Departamento de Lenguajes y Ciencias de la Computación Universidad de Málaga

Bases de Datos

(Ingeniería Técnica en Informática de Sistemas)

Tema 2. Bases de Datos Relacionales: SQL

E.T.S.I. Informática

J. Galindo Gómez y E. Soler Castillo

SQL (Structured Query Language)

- **<u>SQL está en continua evolución:</u>** Es una evolución del lenguaje SEQUEL de D.D. Chamberlin y R.F. Boyce (1974) y fue implementado por primera vez por IBM en su BDR llamado SYSTEM R.
  - **<u>ISO</u>** (*International Standards Organization*) y **<u>ANSI</u>** (*American National Standards Institute*) desarrollaron una versión estándar en 1986, llamada **SQL86** o **SQL1**. Posteriormente, se desarrolló **SQL92** o **SQL2**. Actualmente se desarrolla **SQL3**, que incluye conceptos de BD orientadas a objetos.
- **<u>SQL es un lenguaje estándar para GESTIÓN de BDR</u>:**
  - **Está incluido en muchos SGBD (DBMS),** como DB2 (de IBM), Oracle, Ingres, Informix, Sybase, Access, SQL Server...
    - Las mismas sentencias sirven en distintos SGBD.
    - Si se usan sólo las características estándares facilita la tarea de migrar de SGBD → Hay funciones no estándar en algunos SGBD.
  - **Fácil de usar y aprender:** Tiene similitudes con el Álgebra Relacional, aunque se parece más al Cálculo y es más fácil e intuitivo que ambos lenguajes formales.
  - Aquí se incluye una introducción a SQL estándar con algunos comentarios sobre el SGBD Oracle.

SQL (Structured Query Language)

- **SQL es un lenguaje COMPLETO:** Incluye sentencias para
  - **DDL y DML:** Permite definir esquemas, consultar, borrar, actualizar...
  - **Definición de vistas:** Para ver la BD de distintas formas.
  - **Seguridad:** Permisos de acceso distintos para cada usuario.
  - **Definir restricciones de integridad:** Integridad referencial...
  - **Especificar control de transacciones:** Para grandes empresas, recuperación de errores, archivos históricos...
  - **Puede incrustarse en lenguajes de alto nivel** (C, C++, Pascal, COBOL...).
  - Permite **operaciones tan complejas** que su total definición es complicada: Sólo veremos un subconjunto de las operaciones posibles.
- **ESQUEMA** (*schema*)**:** Conjunto de elementos (tablas, vistas, permisos...) que pertenecen a la misma BD. Cada esquema tiene un nombre y un usuario propietario del esquema**.**
- **CATÁLOGO** (*catalog*)**:** Conjunto de esquemas. Tiene un esquema especial llamado `INFORMATION_SCHEMA` que provee información sobre los demás esquemas, usuarios autorizados, definiciones de dominio, restricciones de integridad referencial (sólo entre tablas del mismo catálogo)...

SQL (Structured Query Language)

- **TIPOS de DATOS** de los atributos de las relaciones**:**
  - **Enteros** de distintos tamaños**:** **`INTEGER`** o **`INT`** y **`SMALLINT`**.
  - **Reales** de distinta precisión**:** **`FLOAT(`**p**`)`**, **`REAL`**, **`DOUBLE PRECISION`**, o el más genérico **`DECIMAL(`**precisión, escala**`)`**

`En Oracle`:

`NUMBER(p,s), donde p="número total de dígitos" (de 1 a 38 en Oracle) y s="número de decimales" (de -84 a 127 en Oracle, con valor 0 por defecto).`

`NUMBER(p,0)` **`ENTERO`**

`NUMBER` **`COMA FLOTANTE`**

Actual Data	Specified As	Stored As
7456123.89	NUMBER	7456123.89
7456123.89	NUMBER(9)	7456124
7456123.89	NUMBER(9,2)	7456123.89
7456123.89	NUMBER(9,1)	7456123.9
7456123.89	NUMBER(6)	exceeds precision
7456123.89	NUMBER(7,-2)	7456100
7456123.89	NUMBER(-7,2)	exceeds precision

SQL (Structured Query Language)

- **TIPOS de DATOS** de los atributos de las relaciones**:**
  - **Caracteres: CHAR(**n**)** o **CHARACTER(**n**)**, n=longitud fija (por defecto n=1). También **VARCHAR(**n**)**, n=longitud máxima

En Oracle, se recomienda:

VARCHAR2(n) con n=4000 como máximo.
Para cadenas más largas usar LONG (máximo 2GB) o CLOB (Character Large OBject, máx. 4GB).
NCHAR y NVARCHAR2 usan el juego de caracteres Nacional definido al crear la BD.
Diferencia entre CHAR y VARCHAR

  - **Cadenas de bits** (para gráficos, sonidos, ficheros binarios...)**: BIT(**n**)**, n=longitud fija o **BIT VARYING(**n**)**, con n=longitud máxima. Por defecto n=1.

En Oracle, se recomienda:

RAW(tamaño_fijo_máx_2000bytes**)** o **LONG RAW** (sin argumento y con un tamaño máximo de 2GB). Últimamente Oracle aconseja usar **LOB** o **BLOB** (*Binary Large OBject*, máximo 4GB), o también **BFILE** para almacenar la localización de un fichero binario.

SQL (Structured Query Language)

- **TIPOS de DATOS** de los atributos de las relaciones**:**
  - **Fecha y Hora: DATE** (año, mes, día: YYYY-MM-DD). **TIME** (horas, minutos, segundos: HH:MM:SS). **TIMESTAMP** incluye ambos

En Oracle,

DATE incluye fecha y hora

- Se usan las funciones **TO_CHAR** y **TO_DATE** para convertir un dato de tipo fecha a texto y viceversa.

  TO_DATE('27-OCT-98', 'DD-MON-YY')

  to_char(to_date('27-10-03','dd-mm-yy'),'dd-mon-yyyy')

  27-oct-2003

SQL (Structured Query Language)

- Aritmética del tipo DATE:

Se puede sumar y restar constantes numéricas así como otras fechas a las fechas. Oracle interpreta las constantes como número de días.

Ejemplos: SYSDATE + 1 mañana
SYSDATE - 7 hace una semana
SYSDATE + (10/1440) dentro de 10 minutos

Restar en atributo FechaAlquiler de SYSDATE devuelve el número de días desde que fue alquilado.

Existe muchas otras funciones como ADD_MONTHS, MONTHS_BETWEEN, NEXT_DAY, ROUND, etc.

DML de SQL: Consultas con SELECT

-
```
SELECT <Lista_Atributos>
    FROM <Lista_Tablas>
    WHERE <Condición>;
```

Esquema Conceptual:
Suministrador (S#, NombreS, Dirección, Ciudad);
Pieza (P#, NombreP, Peso, Cantidad);
Suministros (S#,P#);

Suministrador

S#	NombreS	Dirección	Ciudad
2	Juan	C/ Pelayo	Málaga
3	Luis	C/ Pato	Málaga
4	Pablo	C/Alfonso X	Granada

Pieza

P#	NombreP	Peso	Cantidad
2	Tornillo	16	20
4	Tuerca	8	20
8	Clavos	7	30

Suministros

S#	P#
2	4
2	8
4	2
3	2

DML de SQL: Consultas con SELECT

- **Ejemplos:**
  - Proyección:
    - "Nombre y Dirección de todos los suministradores"

```
SELECT NombreS, Direccion
FROM Suministrador;
```

Ojo al ;

  - Selección:
    - "Todos los datos de los suministradores de Málaga"

```
SELECT * FROM Suministrador
WHERE NombreS= 'Malaga';
```

CASE SENSITIVE

    - "Piezas que pesen más de 15 gramos y que su Cantidad sea menor o igual que 30":

```
SELECT * FROM Pieza
WHERE Peso>15 AND Cantidad<=30;
```

    - "Piezas de las que se ignore el Peso":

```
SELECT * FROM Pieza
WHERE Peso IS NULL;
```

No usar Peso=NULL. Cada NULL es distinto a otro.

Consultas con SELECT: Observaciones

- **Cláusula SELECT:**
  - Puede ponerse un ***** para indicar que se recuperen **todos los atributos** de todas las tablas de la cláusula FROM.
  - Puede usarse también el formato **<Tabla>.*** para referirse a **todos** los atributos de esa tabla.
  - Pueden **renombrarse** los nombres de las columnas del resultado, poniendo el nuevo nombre justo después del atributo (usar comillas dobles si son varias palabras).

```
SELECT nombrep "Nombre de la pieza", peso/1000 Kg
FROM Pieza;
```

Nombre de la pieza	Kg
Tornillo	0.016
Tuerca	0.008
Clavos	0.007

Consultas con SELECT*: Observaciones*

- **Cláusula FROM:** Para especificar tablas, vistas o instantáneas (*snapshot*).
  - Pueden ir varias tablas (vistas o instantáneas)
  - Pueden ponerse alias a las tablas: **<Tabla> <Alias>**.
  - Pueden usarse los alias en el SELECT (delante) y en el WHERE (detrás)

```
SELECT p.nombrep, p.peso FROM pieza p
WHERE p.cantidad>= 20;
```

- **Cláusula WHERE:**
  - Se usan los operadores relacionales <,>,<=,>=,<>,!=,=,LIKE
    - Comparación de Fechas, Caracteres, Números
    - Caracteres: UPPER, LOWER, LIKE '_ose%'
  - La condición puede usar los op. lógicos **NOT**, **AND** y **OR**.
  - **Si no existe cláusula WHERE:** Se supone que se recuperan todas las tuplas (como si la condición fuese siempre verdad).
    - Si no existe cláusula **WHERE** y en la cláusula **FROM** hay varias tablas, se obtiene el **Producto Cartesiano:** Por tanto, si hay varias tablas la condición establece una selección de tuplas del Producto Cartesiano.
    - Las condiciones de una **operación de REUNIÓN** hay que explicitarlas.

