

**4.1. Introducción: SQL, SQL3 y PL/SQL**

**4.2. Vistas y Diseño Externo**

**4.3. Bloques y Procedimientos**

**4.4. Disparadores**

# Tema 4.1: Introducción



- Desde 'standard' query language al estándar real SQL92 y su extensión SQL3.

Incorporan elementos, paradigmas (OODB) y mecanismos procedimentales.

- Extensiones procedimentales: bloques, bucles, inst. condicionales, ... *~ Añadir a base nuevos funcionamiento.*

- Comportamiento activo (reglas ECA) *Capturar eventos de datos, para que cuando ocurra cierto evento ocurra x. Disparador (trigger)*

- Diseño externo: control de privilegios (usuarios y roles), vistas, ...

- Diseño físico: índices, clusters, ...

Algunas implementaciones incluyen elementos y mecanismos propios:

- ECA temporal, DB spatial, distribución y paralelismo, ...

## Tema 4.2: Concepto de Vista

La tabla *personas* tiene cuatro columnas

Qué no te digo que tiene tres!

Lo que puede ver cada usuario, pero no todos ven todas las tablas, atrib. o filas.

Es una definición de tabla del que el usuario puede ver, es una redun-  
dancia controlada. no es que se lo muestre  
Pero a la hora de crearlos se hace de distinta manera. Pero esto  
están sup.

Una vista debe ser operativa, consultar, modificar e insertar.  
La BD lo hace cuando a trivial, cuando no lo es el programador lo debe definir.

Grado entre 1 y todas las columnas de la table

La *relación base* será la conjunción de todas ellas (esq. lógico global)  
Cardinalidad al menos 1 pero como mucho todas.

Se definirá aparte la versión específica de cada tipo de usuario

Existirán por tanto diversos esquemas (externos) y uno sólo global.

La vista puede reducir el grado y/o la cardinalidad de la relación base, e incluso fusionar varias relaciones en una sola.

Si están en otra table física, hay que actualizar los datos, está controlada. Es más eficiente, lo tienes cerca. Ocupa más, pero priorizamos tiempo

## CREACIÓN DE UNA VISTA *si es física.*

```
CREATE [MATERIALIZED] VIEW <nombre de tabla>
    [( <nombre de columna> [, <nombre de columna> ] ... )
    AS <expresión de consulta> [WITH CHECK OPTION]
```

*~ podemos cambiar el nombre de las columnas entre ( col, col )*

- Las inserciones y las modificaciones solicitadas sobre la vista se realizarán **sobre las tablas fuente** (involucradas en la consulta). *Comprueba y cancela que una fila se haya de la vista, si se solo no lo hace.*
- Las columnas obligatorias omitidas deberán adquirir un valor por otros medios (valor por defecto, o por disparador).
- La vista materializada tiene datos almacenados (conveniente si las tablas fuente son remotas, o la consulta es compleja).
- La snapshot (create snapshot) es materializada y constante.
- WITH CHECK OPTION comprueba en actualizaciones que ningún elemento modificado sea excluido de la vista (eoc, cancela).

- **Vigente / borrador / histórico**

```
CREATE TABLE doc(..., fecha_ini DATE, fecha_fin DATE,...);
```

```
CREATE VIEW borrador AS
```

```
    SELECT * FROM doc WHERE fecha_ini IS NULL;
```

```
CREATE VIEW historico AS
```

```
    SELECT * FROM doc WHERE fecha_fin IS NOT NULL;
```

```
CREATE VIEW vigente AS
```

```
    SELECT * FROM doc WHERE fecha_ini IS NOT NULL  
        AND fecha_fin IS NULL;
```

- **Fusión** de relaciones y atributos **derivados**

```
CREATE TABLE refs(ref NUMBER(8) PRIMARY KEY,  
                  nombre VARCHAR2(25) NOT NULL,  
                  tipo VARCHAR2(5),  
                  coste NUMBER (8,2) NOT NULL, ...);
```

```
CREATE TABLE vats(tipo VARCHAR2(5) PRIMARY KEY,  
                  iva NUMBER(2,2) NOT NULL, ...);
```

```
CREATE VIEW productos(nombre, ref, precio) AS  
    SELECT a.nombre, a.ref, a.coste*(1+b.iva)  
    FROM refs a NATURAL JOIN vats b;
```

# Tema 4.2: Estática Relacional: Tipos de Relación

## Clases de relación:

A) Persistentes: sólo se borran con una acción explícita del usuario

- Relaciones base: Se corresponden con el nivel lógico, tienen existencia por sí mismas y se crean de manera explícita.
- Vistas: Se corresponden con el esquema externo, son derivadas, nominadas, no tienen datos almacenados, solo se almacena su definición en términos de otras relaciones (redundancia lógica).
- Vistas materializadas: Se corresponden con el nivel interno, son derivadas como las vistas, pero tienen datos almacenados (red. física).

