Universidade da Coruña
Departamento de Computación

Introducción a SQL sobre Oracle

Luis A. González Ares
lgares@udc.es

Lenguaje SQL

Contenido

- Lenguaje SQL sobre Oracle.
  - Descripción de las características y elementos fundamentales del lenguaje.
  - Ejercicios sobre la sentencia `SELECT`.
  - Otras sentencias DML.
  - Sentencias DDL.
  - Vistas.
  - Indices.
  - Seguridad.
  - Transacciones.
  - Catálogo.

Medios de apoyo

1. Servidor `oracle` o `xurxo` con el SGBD Oracle v 9.2
2. Fotocopiadora.

Nota

El presente material es un resumen de lo impartido en las clases de la Facultad de Informática, que se entrega como documento de apoyo.

Lenguaje SQL

Introducción a SQL

SQL (*Structured Query Language* : Lenguaje de Consulta Estructurado)
Acceder a los datos de un SGBD relacional (crear, consultar, modificar).

Evolución:

- SEQUEL (*Structured English QUEry Language*). IBM para SYSTEM R.
- SQL-86 o SQL1. ANSI e ISO en 1986. Revisado en 1989.
- SQL-92 o SQL-2. 1992.
- SQL:1999, SQL:2003.

Dos vertientes fundamentales en los lenguajes de BD relacionales:

- DDL o LDD (lenguaje de definición de datos).
- DML o LMD (lenguaje de manipulación de datos).

Otras características:

- Aspectos de control (seguridad, transacciones, concurrencia, etc.),
- Posibilidad de incluir SQL en lenguajes de propósito general como C, Pascal, Java, etc.
- Lenguaje no procedural (el qué frente al cómo)
- Actúa sobre un conjunto de registros o filas.
- Base teórica: se fundamenta en el álgebra relacional y en el cálculo relacional.
- Lenguaje para todo tipo de usuarios de un SGBD relacional.
- Trabaja con tablas, columnas y filas, no con relaciones, atributos y tuplas. Una tabla es diferente a una relación, ya que puede tener filas repetidas.
- No es un lenguaje de propósito general.

Lenguaje SQL

SQL*Plus de Oracle

Los fabricantes amplian las características del estándar pero dejan elementos sin implementar.

SQL*Plus: Interface de Oracle que permite ejecutar sentencias SQL y comandos propios.

Entrar en SQL*Plus (servidor `oracle` o `xurxo`):

```
$ sqlplus / [@filename]
```

Comandos SQL*Plus. Uno por fila:

```
SQL> DESC[RIBE] emp

Nombre                                    Nulo?    Tipo
----------------------------------------- -------- ----------------------------

EMPNO                                     NOT NULL NUMBER(4)
ENAME                                              VARCHAR2(10)
JOB                                                VARCHAR2(9)
MGR                                                NUMBER(4)
HIREDATE                                           DATE
SAL                                                NUMBER(7,2)
COMM                                               NUMBER(7,2)
DEPTNO                                             NUMBER(2)
```

Sentencias SQL: Multifila y acaban en `;`

```
SQL> SELECT *
     FROM   emp;
```

Salir de SQL*Plus:

```
SQL> EXIT
```

Lenguaje SQL

SQL*Plus de Oracle (cont.)

Crear el fichero de configuración `login.sql` y ubicarlo en el directorio de llamada a SQL*Plus. Por ejemplo:

```
$ cat  login.sql
def_editor=vi
set pages 50
set feed 1
column empno  format  9990
column mgr    format  9990
column sal    format  99,990
column comm   format  99,990
column deptno format  90
```

SQL*Plus usa un buffer para almacenar la última sentencia SQL ejecutada (no afecta a los comandos SQL*Plus). Incorpora un editor de líneas elemental para realizar cambios mínimos en esa sentencia SQL, como visualizarla (LIST), editarla (ED), etc.

Algunas órdenes muy habituales son:

```
L[IST] [n [m]]            Visualiza líneas del buffer
ED[IT] fichero[.sql]      Edita un fichero [.sql]
R[UN]                     Ejecuta lo almacenado en el buffer
/                         Ejecuta lo almacenado en el buffer
SAV[E] fichero[.sql]      Almacena el buffer en un fichero
GET fichero[.sql]         Pasa al buffer un fichero
```

Lenguaje SQL

SQL*Plus de Oracle (cont.)

