.puesto%TYPE) IS
        SELECT sueldo FROM mecanico WHERE puesto = cpuesto;
    csueldo mecanico.sueldo%TYPE;
    vsuma NUMBER;
BEGIN
    vsuma:=0;
    FOR sueldo IN sueldos(vpuesto) LOOP
        vsuma:=vsuma+sueldo;
    END LOOP;
    RETURN vsuma;
END;
```

```
/

SQL> SELECT DISTINCT puesto,sumar_puesto(puesto) "Nomina puesto"
      FROM mecanico;
```



1.7. Procedimientos almacenados

Programación del servidor: PL/SQL

- Los procedimientos y funciones se pueden agrupar en **paquetes**:
 - Modularidad
 - Diseño de aplicaciones: especificación e implementación independientes
 - Ocultamiento de información
 - Código compartido: variables y cursores persisten durante las sesiones y se comparten por los procedimientos y funciones
 - Mejora en la ejecución: el paquete se carga en memoria reduciéndose la tasa de I/O
- Especificación (parte visible)

```
CREATE PACKAGE nombre_paquete AS
    [variables y cursores públicos]
    cabeceras de funciones y procedimientos
END;
```

```
CREATE PACKAGE nomina_mecanicos AS
    total_nomina NUMBER;
    FUNCTION sumar_puesto (vpuesto IN mecanico.puesto%TYPE) RETURN NUMBER;
END;
```



1.7. Procedimientos almacenados

Programación del servidor: PL/SQL

- Implementación (parte oculta)

```
CREATE PACKAGE BODY nombre_paquete AS
    [variables y cursores privados]
    implementación de funciones y procedimientos
    [BEGIN
        sentencias_inicialización]
END;
```

- Llamada

- Variables y cursores públicos: `nombre_paquete.nombre_variable`
- Procedimientos y funciones: `nombre_paquete.nombre_procedimiento(...)`



1.7. Procedimientos almacenados

Programación del servidor: PL/SQL

```
CREATE PACKAGE BODY nomina_mecanicos AS

  FUNCTION sumar_puesto (vpuesto IN mecanico.puesto%TYPE) RETURN NUMBER IS
    CURSOR sueldos (cpuesto IN mecanico.puesto%TYPE) IS
      SELECT sueldo FROM mecanico WHERE puesto = cpuesto;
    csueldo mecanico.sueldo%TYPE;
    vsuma NUMBER;
  BEGIN
    OPEN sueldos(vpuesto);
    vsuma:=0;
    LOOP
      FETCH sueldos INTO csueldo;
      EXIT WHEN sueldos%NOTFOUND;
      vsuma:=vsuma+csueldo;
    END LOOP;
    CLOSE sueldos;
    RETURN vsuma;
  END;

  BEGIN
    total_nomina:=0;
  END;

  /

  BEGIN
    nomina_mecanicos.total_nomina := nomina_mecanicos.total_nomina +
                                     nomina_mecanicos.sumar_puesto('CHAPA');
  END;
```



1.7. Disparadores

Programación del servidor: PL/SQL

- Regla ECA: Evento - Condición - Acción
- Código almacenado asociado a una tabla
- El código se ejecuta (dispara) cuando se produce un determinado evento (inserción, actualización o borrado) sobre la tabla
- Los DBMSs se convierten en software activo, que genera eventos que son capturados por el disparador
- Útil para:
 - Generar valores de columnas derivadas
 - Validaciones y restricciones complejas sobre datos de entrada
 - Mantenimiento de auditorias, información resumen o copias de datos
- Pero debe usarse con cautela:
 - No debe enmascarar errores de diseño (por ejemplo mantener valores duplicados o errores de normalización)
 - Lógica sencilla: no deben tener más de 30 líneas, si recursividad
 - No deben sustituir a otros mecanismos de mantenimiento de la integridad (*constraints*), tales como claves externas, CHECK, UNIQUE, DELETE CASCADE, NOT NULL, etc.



1.7. Disparadores

Programación del servidor: PL/SQL

- Definición del disparador

Sentencia de disparo