Tabla con todos los datos

Redefinición lógica de una tabla sin datos propio

Redefinición física de una tabla con datos propio. Redundancia contra y actualizac.

Copia de los datos congelada, para gestionar los datos de un momento y luego se elimina.

Instantáneas (snapshots): fotografía de la tabla en un momento del tiempo (almacenada físicamente); orientada al proceso atómico.

B) Temporales: desaparecen al ocurrir un determinado evento (sin especificar una acción de borrado). Por ejemplo, al acabar la sesión o una transacción. Pueden ser de usuario (local temp table, cursor) o de sistema (workspace).

transaccional: se borra con o de sesión: se borra al salir.

- **Diseño Externo:** existen dos perfiles de usuario: *oficina* y de *RRHH*...

```
CREATE TABLE empleados_ALL
```

```
(DNI NUMBER(8) PRIMARY KEY,  
 nombre VARCHAR2(25) NOT NULL,  
 tlf NUMBER(9) UNIQUE,  
 salario NUMBER (8,2), ...);
```

```
CREATE VIEW empleados(nombre, telefono) AS
```

```
SELECT nombre, tlf FROM empleados_ALL;
```

```
CREATE MATERIALIZED VIEW asalariados AS
```

```
SELECT nombre, DNI, salario FROM empleados_ALL;
```

- ¿podrán los usuarios borrar/actualizar/**insertar** en estas vistas?



## Gestión de privilegios de usuario en PL/SQL

- Elementos: Usuarios, perfiles y Roles

```
CREATE USER <username> IDENTIFIED BY <password>
[DEFAULT TABLESPACE <tablespace>]
[QUOTA <size> ON <tablespace>]
[PROFILE <profilename>]
[PASSWORD EXPIRE]
[ACCOUNT {LOCK|UNLOCK}] ;
```

local user

nº sesiones  
tiempo conex.  
nº accesos  
CPU  
RAM ...

```
CREATE PROFILE <profilename> LIMIT <resources>;
```

```
CREATE ROLE <rolename> ~ conjunto de privilegios
{NOT IDENTIFIED| IDENTIFIED BY <password>;}
```

## Gestión de privilegios de usuario en PL/SQL

- Privilegios: GRANT (conceder) y REVOKE (revocar)

```
- GRANT { <rolename> | <sys_privileges> | ALL PRIVILEGES }  
      TO <users/roles> [WITH ADMIN OPTION];
```

INSERT|DELETE|UPDATE|SELECT|...  
- table  
- view  
- materialized view  
- ...

CREATE|ALTER|DROP  
- session  
- user  
- role  
- table  
- tablespace  
- trigger  
- index  
- cluster  
- ...

```
- GRANT { <object_privileges> | ALL PRIVILEGES }  
      [(column [, ...])] ON [schema.]<object> TO <users/roles>  
      [WITH HIERARCHY OPTION] [WITH GRANT OPTION];
```

```
- REVOKE <privileges> [ON <object>] FROM <users/roles>;
```

## Tema 4.3: SQL procedimental - Bloque

Estructura de un bloque: tiene tres partes: **declaraciones**, **cuerpo**, y **excepciones**

```
[DECLARE
  varname type; [...] ]
```

~ se declara antes del bloque.

```
BEGIN
```

```
<código procedimental>
```

```
[EXCEPTION
```

```
WHEN ... THEN ...; [...] ]
```

~ se declaran al final las excepciones del propio bloque de código.

```
END;
```

Para hacer try/catch se anidan begin's

- Los bloques nominados (*function*, *procedure*, ...) omiten la keyword DECLARE
- Se puede almacenar un bloque convirtiéndolo en función o procedimiento

Se omit  
DECLARE

```
CREATE OR REPLACE PROCEDURE name(params) IS
  <declaraciones>
  BEGIN
    <código>
  END;
```

} Código

- Las **funciones** deben incluir la instrucción return(valor)

```

CREATE OR REPLACE FUNCTION name(params) RETURN CHAR IS
<declaraciones>
BEGIN
    <código>
    RETURN <valor>;
END;

```

*~ tipo de valor*

*Código*

- Las **invocaciones** a procedimientos almacenados deben hacerse siempre desde dentro de un bloque (procedimiento/disparador/...) o con la instrucción EXEC

```

BEGIN nombre (parámetro); END;
EXEC my_proc("Hello world");

```

*— debe otro bloque.*

*~ desde la consola*

- **PACKAGE** es una colección de variables y procedimientos almacenados
- Su creación requiere dos pasos: descripción e implementación (cuerpo)
- Descripción del paquete

```
CREATE OR REPLACE PACKAGE my_package AS  
    <declaración variables, cabeceras proc., etc>  
END my_package;
```

*{ ~ cabeceras  
variable global*

- Implementación (cuerpo del paquete)

```
CREATE OR REPLACE PACKAGE BODY my_package AS  
    <descripción completa de cada procedimiento>  
END my_package;
```

*{ completos      { variables para el  
paquete, no globales.*

## Tema 4.4: Concepto de Disparador

Se incluyen en el estándar SQL3 (1999)

- Son procedimentales (no declarativos).