Consultas con SELECT*: Observaciones*

- **Null en Funciones SQL**

  Todas las funciones escalares (excepto NVL y TRANSLATE) devuelven null cuando se les pasa un null

  argumento. Se puede usar la función NVL para devolver un valor cuando hay un null.

  NVL(COMM,0) devuelve 0 si COMM es null o el valor de COMM si no es null.

  La mayoría de las funciones de agregación ignoran nulls.

  Media de 1000, null, null, null, y 2000 → (1000+2000)/2 = 1500.
- **Nulls con Operadores de Comparación**

  Usar IS NULL y IS NOT NULL.

  Al usar otros operadores → UNKNOWN

  Oracle considera 2 nulls iguales si aparecen en claves compuestas.

Ejemplos:

– "Números de Suministradores que Suministren una Pieza de 16 gramos":

```
SELECT S# FROM Pieza, Suministros;
```

Pieza

P#	NombreP	Peso	Cantidad
2	Tornillo	16	20
4	Tuerca	8	20
8	Clavos	7	30

Suministros

S#	P#
2	4
2	8
4	2
3	2

P#	NombreP	Peso	Cantidad	S#	P#
2	Tornillo	16	20	2	4
2	Tornillo	16	20	2	8
2	Tornillo	16	20	4	2
2	Tornillo	16	20	3	2
4	Tuerca	8	20	2	4
4	Tuerca	8	20	2	8
4	Tuerca	8	20	4	2
4	Tuerca	8	20	3	2
8	Clavos	7	30	2	4
8	Clavos	7	30	2	8
8	Clavos	7	30	4	2
8	Clavos	7	30	3	2

```
SELECT S# FROM Pieza,
       Suministros
   WHERE Peso = 16;
```


Ejemplos:

– "Números de Suministradores que Suministren una Pieza de 16 gramos":

```
SELECT S# FROM Pieza, Suministros
WHERE Pieza.P# = Suministros.P#;
```

P#	NombreP	Peso	Cantidad	S#	P#
2	Tornillo	16	20	4	2
2	Tornillo	16	20	3	2
4	Tuerca	8	20	2	4
8	Clavos	7	30	2	8

```
SELECT S# FROM Pieza, Suministros
   WHERE Pieza.P# = Suministros.P# AND Peso=16;
```

S#
4
3

DML de SQL: Consultas con SELECT

- **Ejemplos:**
  - "Nombres de Suministradores que Suministren una Pieza de 16 gramos":

```
SELECT NombreS FROM Suministrador, Pieza, Suministros
       WHERE Suministros.P# = Pieza.P#
  AND Suministros.S# = Suministrador.S# AND Peso=16;
```

Suministrador

S#	NombreS	Dirección	Ciudad
2	Juan	C/ Pelayo	Málaga
3	Luis	C/ Pato	Málaga
4	Pablo	C/Alfonso X	Granada

Suministros

S#	P#
2	4
2	8
4	2
3	2

Pieza

P#	NombreP	Peso	Cantidad
2	Tornillo	16	20
4	Tuerca	8	20
8	Clavos	7	30

  - "Nombres de Piezas suministradas desde Málaga": Usando **alias** de tablas.

```
SELECT NombreP FROM Suministrador S, Pieza P,
  Suministros SP
WHERE SP.P# = P.P# AND SP.S# = S.S# AND Ciudad='Málaga';
```


Consultas con SELECT: Ejemplos

- **Ejemplos:** **Empleado** (NIF, Nombre, Salario, Dpto, NIFSupervisor);
  - "Nombres de los empleados supervisores y sus empleados supervisados":

```
SELECT E1.Nombre Supervisor, E2.Nombre Supervisado
FROM   Empleado E1, Empleado E2
WHERE  E1.NIF = E2.NIFSupervisor;
```

  - "Nombres de Pares de Empleados que cobren lo mismo"

```
SELECT E1.Nombre, E2.Nombre
FROM   Empleado E1, Empleado E2
WHERE  E1.Salario = E2.Salario AND E1.NIF < E2.NIF;
```

  - "Datos de Suministradores de Piezas de 5 gramos o cuya cantidad sea menor a 9":

```
SELECT S.*  FROM Suministrador S, Pieza P, Suministros SP
  WHERE SP.P#=P.P# AND SP.S#=S.S# AND(Peso=5 OR Cantidad<9);
```

  - "Mostrar todas las posibles combinaciones de pares (Suministrador, Pieza), mostrando el número y el nombre para cada Suministrador y cada Pieza":

```
SELECT S#, NombreS, P#, NombreP  FROM Suministrador, Pieza;
```

  - "Números de Suministrador y Pesos (si se conocen) de todas las piezas que suministren":

```
SELECT DISTINCT S#, Peso  FROM Suministros SP, Pieza P
  WHERE SP.P#=P.P# AND Peso IS NOT NULL;
```

    Si hay varias piezas con el mismo peso suministradas por el mismo suministrador, se eliminan, para que no aparezcan dos veces.

Consultas con SELECT*: Multiconjuntos*

- **Tuplas Repetidas:** Las tablas en SQL no son conjuntos de tuplas sino **Multiconjuntos**, pues admiten duplicados. En las consultas:
  - Eliminar duplicados es una tarea costosa (ordenar primero es mejor).
  - Los duplicados pueden interesar al usuario.
  - En funciones de agregación no se desean eliminar duplicados.
  - Una tabla con `PRIMARY KEY/UNIQUE` no admite duplicados
- **Operaciones de Conjuntos:** Pueden usarse entre varias subconsultas, actuando sobre los conjuntos de tuplas de cada una:
  - **`UNION`: Unión SIN duplicados** de las tuplas de las consultas.
  - **`UNION ALL`: Unión CON duplicados**.
  - **`INTERSECT`: Intersección** de conjuntos
  - **`MINUS` (o `EXCEPT`): Diferencia** (resta) de conjuntos.
  - **Ejemplo:** "Números de Suministradores que NO suministren la Pieza 8":

```
SELECT S# FROM Suministrador
MINUS  SELECT S# FROM Suministros WHERE P<>8;
```


Consultas con SELECT*: Operaciones*

- **Comparación de cadenas:**
  Puede usarse el comparador **`[NOT] LIKE`** con los caracteres **"%"** (para cualquier número de caracteres) y **"_"** (para un único carácter). "_" y "\%" son sendos símbolos. El operador **"||"** concatena cadenas.

  - "Seleccionar Empleados que tengan un apellido López":

```
SELECT * FROM Empleado WHERE Nombre LIKE '%López%';
```

  - "Seleccionar Valores que hayan bajado un porcentaje acabado en 5":

```
SELECT * FROM Valores WHERE Variacion LIKE '-_5\%';
```

- Ejemplo de concatenación:

```
SELECT 'el nombre es ' || nombre FROM empleados;
'ELNOMBREES'||NOMBRE
---------------------------------
el nombre es TORRES, HORACIO
el nombre es VAZQUEZ, HONORIA
el nombre es CAMPOS, ROMULO
el nombre es PEREZ, JULIO
```

Consultas con SELECT*: Operaciones*

- **Operaciones Aritméticas** (+,-,*,/):
  - "Productos con su porcentaje de descuento posible aplicado si el descuento aplicado no supera los 5 Euros": Esquema **Producto**(Nombre, Precio, Descuento...).

```
SELECT Precio, Descuento "Porcentaje Descuento",
       Precio-(Precio*Descuento/100) "Precio Final"
FROM Producto WHERE (Precio*Descuento)/100 < 5;
```

- **Ordenar Resultados**:

  Cláusula **ORDER BY** seguida de una lista de atributos o de posiciones en la lista de atributos del **SELECT**.
  - Tras cada atributo puede ponerse:
    - **ASC** (ordenación ascendente, por defecto).
    - **DESC** (ordenación descendente).
  - "Seleccionar Empleados y sus sueldos aumentados un 10% si su sueldo inicial está entre 1 y 3 millones de euros, ordenando descendentemente según su sueldo incrementado"**:**

```
SELECT Nombre, 1.1*Sueldo FROM Empleado
WHERE Salario BETWEEN 1 AND 3
        ORDER BY 2 DESC;
```


SELECT*: Pseudocolumnas en Oracle*

- **Pseudocolumnas**:

  Son valores que se comportan como si fueran columnas, pero no tienen valores almacenados en la BD. No se pueden insertar, actualizar o borrar sus valores.
  - **ROWID:** Es la dirección de una fila concreta en una tabla concreta. Permite averiguar cómo se almacenan las filas de una tabla. Además, es la forma más rápida de acceder a una fila particular.
    - No debe usarse como llave primaria, pues el SGBD puede reorganizarlas (al borrar o insertar una fila, por ejemplo).
    - Ejemplo: `SELECT ROWID,NIF FROM Empleado WHERE Dpto=20;`
  - **ROWNUM:** Es el número de orden de cada fila en una consulta particular.
    - Se asigna en el orden en el que las tuplas van siendo seleccionadas.
    - Ejemplo: "Selecciona los empleados con mayor salario"
      `SELECT ROWNUM, e.* FROM Empleado e ORDER BY Salario DESC;`
    - Consulta que no devuelve nada: `SELECT * FROM Empleado WHERE ROWNUM>1;`
      - La primera fila recuperada es la 1 y la condición es falsa, por lo que NO se recupera. La segunda tupla es ahora la primera y también hace que la condición sea falsa.
    - Se puede utilizar para actualizar un atributo de cierta tabla, con valores únicos y correlativos.