```
CREATE TRIGGER nombre_disparador  
  {BEFORE | AFTER | INSTEAD OF} {DELETE | INSERT | UPDATE [OF atributo]}  
  [OR {DELETE | INSERT | UPDATE [OF atributo]}]...  
ON nombre_tabla  
  [FOR EACH ROW [WHEN condición]]  
bloque_sentencias;
```

Tipo de disparador



1.8 Disparadores

Programación del servidor: PL/SQL

- Tipo de disparador depende de:
 - cuándo se ejecuta: `BEFORE` ó `AFTER`
 - cuántas veces se ejecuta:
 - disparador de sentencia:
 - el código se ejecuta una única vez
 - disparador de tupla (`FOR EACH ROW`):
 - el código se ejecuta una vez por cada tupla afectada por la sentencia DML
 - permite acceder a los valores de las tuplas afectadas por la sentencia DML (:new y :old)
 - permite modificar los valores nuevos de las tuplas (sólo para `BEFORE`) (:new.xxx := ...)
 - Si hay varios disparadores para la misma sentencia DML, el orden de ejecución es:
 1. `BEFORE` de sentencia
 2. para cada tupla afectada por la sentencia DML:
 - 2.1 `BEFORE` de tupla
 - 2.2 sentencia DML
 - 2.3 `AFTER` de tupla
 3. `AFTER` de sentencia



- Dentro del código del disparador
 - Acceso a los valores antiguos y nuevos en disparadores de tupla `:old` y `:new` (sin `:` cuando se utilizan en cláusula `WHEN`)
 - Si el disparo se puede producir por más de una operación:

```
INSERTING, DELETING y UPDATING [ ('nombre_columna') ]
```

```
BEGIN
```

```
  IF INSERTING THEN ...
```

```
  ELSIF UPDATING ('sueldo') THEN ...
```

```
  END IF;
```

```
END;
```


1.8. Disparadores

Programación del servidor: PL/SQL

Disparador para controlar que la subida de sueldos, para trabajadores no directivos, no sea superior al 5%

```
CREATE TRIGGER aumento_sueldo
  BEFORE UPDATE OF sueldo ON mecanico
  FOR EACH ROW WHEN (new.puesto != 'DIRECCION')
DECLARE
  subida NUMBER;
BEGIN
  subida := :new.sueldo - :old.sueldo;
  IF (subida > :old.sueldo * 0.05) THEN
    RAISE_APPLICATION_ERROR(-20000, 'Subida excesiva');
  END IF;
END;
```

Si se provoca una excepción, no tienen efecto ni las sentencias del disparador ni la sentencia que causó el disparo



1.8. Disparadores

Programación del servidor: PL/SQL

Disparador para **establecer** que la subida de sueldos, para trabajadores no directivos, sea como máximo de un 5%

```
CREATE TRIGGER aumento_sueldo
  BEFORE UPDATE OF sueldo ON mecanico
  FOR EACH ROW WHEN (new.puesto != 'DIRECCION')
DECLARE
  subida NUMBER;
BEGIN
  subida := :new.sueldo - :old.sueldo;
  IF (subida > :old.sueldo * 0.05) THEN
    :new.sueldo := :old.sueldo * 1.05;
  END IF;
END;
```



1.8 Disparadores

Programación del servidor: PL/SQL

Los disparadores de sustitución (**INSTEAD OF**) :

- sólo pueden definirse sobre vistas
- disponibles a partir de Oracle 8
- se activan en lugar de la orden DML que provocó el disparo
- deben estar definidos a nivel de fila.

```
CREATE VIEW vista AS
SELECT edificio, sum(numero_asientos) FROM habitaciones
GROUP BY edificio;
```

Ejemplo:

```
DELETE FROM vista WHERE edificio='edificio 7'; /*BORRADO INVÁLIDO*/
```

Sin embargo, se puede crear un disparador de sustitución que efectúe el borrado equivalente pero sobre la tabla habitaciones.

