Introducción a las Bases de Datos TUPAR

Cursada 2008

Clase 6: Restricciones

Facultad de Ciencias Exactas
Universidad Nac. Centro de la Pcia. de Bs. As.

Corrección y consistencia de los datos

- Implícitas
  - Del modelo de datos
  - Especificada y representada en el esquema
- Explícitas
  - Restricciones adicionales del UdeD
  - No pueden ser representadas en el modelo
- Inherentes
  - Se asumen por la definición del modelo de datos
  - No se tienen que especificar
  - Ej. Los atributos son atómicos, las relaciones son simétricas.

Clasificación de restricciones

- De estado
  - Restringen los valores de la BD
  - Es consistente si satisface todas las restricciones de estado
- De transición
  - Restringen el cambio de estado, no un estado individual
  - ej. El sueldo de un empleado no puede decrecer (requiere conocer el estado 'viejo' y el estado 'nuevo' para compararlos)

Clasificación de restricciones de estado

- Unicidad: no puede haber dos valores iguales de un mismo atributo
- Integridad de Entidades: una clave primaria no puede ser ni total ni parcialmente nula
- No Nulidad: los valores de un atributo no pueden ser nulos
- Dominios (conjuntos de valores): todos los valores de un dominio deben pertenecer a un dominio definido
- Cardinalidad: el número de valores para un atributo debe caer en un rango definido (ej. Numero de padres puede ser 0, 1 ó 2)

Clasificación de la restricciones de estado

- Cardinalidad de relación: el número de veces que una entidad participa de una relación (ej. Los estudiantes sólo pueden inscribirse en un máximo de 5 cursos)
- Participación en una relación: participación obligatoria u opcional en una relación.
- Inclusión: todos los valores de un atributo son también valores de otro atributo (ej. Conj. de delegados de curso $\subset$ conj. de alumnos del curso)

Clasificación de la restricciones de estado

- Cobertura: todos los valores de un atributo son también valores de uno de otros conjuntos de atributos (ej. autos $\cup$ botes $\cup$ aviones = vehículos)
- Exclusión: los valores de un atributo no pueden ser al mismo tiempo valores de otro atributo (ej. Nros de alumnos libres y de alumnos regulares)
- Referencial: un valor de un atributo se garantiza que existe como valor de otro atributo, generalmente la clave de otra entidad

Restricciones Generales

Consisten en predicados o condiciones sobre uno o más atributos.

Conocidas generalmente como Reglas del Negocio (business rules)

- Inter-atributo
  - Fecha de nac < fecha de contratación
  - Cantidad comprada = cantidad enviada
- Funciones de dominio
  - promedio de alumno >= 4
- Atributos derivados
  - Nota final en curso = (nota examen + nota lab) / 2

Especificación de restricciones en el M. Relacional

- Inherentes
  - Ya están en el modelo (ej. Valores de dominio atómicos)
- Implícitas
  - en el DDL (Data Definition Language) (ej. Integridad referencial)
- Explícitas
  - Declarativamente *checks, assertions o triggers*
  - Proceduralmente *transacciones*

  (ej. Un docente no puede dirigir más de 3 tesis simultáneamente)

Restricciones de Dominio

Especifican los valores válidos para un atributo (además del tipo).

El estandar provee dos mecanismos:

✓ **mediante la claúsula CHECK**

```
precio  NUMERIC(4,2)  NOT NULL  CHECK (precio > 0)
tdoc  VARCHAR(3) NOT NULL  CHECK (tdoc IN ('DNI', 'LE', 'LC', 'CI'))
```

✓ **mediante la definición explícita de DOMINIOS**

```
CREATE DOMAIN  nom_dom AS tipo
    [ DEFAULT valor_defecto ] [ CHECK condicion ];

CREATE DOMAIN  tipo_doc AS VARCHAR(3)
    DEFAULT 'DNI' CHECK (value IN ('DNI', 'LE', 'LC', 'CI') );

tdoc  tipo_doc  NOT NULL,
```