Una lista más amplia de órdenes:

```
L[IST] [n [m]]            Visualiza líneas del buffer
A[PPEND] texto            Añade texto al final de la línea
C[HANGE] /viejo/nuevo/    Cambia una cadena de caracteres por otra
C[HANGE] /texto/          Una forma de "cambiarla"
CL[EAR] BUFF[ER]          Elimina los datos del buffer
DEL    [n [m]]            Suprime líneas
I[NPUT] [texto]           Añade una línea nueva despues de la "actual"
R[UN]                     Ejecuta lo almacenado en el buffer
/                         Ejecuta lo almacenado en el buffer

n                         Nos posiciona en una línea del buffer
n texto                   Introduce una línea con "texto" como contenido
                          antes de la línea "n".
0 texto                   Lo mismo, pero antes de la primera línea.
SAV[E] fichero[.sql]      Almacena el buffer en un fichero
GET fichero[.sql]         Pasa al buffer un fichero
STA[RT] fichero[.sql]     Ejecuta un fichero [.sql]
@fichero[.sql]            Idem
ED[IT] fichero[.sql]      Edita un fichero [.sql]
SPO[OL] [fichero[.lst]]   Define un fichero de registro de sesión
SPO[OL] {OFF/OUT}         Finaliza el registro de la sesión
HELP [tema]               Ayuda elemental sobre algunos temas
HO[ST] comando            Ejecuta un comando del sistema operativo
!comando                  Ejecuta un comando del sistema operativo
```

Lenguaje SQL

Sentencias SQL

- Lenguaje de definición de datos: Permite crear, manipular y eliminar estructuras

`CREATE TABLE`	Crea una tabla.
`DROP TABLE`	Elimina una tabla.
`CREATE VIEW`	Crea una vista o tabla virtual.
`DROP VIEW`	Elimina una vista.
`ALTER TABLE`	Modifica la estructura de una tabla.
`CREATE INDEX`	Crea un índice para una tabla.
...	

- Lenguaje de manipulación de datos: Permite consultar, modificar, introducir y eliminar datos de las estructuras de una BD relacional.

`SELECT`	Obtiene datos de la BD.
`INSERT`	Introduce datos.
`UPDATE`	Actualiza datos.
`DELETE`	Borra datos.

- (Oracle) Lenguaje de control de datos: Sentencias que realizan otras funciones, por ejemplo sobre proceso transaccional, concurrencia, etc.

`COMMIT`	Confirma una transacción.
`ROLLBACK`	Anula una transacción.
`GRANT`	Otorga permisos a usuarios en objetos de la BD.
`REVOKE`	Elimina permisos a usuarios en objetos de la BD.
...	

Por ejemplo:

```
DROP TABLE mitabla;
```

```
SELECT empno, ename
FROM   emp;
```

Lenguaje SQL

Tipos de Datos

Cada columna de una tabla tiene un tipo de dato asignado, que determina los valores posibles y las operaciones permitidas sobre esos valores.

Un tipo de dato puede asignarse a un dominio, con lo que se asignará a cada columna de ese dominio.

- Tipos de datos numéricos
  - Oracle
    `NUMERIC(m[,n])` o `NUMBER(m[,n])`
    Ej.: `NUMBER(3)` `NUMBER(5,2)`
  - SQL2
    `INTEGER` o `INT`, `SMALLINT`
    `FLOAT`, `REAL`, `DOUBLE PRECISION`
    `DECIMAL(m,n)` o `DEC(m,n)` o `NUMERIC(m,n)`
- Tipos de datos carácter o alfanuméricos
  - Oracle
    `CHAR[(n)]`
    Ej.: `CHAR(20)` `CHAR`
    Ej.: `'micasa'` `'A'`
    `VARCHAR2[(n)]`
    `LONG`
  - SQL2
    `CHAR(n)` o `CHARACTER(n)`
    `VARCHAR(n)`, `CHARVARYING(n)`, `CHARACTER VARYING(n)`
- Tipos de datos temporales
  - Oracle
    `DATE`
    Ej.: `'06-JAN-2002'`
    Ej.: `'06-JAN-2002 13:21'`
  - SQL2
    `DATE` `YEAR, MONTH, DAY`
    `TIME` `HOUR, MINUTE, SECOND`
    `TIMESTAMP`
    `INTERVAL`

Lenguaje SQL

Tablas de los ejemplos

Tenemos una tabla de empleados (EMP) y otra de departamentos (DEPT).

```
SQL> DESC[RIBE] emp

Nombre                                    Nulo?    Tipo
----------------------------------------- -------- ----------------------------
EMPNO                                     NOT NULL NUMBER(4)
ENAME                                              VARCHAR2(10)
JOB                                                VARCHAR2(9)
MGR                                                NUMBER(4)
HIREDATE                                           DATE
SAL                                                NUMBER(7,2)
COMM                                               NUMBER(7,2)
DEPTNO                                             NUMBER(2)


SQL> DESCRIBE dept

Nombre                                    Nulo?    Tipo
----------------------------------------- -------- -------------
DEPTNO                                    NOT NULL NUMBER(2)
DNAME                                              VARCHAR2(14)
LOC                                                VARCHAR2(13)
```

Los datos de cada tabla se obtienen con la sentencia SELECT:

```
SQL> SELECT *                          SQL> SELECT *
     FROM   emp;                            FROM   dept;
```