SELECT*: Funciones en Oracle*

- **Funciones:** Oracle permite utilizar funciones ya definidas, que simplifican ciertas operaciones. Además, también permite crear nuestras propias funciones. Algunos Ejemplos:
  - **Funciones Numéricas:**
    - **ABS** (valor absoluto),
    - **SINH** (seno hiperbólico),
    - **TAN** (tangente),
    - **POWER(base,exp)** (potencia),
    - **LN** (logaritmo neperiano),
    - **ROUND** (redondear números),
    - **SIN** (seno),
    - **COS** (coseno),
    - **SQRT** (raíz cuadrada),
    - **EXP** (exponencial con e = 2.71828183),
    - **LOG(b,n)** (logaritmo en base **b** de **n**),
    - **TRUNC** (truncar números)...
  - **Funciones de Caracteres:**
    - **CHR** (carácter que corresponde a un número),
    - **LOWER** (devuelve la cadena del argumento con todas las letras en minúsculas),
    - **UPPER** (pasa a mayúsculas),
    - **LPAD**/**RPAD** (ajusta a la izquierda/derecha una cadena),
    - **LTRIM**/**RTRIM** (quita ciertos caracteres de la izquierda/derecha de una cadena),
    - **SUBSTR** (extrae una subcadena de una cadena),
    - **REPLACE** (reemplaza subcadenas),
  - **Funciones de Caracteres que devuelven números:**
    - **ASCII** (código ASCII de un carácter),
    - **LENGTH**/**LENGTHB** (longitud de una cadena en caracteres/bytes),
    - **INSTR** (buscar caracteres en una cadena)...

SELECT*: Funciones en Oracle*

- **Funciones:**
  - **Funciones de Fecha:**
    - **ADD_MONTHS** (suma meses a una fecha),
    - **LAST_DAY** (devuelve el último día del mes de una fecha dada),
    - **SYSDATE** (fecha y hora actual del sistema)...
  - **Funciones de Conversión:**
    - **TO_CHAR** (convierte fechas y números a cadenas de caracteres),
    - **TO_DATE** (convierte una cadena a un dato de tipo fecha **DATE**),
    - **TO_LOB** (convierte datos de tipo **LONG**/**LONG RAW** a tipo **LOB**),
    - **TO_NUMBER** (convierte una cadena con un número a un número)...

      Ej.: **SELECT TO_CHAR(SYSDATE, 'MM-DD-YYYY HH24:MI:SS') "Fecha" FROM DUAL;**

      (**DUAL** es una tabla usada cuando se necesita poner algo en el **FROM**)
  - **Otras funciones:**
    - **GREATEST/LEAST** (devuelve el mayor/menor valor de una lista de valores),
    - **NVL** (para cambiar valores **NULL** por otros valores),
    - **USER**/**UID** (devuelven el nombre/número del usuario actual de la sesión)

Subconsultas o Consultas Anidadas

- **Subconsulta o Consulta anidada** (*nested query*)**:** Sentencia **SELECT** incluida dentro de otra **SELECT** (llamada consulta externa, *outer query*).
  - Una **subconsulta** genera un **multiconjunto** de valores: Podemos ver si un valor **pertenece** al multiconjunto (**IN**), si es mayor, igual... que **todos** (**ALL**) o **alguno** (**SOME**, **ANY**) de los valores del multiconjunto:

Las **subconsultas** suelen ser **ineficientes.**

```
• SELECT * FROM Pieza, Suministrador
  WHERE (S#,P#) IN (SELECT * FROM Suministros);
```

Sustituye a una doble Reunión Natural.

```
• SELECT * FROM Pieza
  WHERE P# IN (SELECT P#
               FROM Suministros SP, Suministrador S
               WHERE SP.S#=S.S# AND Ciudad='Málaga')
     OR P# IN (SELECT P# FROM Pieza
               WHERE Peso IS NOT NULL AND Cantidad IN (15,20));
```

☺J☺: No es la mejor manera de efectuar esta consulta.

```
• SELECT Nombre FROM Empleado
  WHERE Salario > ALL (SELECT Salario FROM Empleado
                       WHERE Dpto IN (1,3,5,7)
                          OR Dpto IN (SELECT Dpto FROM Empleado
                                      WHERE NIFSupervisor=999));
• SELECT Nombre FROM Empleado E
  WHERE  Dpto IN (SELECT Dpto FROM Empleado
                  WHERE Salario > E.Salario );
```

SUBCONSULTA correlacionada: Usa atributo de la consulta externa. **Se evalúa una vez por cada tupla de la consulta externa.**

Cuantificador Existencial: EXISTS

- **Función [NOT] EXISTS:** Comprueba si una subconsulta (normalmente correlacionada) es o no **vacía** (si recupera o no alguna tupla).
  - **Ejemplos:**
    - "Empleados tales que **NO existen** compañeros en el mismo Dpto.":

```
SELECT Nombre FROM Empleado E
  WHERE  NOT EXISTS (SELECT * FROM Empleado
        WHERE Dpto = E.Dpto AND NIF<>E.NIF);
```

    - "Empleados tales que **existen** compañeros de Dpto. que cobran más que ellos":

```
SELECT Nombre FROM Empleado E
  WHERE  EXISTS (SELECT * FROM Empleado
                 WHERE Salario > E.Salario AND Dpto = E.Dpto);
```

    - "Piezas de las que haya menos de 100 y **NO exista** actualmente ningún suministrador":

```
SELECT * FROM Pieza P WHERE Cantidad<100 AND NOT EXISTS
  (SELECT * FROM Suministros SP WHERE SP.P#=P.P#);
```

    - "Números de suministradores para los que **NO existen** piezas que **NO** sean suministradas por ellos":

```
SELECT S# FROM Suministros SP1 WHERE NOT EXISTS
  (SELECT * FROM Pieza P WHERE NOT EXISTS
    (SELECT * FROM Suministros SP2
     WHERE SP2.S#=SP1.S# AND SP2.P#=P.P#));
```

Funciones de Grupo o de Agregación

- **Funciones de Grupo o de Agregación:** Son funciones que se aplican sobre un grupo de valores del mismo dominio. Se aplican sobre un grupo de tuplas (o de atributos concretos de esas tuplas).
  - **COUNT:** Cuenta el número de tuplas del grupo (indicadas por *****).
    - "¿Cuántas piezas hay que pesen más de 15 gramos?":
      ```
      SELECT COUNT(*) FROM Pieza WHERE Peso>15;
      ```
    - "¿Cuántos empleados hay en el Departamento de I+D?":
      ```
      SELECT COUNT(*) FROM Empleado E, Dpto D
      WHERE E.Dpto=D.NumDpto AND D.Nombre='I+D';
      ```
    - "¿Cuántos salarios **distintos** hay entre los empleados?":
      ```
      SELECT COUNT(DISTINCT Salario) FROM Empleado;
      ```
    - "¿Cuántas piezas hay de las que se ignore su peso?":
      ```
      SELECT COUNT(*) FROM Pieza WHERE Peso IS NULL;
      ```
  - **Observaciones:**
    - Los **NULL** se ignoran, excepto por **COUNT(*).**
    - **DISTINCT** elimina los valores duplicados (y no cuenta los **NULL**).
    - En el lugar de **DISTINCT** se puede usar **ALL** (opción por defecto).
    - **DISTINCT** y **ALL** pueden usarse en todas las funciones de grupo.

COUNT(Salario) es lo mismo que **COUNT(*)** si no hay Nulls

Funciones de Grupo o de Agregación

- **SUM, MAX, MIN, AVG, STDEV, VARIANCE:** Calcula la **suma** de los valores del grupo, el valor **mayor**, el valor **menor**, la **media aritmética** (*average*), la **desviación típica** (*standard deviation*) y la **varianza** (el cuadrado de la desviación típica).
  - ```
    SELECT
    SUM(Salario), MAX(Salario), MIN(Salario),
    AVG(Salario), VARIANCE(Salario),
    STDDEV(Salario), '=', SQRT(VARIANCE(Salario))
    FROM Empleado;
    ```
  - ```
    SELECT AVG(DISTINCT Salario) "Media"
    FROM Empleado
    WHERE Dpto NOT BETWEEN 5 AND 9;
    ```
  - ```
    SELECT SUM(Salario), AVG(Salario)
    FROM Empleado E, Dpto D
    WHERE E.Dpto=D.NumDpto AND D.Nombre='I+D';
    ```
- **Observación: DISTINCT** y **ALL** no tienen efecto en las funciones **MAX** y **MIN**.
    ```

Funciones de Grupo: *Ejemplos en subconsultas*

- **Funciones de Grupo en Subconsultas** (correlacionadas o no):
  - "Nombre de los suministradores que suministran más de 5 piezas":
    ```
    SELECT NombreS FROM Suministrador S
           WHERE 5 < (SELECT COUNT(*) FROM Suministros
                      WHERE S#=S.S#);
    ```
  - "Nombre de las Piezas que pesan más que la media":
    ```
    SELECT NombreP FROM Pieza
           WHERE Peso > (SELECT AVG(Peso) FROM Pieza);
    ```
  - "Nombre de la pieza o piezas que pesen más":
    ```
    SELECT NombreP FROM Pieza
           WHERE Peso = (SELECT MAX(Peso) FROM Pieza);
    ```
  - "Nombre de piezas que son suministradas por varios suministradores":
    ```
    SELECT NombreP FROM Pieza
           WHERE 2 <= (SELECT COUNT(*) FROM Suministros
                       WHERE P#=Pieza.P#);
    ```
  - "Nombre de piezas que provengan de 3 suministradores de Málaga":
    ```
    SELECT NombreP FROM Pieza P WHERE 3 =
    (SELECT COUNT(*) FROM Suministros SP, Suministradores S
     WHERE P#=P.P# AND SP.S#=S.S# AND Ciudad='Málaga');
    ```


Agrupación con `GROUP BY`

- **Las Funciones de Grupo se pueden aplicar a <u>subgrupos</u> de entre las tuplas recuperadas** (no sólo al grupo formado por TODAS las tuplas recuperadas).
- La **<u>Cláusula</u>** **<u>`GROUP BY`</u>** seguida de una lista de atributos permite agrupar las tuplas en grupos que tengan los mismos valores en todos los atributos de esa lista.
- En una consulta con **`GROUP BY` todos** los elementos seleccionados deben ser:
  - Expresiones de la cláusula **`GROUP BY`**,
  - Expresiones con funciones de grupo, o
  - Constantes.
- Ejemplos:
  - "Para cada departamento, recuperar el número de empleados que tiene, su salario medio, y su mayor y menor salario":
    ```
    SELECT  Dpto, COUNT(*), AVG(Salario), MAX(Salario),
            MIN(salario)
    FROM Empleado  GROUP BY Dpto;
    ```
  - "Para cada pieza, recuperar el número de suministradores que tiene":
    ```
    SELECT P.P#, NombreP, COUNT(*)
    FROM Pieza P, Suministros SP
    WHERE P.P#=SP.P#  GROUP BY P.P#, NombreP;
    ```

Agrupación con GROUP BY *y* HAVING

- **Cláusula HAVING:** Establece una condición sobre los grupos, para que sólo se recuperen los grupos que la cumplen:
  - "Para cada pieza, indicar cuántos suministradores tiene, si son varios":