Luego se puede utilizar el dominio en todas las tablas que se requiera

Restricciones en la creación de tablas

- ✓**NOT NULL: establece que la columna no puede tener un valor nulo.**
- ✓**UNIQUE: evita valores repetidos en una o más columnas. Determina claves alternativas**
- ✓**DEFAULT: establece un valor por defecto para la columna (si no se le asigna ninguno).**
- ✓**CHECK: comprueba que se satisfaga una condición determinada.**
- ✓**PRIMARY KEY: establece el conjunto de columnas que forman la clave primaria de la tabla.**
- ✓**FOREIGN KEY: indica que el contenido de esa columna estará contenido en una columna de otra tabla (o de la misma tabla).**

Todas las restricciones pueden estar precedidas por un nombre mediante la claúsula CONSTRAINT nombre_restriccion

Integridad de Dominios en SQL

```
create domain Tipo_nombre as char(20);
create table estudiante
   (NroMatr number(8) primary key,
   Nombre tipo_nombre,
   carrera char(30) check (carrera in ('IngS','TUPAR', 'PF', 'PM')),
   IdTutor number(4),
   AnioCursa number(1) not null,    etc..... );

   create table Docente
   (NroLeg number(4) primary key,
   NombreD Tipo_nombre,
   titulo char(4) check (titulo in  ('MSc','Ing','Lic','Prof')),
   NroOficina char(6),
   Categ number(4),
   Supervisor number(4),        etc.... );
```


Restricciones de Nulidad

Algunas columnas requieren un valor válido, no se permiten nulos.

El standard permite especificar como NOT NULL tales columnas en las sentencias CREATE TABLE y ALTER TABLE.

```
CREATE TABLE Categoria
( IdCategoria     INTEGER NOT NULL,
  Descripcion     VARCHAR(50) );

ALTER TABLE PRODUCTO
  ADD COLUMN  Descrip_P  VARCHAR(50) NOT NULL ;
```

Integridad de Entidades

La clave primaria de una tabla debe contener un valor único (uno o más atributos), no nulo para cada fila.

El standard provee la cláusula PRIMARY KEY en las sentencias CREATE TABLE y ALTER TABLE.

También es posible asegurar la unicidad de *claves alternativas* en la tabla mediante la cláusula UNIQUE.

```
CREATE TABLE Cliente
 ( Cuit  CHAR(11)  NOT NULL,
id_cliente  INTEGER NOT NULL,
    . . .
    PRIMARY KEY (Cuit),
    UNIQUE (id_cliente)  );
```

```
ALTER TABLE Presentacion ADD
PRIMARY KEY (Codigo_P,
Id_presentacion);
```


Restricciones de Tabla

Condición sobre uno o más atributos de una tabla, aunque pueden hacer referencia a otras tablas.

```
[ CONSTRAINT  nom_restriccion ]  CHECK ( expresión condicional)
```

```
CREATE TABLE Presentacion
( ...
  CONSTRAINT
precio_positivo
CHECK ( precio > 0 ) );
```

El precio de un artículo debe ser positivo

```
  ALTER TABLE Presentacion
  ADD CONSTRAINT  chequeo_limites
        CHECK ( lim_inferior < lim_superior ) );
```

Se debe chequear que el límite inferior de cada presentación sea menor que el superior

Restricciones de Tabla

Las condiciones de venta de Cond_Pedido deben estar definidas en la tabla Cond.Vta.

```
ALTER TABLE Cond_Pedido
ADD CONSTRAINT  ch_cond_vta
        CHECK ( IdCond  IN  ( SELECT IdCond
                              FROM COND_VTA ) );
```

Un producto no puede tener más de 3 presentaciones de precio superior a $100

```
CREATE TABLE Presentacion
  ( . . .
   CONSTRAINT cant_present
        CHECK ( NOT EXISTS ( SELECT * FROM Presentacion
                             WHERE ( precio >100 )
                             GROUP BY Codigo_P
                             HAVING COUNT (*) > 3)) );
```

Restricciones de Integridad Referencial (RIRs)

Una RIR se establece entre dos relaciones (padre e hija) y permite mantener la consistencia entre las tuplas de ambas.