Lenguaje SQL

Tablas de los ejemplos(cont.)

```
SQL> SELECT  *  FROM    emp

EMPNO ENAME      JOB          MGR HIREDATE     SAL     COMM DEPTNO
----- ---------- --------- ----- -------- ------- ------- ------
 7369 SMITH      CLERK      7902 17/12/80     800              20
 7499 ALLEN      SALESMAN   7698 20/02/81   1,600     300      30
 7521 WARD       SALESMAN   7698 22/02/81   1,250     500      30
 7566 JONES      MANAGER    7839 02/04/81   2,975              20
 7654 MARTIN     SALESMAN   7698 28/09/81   1,250   1,400      30
 7698 BLAKE      MANAGER    7839 01/05/81   2,850              30
 7782 CLARK      MANAGER    7839 09/06/81   2,450              10
 7788 SCOTT      ANALYST    7566 09/12/82   3,000              20
 7839 KING       PRESIDENT       17/11/81   5,000              10
 7844 TURNER     SALESMAN   7698 08/09/81   1,500       0      30
 7876 ADAMS      CLERK      7788 12/01/83   1,100              20
 7900 JAMES      CLERK      7698 03/12/81     950              30
 7902 FORD       ANALYST    7566 03/12/81   3,000              20
 7934 MILLER     CLERK      7782 23/01/82   1,300              10

SQL> SELECT  *  FROM    dept

DEPTNO DNAME          LOC
------ -------------- -------------
    10 ACCOUNTING     NEW YORK
    20 RESEARCH       DALLAS
    30 SALES          CHICAGO
    40 OPERATIONS     BOSTON
```

Valores nulos

El valor nulo `NULL` representa la ausencia de información, o bien por desconocimiento del dato, o bien porque no procede.

Debe diferenciarse de cualquier otro valor, entre ellos del valor 0 si se trata de un dato numérico, y de la cadena de caracteres vacía, si es un dato de tipo carácter.

Una columna de una tabla podrá admitir valores nulos (`NULL`) o no (`NOT NULL`). Por defecto admite nulos.

Una forma de indicar la no admisión de valores nulos:

```
CREATE TABLE emp(
   empno    VARCHAR2(4) NOT NULL,
   ename    VARCHAR2(15) NOT NULL,
   job      VARCHAR2(9),
   mgr      VARCHAR2(4),
   hiredate DATE,
   sal      NUMBER(7,2),
   comm     NUMBER(7,2),
   deptno   VARCHAR2(2)
)
```

Incluyendo un valor nulo en una fila:

```
INSERT INTO emp VALUES('1245', 'José', 'Analista', NULL,
  '12-Jan-1997', 34000, 300.05, '12')


INSERT INTO emp VALUES('1245', 'José', 'Analista', NULL,
   TO_DATE('12/01/1997', 'dd/mm/yyyy'), 34000, 300.05, '12')
```

Lenguaje SQL

Expresiones

Una expresión es la formulación de una secuencia de operaciones, o sea, una combinación de operadores, operandos y paréntesis, que, cuando se ejecuta, devuelve un único valor como resultado.

Los operandos pueden ser constantes, nombres de columna, funciones, otras expresiones y otros elementos.

El tipo de dato de cada operando de una expresión debe ser el mismo. Si un operando es nulo, el resultado también es nulo.

Operadores numéricos: `+` `-` `*` `/`

Operador alfanumérico: `||`

Ejemplos:

```
3
'Casa'
3+2
'A'|| 'BC'
ENAME
SAL*1.5
0.5 * COMM
SAL + COMM
(Select COMM FROM EMP WHERE EMPNO = 7499)
```

Contraejemplos:

```
SAL < 1500

COMM < SAL / 3
```

Lenguaje SQL

La sentencia SELECT

La sentencia `SELECT` permite seleccionar u obtener datos de una o de varias tablas.

Parte de una o de varias tablas de la BD y el resultado es otra tabla, denominada a veces tabla resultado, pero que no formará parte de la BD – no queda almacenada en la BD–.

```
SELECT     [DISTINCT | ALL] {* | <expr1>[, <expr2>] ...}
FROM       <tabla1>[, <tabla2>, ...]
[WHERE     <condicion_where>]
[GROUP BY  <group_expr1>[, <group_expr2>, ...]
[HAVING    <condicion_having>]
[ORDER BY  <expr_orderby1 [ASC | DESC]>[, ...]]
```

La sentencia `SELECT` tiene varias cláusulas o partes diferentes con una función específica.

La sentencia `SELECT` básica está formada por las clásulas `SELECT`, `FROM`, `WHERE` y `ORDER BY`.

El orden de ejecución de las cláusulas y la función de cada una es:

1. `FROM` (obligatoria)
   Determina la tabla o tablas de la que se seleccionarán los datos.
2. `WHERE` (optativa)
   Indica un predicado que expresa la condición que debe cumplir cada fila que interviene en la consulta. Así la consulta se restringe a las filas que cumplen la condición.
3. `SELECT` (obligatoria)
   Incluye los datos que se solicitan en la consulta, normalmente una o varias expresiones. Alternativamente un `*` indica todas las columnas de las tablas involucradas. Si hubiese filas repetidas, por defecto aparecen, pero no lo hacen si se incluye `DISTINCT`.
4. `ORDER BY` (optativa)
   Permite determinar el criterio de ordenación de las filas de la tabla resultado. Sin ella obtendremos las mismas filas, pero puede que en órdenes distintos, según la estrategia seguida por el SGBD para extraer los datos.