```
SELECT P.P#, NombreP, COUNT(*) FROM Pieza P, Suministros
  WHERE P.P#=Suministros.P#
        GROUP BY P.P#, NombreP HAVING COUNT(*)>1;
```

  - "Departamentos y el número de sus empleados de aquellos que tienen más de 5 empleados y la media de su salario es mayor que 6":

```
SELECT Dpto, COUNT(*) FROM Empleado
  GROUP BY Dpto HAVING COUNT(*)>5 AND AVG(Salario)>6;
```

  - "Recuperar los departamentos y el número de sus empleados que cobran más de 2.5 de entre aquellos que tienen más de 5 empleados que cobren más que esta cantidad":

```
SELECT Dpto, COUNT(*) FROM Empleado
  WHERE Salario>2.5 GROUP BY Dpto HAVING COUNT(*)>5;
```

  - "Recuperar los departamentos y el número de sus empleados que cobran más de 2.5 de entre aquellos que tienen más de 5 empleados":

```
SELECT Dpto, COUNT(*) FROM Empleado
  WHERE Salario>2.5  AND Dpto IN
  (SELECT Dpto FROM Empleado GROUP BY Dpto
   HAVING COUNT(*)>5 )     GROUP BY Dpto;
```


Resumen del Comando SELECT

- **SELECT** <Select_list>
  - Expresiones a recuperar (atributos, funciones, operaciones...).
- **FROM** <Table_list>
  - Tablas necesarias para la consulta (incluyen tablas de reunión, subconsultas...).
- **[WHERE** <Condición> **]**
  - Condición de selección de tuplas, incluyendo condiciones de reunión.
- **[GROUP BY** <Atributos_para_Agrupar> **]**
  - Atributos por los que agrupar el resultado (cada grupo tendrá los mismos valores en estos atributos). Las funciones de grupo o de agregación se aplicarán sobre cada uno de estos grupos. Se aplicarán sobre todas las tuplas si no existe esta cláusula.
- **[HAVING** <Condición_de_Grupo> **]**
  - Condición sobre los grupos (no sobre las tuplas).
- **[ORDER BY** <Atributos_para_Ordenar> **]**
  - Atributos por los que ordenar. El orden de estos atributos influye.

Insertar Tuplas: Comando `INSERT`

- **INSERTA** una o varias tuplas en una relación. Formato:

```
INSERT INTO <Tabla> VALUES (a1, a2, ..., an);
```

  - **Lista de valores:** En el mismo orden en el que fue creada la tabla con **`CREATE TABLE`**.
  - **Ej.:** `INSERT INTO pieza VALUES (4,'Teja',50,1000);` (P# → 4, NombreP → 'Teja', Peso → 50, Cantidad → 1000)
- Puede especificarse sólo un **subconjunto de atributos**. Formato:
  `INSERT INTO <Tabla>(<Lista_Atribs>) VALUES (...);`
  - **Lista de atributos:** Nombres de los atributos en los que se desea insertar algún valor. Los valores deben estar en ese orden.
    - Atributos ausentes: Se les asigna su valor por defecto o NULL.
    - Deben estar todos los atributos que no admiten NULL y que además no tienen valor por defecto (restr. **`NOT NULL`** y **`DEFAULT`**).
- **Insertar varias tuplas:** Separadas por comas y entre paréntesis.
  - Se puede sustituir la cláusula **`VALUES`**`(...)` por una subconsulta: Se insertarán TODAS las tuplas recuperadas por esa subconsulta.
- **SGBD:** Debe gestionar que se cumplan las restricciones (especialmente la de integridad referencial). Si no, lo tendrá que hacer el usuario.

Comando `INSERT` *con Subconsultas*

- **Insertar el resultado de una subconsulta:** Formato:

```
INSERT INTO <Tabla> <Subconsulta>;
```

  - **Ej.:** Suponemos que la tabla `TOTAL` está creada correctamente.

```
INSERT INTO TOTAL
        SELECT S.S#, NombreS, COUNT(*)
        FROM Suministros SP, Suministrador S
        WHERE SP.S# = S.S#
        GROUP BY S.S#, NombreS
        ORDER BY 2;
```

  - **Observaciones:**
    - La tabla `TOTAL` es una tabla normal: Puede consultarse, modificarse y borrarse como cualquier otra tabla.
    - La tabla `TOTAL` no cambiará si se producen cambios en las tablas de las que se ha extraído su información. Sus datos pertenecerán a la fecha en la que se ejecutó la sentencia **`INSERT`**.
    - Si deseamos que la tabla esté actualizada, crear una **VISTA** (con **`CREATE VIEW`**).

Borrar Tuplas: Comando DELETE

- **BORRA** una o varias tuplas en una relación. Formato:

```
DELETE [FROM] <Tabla> [<Alias>] [WHERE <Cond>];
```

  - **Borra** las tuplas que cumplan la condición**.**
  - **Puede implicar el borrado de otras tuplas en otras tablas,** para que se cumpla una restricción de integridad referencial.
  - **Si se omite la cláusula WHERE** (y su condición), se borran todas las tuplas de la tabla: La tabla quedará vacía (pero sigue existiendo).
  - **Ejemplos:**
    - `DELETE Suministrador WHERE S# IN (2,4,8);`
    -
```
DELETE Suministros SP
      WHERE S# IN (SELECT S# FROM Suministrador
                   WHERE Ciudad='Tegucigalpa');
```
    -
```
DELETE Pieza
      WHERE Peso-250 > (SELECT AVG(Peso) FROM Pieza
      WHERE Peso > (SELECT AVG(Peso)
                    FROM Pieza) );
```


Actualizar Tuplas: Comando UPDATE

- **ACTUALIZA** o **MODIFICA** tuplas de una relación. Formato:

```
UPDATE <Tabla> [<Alias>] SET <Actualizaciones>
                    [ WHERE <Cond> ];
```

  - `<Actualizaciones>` puede ser**:**
    - **1.** Una lista de asignaciones separadas por coma del tipo:
      `<Atributo> = <Expresión>`
      (Una expresión puede ser una subconsulta que recupere sólo un valor).
    - **2.** Una lista de atributos entre paréntesis a los que se le asigna una subconsulta: `(<A1>, ..., <Am>) = (<Subconsulta>)`
  - La cláusula **WHERE** selecciona las tuplas a modificar.
  - Modificar la **llave primaria** puede acarrear la modificación de llaves externas en otras tablas.

- Ejs.:
- `UPDATE Pieza SET Nombre='Eje', Peso=9 WHERE P#=5;`
-
```
UPDATE Pieza SET Cantidad=Cantidad+100
      WHERE P# IN (SELECT P# FROM Suministros
                   WHERE S# IN (3,7));
```
-
```
UPDATE Pieza SET (Peso,Cantidad)=(SELECT Peso,
      Cantidad FROM Pieza WHERE P#=6) WHERE P#=5;
```

DDL de SQL: CREATE TABLE

- **Comando CREATE TABLE:** Crea una nueva relación/tabla **base** con su nombre, atributos (nombres y dominios) y restricciones:

```
CREATE TABLE <NombreTabla>
  (<NombreA1> <TipoA1> [DEFAULT lit] <RestricA1>,
    ...
   <RestriccionesTabla>);
```


DDL de SQL: CREATE TABLE

- **Restricciones de Atributos** (puede haber varias por cada uno):
  - **NOT NULL**
  - **PRIMARY KEY**
  - **CHECK (**<Condición>**)**
  - **UNIQUE**
  - **DEFAULT** <Valor>
  - **REFERENCES...**
- **Restricciones de Tabla:** Pueden tener un **Nombre**, que se asigna al principio, con el formato: **CONSTRAINT <nombreR>** (las rest. de Atrib. también)
  - **PRIMARY KEY (**<Atributos de la Llave Primaria>**)**
  - **CHECK (**<Condición>**)**
  - **UNIQUE (**<Llave Candidata o Secundaria>**)**
  - **FOREIGN KEY (**<Llave Externa>**) REFERENCES** <Tabla>**(**<Atributos>**)**