Si una clave extranjera en la tabla hija contiene un valor, éste debe referir a un valor existente en la tabla padre.

El standard provee la cláusula FOREIGN KEY en las sentencias CREATE TABLE y ALTER TABLE.

```
[ FOREIGN KEY ( lista_columnas )
      REFERENCES  nom_tabla [ (lista_columnas) ]
      [ MATCH { PARTIAL | SIMPLE |   FULL } ]
      [ ON UPDATE  accion_referencial ]
      [ ON DELETE  accion_referencial ]
```

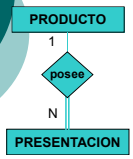
Acciones referenciales: CASCADE, RESTRICT, SET NULL, SET DEFAULT, NO ACTION

Integridad Referencial en SQL

Propagacion vs rechazo

- Restrict – No Action → restringen las actualizaciones. Son posibles cuando no hay referencias
- Cascade – Set Null – Set Default → propagan las actualizaciones a sus referencias

Restricciones de Integridad Referencial (RIRs)




```
CREATE TABLE Producto
   ( Código_P   INTEGER,
     ……..
     PRIMARY KEY (Codigo_P) );

CREATE TABLE Presentacion
   ( Código_P   INTEGER,
     Id_presentación  INTEGER,
     lim_inferior   INTEGER,
     lim_superior   INTEGER,
     ……..
   PRIMARY KEY (Codigo_P, Id_presentacion)
   FOREIGN KEY (Codigo_P) REFERENCES Producto
     ON UPDATE CASCADE
     ON DELETE RESTRICT );
```

Restricciones de Integridad Referencial (RIRs)


```
CREATE TABLE  escribe
 ( id_autor  number(3) NOT NULL,
   id_libro number (13) NOT NULL,
   CONSTRAINT PK_clave_escribe PRIMARY KEY ( autor, libro ),
   CONSTRAINT FK1_autor  FOREIGN KEY (id_autor)
      REFERENCES  autor (ID_aut) ON UPDATE  CASCADE,
   CONSTRAINT FK2_libro FOREIGN KEY (id_libro)
      REFERENCES  libro (ISBN) ON UPDATE  CASCADE);
```


Integridad Referencial en SQL

```
create table Tutoria
   (AnioCurso number(8),
    IdTutor number(4) constraint FK_tut
        references Docente(NroLeg)
        on delete set null on update cascade),
    constraint PK primary key (AnioCurso));

create table Docente
   (NroLeg number(4) primary key,
   ……..
   Supervisor number(4),   not null default '22'
   constraint FK_Sup
        foreign key (Supervisor)
        references Docente(NroLeg)
        on delete set default on update cascade);
```


Tipos de matching (sincronía) en las RIRs

- ***Simple o débil***, una RIR se satisface si cualquier componente de la FK es nulo o si hay una fila en la tabla padre que hace matching.
- ***Parcial,*** las componentes de la FK deben ser todos nulos o debe haber al menos una fila en la tabla padre que satisfaría la RIR si los otros valores nulos fueran correctamente sustituidos.
- ***Full***, para este caso los valores de las columnas de la FK deben ser todos nulos o ser valores válidos de una clave en la tabla referenciada.

INTRODUCCIÓN A LAS BASES DE DATOS

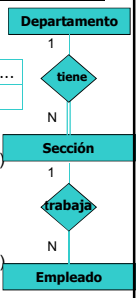
Ejemplos de tipo de matching

sección

IdDpto	IdSec	nombre	…
d1	sec1		
d1	sec2		
d2	sec1		

empleado

IdEmp	IdDpto	IdSec	…



INSERT INTO empleado VALUES ('Emp1', NULL, NULL, ...)
los 3 tipos de matching

INSERT INTO empleado VALUES ('Emp2', 'd2', NULL, ...)
SIMPLE, PARCIAL

INSERT INTO empleado VALUES ('Emp3', NULL, 'sec3', ...)
SIMPLE

INSERT INTO empleado VALUES ('Emp4', 'd2', 'sec2', ...)
ninguno de los 3 tipos de matching