Predicados elementales

Un predicado expresa una condición y como en BD relacionales existe una lógica de tres valores, verdadero, falso y nulo, la evaluación de una condición puede ser `TRUE`, `FALSE` o `NULL`.

Un predicado puede aparecer en una cláusula `WHERE` evaluando la condición de cada fila de las tablas involucradas, de forma que sólo las filas que cumplen la condición permanecen involucradas en la consulta, ignorando las restantes.

Los predicados más elementales son los de comparación, que comparan dos expresiones según un operador de comparación, que puede ser: `<` `<=` `=` `!=` `<>` `>=` `>`

Ejemplos iniciales

1.– Obtener todos los datos de la tabla de empleados:

```
SELECT      *
FROM        emp
```

2.– Obtener los códigos y nombres de los empleados que trabajan en el departamento 10:

```
SELECT      empno, ename
FROM        emp
WHERE       deptno = 10

EMPNO ENAME
----- ----------
 7782 CLARK
 7839 KING
 7934 MILLER
```

Ejemplos iniciales (cont.)

3.– Seleccionar todos los datos del departamento 40:

```
SELECT     *
FROM       dept
WHERE      deptno = 40

DEPTNO DNAME          LOC
------ -------------- -------------
    40 OPERATIONS     BOSTON
```

4.– Seleccionar los datos de los empleados del departamento 40:

```
SELECT     *
FROM       emp
WHERE      deptno = 40
```

5.– Obtener los datos de los departamentos ordenados por el nombre del departamento:

```
SELECT     *                                SELECT     *
FROM       dept                             FROM       dept
ORDER BY   dname                            ORDER BY   2

DEPTNO DNAME          LOC
------ -------------- ---------
    10 ACCOUNTING     NEW YORK
    40 OPERATIONS     BOSTON
    20 RESEARCH       DALLAS
    30 SALES          CHICAGO
```

Ejemplos iniciales (cont.)

6.– Obtener la comisión, departamento y nombre de los empleados cuyo salario sea inferior a 1.900, ordenándolos por departamento en orden creciente, y por comisión en orden decreciente dentro de cada departamento:

```
SELECT   comm AS COMISION, deptno AS DEPARTAMENTO, ename    NOMBRE
FROM     emp
WHERE    sal < 1900
ORDER BY deptno, comm DESC
```

```
  COMISION DEPARTAMENTO NOMBRE
---------- ------------ ----------
                    10 MILLER
                    20 SMITH
                    20 ADAMS
                    30 JAMES
      1400          30 MARTIN
       500          30 WARD
       300          30 ALLEN
         0          30 TURNER
```

Elementos de interés:

- Diversos criterios jerarquizados de ordenación.
- Asignación de nombres a las columnas del resultado.

Ejemplos iniciales (cont.)

7.– Hallar todas las combinaciones diferentes de valores de puesto de trabajo (JOB) y año de contratación en el departamento 30:

```
SELECT   job, TO_CHAR(hiredate,'yyyy') CONTRATADO
FROM     emp
WHERE    deptno = 30

JOB       CONT
--------- ----
SALESMAN  1981
SALESMAN  1981
SALESMAN  1981
MANAGER   1981
SALESMAN  1981
CLERK     1981


SELECT   DISTINC job, TO_CHAR(hiredate,'yyyy') CONTRATADO
FROM     emp
WHERE    deptno = 30

JOB       CONT
--------- ----
MANAGER   1981
SALESMAN  1981
CLERK     1981
```

- El `DISTINCT` afecta a toda la fila.

Ejemplos iniciales (cont.)

8.– Obtener las diferentes comisiones de los empleados cuyo salario es inferior a 1900, ordenándolas de forma decreciente.

```
SELECT   comm AS COMISION
FROM     emp
WHERE    sal < 1900
ORDER BY comm DESC

  COMISION
----------



      1400
       500
       300
         0

8 filas seleccionadas
```

```
SELECT   DISTINCT comm AS COMISION
FROM     emp
WHERE    sal < 1900
ORDER BY comm DESC

  COMISION
----------

      1400
       500
       300
         0

5 filas seleccionadas
```

- Efecto del `NULL` en el `DISTINCT` (en este caso dos nulos se consideran *iguales*).

Predicados

Los predicados expresan condiciones.

Si aparecen en una cláusula `WHERE` indican una condición que las filas de las tablas involucradas deben cumplir.

Pueden aparecer también en una cláusula `HAVING`, como veremos posteriormente.

Pueden ser simples, si incluyen una única condición, o compuestos si constan de varias, o sea, si combinan varios predicados simples, unidos por los operadores `AND`, `OR` y `NOT`.

Predicados simples

Inicialmente veremos los predicados simples en su formato más elemental. Son los siguientes:

- De comparación.
  Comparan dos expresiones según un operador de comparación, que puede ser: `< <= = != <> >= >`

```
SELECT   *
FROM     emp
WHERE    sal <= 1900
```

- De valor nulo.
  Los predicados de comparación no sirven para determinar los valores nulos. Por ejemplo, no es válido `COMM = NULL` porque sería discernir si un valor que puede ser desconocido es igual a otro también desconocido.
  Se requiere un predicado especial, con formato: `<expr> IS [NOT] NULL`

```
SELECT   *
FROM     emp
WHERE    comm IS NULL
```

```
SELECT   *
FROM     emp
WHERE    comm IS NOT NULL
```

Lenguaje SQL

Predicados simples (cont.)