```
[ON DELETE {CASCADE | SET NULL | SET DEFAULT}]
[ON UPDATE {CASCADE | SET NULL | SET DEFAULT}]
```

Si se **borra** la llave referenciada, se borran las tuplas que la referencian
Si se **actualiza** la llave referenciada, se actualizan las tuplas que la referencian

Si se **borra/actualiza** la llave referenciada, se ponen a NULL los valores que la referencian (llave externa).

Si se **borra/actualiza** la llave referenciada, se ponen los valores que la referencian a su valor por defecto.

Las Restricciones en Oracle

- **Propiamente, <u>Oracle no distingue entre restricciones</u> de tabla y restricciones de Atributo (o de Columna):**
  - Oracle las almacena todas en la vista **`USER_CONSTRAINTS`** del Diccionario de Datos, con atributos como:
    - **`CONSTRAINT_NAME:`** Nombre de la restricción. Si no se le ha dado uno, Oracle le asigna uno con un código.
    - **`TABLE_NAME:`** Nombre de la tabla con dicha restricción.
    - **`CONSTRAINT_TYPE:`** Es un carácter (**`P`** para **`PRIMARY KEY`**, **`U`** para **`UNIQUE`**, **`R`** para una restricción de integridad referencial, **`C`** para una restricción de tipo **`CHECK`** (o **`NOT NULL`**) con la condición almacenada en el atributo **`SEARCH_CONDITION`**...)
    - **`STATUS:`** Estado de la restricción (**`ENABLE`** o **`DISABLE`**).
  - La diferencia entre restricciones de tabla y de atributo es sólo a nivel sintáctico sobre dónde y cómo deben escribirse:
    - Las restricciones que involucren varios atributos deben ser consideradas forzosamente como restricciones de tabla.
    - Las restricciones **`NOT NULL`** y **`DEFAULT`** son forzosamente restricciones de atributo, aunque estrictamente hablando **`DEFAULT`** no es una restricción.
- **Oracle 8 no implementa las opciones de `ON UPDATE` ni la opción `ON DELETE SET DEFAULT`.**
  - Por defecto, Oracle no permite borrar una tupla si existe una o varias tuplas que estén haciendo referencia a algún valor de la tupla que se intenta borrar.
- **Por defecto, las restricciones de tipo `CHECK` sólo se exigen si los atributos involucrados tienen valores distintos de NULL.**

DDL de SQL estándar: `DROP` *y* `ALTER`

- **<u>Borrar Esquemas y Tablas</u>: `DROP`**
  - **`DROP SQUEMA <NombreEsquema [CASCADE | RESTRICT]`**
    - **`CASCADE`** borra el esquema totalmente y **`RESTRICT`** sólo si está vacío.
  - **`DROP TABLE <NombreTabla> [CASCADE CONSTRAINTS | RESTRICT]`**
    - **`CASCADE:`** Borra la tabla (y su contenido). Si hay referencias sobre ella (llaves externas en otras tablas), dichas restricciones son borradas.
    - **`RESTRICT:`** Borra la tabla si no hay referencias sobre ella.
    - En Oracle **`RESTRICT`** no existe y es la opción por defecto. Para borrar las restricciones que hacen referencia a la tabla se usa **`CASCADE CONSTRAINTS`**.

DDL de SQL estándar: DROP *y* ALTER

- **Modificar Tablas**: ALTER TABLE <NombreTabla> <ACCIÓN>
  - **Añadir columna:** ADD (<NombreA, Tipo, Restric_de_Columna>);
    - En Oracle, para modificar un atributo se usa **MODIFY** en vez de **ADD**.
  - **Borrar columna:** DROP <NombreA> [CASCADE|RESTRICT];
  - **Añadir restric. de Atributo:** ALTER <NombreA> SET <RestricA>;
  - **Borrar restric.:** ALTER <NombreA> DROP <TipoRA: DEFAULT...>;
  - **Borrar restric. de tabla** (debe tener un nombre)**:** DROP CONSTRAINT <NombreC> CASCADE;
  - **Añadir restric. de tabla:** ADD (<Restric_de_Tabla>);

ALTER TABLE *en Oracle 8*

- **Modificar Tablas**: ALTER TABLE <NombreTabla> <ACCIÓN>
  - **Añadir Columna:** ADD (<NombreA> <Tipo> [<Restric_Columna>]);
  - **Añadir Restricción de Tabla:** ADD (<Restric_de_Tabla>);
  - **Modif. Col.:** MODIFY (<NombreA> [<Tipo>] [DEFAULT <expr>][[NOT] NULL]);
  - **Estado de una Restricción:** MODIFY CONSTRAINT <NombreR> <Estado>;
    - donde **<Estado>** es: (el **<Estado>** puede seguir a toda restricción)
      - **[NOT] DEFERRABLE:** Indica si el chequeo de la restricción se aplaza al final de la transacción o no se aplaza (por defecto), comprobándola tras la sentencia DML.
      - **ENABLE [VALIDATE|NOVALIDATE]:** Activa la restricción para los nuevos datos (opción por defecto). **VALIDATE** (opción por defecto) exige que se compruebe si la restricción es *válida* en los datos antiguos (lo que tenía previamente la tabla).
      - **DISABLE**: Desactiva la restricción.
    - Si a una restricción no se le ha asignado ningún nombre, Oracle le asigna un nombre. Puede consultarse en la vista **USER_CONSTRAINTS** del diccionario.

ALTER TABLE *en Oracle 8*

- **Modificar Tablas**: `ALTER TABLE <NombreTabla> <ACCIÓN>`
  - **Borrar R. por Tipo:** `DROP {PRIMARY KEY | UNIQUE(<Cols>)} [CASCADE];`
    - No pueden borrarse esas dos restricciones si existe una llave externa que referencie sus atributos**:** Con **CASCADE** se borran todas esas llaves externas.
  - **Borrar Restricción por Nombre:** `DROP CONSTRAINT <NombreR>;`
    - Obligatorio para las restricciones de tipo **CHECK** y **REFERENCES**.
  - **Borrar Columna:** `DROP COLUMN <NombreA> [CASCADE CONSTRAINTS];`
  - **Renombrar Tabla:** `RENAME TO <Nuevo_Nombre_Tabla>;`

DDL de SQL: Ejemplos

```
– CREATE TABLE CLIENTE (
      NIF    CHAR(12) NOT NULL,
      Nombre CHAR(80) NOT NULL,
      Edad   NUMBER(2) CONSTRAINT Edad_Pos CHECK (Edad>0),
      CONSTRAINT PK_Cliente PRIMARY KEY(NIF));
– CREATE TABLE MASCOTA (
    NIF_Owner CONSTRAINT Sin_Propietario
              REFERENCES CLIENTE(NIF)
              ON DELETE CASCADE,
    Nombre CHAR(30) NOT NULL,
    Fecha_Nac DATE,
    Especie CHAR(30) DEFAULT 'Perro' NOT NULL,
    CONSTRAINT PK_Mascota PRIMARY KEY(NIF_Owner,Nombre));
– ALTER TABLE CLIENTE ADD Telefono CHAR(20);
– ALTER TABLE CLIENTE DROP COLUMN Edad CASCADE CONSTRAINTS;
– ALTER TABLE MASCOTA DROP CONSTRAINT Sin_Propietario;
– ALTER TABLE MASCOTA MODIFY Especie NULL;
– ALTER TABLE MASCOTA DROP PRIMARY KEY;
– ALTER TABLE MASCOTA ADD PRIMARY KEY (NIF_Owner,Nombre);
```

Observación: En una llave externa no es necesario poner el tipo. En ese caso, copia el tipo del atributo padre.

Creación de Vistas: `CREATE VIEW`

- **VISTA:** Es una **tabla virtual** cuyas tuplas derivan de otras tablas (que pueden ser tablas base o también otras vistas).
  - Sus tuplas no se almacenan sino que se calculan a partir de las tablas de las que dependa.
  - Son útiles para usar, como si fueran tablas, consultas que se efectúan frecuentemente, y también para cuestiones de **seguridad**.
  - **Formato:**

```
CREATE [OR REPLACE] [[NO] FORCE] VIEW
<NombreV> [(<Lista_Atrbs>)] AS (<Subconsulta>)
[WITH READ ONLY];
```

    - Crea la vista `<NombreV>`, asociada a la subconsulta especificada.
    - La **lista de atributos** es el nombre de los atributos de la vista: Por defecto toma los nombres de los atributos de la subconsulta. Son necesarios si los atributos son calculados (funciones de grupo...).
    - **`OR REPLACE`:** Permite modificar una vista ya existente sin borrarla.
    - **`WITH READ ONLY`:** Indica que no se permitirán borrados, inserciones o actualizaciones en la vista.
    - **`FORCE`:** Fuerza a que la vista se cree aunque no existan los objetos que se usan en ella (tablas, otras vistas...) o no se tengan privilegios suficientes. Esas condiciones serán necesarias para usar la vista. La opción contraria es **`NO FORCE`**, que es la opción por defecto.

Vistas: Ejemplos y Observaciones

- **Ejemplos:**
  -
```
CREATE OR REPLACE VIEW SumiNombres
      AS (SELECT NombreS, NombreP
          FROM Suministros SP,Suministrador S,Pieza P
          WHERE SP.S#=S.S# AND SP.P#=P.P#);
```
  -
```
CREATE OR REPLACE VIEW Cantidad(NombreS,NumPiezas)
      AS (SELECT NombreS, COUNT(*)
          FROM Suministros SP, Suministrador S
          WHERE SP.P#=S.P#
          GROUP BY NombreS);
```

OJO: ¿Qué ocurre si varios Suministradores tienen el mismo nombre?

- **Observaciones:**
  - Las vistas pueden **consultarse como si fueran tablas**.
  - Una vista está **siempre actualizada** (*up to date*): Si se modifican las tablas de las que depende, la vista reflejará esos cambios.
  - Para que la vista **NO se actualice,** no hay que crear una vista, sino una "instantánea", "foto" o vista materializada (*materialized view*) de la BD (con **`CREATE SNAPSHOT`**, o bien con **`CREATE MATERIALIZED VIEW`**).
  - Para **borrar una vista** que ya no es útil: **`DROP VIEW`** `<NombreV>;`

Vistas: Implementación y Modificación

- **Modificar directamente una vista puede ser complicado:**
  - Es **fácil** si es una vista sobre una tabla, sin funciones de agregación e incluye la llave primaria (o una candidata). En general, se consideran vistas **NO actualizables** si están definidas sobre varias tablas (en una reunión) o si usan agrupamiento y/o funciones de agregación.
  - A veces, una **modificación en una vista** puede traducirse de varias formas en las relaciones base, lo cual implica que tal modificación no debe admitirse por ser **ambigua**. A veces, se escoge la más probable.
    - **Ej.:** Si en la vista `SumiNombres` se modifica el nombre de una Pieza: ¿Queremos modificar el nombre de la pieza o queremos modificar la pieza que es suministrada por el correspondiente suministrador?

Control de Transacciones

- **SQL permite controlar la ejecución de una transacción:**
  - **Una transacción empieza con** la primera orden SQL después de conectarse a la base de datos o con la primera orden SQL después de terminar la transacción anterior.
  - **Una transacción termina con: `COMMIT`** o **`ROLLBACK`**.
- **Órdenes SQL de Control de Transacciónes:**
  - **`COMMIT`:** Si la transacción terminó bien y se deben **guardar los cambios** efectuados a la base de datos.
    - Además, **`COMMIT`** hace que los cambios sean visibles por otras sesiones y que se liberen los bloqueos establecidos por la transacción.
    - Hasta que no se usa **`COMMIT`** los cambios no son permanentes, sino temporales y sólo pueden ser vistos por la sesión que los hizo.
  - **`ROLLBACK`:** Si la transacción no terminó bien y se deben **deshacer los cambios** efectuados a la base de datos.
    - **`ROLLBACK`** también libera los bloqueos establecidos.
    - Esta es la opción por defecto si una sesión se desconecta de la base de datos SIN terminar la transacción.
  - **`SAVEPOINT <Nombre>`:** Para deshacer sólo parte de la transacción, se pueden poner varios **puntos de salvaguarda** con distinto nombre cada uno.
    - Tras esto, la orden **`ROLLBACK TO SAVEPOINT <Nombre>`**, deshace todo lo hecho desde el punto **<Nombre>** y libera los bloqueos establecidos tras ese punto de salvaguarda.
    - La transacción no termina con este tipo de **`ROLLBACK`**.

Disparadores: TRIGGER

- Es un bloque PL/SQL que **se Ejecuta de forma Implícita** cuando se ejecuta cierta **Operación DML: INSERT, DELETE o UPDATE.**
  - Contrariamente, los procedimientos y las funciones se ejecutan haciendo una llamada **Explícita** a ellos.
  - **Un Disparador No admite Argumentos.**
- **Utilidad:** Sus aplicaciones son inmensas, como por ejemplo:
  - **Mantenimiento de Restricciones de Integridad complejas.**
    - Ej: Restricciones de Estado (como que el sueldo sólo puede aumentar).
  - **Auditoría de una Tabla,** registrando los cambios efectuados y la identidad del que los llevó a cabo.
  - **Lanzar cualquier acción** cuando una tabla es modificada.
- **Formato:**