INTRODUCCIÓN A LAS BASES DE DATOS

Restricciones Generales

```
CREATE ASSERTION nom_asser CHECK
   (<predicado>)

CREATE ASSERTION salario_subordinado_valido
  CHECK ( NOT EXIST ( SELECT * FROM Empleado E,
    EMPLEADO M, DEPARTAMENTO D
            WHERE E.SUELDO > M.SUELDO AND
                        E.DTO_NUM= D. DTO_NUM
  AND                       D.JEFE=M.ID_EMPL)
  );
```

El sueldo de los empleados de un departamento no puede ser mayor al sueldo del gerente de ese departamento.

INTRODUCCIÓN A LAS BASES DE DATOS

Restricciones en Oracle

- No implementa DEFINE DOMAIN, tiene CREATE TYPE.
- Acciones referenciales para DELETE: RESTRICT (por defecto, NO debe declararse), SET NULL, CASCADE
- Acciones referenciales para UPDATE: RESTRICT, CASCADE, SET NULL O SET DEFAULT. Por defecto la acción referencial es RESTRICT y NO debe declararse.
- No admite subconsultas en los CONSTRAINTS (check de tabla).
- No implementa ASSERTIONS.

REPASO SQL estandar

```
CREATE TABLE  nom_tabla
{ ( nom_columna  tipo_dato  [ NOT NULL ] [ UNIQUE ]
     [ DEFAULT valor_defecto ] [ check (condicion) ] [ ,.... ] }
   [ PRIMARY KEY ( lista_columnas ) ]
   [ UNIQUE ( lista_columnas ) ] [ , ... ]
   [ FOREIGN KEY ( lista_columnas )
      REFERENCES  nom_tabla [ (lista_columnas) ]
      [ MATCH { PARTIAL | FULL } ]
      [ ON UPDATE  accion_referencial ]
      [ ON DELETE  accion_referencial ] [ ,.....] }
   { [ CHECK (condicion) ] [ , ..... ] } );
```


Repaso restricciones en SQL estandar

```
ALTER TABLE  nom_tabla
[ ADD [COLUMN] nom_columna  tipo_dato [ NOT NULL ]
[UNIQUE][ DEFAULT valor_defecto ] [ check (condicion) ] ]
[ DROP [ COLUMN ] nom_columna [ RESTRICT | CASCADE ] ]
[ ADD CONSTRAINT nom_restric  def_restric_columna ]
[ DROP CONSTRAINT nom_restric [ RESTRICT | CASCADE ] ]
[ ALTER [COLUMN] SET DEFAULT valor_defecto ]
[ ALTER [COLUMN] DROP DEFAULT ];
```

donde def_restric_columna es una de las siguientes claúsulas:

PRIMARY KEY, UNIQUE, FOREIGN KEY, CHECK

Control de Consistencia en Transacciones

- Una unidad atómica de procesamiento (todo o nada) que ejecuta accesos o actualizaciones en la BD, tomando como entrada un estado consistente (y correcto!!) y devolviendo otro estado consistente (y correcto!!!)
- Una colección de acciones que hacen transformaciones consistentes de estados preservando la consistencia global
- Unidad indivisible de procesamiento

Restricciones manejadas mediante procedimientos

- Problemas:
  - Carga de trabajo para el programador
  - Restricciones cambiantes
  - Administración de consistencia descentralizada (GRAVE!!!)
  - Registro de transformaciones descentralizado
- Oracle (y otros DBMS) → proveen lenguaje genérico o propietario (ej. C o PL/SQL) para escribir transacciones
- Se salvan en el Data Dictionary (catálogo de la BD) como
  - Funciones
  - Procedimientos
  - Paquetes

SQL Procedural

Posibilita el uso de código procedural conjuntamente con sentencias SQL que son almacenadas dentro de la BD. El código procedural es ejecutado por el DBMS cuando es invocado (directa o indirectamente) por el usuario de la BD.