- De rango (`BETWEEN`).
  Compara si los valores de una expresión están o no entre los de otras dos.
  Formato: `<expr1> [NOT] BETWEEN <expr2> AND <expr3>`
  Expresividad: `e1 BETWEEN e2 AND e3` $\equiv$ `(e1 >= e2) AND (e1 <= e3)`

```
SELECT * FROM emp WHERE sal BETWEEN 1500 AND 3000
```

- De pertenencia a un conjunto (`IN`).
  Comprueba si el valor de una expresión coincide con alguno de los incluidos en una lista de valores.
  Formato: `expr [NOT] IN (expr1[, expr2, ...])`
  Expresividad: `e IN (a, b)` $\equiv$ `(e = a) OR (e = b)`

```
SELECT   *                          SELECT ename
FROM     emp                        FROM   emp
WHERE    deptno IN (10, 30)         WHERE  job IN ('CLERK','SALESMAN')
```

- De correspondencia con un patrón o modelo (`LIKE`).
  Comprueba si el valor de una expresión alfanumérica se corresponde con un modelo.
  El modelo puede incluir dos caracteres que actúan como comodines:
  `_` Indica un único carácter, incluido el blanco.
  `%` Indica una cadena de caracteres de cualquier longitud, incluida la cadena vacía.
  Formato: `expr_cart [NOT] LIKE modelo`

```
SELECT   *                          SELECT   *
FROM     emp                        FROM     emp
WHERE    ename LIKE '%NE%'          WHERE    ename LIKE '_____'
```

Predicados compuestos

Son la unión de dos o más predicados simples mediante los operadores lógicos `AND`, `OR` y `NOT`.

Al existir una lógica de tres valores, debemos considerar el efecto del valor `NULL`:

X	Y	X AND Y	X OR Y	NOT X
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	NULL	NULL	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	NULL	FALSE	NULL	TRUE
NULL	TRUE	NULL	TRUE	NULL
NULL	FALSE	FALSE	NULL	NULL
NULL	NULL	NULL	NULL	NULL

a) Obtener los nombres, salarios y fechas de contratación de los empleados que, o bien ingresaron después de 1-6-81, o bien tienen un salario inferior a 1.500. Clasificar los resultados por fecha y nombre.

b) Nombre de los empleados que ganan más de 2.500 en total (salario más comisión).

a)
```
SELECT   ename, sal, hiredate
FROM     emp
WHERE    hiredate > '1-jun-1981'
   OR    sal < 1500
ORDER BY 3,1
```

b)
```
SELECT  ename
FROM    emp
WHERE   sal > 2500
   OR   sal + comm > 2500
```

Composición de consultas con operadores conjuntistas

SQL proporciona tres operadores conjuntistas: unión (`UNION`), intersección (`INTERSECT`) y diferencia (`EXCEPT`). Permiten realizar las operaciones de conjuntos sobre el resultado de dos o más consultas.

El formato simplificado es:

```
consulta1
{UNION | INTERSECT | EXCEPT}  [ALL | DISTINCT]
consulta2
```

Las consultas deben tener una estructura compatible – *compatibles para la unión* –, lo que implica, al menos, el mismo número de elementos y la correspondencia entre tipos de datos de una consulta y de la otra. Estas condiciones pueden exigerse con más o menos rigurosidad por los fabricantes.

Los términos `ALL` y `DISTINCT` hacen que aparezcan o no las filas repetidas. El defectivo es el `DISTINCT`.

Oracle sustituye `EXCEPT` por `MINUS`.

```
SELECT   ename, empno
FROM     emp
WHERE    sal > 3000
UNION
SELECT   dname, deptno
FROM     dept
ORDER BY 1
```

```
ENAME          EMPNO
-------------- -----
ACCOUNTING        10
KING            7839
OPERATIONS        40
RESEARCH          20
SALES             30
```

- Efecto del `ORDER BY`.

Lenguaje SQL

Funciones

Pueden formar parte de una expresión. Constan de un nombre y pueden tener uno o más argumentos entre paréntesis. El resultado es un valor.