```
CREATE [OR REPLACE] TRIGGER <NombreT>
   {BEFORE|AFTER} <Suceso_Disparo> ON <Tabla>
   [FOR EACH ROW [WHEN <Condición_Disparo>]]
<Bloque_del_TRIGGER>;
```

47

Elementos de un TRIGGER

- **Nombre:** Se recomienda que identifique su función y la tabla sobre la que se define.
- **<Suceso_Disparo>** es la operación **DML** que efectuada sobre **<Tabla>** disparará el *trigger*.
  - Puede haber varios sucesos separados por la palabra **OR.**
- **Tipos de Disparadores:** Hay **12 Tipos** básicos según**:**
  - **Orden, Suceso u Operación DML: INSERT, DELETE o UPDATE.**
    - Si la orden **UPDATE** lleva una lista de atributos el Disparador sólo se ejecutará si se actualiza alguno de ellos**: UPDATE OF <Lista_Atributos>**
  - **Temporización:** Puede ser **BEFORE (anterior) o AFTER (posterior).**
    - Define si el disparador se activa **antes o después** de que se ejecute la operación **DML** causante del disparo.
  - **Nivel:** Puede ser **a Nivel de Orden** o **a Nivel de Fila (FOR EACH ROW).**
    - **Nivel de Orden** (*statement trigger*)**:** Se activan sólo una vez, antes o después de la **Orden** u operación **DML**.
    - **Nivel de Fila** (*row trigger*)**:** Se activan una vez por cada **Fila** afectada por la operación **DML** (una misma operación **DML** puede afectar a varias filas).

48

Elementos de un TRIGGER

- **Ejemplo:** Guardar en una tabla de control la fecha y el usuario que modificó la tabla `Empleados`:
  (NOTA: Este trigger es **AFTER** y a nivel de orden)

```
CREATE OR REPLACE TRIGGER Control_Empleados
  AFTER INSERT OR DELETE OR UPDATE ON Empleados
BEGIN
  INSERT INTO Ctrl_Empleados (Tabla,Usuario,Fecha)
    VALUES ('Empleados', USER, SYSDATE);
END Control_Empleados;
```

49

Orden en la Ejecución de Disparadores

- Una tabla puede tener distintos Tipos de **Disparadores** asociados a una misma **orden DML.**
- En tal caso, el **Algoritmo de Ejecución** es:
  - **1. Ejecutar, si existe, el disparador tipo BEFORE a nivel de orden.**
  - **2. Para cada fila a la que afecte la orden:** (esto es como un bucle, para cada fila)
    - **a) Ejecutar, si existe, el disparador BEFORE a nivel de fila,** sólo si dicha fila cumple la condición de la cláusula **WHEN** (si existe).
    - **b) Ejecutar la propia orden.**
    - **c) Ejecutar, si existe, el disparador AFTER a nivel de fila,** sólo si dicha fila cumple la condición de la cláusula **WHEN** (si existe).
  - **3. Ejecutar, si existe, el disparador tipo AFTER a nivel de orden.**

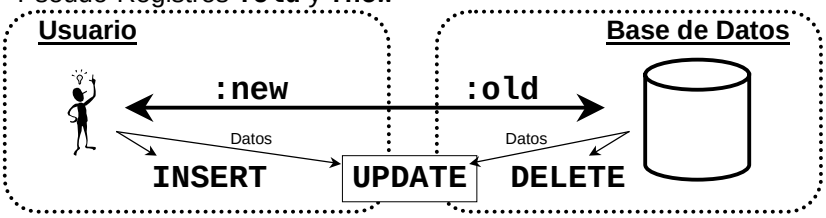
50

Disparadores de Fila: `:old` *y* `:new`

- **Disparadores a Nivel de Fila:**
  - Se ejecutan **una vez por cada Fila** procesada por la orden **DML** disparadora.
  - Para acceder a la **Fila Procesada** en ese momento, se usan dos **Pseudo-Registros** de tipo **<TablaDisparo>%ROWTYPE: `:old` y `:new`**

Orden DML	`:old`	`:new`
`INSERT`	**No Definido: NULL** ∀ campo.	**Valores Nuevos** a Insertar.
`UPDATE`	**Valores Originales** (antes de la orden).	**Valores Actualizados** (después).
`DELETE`	**Valores Originales** (antes de la orden).	**No Definido: NULL** ∀ campo.

  - **Esquema** para la comprensión del significado y utilidad de los Pseudo-Registros `:old` y `:new`



51

Disparadores a Nivel de Fila: Ejemplo

- **Ejemplo:** Programar un Disparador que calcule el campo código de Pieza (P#) cada vez que se inserte una nueva pieza**:**

```
CREATE OR REPLACE TRIGGER NuevaPieza
  BEFORE INSERT ON Pieza FOR EACH ROW
BEGIN
  -- Establecer el nuevo número de pieza:
  SELECT MAX(P#)+1 INTO :new.P# FROM Pieza;
  IF :new.P# IS NULL THEN
     :new.P# := 1;
  END IF;
END NuevaPieza;
```

  - Cuando se ejecute la orden **DML** se emplean los valores del **Pseudo-Registro `:new`**. Una orden válida sería la siguiente, sin que produzca un error por no tener la llave**:**

```
INSERT INTO Pieza (Nombre, Peso)
       VALUES ('Alcayata', 0.5);
```

    - En este caso, si se suministra un valor para la llave primaria, tampoco producirá un error, pero ese valor será ignorado y sustituido por el nuevo valor calculado por el disparador.
- **Modificar Pseudo-Registros:**
  - No puede modificarse el **Pseudo-Registro `:new`** en un disparador **AFTER** a nivel de fila.
  - El **Pseudo-Registro `:old`** nunca se modificará**:** Sólo se leerá.

52

Disparadores a Nivel de Fila: WHEN

- **Formato: ... FOR EACH ROW WHEN <Condición_Disparo>**
  - Sólo es válida en Disparadores **a Nivel de Fila** y siempre es **opcional**.
  - Si existe, el cuerpo del disparador se ejecutará **sólo para las filas que cumplan la Condición de Disparo** especificada.
  - En la **Condición de Disparo** pueden usarse los Pseudo-Registros **:old** y **:new,** pero en ese caso **no se escribirán los dos puntos** (**:**), los cuales son obligatorios en el cuerpo del disparador.
  - **Ejemplo:** Deseamos que los precios grandes no tengan más de 1 decimal. Si tiene 2 ó más decimales, redondearemos ese precio**:**

```
-- Si el Precio>200, redondearlos a un decimal:
CREATE OR REPLACE TRIGGER Redondea_Precios_Grandes
  BEFORE INSERT OR UPDATE OF Precio ON Pieza
  FOR EACH ROW WHEN (:new.Precio > 200)
BEGIN
  :new.Precio := ROUND(:new.Precio,1);
END Redondea_Precios_Grandes;
```

- Se puede escribir ese disparador sin la cláusula **WHEN**, usando un **IF:**

