Tema 4: Relacional Avanzado

4.1. Introducción: SQL, SQL3 y PL/SQL

4.2. Vistas y Diseño Externo

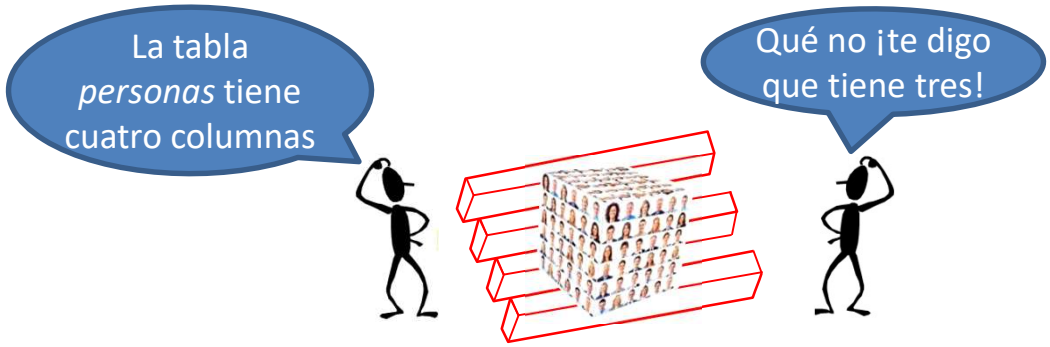
4.3. Bloques y Procedimientos

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,…
  - Comportamiento activo (reglas ECA)
  - 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



- Un solo objeto (tabla) admite varias definiciones para distintos tipos de usuario
- La *relación base* será la conjunción de todas ellas (esq. lógico global)
- 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.

Tema 4.2: Creación de Vistas

CREACIÓN DE UNA VISTA

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

- Las inserciones y las modificaciones solicitadas sobre la vista se realizarán sobre las tablas fuente (involucradas en la consulta).
- 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).

Tema 4.2: Ejemplos de uso de Vistas

- **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;
```

Tema 4.2: Ejemplos de uso de Vistas

- **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).
- 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).

Tema 4.2: Ejemplos de uso de Vistas

- **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?

Tema 4.2: Control de Privilegios (LCD)

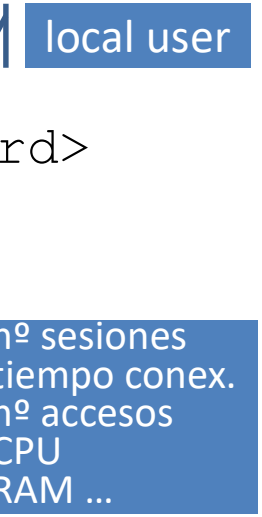
Gestión de privilegios de usuario en PL/SQL

- 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}] ;

CREATE PROFILE <profilename> LIMIT <resources>;

CREATE ROLE <rolename>
  {NOT IDENTIFIED| IDENTIFIED BY <password>};
```



Tema 4.2: Control de Privilegios (LCD)

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];
```

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

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

```
- 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; […] ]
BEGIN
    <código procedimental>
[EXCEPTION
    WHEN … THEN …; […] ]
END;
```

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

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

Tema 4.3: PL/SQL procedimental: Bloques

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

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

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

```
BEGIN my_proc("Hello world"); END;

EXEC my_proc("Hello world");
```

Tema 4.3: PL/SQL procedimental: Paquete

- ***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;
```

- Implementación (cuerpo del paquete)

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

Tema 4.4: Concepto de Disparador

Se incluyen en el estándar SQL3 (1999)

• Son procedimentales (no declarativos).

• 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>
        <evento disparador>
        ON <nombre tabla>
        <nivel de activacion>
        <bloque definiendo la acción disparada>
```

Tema 4.4: Estática Relacional, disparadores

- 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}]
```

en Oracle, por defecto 'STATEMENT'

:old , :new

old_table , new_table (oracle.javadb)

Tema 4.4: Estática Relacional, disparadores

- Temporalidad del disparador:

```
<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.

Tema 4.4: Disparador como regla ECA

```
CREATE [OR REPLACE] TRIGGER nombre
{BEFORE | AFTER | INSTEAD OF}
{INSERT|DELETE|UPDATE [OF <atributo>]} ON <tabla>
```

EVENTO

```
[FOR EACH ROW | STATEMENT]
[FOLLOWS disparador]
[ENABLE | DISABLE]
[WHEN condición]
```

CONDICIÓN

```
{ CALL procedimiento(parámetros)  |  <bloque> } ;
```

ACCIÓN

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

Tema 4.4: Disparador compuesto

```
CREATE [OR REPLACE] TRIGGER nombre
FOR {INSERT|DELETE|UPDATE [OF <atributo>]} ON <tabla>
[FOLLOWS …] [ENABLE|DISABLE] [WHEN …]
```

EVENTO

CONDICIÓN