Hay muchas funciones y pueden clasificarse de diferentes formas. Un breve lista:

- Numéricas o aritméticas:
  - `SQRT(<exp_numérica>)` Raíz cuadrada. Ej.: `SQRT(81)`
  - `ABS(<exp_numérica>)` Valor absoluto. Ej.: `ABS(-11)`
  - `POWER(<exp1>,<exp2>)` Potencia. Ej.: `POWER(9,2) = 81`

- Alfanuméricas o de cadenas de caracteres:
  - `SUBSTR(<exp1>,<exp2>[,<exp3>])` Subcadena de `<exp1>` empezando en la posición `<exp2>` y de longitud `<exp2>`.
    Ej.: `SUBSTR('Materia',2,4) = 'ater'` `SUBSTR('Materia',5) = 'ria'`
  - `UPPER(<exp_carácter>)` Pasa a mayúsculas. Ej.: `UPPER('Materia') = 'MATERIA'`
  - `LOWER(<exp_carácter>)` Pasa a minúsculas. Ej.: `LOWER('Materia') = 'materia'`

- De fecha y tiempo:
  - `CURRENT_DATE` Fecha actual del sistema. Ej.: `SELECT CURRENT_DATE FROM DUAL`

- Otras (específicas de Oracle):
  - `NVL(<exp1>,<exp2>)` Asignación del valor de `<exp2>` a la expresión `<exp1>` si ésta es nula.
    Ej.: `NVL(sal + cmm, sal)`
  - `DECODE` Asignación condicional de resultados.

```
DECODE(<exp>,<valor1>,<resultado1>
          [, <valor2>,<resultado2>, ...]
          [,<resultado_en_otro_caso>]
```

```
DECODE(DEPTNO, '10','CONTABILIDADE',
               '30','VENTAS')
```

Funciones de agrupamiento o colectivas

Otros nombres: agregadas, de columna, ...

Las funciones vistas hasta ahora obtienen un valor para cada fila de la tabla sobre la que se realiza la consulta:

```
SELECT   SQRT(sal)          ==>  Para cada fila, obtiene la raíz cuadrada
FROM     emp                     del salario de cada empleado
```

Las funciones de agrupamiento obtienen un valor que hace referencia a un conjunto de filas:

```
SELECT   MAX(sal)           ==>  Obtiene una única fila con el máximo
FROM     emp                     de los salarios de los empleados
```

Formato: `<func>(<expr>)`

Muchas permiten los términos `ALL` y `DISTINCT`: `<func>([ALL|DISTINCT]<expr>)`.

Si aparece `DISTINCT` se eliminan los valores repetidos del argumento, antes de calcular la función.

Las más frecuentes son:

`AVG`	Media	`COUNT`	Número de valores
`MAX`	Máximo	`MIN`	Mínimo
`SUM`	Suma	`VAR`	Varianza

Si se incluye una función de agrupamiento en la cláusula `SELECT` todas las expresiones en dicha cláusula deben tener un valor único para el conjunto de filas.

Lenguaje SQL

Funciones de agrupamiento o colectivas (cont.)

1.– Obtén la media de los salarios del departamento 30.

```
SELECT      AVG(sal)
FROM        emp
WHERE       deptno = 30
```

El conjunto es el formado por las filas del departamento 30.

2.– Obtén la media de los salarios y el nombre alfabéticamente más alto de los empleados cuyo salario es mayor que 1500.

```
SELECT      AVG(sal), MAX(ename)
FROM        emp
WHERE       sal > 1500
```

El conjunto es el formado por las filas con salario mayor que 1500.

3.– Obtén la media de los salarios de los empleados.

```
SELECT      AVG(sal)
FROM        emp
```

¿Cuál es el conjunto?

4.– !?

```
SELECT      AVG(sal)
FROM        emp
WHERE       sal > MAX(sal)
```

¿Cuál es el conjunto?

5.– !?

```
SELECT      MAX(sal), sal
FROM        emp
```

¿Cuál es el problema?

Funciones de agrupamiento o colectivas (cont.)

Formato de COUNT:

La función COUNT permite un formato especial, quedando de la forma:

- COUNT([ALL] <columna>) Devuelve el número de filas con valores diferentes a NULL en la columna <columna>.
- COUNT(DISTINCT <columna>) Devuelve el número de filas con valores distintos y diferentes a NULL en la columna <columna>.
- COUNT(*) Obtiene el número de filas de la tabla.

Ej.: Dada la tabla t1 con una columna C y cuatro filas:

```
C
----
NULL
1
2
2
```

la siguiente consulta, obtendrá:

```
SELECT COUNT(*), COUNT(c), COUNT(ALL c), COUNT(DISTINCT c)
FROM   t1

  COUNT(*)   COUNT(C) COUNT(ALLC) COUNT(DISTINCTC)
---------- ---------- ---------- ----------------
         4          3          3                2
```

Lenguaje SQL

Cláusula `GROUP BY`

Hemos visto las funciones de agrupamiento actuando sobre un conjunto formado por la totalidad de las filas involucradas en la consulta.

En SQL pueden formarse diversos conjuntos que vedrán dados por la igualdad en los valores de una o más columnas, denominadas columnas de agrupamiento, que se incluyen en una cláusula específica, `GROUP BY`, que actúa después de la `WHERE`..

Formato: `GROUP BY <columna1>[, <columna2>, ...]`

Una serie de filas forman un conjunto si tienen los mismos valores para una o más columnas.

Si consideramos que la última columna es la de agrupamiento:

...	8	a
...	5	b
...	4	a
...	7	a
...	2	c
...	1	b

$\Rightarrow$

...	8	a
...	4	a
...	7	a

...	5	b
...	1	b

...	2	c

Cláusula `GROUP BY` (cont.)