```
... BEGIN
      IF :new.Precio > 200 THEN
         :new.Precio := ROUND(:new.Precio,1);
...   END IF;
```

53

Predicados INSERTING, DELETING, *y* UPDATING

- En los disparadores que se ejecutan cuando ocurren **diversas Operaciones DML (INSERT, DELETE o UPDATE),** pueden usarse 3 **Predicados Booleanos** para conocer la operación disparadora**:**
  - **INSERTING** Vale **TRUE** si la orden de disparo es **INSERT**.
  - **DELETING** Vale **TRUE** si la orden de disparo es **DELETE**.
  - **UPDATING** Vale **TRUE** si la orden de disparo es **UPDATE**.
- **Ejemplo:** Guardar en una tabla de control la fecha, el usuario que modificó la tabla Empleados y el **Tipo de Operación** con la que modificó la tabla. Usando los Pseudo-Registros **:old** y **:new** también se pueden registrar los valores antiguos y los nuevos (si procede)**:**

```
CREATE OR REPLACE TRIGGER Control_Empleados
  AFTER INSERT OR DELETE OR UPDATE ON Empleados
BEGIN
  IF INSERTING THEN
     INSERT INTO Ctrl_Empleados (Tabla,Usuario,Fecha,Oper)
       VALUES ('Empleados', USER, SYSDATE,'INSERT');
  ELSIF DELETING THEN
     INSERT INTO Ctrl_Empleados (Tabla,Usuario,Fecha,Oper)
       VALUES ('Empleados', USER, SYSDATE,'DELETE');
  ELSE
     INSERT INTO Ctrl_Empleados (Tabla,Usuario,Fecha,Oper)
       VALUES ('Empleados', USER, SYSDATE,'UPDATE');
   END
END Control_Empleados;
```

54

Disparadores de Sustitución: `INSTEAD OF`

- **Disparadores de Sustitución:** Disparador que se ejecuta en lugar de la orden DML (ni antes ni después, sino sustituyéndola).
  - Cuando se intenta **modificar una vista** esta modificación puede no ser posible debido al formato de esa vista.
  - **Características:**
    - **Sólo pueden definirse sobre Vistas.**
    - **Se activan en lugar de la Orden DML** que provoca el disparo, o sea, la orden disparadora no se ejecutará nunca.
    - **Deben tener Nivel de Filas.**
    - Se declaran usando **`INSTEAD OF`** en vez de **`BEFORE/AFTER`**.
- **Ejemplo:** Supongamos la siguiente vista:

```
CREATE VIEW Totales_por_Suministrador AS
  SELECT S#, MAX(Precio) Mayor, MIN(Precio) Menor
  FROM Suministros SP, Pieza P
  WHERE SP.P# = P.P#  GROUP BY S#;
```

  - Disparador que borre un suministrador si se borra una tupla sobre esa vista:

```
CREATE OR REPLACE TRIGGER Borrar_en_Totales_por_S#
  INSTEAD OF DELETE ON Totales_por_Suministrador
  FOR EACH ROW
BEGIN
  DELETE FROM Suministrador WHERE S# = :old.S#;
END Borrar_en_Totales_por_S#;
```

55

Disparadores: Observaciones

- Un *trigger* puede contener cualquier orden legal en un bloque PL/SQL, con las siguientes **Restricciones:**
  - **No puede emitir órdenes de Control de Transacciones: `COMMIT`, `ROLLBACK` o `SAVEPOINT`.**
    - El disparador se activa como parte de la orden que provocó el disparo y, por tanto, forma parte de la misma transacción. Cuando esa transacción es confirmada o cancelada, se confirma o cancela también el trabajo realizado por el disparador.
  - Cualquier **Subprograma** llamado por el disparador **tampoco puede emitir órdenes de control de transacciones.**
  - **Un disparador puede tener Restringido el acceso a ciertas tablas.**
    - Dependiendo del tipo de disparador y de las restricciones que afecten a las tablas, dichas tablas pueden ser **mutantes**.
- **Diccionario de Datos:** Todos los datos de un **`TRIGGER`** están almacenados en la vista **`USER_TRIGGERS`**, en columnas como **`OWNER`**, **`TRIGGER_NAME`**, **`TRIGGER_TYPE`**, **`TABLE_NAME`**, **`TRIGGER_BODY`**...
- **Borrar un Disparador: `DROP TRIGGER <NombreT>;`**
- **Habilitar/Deshabilitar un Disparador,** sin necesidad de borrarlo:

```
ALTER TRIGGER <NombreT> {ENABLE | DISABLE};
```

56

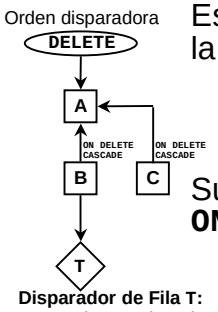
Disparadores: Tablas Mutantes

- **Tabla de Restricción o Tabla Padre** (*constraining table, Parent*)**:** Tabla a la que referencia una llave externa en una **Tabla Hijo** (*Child*), por una Restricción de Integridad Referencial.
- **Tabla Mutante** (*mutating table*)**:** Una tabla es mutante**:**
  - **1. Si está modificándose "actualmente" por una orden DML (`INSERT`, `DELETE` o `UPDATE`).**
  - **2. Si está siendo leída por Oracle para forzar el cumplimiento de una restricción de integridad referencial.**
  - **3. Si está siendo actualizada por Oracle para cumplir una restricción con `ON DELETE CASCADE`.**
    - **Ejemplo:** Si la cláusula `ON DELETE CASCADE` aparece en la tabla **Suministros**, borrar una **Pieza** implica borrar todos sus **Suministros.**
      - **Pieza** (y Suministrador) es una Tabla de Restricción de **Suministros**.
      - Al borrar una pieza ambas tablas serán mutantes, si es necesario borrar suministros. Si se borra una pieza sin suministros sólo será mutante la tabla **Pieza**.
- **Un Disparador de Fila se define sobre una Tabla Mutante** (casi siempre)**.**
  - En cambio, un **Disparador de Orden NO** (casi nunca)**:** El disparador de Orden se ejecuta antes o después de la orden, pero no a la vez.

57

Disparadores: Tablas Mutantes

- **Las Órdenes SQL en el cuerpo de un disparador tienen las siguientes 2 Restricciones por las que NO PUEDEN:**
  - **1. Leer o Modificar ninguna Tabla Mutante de la orden que provoca el disparo.**



Disparador de Fila T: No puede acceder a las tablas **A**, **B** y **C**, con esa orden disparadora.

Esto incluye lógicamente la tabla de dicha orden y la tabla del disparador, que pueden ser distintas.

Ej.: Borramos una tupla de una tabla **A**. Esto implica borrar tuplas de una tabla **B** (que tiene una restricción `ON DELETE CASCADE` sobre **A**). Si este segundo borrado sobre **B** dispara algún disparador **T**, entonces, ese disparador no podrá ni leer ni modificar las tablas **A** y **B** pues son **mutantes**. Si lo hace se producirá un error.

Sus tablas de restricción afectadas por la cláusula `ON DELETE CASCADE`, también son mutantes.

Ej.: Si borrar en **A** implica borrar en una tercera tabla **C**, el disparador sobre **B** no podrá tampoco acceder a **C**, si ese disparador se activa por borrar sobre **A** (aunque en **C** no se borre nada). Sí podrá acceder a **C** si se activa por borrar directamente sobre **B**. También podría acceder a **C** si ésta no tuviera el `ON DELETE CASCADE` pero hay que tener en cuenta que NO se podrán borrar valores en **A**, si están siendo referenciados en **C** (ORA-2292).

  - **2. Modificar las columnas de clave primaria, única o externa de una Tabla de Restricción a la que la tabla del disparador hacen referencia:** Sí pueden modificarse las otras columnas.

58

¿Qué Tablas son Mutantes?

- Al escribir disparadores es importante responder a **Dos Preguntas:**
  - **¿Qué Tablas son Mutantes?**
  - **¿Cuando esas Tablas Son mutantes?**
- **¿Qué Tablas son Mutantes?** → Para responder a esa pregunta es necesario conocer el tipo de orden **DML** que se está ejecutando.
  - **1.** Son mutantes las tablas afectadas por una operación **`INSERT`**, **`DELETE`** o **`UPDATE`**.
  - **2.** Si una **tabla Hijo** (p.e. `Empleados`) tiene un atributo llave externa (`Dpto`) a otra **tabla Padre** (`Departamentos`), **ambas** tablas serán mutantes si:
    - **Insertamos** (**`INSERT`**) en la tabla **Hijo:** Comprobar valores en la tabla padre.
    - **Borramos** (**`DELETE`**) de la tabla **Padre:** Impedir que tuplas hijas se queden sin padre.
    - **Actualizamos** (**`UPDATE`**) la tabla **Padre** o la tabla **Hijo:** Las 2 operaciones anteriores.
  - **3.** Si existe la restricción **`ON DELETE CASCADE`**, esta implica que si se borra de la **tabla Padre**, se borrarán las tuplas relacionadas en la **tabla Hijo** y, a su vez, pueden borrarse tuplas de tablas hijos de esa **tabla Hijo**, y así sucesivamente. En ese caso todas esas tablas serán mutantes.
    - En disparadores activados por un **`DELETE`**, es importante tener en cuenta si pueden ser activados por un borrado en cascada y, en ese caso no es posible acceder a todas esas tablas mutantes.

59

¿Cuándo son las Tablas Mutantes?

- **Las 2 Restricciones anteriores en las órdenes SQL de un Disparador se aplican a:**
  - **Todos los Disparadores con Nivel de Fila.**
    - **Excepción:** Cuando la orden disparadora es un **`INSERT`** que afecta a una única fila, entonces esa tabla disparadora no es considerada como mutante en el **Disparador de Fila-Anterior**.
      - Con las órdenes del tipo **`INSERT INTO Tabla SELECT...`** la tabla del disparador será mutante en ambos tipos de disparadores de Fila si se insertan varias filas.
      - Este es el único caso en el que un disparador de Fila puede leer o modificar la tabla del disparador.
  - **Los Disparadores a Nivel de Orden cuando la orden de disparo se activa como resultado de una operación `ON DELETE CASCADE` (al borrar tuplas en la tabla Padre).**
- **Los ERRORES por Tablas Mutantes se detectan y se generan en Tiempo de Ejecución y no de Compilación** (ORA-4091)**.**

60

Tablas Mutantes: Disparador de Ejemplo

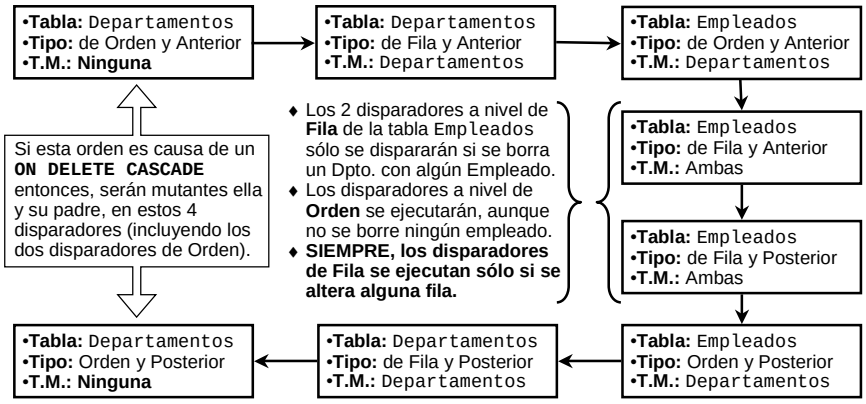
- **Disparador** que modifique el número de empleados de un departamento (columna `Departamentos.Num_Emp`) cada vez que sea necesario.
  - Ese número cambia al **INSERTAR** o **BORRAR** uno o más empleados, y al **MODIFICAR** la columna `Dpto` de la tabla `Empleados`, para uno o varios empleados.
  - La tabla `Departamentos` es una tabla de restricción de la tabla `Empleados`, pero el Disparador es correcto, porque modifica `Num_Emp`, que no es la llave primaria.
  - Este disparador no puede consultar la tabla `Empleados`, ya que esa tabla es mutante: `SELECT COUNT(*) FROM Empleados WHERE Dpto = :new.Dpto;`