```
COMPOUND TRIGGER
[DECLARE …]
BEFORE STATEMENT IS
BEGIN … cuerpo del trigger
END BEFORE STATEMENT;
BEFORE EACH ROW IS
BEGIN … cuerpo del trigger
END BEFORE EACH ROW;
AFTER EACH ROW IS
BEGIN … cuerpo del trigger
END AFTER EACH ROW;
AFTER STATEMENT IS
BEGIN … cuerpo del trigger
END AFTER STATEMENT;
END nombre;
```

ACCIÓN 1: BS

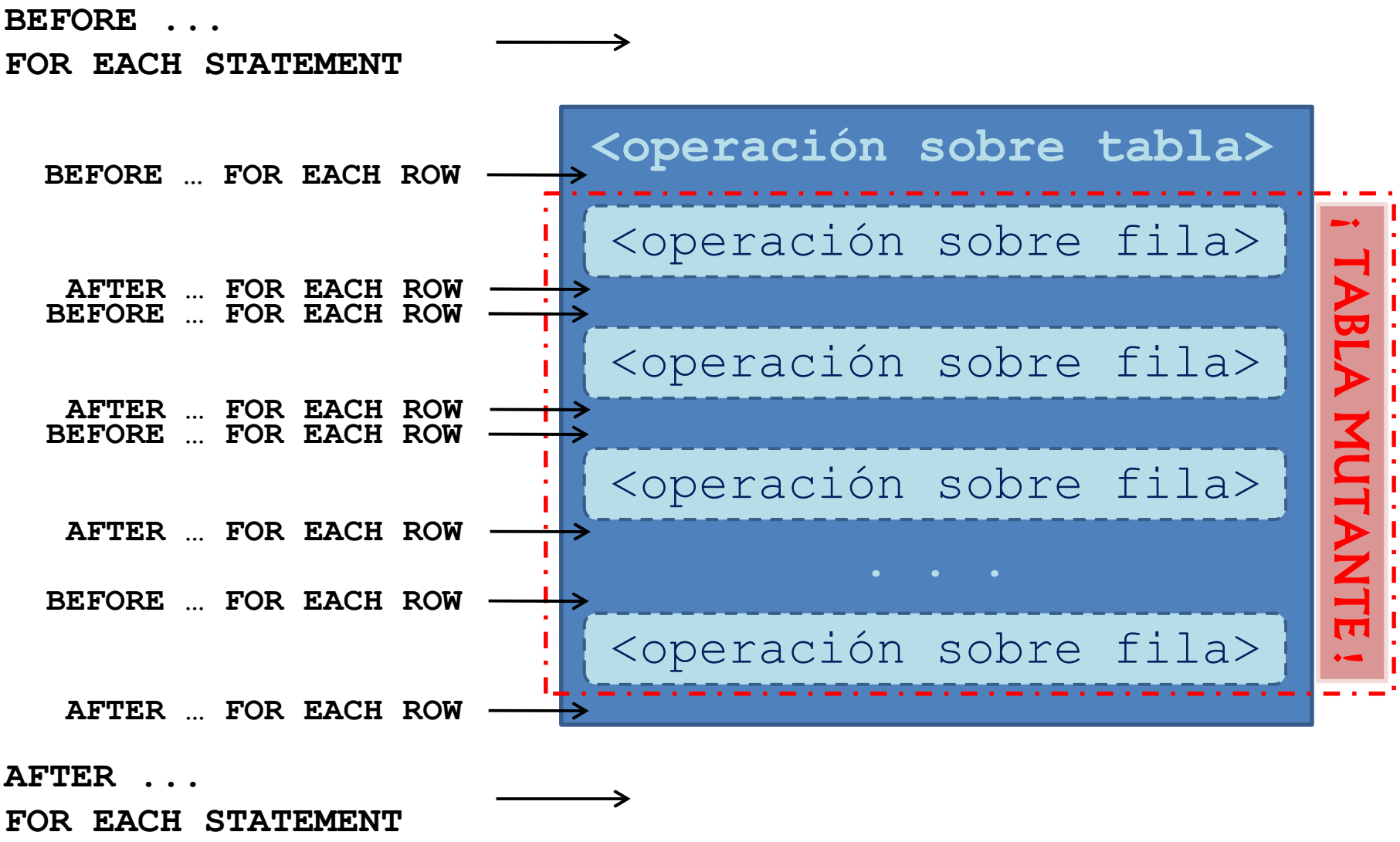
ACCIÓN 2: BER

ACCIÓN 3: AER

ACCIÓN 4: AS

Por ser compuesto comparte una zona declarativa global

Tema 4.4: El error de TABLA MUTANTE



Tema 4.4: Estática Relacional, disparadores

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
FOR EACH ROW
 BEGIN
   UPDATE DEPARTAMENTO
      SET presupuesto = presupuesto + :NEW.sueldo
      WHERE cod_dep = :NEW.dep     ;
   INSERT INTO tabla_control
      VALUES(:NEW.cod_emp,USER,SYSDATE);
 END;
```

Evento

Granularidad o tiempo de activación

¿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

Evento

Granularidad o tiempo de activación

```
CREATE OR REPLACE TRIGGER ACT_precios
BEFORE INSERT ON Productos
FROM EACH ROW
   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.

Tema 4.4: Estática Relacional, disparadores

<u>Ejemplo</u> 3: Impedir el borrado de evidencias en la tabla *Pruebas*.

Temporalidad

Evento

Granularidad: STATEMENT

```
CREATE OR REPLACE TRIGGER NO_DELETE
BEFORE DELETE ON Pruebas
    DECLARE no_borres EXCEPTION;
BEGIN
    RAISE no_borres;
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***

Tema 4.4: Gestión de Restricciones de Rechazo

- Las restricciones de rechazo impiden actualizaciones no válidas, y en ocasiones dificultan operaciones válidas:
  - 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)

Tema 4.4: Est. Relacional – Más disparadores

Disparadores DDL

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

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

Disparadores DB:

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}
```