*Código que se ejecuta ante un determinado evento.*

- A diferencia de las restricciones de rechazo, la acción de los disparadores es definida por el usuario.

<def\_trigger> ::=

**CREATE OR REPLACE TRIGGER** [<nombre>]

<tiempo activación acción>

*quando ⇒ Antes de, después de, en vez de*

<evento disparador>

*porque*

**ON** <nombre tabla>

*donde*

<nivel de activación>

*Si se ejecuta granularidad. ⇒ Para cada fila o para la operación propia. insert*

<bloque definiendo la acción disparada>

*La acción a ejecutar*

- Lista de alias:

```
<alias> ::= OLD [AS]<nombre correlacion valores antig.>
| NEW [AS]<nombre correlacion valores antig.>
| OLD_TABLE [AS]<nombre correlacion valores antig.>
| NEW_TABLE [AS]<nombre correlacion valores antig.>
```

OLD/NEW indica el valor antes/después de la columna antes de borrar, modificar o insertar

- Nivel de activación:

```
<nivel activación acción>::=
```

```
[FOR EACH {ROW | STATEMENT}]
```

:old, :new

old\_table, new\_table  
(oracle.javadb)

en Oracle,  
por defecto  
'STATEMENT'

el nivel tambien  
existe, pero no  
en el gestor.

valor de la fila operada antes de  
valor de la fila después de la operación

### - Temporalidad del disparador:

*Reescribir.*

<tiempo activación acción> ::= AFTER / BEFORE / INSTEAD OF

“Instead of” se utiliza asociado a una vista para realizar operaciones de actualización que no están permitidas en las vistas.

### - Definición del evento:

<evento disparador> ::= INSERT / DELETE /  
UPDATE [OF <lista columnas disparador>]

Se podrían componer eventos con las operaciones lógicas básicas entre estos eventos.

*Alarma  
disparador indicada  
ya que la base  
en la fila  
de terminada.*



# Tema 4.4: Disparador como regla ECA

**CREATE** [OR REPLACE] **TRIGGER** nombre

EVENTO

{**BEFORE** | **AFTER** | **INSTEAD OF**}

{**INSERT**|**DELETE**|**UPDATE** [OF <atributo>]} **ON** <tabla>

[**FOR EACH ROW** | **STATEMENT**]

[**FOLLOWS** disparador]

*se ejecutará después del indicado. Secuencia de disparo*

CONDICIÓN

[**ENABLE** | **DISABLE**]

[**WHEN** condición]

{ **CALL** procedimiento (parámetros)

*Lo llama*

| <bloque> } ;

ACCIÓN

```
<bloque> :=  [DECLARE ...]
              BEGIN
                cuerpo del trigger
              [EXCEPTION
                [WHEN <excp> THEN... ]
              END;
```

# Tema 4.4: Disparador compuesto

**CREATE** [OR REPLACE] **TRIGGER** nombre

EVENTO

**FOR** {**INSERT**|**DELETE**|**UPDATE** [OF <atributo>]} **ON** <tabla>

[**FOLLOWS** ...] [**ENABLE**|**DISABLE**] [**WHEN** ...]

CONDICIÓN

**COMPOUND TRIGGER**

[**DECLARE** ...]

**BEFORE STATEMENT IS**

**BEGIN** ... cuerpo del trigger  
**END BEFORE STATEMENT;**

ACCIÓN 1: BS

**BEFORE EACH ROW IS**

**BEGIN** ... cuerpo del trigger  
**END BEFORE EACH ROW;**

ACCIÓN 2: BER

**AFTER EACH ROW IS**

**BEGIN** ... cuerpo del trigger  
**END AFTER EACH ROW;**

ACCIÓN 3: AER

**AFTER STATEMENT IS**

**BEGIN** ... cuerpo del trigger  
**END AFTER STATEMENT;**

ACCIÓN 4: AS

**END** nombre;

Por ser compuesto  
comparte una zona  
declarativa global

Comparten la declaración, y se pueden definir  
variables para los 4 posibles y todos quedan en el trigger.

# Tema 4.4: El error de TABLA MUTANTE

BEFORE ...