```
CREATE OR REPLACE TRIGGER Cuenta_Empleados
  BEFORE DELETE OR INSERT OR UPDATE OF Dpto ON Empleados
  FOR EACH ROW
BEGIN
  IF INSERTING THEN
     UPDATE Departamentos SET Num_Emp = Num_Emp+1
       WHERE NumDpto=:new.Dpto;
  ELSIF UPDATING THEN
     UPDATE Departamentos SET Num_Emp = Num_Emp+1
       WHERE NumDpto=:new.Dpto;
     UPDATE Departamentos SET Num_Emp = Num_Emp-1
       WHERE NumDpto=:old.Dpto;
  ELSE
     UPDATE Departamentos SET Num_Emp = Num_Emp-1
       WHERE NumDpto=:old.Dpto;
  END IF;
END;
```

61

Tablas Mutantes: Ejemplo Esquemático

- Sea una **tabla Padre** `Departamentos`, y una **Hijo** `Empleados` cuyo atributo llave externa es `Dpto` con la restricción **`ON DELETE CASCADE`** que fuerza a borrar todos los empleados de un departamento si su departamento es borrado.
- Supongamos que para la orden **`DELETE`** están implementados los 4 tipos de disparadores posibles (de fila y de órden, anterior y posterior) para las dos tablas.
- Si se ejecuta una orden **`DELETE`** sobre la **tabla Padre** `Departamentos`, se ejecutarán los siguientes disparadores, con las tablas mutantes indicadas, en este orden:



62

Tablas Mutantes: Esquema Padre/Hijo

Órdenes sobre `Departamentos` **(Tabla Padre):**

DELETE:

Los 4 disparadores de la tabla Hijo `Empleados` sólo se dispararán si se borra un Departamento que tenga algún Empleado.

Si esta orden sobre el Padre es causa de un **ON DELETE CASCADE** sobre una tabla **X** (padre del padre) entonces, serán mutantes ella y su padre **X**, en sus 4 disparadores. Ambas tablas también serán mutantes en los 4 disparadores de la tabla Hijo `Empleados`.

INSERT: Su tabla Hijo no se ve afectada: No se disparan sus disparadores.

UPDATE: Su tabla Hijo no se ve afectada, porque sólo se permiten actualizar valores no referenciados en sus tablas Hijos (ORA-2292).

Órdenes sobre `Empleados` **(Tabla Hijo):** No se dispara ningún disparador del Padre.

DELETE: No afecta a la tabla Padre y sólo es mutante la Hijo.

INSERT:

- **Insertar una fila:**

  Las tabla Hijo **no** es mutante en el disparador de **Fila-Anterior** (a pesar de ser la tabla del disparador) y **sí** lo es en el de **Fila-Posterior**.

  La tabla Padre es mutante sólo si se intenta modificar el atributo al que hace referencia la tabla Hijo y sólo en el disparador de **Fila-Posterior**, ya que durante la ejecución de disparador de Fila-Anterior aún no está mutando ninguna tabla.

- **Insertar varias filas:** Las tablas Padre e Hijo son mutantes en los dos disparadores de **Fila**, pero la tabla Padre sólo si se modifica el atributo al que hace referencia la tabla Hijo.

UPDATE: Las tablas Padre e Hijo son mutantes en los dos disparadores de **Fila**, pero la tabla Padre sólo si se modifica el atributo al que hace referencia la tabla Hijo.

63

Tablas Mutantes: Solución

- El disparador anterior `Cuenta_Empleados` tiene dos inconvenientes:
  - Modifica el campo `Num_Emp` aunque este no tenga el valor correcto.
  - Si se modifica directamente ese campo, quedará incorrecto siempre.
- **Solución al Problema de la Tabla Mutante:**
  - Las tablas mutantes surgen básicamente en los disparadores a nivel de Fila. Como en estos no puede accederse a las tablas mutantes, la **Solución es Crear Dos Disparadores:**
    - **A Nivel de Fila:** En este guardamos los valores importantes en la operación, pero no accedemos a tablas mutantes.
      - Estos valores pueden guardarse en:
        - » **Tablas de la BD** especialmente creadas para esta operación.
        - » **Variables o tablas PL/SQL de un paquete:** Como cada sesión obtiene su propia instancia de estas variables no tendremos que preocuparnos de si hay actualizaciones simultáneas en distintas sesiones.
    - **A Nivel de Orden Posterior (AFTER):** Utiliza los valores guardados en el disparador a Nivel de Fila para acceder a las tablas que ya no son mutantes.

64

Tablas Mutantes: Solución

```
CREATE OR REPLACE PACKAGE Empleados_Dpto AS
  TYPE T_Dptos IS TABLE OF Empleados.Dpto%TYPE
    INDEX BY BINARY_INTEGER;
  Tabla_Dptos T_Dptos;
END Empleados_Dpto;

CREATE OR REPLACE TRIGGER Fila_Cuenta_Empleados
  AFTER DELETE OR INSERT OR UPDATE OF Dpto ON Empleados FOR EACH
   ROW
DECLARE Indice BINARY_INTEGER;
BEGIN Indice := Empleados_Dpto.Tabla_Dptos.COUNT + 1;
      IF INSERTING   THEN
           Empleados_Dpto.Tabla_Dptos(Indice)  := :new.Dpto;
      ELSIF UPDATING THEN
           Empleados_Dpto.Tabla_Dptos(Indice)  := :new.Dpto;
           Empleados_Dpto.Tabla_Dptos(Indice+1):= :old.Dpto;
      ELSE Empleados_Dpto.Tabla_Dptos(Indice)  := :old.Dpto;
      END IF;
END Fila_Cuenta_Empleados;
```

65

Tablas Mutantes: Solución

```
CREATE OR REPLACE TRIGGER Orden_Cuenta_Empleados
  AFTER DELETE OR INSERT OR UPDATE OF Dpto ON Empleados
DECLARE Indice BINARY_INTEGER;
        Total  Departamentos.Num_Emp%TYPE;
        Departamento Departamentos.NumDpto%TYPE;
BEGIN
  FOR Indice IN 1..Empleados_Dpto.Tabla_Dptos.COUNT LOOP
     Departamento := Empleados_Dpto.Tabla_Dptos(Indice);
     SELECT COUNT(*) INTO Total FROM Empleados WHERE Dpto    =
   Departamento;
     UPDATE Departamentos SET Num_Emp = Total  WHERE NumDpto =
   Departamento;
  END LOOP;
  Empleados_Dpto.Tabla_Dptos.DELETE;
END Orden_Cuenta_Empleados;
```

66

Tablas Mutantes: Observaciones

- **<u>Variables Contenidas en un Paquete</u>:**
  - Son Variables Globales de la Sesión.
  - Son visibles para todos los programas de cada sesión.
- **<u>En el Disparador a Nivel de Orden Posterior</u>:**
  - Es necesario **borrar la tabla PL/SQL**, para que la siguiente orden empiece a introducir valores en esa tabla a partir de la posición 1.
    - También podría declararse otra variable en el paquete que mantuviera el número de elementos de la tabla, poniéndola a cero al final. Esto evita tener que usar los atributos de la tabla PL/SQL (**`COUNT`** y **`DELETE`**).
  - Hay que tener en cuenta que se ejecutará **después** de la orden, o sea, que los datos ya estarán actualizados.
  - Si este disparador produce un **error** la orden se deshará (*rollback*). Ese error puede ser:
    - Producido por un error en el disparador o por una operación no válida del disparador.
    - Generado por el disparador de forma explícita porque se haya detectado que la orden disparadora no es admisible (por cualquier razón).
      - Esto se hace con el procedimiento **`RAISE_APPLICATION_ERROR`**.
- **Por supuesto, cualquier orden SQL de cualquier bloque PL/SQL debe <u>Respetar</u> siempre todas las <u>Restricciones de Integridad</u>.**

67