– Si deseamos obtener el máximo del salario de los empleados para cada departamento, tendremos que formar tantos conjuntos como departamentos distintos que tienen empleados haya, agrupar en cada conjunto a los empleados del mismo departamento, y obtener el máximo del salario para cada uno de los conjuntos.

Lo que se pide es la función de agrupamiento `MAX(sal)` sobre cada conjunto, siendo el criterio de formación de los conjuntos la igualdad en la columna del código del departamento:

```
SELECT   MAX(sal), deptno
FROM     emp
GROUP BY deptno
```

```
  MAX(SAL) DEPTNO
---------- ------
      5000     10
      3000     20
      2850     30
```

Si se incluye una cláusula `GROUP BY` sólo se permiten en la cláusula `SELECT` expresiones que tengan un valor único para cada conjunto formado, esto es, que incluyan constantes, columnas de agrupamiento, funciones de agrupamiento o combinaciones de ellas:

```
SELECT   MAX(sal), deptno, empno
FROM     emp
GROUP BY deptno
```

```
!?
```

Cláusula GROUP BY (cont.)

1.– Obtén el máximo, mínimo y la suma de los salarios de cada departamento, además del número de empleados de cada departamento:

```
SELECT     MAX(sal), MIN(sal), SUM(sal), COUNT(empno), deptno
FROM       emp
GROUP BY   deptno

  MAX(SAL)   MIN(SAL)   SUM(SAL) COUNT(EMPNO) DEPTNO
---------- ---------- ---------- ------------ ------
      5000       1300       8750            3     10
      3000        800      10875            5     20
      2850        950       9400            6     30
```

Los conjuntos se forman después del efecto de la cláusula WHERE:

2.– Obtén el máximo, mínimo y la suma de los salarios de cada departamento, además del número de empleados de cada departamento, considerando los empleados cuyo salario es superior a 1500:

```
SELECT     MAX(sal), MIN(sal), SUM(sal), COUNT(empno), deptno
FROM       emp
WHERE      sal > 1500
GROUP BY   deptno

  MAX(SAL)   MIN(SAL)   SUM(SAL) COUNT(EMPNO) DEPTNO
---------- ---------- ---------- ------------ ------
      5000       2450       7450            2     10
      3000       2975       8975            3     20
      2850       1600       4450            2     30
```

Cláusula GROUP BY (cont.)

3.– Obtén los diferentes empleos que hay para cada departamento, incluyendo cuantos empleados lo desempeñan en cada departamento, ordenando por empleo y departamento:

```
SELECT     job, deptno, COUNT(*)
FROM       emp
GROUP BY   deptno, job
ORDER BY   job, deptno

JOB       DEPTNO   COUNT(*)
--------- ------ ----------
ANALYST       20          2
CLERK         10          1
CLERK         20          2
CLERK         30          1
MANAGER       10          1
MANAGER       20          1
MANAGER       30          1
PRESIDENT     10          1
SALESMAN      30          4
```

El mismo resultado se obtendría con:

```
SELECT     job, deptno, COUNT(*)
FROM       emp
GROUP BY   job, deptno
ORDER BY   job, deptno
```

Condiciones sobre los conjuntos

De la misma forma que la cláusula `WHERE` indica una condición sobre las filas, la cláusula `HAVING` permite establecer una condición sobre los conjuntos. Sólo los conjuntos que cumplan la condición del `HAVING` seguirán formando parte de la consulta.

La condición impuesta en el `HAVING` tiene que ser aplicable a cada conjunto formado por la acción de la cláusula `GROUP BY`.

```
SELECT     MAX(sal), MIN(sal), SUM(sal), COUNT(empno), deptno
FROM       emp
WHERE      sal > 1500
GROUP BY   deptno

  MAX(SAL)   MIN(SAL)   SUM(SAL) COUNT(EMPNO) DEPTNO
---------- ---------- ---------- ------------ ------
      5000       2450       7450            2     10
      3000       2975       8975            3     20
      2850       1600       4450            2     30


SELECT     MAX(sal), MIN(sal), SUM(sal), COUNT(empno), deptno
FROM       emp
WHERE      sal > 1500
GROUP BY   deptno
HAVING     MIN(sal) > 1600

  MAX(SAL)   MIN(SAL)   SUM(SAL) COUNT(EMPNO) DEPTNO
---------- ---------- ---------- ------------ ------
      5000       2450       7450            2     10
      3000       2975       8975            3     20
```

Condiciones sobre los conjuntos (cont.)

Para que la condición impuesta en el `HAVING` sea aplicable a cada conjunto, en dicha cláusula sólo se permiten expresiones que incluyan constantes, columnas de agrupamiento y funciones de agrupamiento:

```
SELECT   MAX(sal), deptno
FROM     emp
GROUP BY deptno
HAVING   COUNT(*) + 1 > 4
```

```
  MAX(SAL) DEPTNO
---------- ------
      3000     20
      2850     30
```

```
SELECT   MAX(sal), deptno
FROM     emp
GROUP BY deptno
HAVING   COUNT(*) > 3
```

Particularmente importante es distinguir si una condición se debe reflejar en la cláusula `WHERE` o en la `HAVING`:

- La condición del `WHERE` se aplica a cada fila de la tabla o tablas determinadas en la cláusula `FROM`.
- La condición del `HAVING` se aplica a cada conjunto de filas generado a partir de la cláusula `GROUP BY`, si existe.