FOR EACH STATEMENT

BEFORE ... FOR EACH ROW

AFTER ... FOR EACH ROW  
BEFORE ... FOR EACH ROW

AFTER ... FOR EACH ROW  
BEFORE ... FOR EACH ROW

AFTER ... FOR EACH ROW

BEFORE ... FOR EACH ROW

AFTER ... FOR EACH ROW

AFTER ...

FOR EACH STATEMENT

*Tabla no estable durante la operación.*  
*Paraproveer si se produce, hay que tener cuidado con insertar*  
*La Mientras opera filas, antes y después es estable. foreach row lo provoca, y statement no lo provoca*

<operación sobre tabla>

<operación sobre fila>

<operación sobre fila>

<operación sobre fila>

...

<operación sobre fila>

**! TABLA MUTANTE !**

*Solución:*



**Ejemplo 1:** El presupuesto de un departamento es la suma de los sueldos de los empleados que pertenecen al mismo. Debe controlarse quién hace cambios.

Temporalidad

```
CREATE OR REPLACE TRIGGER presupuesto_departamento  
AFTER INSERT ON EMPLEADO
```

Evento

```
FOR EACH ROW
```

Granularidad o tiempo de activación

```
BEGIN
```

```
    UPDATE DEPARTAMENTO
```

```
        SET presupuesto = presupuesto + :NEW.sueldo
```

```
        WHERE cod_dep = :NEW.dep      ;
```

```
    INSERT INTO tabla_control
```

```
        VALUES (:NEW.cod_emp, USER, SYSDATE) ;
```

```
END;
```

**¿por qué esa granularidad?** Notar que depende de la acción.

**Ejemplo 2:** El precio de los productos debe actualizarse al coste base más IVA.

Temporalidad

CREATE OR REPLACE TRIGGER ACT\_precios

BEFORE INSERT ON Productos Evento

FROM EACH ROW Granularidad o tiempo de activación

DECLARE iva NUMBER(3,2);

BEGIN

SELECT porcentaje INTO iva FROM tipos\_iva

WHERE tipo\_iva = :NEW.tipo\_iva;

NEW.precio := :NEW.coste\_base \* (1+iva);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN DBMS\_OUTPUT.PUT\_LINE('wrong type');

WHEN OTHERS THEN DBMS\_OUTPUT.PUT\_LINE('Some error occurred');

END;

**¿por qué esa temporalidad?** Notar que la temporalidad depende de la semántica.

### Ejemplo 3: Impedir el borrado de evidencias en la tabla *Pruebas*.

*de Rechazo.*

Temporalidad

```
CREATE OR REPLACE TRIGGER NO_DELETE  
BEFORE DELETE ON Pruebas
```

Evento

Granularidad: STATEMENT

```
DECLARE no_borres EXCEPTION;
```

```
BEGIN
```

*En Oracle*

```
RAISE no_borres;
```

*RAISE APPLICATION ERROR*

```
EXCEPTION
```

```
WHEN no_borres THEN
```

```
DBMS_OUTPUT.PUT_LINE('Do not do that!');
```

```
END;
```

Observa la temporalidad y la granularidad en este caso → *Semántica de Rechazo*

- Las restricciones de rechazo impiden actualizaciones no válidas, y en ocasiones dificultan operaciones válidas: *Desactivar una Constraint, pero con cabeza.*
  - auto-referencias FK
  - referencias de exclusión mutua (se debe intentar evitarlas, en lo posible...)
  - etc.
- Para estos casos, se cuenta con un abanico de soluciones:
  - Hacer la operación en una sola instrucción (p.e., insertar varias tuplas)
  - Hacer la op. en falso (con valores nulos o erróneos) y luego modificarla
  - **Deshabilitar restricciones y/o disparadores**
    - `ALTER TABLE <table-name> {DISABLE|ENABLE} CONSTRAINT <c_name>;`
    - `ALTER TABLE <table-name> {DISABLE|ENABLE} ALL TRIGGERS;`
    - `ALTER TRIGGER <trigger-name> {DISABLE | ENABLE};`
  - **Diferir restricciones** (que no se comprobarán hasta hacer COMMIT)

### Disparadores DDL *Sobre las tablas de metadatos.*

Son disparadores *sobre el catálogo*. Eventos que se pueden capturar:

- CREATE
- ALTER
- DROP
- GRANT
- REVOKE
- ...
- DDL (cualquiera)

### Disparadores DB: *Para crear tablas de auditorías. Para almacenar eventos*

Capturan eventos sobre la base de datos (o sobre el esquema).

```
{AFTER STARTUP | BEFORE SHUTDOWN | AFTER DB_ROLE_CHANGE}  
ON DATABASE
```

```
{AFTER LOGON | BEFORE LOGOFF | AFTER SERVERERROR | AFTER SUSPEND}  
ON {DATABASE | SCHEMA}
```