Ventajas:

Aísla partes comunes existentes en las aplicaciones, delegándolas en el DBMS.

Desventajas:

Cada proveedor de BD tiene su propio lenguaje procedural.
El SQL-1999 incorporó estas características,
Poco de lo definido anteriormente se ajustaba a un estándar.

Extensión en SQL-1999

SQL-1999 ha extendido el SQL-1992 en varios aspectos, uno de ellos el procedural.

Extensiones Procedimentales:

Sentencias compuestas (agruparlas en bloques Begin - End)

Declaración de variables y constantes.

Sentencias de Flujo de control:

- ✓**If-then-else (elsif)**
- ✓**While**
- ✓**For (itera sobre los elementos de una tabla resultado)**
- ✓**Loop y repeat**

Utilidad SQL Procedural

Se puede utilizar SQL Procedural para definir:

✓Trigger: Es un procedimiento que es invocado automáticamente por el DBMS en respuesta a un evento especifico de la BD.

✓Stored Procedure: Es un procedimiento que es invocado explícitamente por el usuario.

✓Función: definida por el usuario para realizar operaciones específicas sobre los datos, y pueden ser invocadas desde un trigger, stored procedure o explícitamente.

Triggers

- Acciones centralizadas definidas en un lenguaje de programación (genérico o propietario).
- Es un procedimiento almacenado que es disparado (ejecutado implícitamente) cuando ocurre un INSERT, UPDATE, o DELETE sobre la tabla a la cual el trigger está asociado.
- Pueden usarse para:
  - Auditoría basada en valores
  - Generación automática de datos
  - Forzado de restricciones complejas
  - Etc.

Estructura de un Trigger

Tiene tres partes básicas:

- Evento
  - sentencia SQL que provoca el disparo
- Condición
  - especifica una expresión Booleana que debe ser TRUE (verdadera) para que el trigger se dispare (ejecute). No se ejecuta si la condición evalúa en FALSE (falso) o UNKNOWN (desconocido).
- Acción
  - procedimiento (sentencias en lenguaje de progr.) que contiene el código a ser ejecutado.

Triggers – componentes

Los componentes de un trigger son:

- ✓ Un nombre único que identifica al trigger dentro de la base
- ✓ Un evento de disparo asociado (INSERT, UPDATE o DELETE).
- ✓ Un tiempo de activación que puede ser BEFORE o AFTER de la ejecución del evento
- ✓ En Oracle, granularidad que puede ser FOR EACH ROW o FOR EACH STATEMENT (opción por defecto)
- ✓ Una condición que puede ser cualquier condición SQL válida
- ✓ Una acción que puede ser un conjunto de sentencias SQL procedurales
- ✓ Tuplas temporarias nombradas new y old → REFERENCING NEW AS … y REFERENCING OLD AS …

Triggers – características principales

- ✓La acción puede ejecutarse antes (BEFORE) del evento disparador, después de él (AFTER) o en vez de él (INSTEAD OF, leer documentación del DBMS).
- ✓La acción puede referirse a valores anteriores y nuevos que se insertaron, eliminaron o actualizaron en el evento que desencadenó la acción.
- ✓Los eventos de actualización pueden especificar un atributo particular o un conjunto de atributos (Depende del DBMS).

Triggers – características principales

- ✓Una condición puede especificarse con una cláusula WHEN, y la acción se ejecutará sólo si se dispara la regla y si se cumple la condición cuando ocurre el evento disparador.
- ✓El programador tiene la opción de especificar que la acción se realice (dependiendo del DBMS):
  - una vez para cada tupla modificada.
  - una vez para todas las tuplas que cambian en una operación de la base de datos.