```
CREATE TRIGGER borra_en_vista
INSTEAD OF DELETE ON vista
FOR EACH ROW
BEGIN
DELETE FROM habitaciones
WHERE edificio = :old.edificio;
END borra_en_vista;
```



- Disparador Compuesto
 - Un mismo disparador es ejecutado en diversos instantes del ciclo de vida del evento
 - Sólo disponible a partir de Oracle 11g en adelante
 - Útiles si hay que compartir datos en diversos instantes del mismo evento

- Definición del disparador compuesto

```
CREATE TRIGGER nombre_disparador
  {FOR {DELETE | INSERT | UPDATE [OF atributo]}
    [OR {DELETE | INSERT | UPDATE [OF atributo]}]}...
  ON nombre_tabla
  [FOR EACH ROW [WHEN condición]]
COMPOUND TRIGGER IS
  [seccion inicial]
  [[variable]]...
  [[subprograma]]...
  [{bloque_sentencia_instante}]...
/
```

```
bloque_sentencia_instante::=
{ {AFTER|BEFORE|INSTEAD OF}  {STATEMENT | EACH ROW}
BEGIN
{bloque_sentencias}
END { {AFTER|BEFORE|INSTEAD OF}  {STATEMENT | EACH ROW}
```



- Ejemplo de disparador compuesto: copia el contenido de una tabla (`ORDER`) en una tabla histórica (`ORDER_ARCHIVE`) automáticamente

```
CREATE OR REPLACE TRIGGER TRG_INS_ORDER
FOR INSERT ON ORDERS
COMPOUND TRIGGER
    TYPE ORDER_T IS TABLE OF ORDER_ARCHIVE%ROWTYPE
    INDEX BY PLS_INTEGER;
    L_ORDERS ORDER_T;
    I NUMBER := 0;

AFTER EACH ROW IS
BEGIN
    I := I+1;
    L_ORDERS(I).ORD_ID := :NEW.ORD_ID;
    L_ORDERS(I).ORD_CODE := :NEW.ITEM_CODE;

END AFTER EACH ROW;

AFTER STATEMENT IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('Statement level loading');
    FORALL J IN 1..L_ORDERS.COUNT
    INSERT INTO ORDER_ARCHIVE VALUES L_ORDERS(J);
    L_ORDERS.DELETE;
    I := 0;

END AFTER STATEMENT;

END;
/
```



1.8 Disparadores

Programación del servidor: PL/SQL

- Tabla Mutante:
 - Tabla que está siendo modificada por un INSERT (de más de una tupla o after), un UPDATE o un DELETE
 - Los **disparadores de tupla** no pueden leer ni modificar las tuplas de una tabla mutante

```
CREATE TRIGGER aumento_sueldo
  BEFORE UPDATE OF sueldo ON mecanico
  FOR EACH ROW
DECLARE
  subida NUMBER;
  media NUMBER;
BEGIN
  SELECT avg(sueldo)
  INTO media
  FROM mecanico;

  subida := :new.sueldo - :old.sueldo;
  IF (subida > media*0.05) THEN
    :new.sueldo := :old.sueldo + media * 0.05;
  END IF;
END;
```



1.8 Disparadores

Programación del servidor: PL/SQL

- Tabla Mutante:

- Solución 1: utilizar una tabla auxiliar y un nuevo disparador de sentencia
- Disparador de tupla:

- Lee de la tabla auxiliar los datos que necesita

- Guarda en la tabla auxiliar los datos que quiere modificar

- Nuevo disparador de sentencia:

- BEFORE: Almacena en la tabla auxiliar los datos que necesita el disparador de tupla

- AFTER: Lee de la tabla auxiliar lo almacenado por el disparador de tupla y realiza los cambios

```
create table aux (media number);
```

```
create or replace trigger aumento_sueldo_sentencia  
before update of sueldo on MECANICOS
```

```
declare
```

```
media_s number;
```

```
begin
```

```
select avg(sueldo) into media_s  
from mecanicos;
```

```
delete from aux;
```

```
insert into aux values (media_s);
```

```
end;
```

```
create or replace trigger aumento_sueldo  
before update of sueldo on MECANICOS
```

```
for each row
```

```
declare
```

```
subida number;
```

```
media_s number;
```

```
begin
```

```
select media into media_s from aux;
```

```
subida := :new.sueldo - :old.sueldo;
```

```
if (subida > media_s*0.05) then
```

```
:new.sueldo := :old.sueldo +  
media_s * 0.05;
```

```
end if;
```

```
end;
```



1.8 Disparadores

Programación del servidor: PL/SQL

- Tabla Mutante:
 - Solución 2: usar único disparador compuesto
 - Instante BEFORE STATEMENT
 - Lee de la tabla auxiliar los datos que necesita
 - Guarda en una variable local del disparador los datos que quiere modificar
 - Instante BEFORE/AFTER EACH ROW:
 - BEFORE: Almacena en la tabla auxiliar los datos que necesita el disparador de tupla
 - AFTER: Lee de la tabla auxiliar lo almacenado por el disparador de tupla y realiza los cambios

```
create or replace trigger aumento_sueldo
```

```
for sueldo on MECANICOS
compound trigger
  media_s number;
  subida number;
before statement is
begin
  select avg(sueldo) into media_s
  from mecanicos;
end before statement;
```

```
before each row
begin
  subida := :new.sueldo - :old.sueldo;
  if (subida > media_s*0.05) then
    :new.sueldo := :old.sueldo +
                  media_s * 0.05;
  end if;
end each row;
end;
```



Restricciones no procedurales

- Más eficiente.
- Más fácilmente compresible/mantenible
- su semántica es bien conocida → pueden ser manipuladas a través del diccionario de datos siempre que se tenga permisos para ello

Restricciones procedurales

- Mayor capacidad expresiva
- Permiten establecer alternativas frente a un error
- Permiten garantizar la consistencia de las transacciones (*transactional consistency*)

1.9 Manejo de excepciones

Programación del servidor: PL/SQL

- Los errores en tiempo de ejecución provocan excepciones
- Tipos de excepciones:
 - Excepciones del sistema /* (1) */
 - Excepciones definidas por el usuario /* (2) */