Triggers – sintaxis:

En SQL-99 la sintaxis de un trigger es la siguiente:

```
CREATE [O REPLACE] TRIGGER <nombre del trigger>

BEFORE | AFTER <evento> ON <nombre-tabla>          } evento
[ REFERENCING  OLD | NEW  AS <nombre-ref>]
[ FOR EACH { ROW | STATEMENT} ]
[ WHEN < condición > ]                             } condición

BEGIN
    cuerpo del trigger                             } acción
END;
```


Triggers (Evento-Condición-Acción)

Evento: INSERT, DELETE O UPDATE [OF <lista-atributos>]

Ej: AFTER DELETE ON nombre_tabla
AFTER UPDATE OF nombre_columna ON nombre_tabla

Condición: puede ser cualquier condición SQL válida. Sólo para disparadores a nivel de fila

Operadores relacionales: <, <=, >, >=, =, <>
Operadores lógicos: AND, OR, NOT

OLD, NEW (Si están el en cuerpo del trigger se referencian como :OLD ó :NEW)

Acción: pueden ser un conjunto de sentencias SQL procedurales.

Para renombrar las tuplas temporarias usar REFERENCING NEW AS y REFERENCING OLD AS.

Triggers - comportamiento

- ✓ **Se disparan automáticamente al producirse el evento**
- ✓ **La acción del trigger es un procedimiento atómico (Si cualquier sentencia del cuerpo del trigger falla, la acción completa del trigger se deshace, incluyendo la sentencia que lo disparó).**
- ✓ **No se pueden incluir sentencias del DDL, ni COMMIT ni ROLLBACK.**
- ✓ **Un trigger BEFORE no debe contener sentencias SQL que alteren datos (INSERT, UPDATE, DELETE)**
- ✓ **Varios triggers pueden activarse ante un mismo evento.**
  - ✓ **Hay bases en donde se puede especificar el orden.**
- ✓ **Si la activación de un trigger T1 dispara otro trigger T2: se suspende la ejecución de T1, se ejecuta el trigger anidado T2 y luego se retoma la ejecución de T1.**

Triggers – for each row – for each statement

DNI	Nombre	Cuota
1	Juan	133
2	Gustavo	222
3	Pedro	452
4	Mariana	164
5	Silvina	223

```
update cliente
   set cuota = cuota * 1.2
where dni > 2;
```

Cantidad de tuplas Afectadas ?

Un trigger for each row, se ejecuta 3 veces, una por cada tupla.

Un trigger for each statement, se ejecuta 1 vez.

Triggers – ejemplo

Ejemplo en Oracle:

```
CREATE OR REPLACE TRIGGER ejemplo
BEFORE INSERT OR UPDATE OR DELETE ON tabla
BEGIN
  IF DELETING THEN
        Acciones asociadas al borrado
  ELSIF INSERTING THEN
        Acciones asociadas a la inserción
  ELSE
        Acciones asociadas a la modificación
  END IF;
END;
/
```


Triggers - ejemplos

```
CREATE TRIGGER Ejemplo_sentencia
AFTER DELETE ON tabla1
REFERENCING OLD AS anterior
BEGIN
  DELETE FROM tabla2 WHERE  tabla2.cod=anterior.cod;
END;
```

```
CREATE TRIGGER Ejemplo-fila
AFTER DELETE ON tabla1
FOR EACH ROW
WHEN ((OLD.nombre='pepe') OR (OLD.edad > 35))
BEGIN
  DELETE FROM tabla2 WHERE tabla2.cod=:OLD.cod;
END;
```

Ejemplo: Trigger en Oracle

Evento

```
CREATE TRIGGER verif_nota_lab
BEFORE INSERT OR UPDATE OF NotaLab ON InscriptoEn
DECLARE valor_nota exception;
```

Condición

```
WHEN (old.NotaLab IS NOT NULL OR new.NotaLab IS NOT
NULL)
```

row trigger

Valores de columna para la tupla actual
Correlación de nombres new/old

Acción

```
FOR EACH ROW
BEGIN
    IF :new.NotaLab < :old.NotaLab
    THEN raise_application_error(-20221,'Nueva nota
      incorrecta');
    END IF;
    );
END;
```


Triggers – ejemplo

Para chequeos de integridad:

RAISE_APPLICATION_ERROR (nro_error, mensaje); [-20000 y -20999]

```
CREATE OR REPLACE TRIGGER ejemplo
BEFORE DELETE ON tabla
FOR EACH ROW
BEGIN
   IF tabla.columna = valor_no_borrable THEN
   RAISE_APPLICATION_ERROR(-20000,'La fila no se puede borrar');
   END IF;
...
END;
```


Triggers – ejemplo

Ejemplo : "chequear que el salario de los empleados se encuentre en el rango correcto"

```
CREATE OR REPLACE TRIGGER chequear_salario
BEFORE INSERT OR UPDATE ON empleado
FOR EACH ROW
WHEN (new.trabajo<>'presidente')
DECLARE
        v_salariomin INTEGER;
        v_salariomax INTEGER;
BEGIN
   SELECT MAX(salario), MIN(salario) FROM empleado
    INTO v_salariomin, v_salariomax
   FROM empleado
   WHERE trabajo=:new.trabajo;
   IF :new.salario < v_salariomin OR :new.salario > v_salariomax THEN
       RAISE_APPLICATION_ERROR(-20001, 'Sueldo fuera de rango');
   END IF;
END;
```

Triggers – ejemplo

Ejemplo : "registrar las modificaciones de los salarios"

```
CREATE TRIGGER Salvar_cambio_Salario
AFTER UPDATE ON EMPLOYEE
FOR EACH ROW
BEGIN
  IF (:old.salario <> :new.salario) THEN
    INSERT INTO salario_historico
        (NroEmp, FechaCambio, UsuarioId, SalarioViejo,
PorcentCambio)
      VALUES
        (:old.NroEmp, 'NOW', user, :old.salario, (:new.salario -
:old.salario) *100 / :old.salario);
END
```


Row and Statement Triggers/ Before and After

- Pueden crearse 3 de cada tipo, uno por cada evento DELETE, INSERT y UPDATE → 12 triggers.
- Pueden crearse triggers que se disparen con mas de un comando

	For Each Sentence	For Each Row
BEFORE	*BEFORE statement trigger*: Oracle dispara el trigger una vez **antes** de ejecutar **la sentencia** disparadora	*BEFORE row trigger*: Oracle dispara el trigger una vez **antes** de modificar **cada fila afectada** por la sentencia disparadora
AFTER	*AFTER statement trigger*: Oracle dispara el trigger una vez **después** de ejecutar **la sentencia** disparadora	*AFTER row trigger*: Oracle dispara el trigger una vez **después** de modificar **cada fila afectada** por la sentencia disparadora

Triggers vs. Restricciones declarativas

Triggers permiten definir y forzar restricciones de integridad, pero no son UNA RESTRICCIÓN !!.

- No chequean los datos ya cargados en la BD.
- Se recomienda usar sólo cuando:
  - una restricción referencial no puede forzarse usando: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, update CASCADE, update and delete SET NULL, update and delete SET DEFAULT
  - las tablas que deben inspeccionarse están en distintos nodos en una base distribuida.
  - no se puede plantear en forma declarativa de ninguna manera (eficiente??)

Stored Procedures (procedimientos almacenados) en Oracle

```
CREATE [O REPLACE] PROCEDURE ProcedureName
    [<parámetros de entrada>]
 AS
    [ <declaración de variables locales>]
 BEGIN
   <sentencias del procedimiento (cuerpo)>
 END;
```


Stored Procedures en Oracle

```
CREATE PROCEDURE Orden_de_Envío ( NroPed CHAR(8) )
AS
   EstadoOrd CHAR(7);
   NroClte INTEGER;
   Pago NUMERIC(15,2);
BEGIN
   ….
END
```


Stored Procedures en Oracle

Ejemplo:

```
CREATE PROCEDURE Ajuste_Rango_Salario (Factor
NUMERIC(18,2)  )
AS
BEGIN
  UPDATE Tarea SET MIN_Salario = MIN_Salario * Factor,
        MAX_Salario = MAX_Salario * Factor;
END;
```

Para ejecutarlo:

```
exec Ajuste_Rango_Salario(1.1);
```