```
DECLARE
    ....
    excep EXCEPTION; /* (2) */
BEGIN
    ...
    IF ... THEN
        RAISE excep; /* (2) */
    END IF;
    ...
    IF... THEN
        RAISE_APPLICATION_ERROR(-2000,'error msg');/* (2) */
    END IF;
    ...
    EXCEPTION
        WHEN TOO_MANY_ROWS THEN /* (1) */
            sentencias_manejo_excepcion_sistema;
        WHEN excep THEN /* (2) */
            sentencias_manejo_excepcion_usuario;
        WHEN OTHERS THEN ...;
    END;
```



1.9 Manejo de excepciones

Programación del servidor: PL/SQL

| Excepcion | Se ejecuta ... | SQLCODE |
|---------------------|---|---------|
| ACCESS_INTO_NULL | El programa intentó asignar valores a los atributos de un objeto no inicializado | -6530 |
| COLLECTION_IS_NULL | El programa intentó asignar valores a una tabla anidada aún no inicializada | -6531 |
| CURSOR_ALREADY_OPEN | El programa intentó abrir un cursor que ya se encontraba abierto. Recuerde que un cursor de ciclo FOR automáticamente lo abre y ello no se debe especificar con la sentencia OPEN | -6511 |
| DUP_VAL_ON_INDEX | El programa intentó almacenar valores duplicados en una columna que se mantiene con restricción de integridad de un índice único (unique index) | -1 |
| INVALID_CURSOR | El programa intentó efectuar una operación no válida sobre un cursor | -1001 |
| INVALID_NUMBER | En una sentencia SQL, la conversión de una cadena de caracteres hacia un número falla cuando esa cadena no representa un número válido | -1722 |
| LOGIN_DENIED | El programa intentó conectarse a Oracle con un nombre de usuario o password inválido | -1017 |
| NO_DATA_FOUND | Una sentencia SELECT INTO no devolvió valores o el programa referenció un elemento no inicializado en una tabla indexada | 100 |
| NOT_LOGGED_ON | El programa efectuó una llamada a Oracle sin estar conectado | -1012 |
| PROGRAM_ERROR | PL/SQL tiene un problema interno | -6501 |



1.9 Manejo de excepciones

Programación del servidor: PL/SQL

| Excepcion | Se ejecuta ... | SQLCODE |
|--------------------------------|--|---------|
| ROWTYPE_MISMATCH | Los elementos de una asignación (el valor a asignar y la variable que lo contendrá) tienen tipos incompatibles. También se presenta este error cuando un parámetro pasado a un subprograma no es del tipo esperado | -6504 |
| SELF_IS_NULL | El parámetro SELF (el primero que es pasado a un método MEMBER) es nulo | -30625 |
| STORAGE_ERROR | La memoria se terminó o está corrupta | -6500 |
| SUBSCRIPT_BEYOND_COUNT | El programa está tratando 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 |
| SUBSCRIPT_OUTSIDE_LIMIT | El programa está referenciando un elemento de un arreglo utilizando un número fuera del rango permitido (por ejemplo, el elemento "-1") | -6532 |
| SYS_INVALID_ROWID | La conversión de una cadena de caracteres hacia un tipo rowid falló porque la cadena no representa un número | -1410 |
| TIMEOUT_ON_RESOURCE | Se excedió el tiempo máximo de espera por un recurso en Oracle | -51 |
| TOO_MANY_ROWS | Una sentencia SELECT INTO devuelve más de una fila | -1422 |
| VALUE_ERROR | Ocurrió un error aritmético, de conversión o truncamiento. Por ejemplo, sucede cuando se intenta calzar un valor muy grande dentro de una variable más pequeña | -6502 |
| ZERO_DIVIDE | El programa intentó efectuar una división por cero | -1476 |



1.10 El diccionario de datos

Programación del servidor: PL/SQL

- El propietario es el administrador del sistema, aunque sólo se puede leer
- Dependiendo del usuario se pueden acceder a unas u otras vistas
- El esquema del diccionario depende de cada fabricante
- A lo largo del curso iremos aprendiendo algunas vistas útiles en el caso de Oracle:
 - USER_TABLES
 - USER_CONSTRAINTS
 - USER_CONS_COLUMNS
 - USER_VIEWS
 - USER_TRIGGERS
 - USER_OBJECTS
- Código de procedimientos, funciones y paquetes
 - USER_SOURCE
- Depuración de errores: errores y warnings
 - USER_ERRORS

Acceso a los disparadores en el diccionario de datos de oracle:

```
select trigger_name.table_name from user_triggers;
```

```
select when_clause, trigger_body from user_triggers  
where trigger_name='AUMENTO_SUELDO';
```



1.11 Ejercicios

Programación del servidor: PL/SQL

1. Supón que estás conectado a una BBDD en Oracle desde un programa Java ejecutándose en el cliente. Dados los siguientes supuestos, indica si usarías una vista, una sentencia SQL o un procedimiento almacenado.
 1. El cliente genera la consulta dinámica sobre la marcha dependiendo de gran cantidad de parámetros
 2. Se trata de una consulta extremadamente compleja que finalmente devuelve un único valor
 3. Una consulta que devuelve una relación que quiero utilizar como subconsulta en otras consultas, ocasionalmente
 4. Un conjunto de consultas que siempre se ejecutan secuencialmente



1.11 Ejercicios

Programación del servidor: PL/SQL

2. Usando las tablas `ALUM`, `ALAS`, `ASIG`, `PROF`, `PROFAS`, escribe las cabeceras de los disparadores para los siguientes supuestos:
 1. Generar una excepción en el caso que un profesor imparta menos de 6 créditos de docencia
 2. No permitir que un alumno se matricule de más de 80 créditos un único año
 3. Mantener un histórico de convocatorias `ALAS_HISTORICO` (`alum`, `asig`, `fecha`, `nota`)
 4. Generar una excepción si un alumno intenta matricularse por séptima vez en la misma asignatura

3. Implementar los disparadores del ejercicio 2. ¿En cuales necesitaría una tabla auxiliar? Razona la respuesta



1.11 Ejercicios

Programación del servidor: PL/SQL

4. La independencia de datos implica:
 1. Si modificamos el modelo físico de datos entonces debemos modificar el modelo lógico de datos
 2. Si modificamos el modelo lógico de datos no es necesario modificar el modelo físico de datos
 3. Cualquier modificación en una BBDD conllevará la necesaria modificación de las aplicaciones que la accedan
 4. La independencia de datos se establece solo entre la capa física y lógica de la BBDD
5. Respecto a las variables en PL/SQL:
 1. Pueden ser de tipo base y de tipo ancla
 2. Las de tipo ancla pueden, entre otras cosas, ser del mismo tipo que una columna de una tabla
 3. Las de tipo ancla pueden, entre otras cosas, almacenar varias filas de una tabla
 4. Las de tipo ancla pueden, entre otras cosas, almacenar una única fila de un cursor
6. Respecto a los procedimientos almacenados:
 1. Es un procedimiento que se ejecuta en el DBMS
 2. Es un procedimiento que se almacena en el DBMS pero se ejecuta en el cliente que lo invoca
 3. Es un procedimiento que se ejecuta en el DBMS pero se almacena en el cliente que lo invoca
 4. Permite compartir código entre distintas sesiones



1.11 Ejercicios

Programación del servidor: PL/SQL

7. Sea la sentencia SQL `INSERT X SET X.a=X.a+1`. Un disparador declarado `"CREATE TRIGGER TRG_UPD_X_A AFTER INSERT FOR EACH ROW"`:
 1. Se ejecutará una única vez en cualquier caso
 2. Se ejecutará una vez antes de actualizarse cada una de las filas de la tabla X
 3. No tendrá acceso a las variables :new ni :old
 4. Tendrá acceso de lectura a la tabla X
8. Sea la sentencia SQL `UPDATE X SET X.a=X.a+1`. Un disparador declarado `"CREATE TRIGGER TRG_UPD_X_A AFTER UPDATE"`
 1. Se ejecutará una única vez en cualquier caso
 2. Se ejecutará una vez antes de actualizarse cada una de las filas de la tabla X
 3. No tendrá acceso a las variables :new ni :old
 4. Tendrá acceso de lectura y escritura a la tabla X
9. Una tabla mutante:
 1. No puede ser leída por un disparador de tupla
 2. No puede ser modificada por un disparador de tupla
 3. Puede ser leída por un disparador de sentencia
 4. Es una tabla a la que le ha picado una araña radioactiva
10. El diccionario de datos:
 1. Viene predefinido por el DBMS, que a su vez es el encargado de mantenerlo actualizado
 2. Un usuario solo puede realizar de operaciones de lectura sobre éste
 3. Describe, entre otras cosas, el modelo lógico de datos de la BBDD
 4. Describe, entre otras cosas, el modelo físico de datos de la BBDD



1.11 Ejercicios (ejercicio 2 resuelto)

Programación del servidor: PL/SQL

2. Usando las tablas `ALUM`, `ALAS`, `ASIG`, `PROF`, `PROFAS`, escribe las cabeceras de los disparadores para los siguientes supuestos:
1. Generar una excepción en el caso que un profesor imparta menos de 6 créditos de docencia

```
create or replace trigger min_creditos_profesor
before update
or delete on PROFAS
for each row
```

2. No permitir que un alumno se matricule de más de 80 créditos un único año

```
create or replace trigger max_creditos_alumno
before update
or insert on ALAS
for each row
```



1.11 Ejercicios (ejercicio 2 resuelto)

Programación del servidor: PL/SQL

2. Usando las tablas ALUM, ALAS, ASIG, PROF, PROFAS, escribe las cabeceras de los disparadores para los siguientes supuestos:

3. Mantener un histórico de convocatorias ALAS_HISTORICO (al, asig, fecha, nota)

```
create or replace trigger historico_convocatorias
after update
or insert
or delete on ALAS
for each row
```

4. Generar una excepción si un alumno intenta matricularse por séptima vez en la misma asignatura

```
create or replace trigger max_matriculas_alumno
before update of codal
or update of codas
or insert on ALAS
for each row
```