Por tanto las expresiones de cada cláusula deben ser computables sobre lo que van dirigido; a filas individuales para la `WHERE`, y a conjuntos para la `HAVING`.

Condiciones sobre los conjuntos (cont.)

Los nulos resultantes en las funciones de agrupamiento se consideran *similares*:

```
SELECT   MAX(sal), deptno, comm
FROM     emp
GROUP BY deptno, comm

  MAX(SAL) DEPTNO    COMM
---------- ------ -------
      5000     10
      3000     20
      2850     30
      1500     30       0
      1600     30     300
      1250     30     500
      1250     30   1,400
```

Y si el predicado de una cláusula `HAVING` es nulo para un conjunto dado, dicho conjunto ya no se tiene en cuenta en la consulta:

```
SELECT   MAX(sal), deptno, comm
FROM     emp
GROUP BY deptno, comm
HAVING   comm > 0

  MAX(SAL) DEPTNO    COMM
---------- ------ -------
      1600     30     300
      1250     30     500
      1250     30   1,400
```

Orden de ejecución de las cláusulas

La sentencia `SELECT` completa está formada por las clásulas `SELECT`, `FROM`, `WHERE`, `GROUP BY` `HAVING` y `ORDER BY`.

El orden de ejecución de las cláusulas y la función de cada una es:

1. `FROM` (obligatoria)
   Determina la tabla o tablas de la que se seleccionarán los datos.
2. `WHERE` (optativa)
   Indica un predicado que expresa la condición que debe cumplir cada fila que interviene en la consulta. Así la consulta se restringe a las filas que cumplen la condición.
3. `GROUP BY` (optativa)
   Incluye las columnas que determinan la formación de conjuntos según la igualdad de valores en dichas columnas.
4. `HAVING` (optativa)
   Indica un predicado que expresa la condición que debe cumplir cada conjunto que interviene en la consulta.
5. `SELECT` (obligatoria)
   Incluye los datos que se solicitan en la consulta, normalmente una o varias expresiones. Alternativamente un * indica todas las columnas de las tablas involucradas. Si hubiese filas repetidas, por defecto aparecen, pero no lo hacen si se incluye `DISTINCT`.
6. `ORDER BY` (optativa)
   Permite determinar el criterio de ordenación de las filas de la tabla resultado. Sin ella obtendremos las mismas filas, pero puede que en órdenes distintos, según la estrategia seguida por el SGBD para extraer los datos.

Subconsultas

En las consultas anteriores las condiciones para seleccionar las filas implicaban, generalmente, a elementos conocidos.

```
... WHERE  sal > 1200 ...                    ... WHERE ename LIKE 'A%' ...
```

Podemos expresar condiciones en las que algún elemento sea desconocido, obteniéndolo previamente con una consulta, o sea, incluir una consulta dentro de otra consulta.

Ej.: Obtener el salario de los empleados cuyo salario es mayor que el del empleado de código 7900.

```
SELECT   empno, ename, sal
FROM     emp
WHERE    sal >   (SELECT sal
                  FROM   emp
                  WHERE  empno = 7900)
```

Subconsulta: una sentencia `SELECT` que aparece en un predicado de otra consulta.

Puede haber subconsultas que lo son, a su vez, de otras subconsultas.

La consulta inicial se conoce como consulta principal.

El resultado de la subconsulta lo utiliza la consulta de nivel superior en algún predicado, pero no aparece como resultado de esa consulta.

Subconsultas (cont.)

En general, si se presenta una subconsulta se comprueba el predicado para cada fila de la tabla, o para cada conjunto en su caso, por lo que se ejecuta la sunconsulta y se sustituye por su resultado.

Una subconsulta suele tener en su cláusula SELECT una expresión, aunque las últimas estandarizaciones permiten que aparezca más de una expresión.

Considerando el caso de una única expresión en la subconsulta, todavía su resultado puede ser:

- Sólo una fila:

```
SELECT   empno, ename, sal                    SELECT   empno, ename, sal
FROM     emp                                  FROM     emp
WHERE    sal > (SELECT sal                    WHERE    sal >= (SELECT AVG(sal)
                FROM   emp                                    FROM   emp)
                WHERE  empno = 7900)
```

- Varias filas:

```
SELECT   empno, ename, sal                    SELECT   *       !?
FROM     emp                                  FROM     emp
WHERE    deptno IN (SELECT deptno             WHERE    sal = (SELECT sal
                    FROM   dept                                FROM   emp
                    WHERE  loc = 'DALLAS'                      WHERE  deptno = 20)
                       OR  loc = 'CHICAGO')
```

- Ninguna fila:

```
SELECT *  FROM emp  WHERE sal >= (SELECT AVG(sal) FROM emp  WHERE deptno = 40)
```

Subconsultas (cont.)

Los predicados que pueden utilizarse en una subconsulta deben tener en cuenta el posible número de filas esperado en ella.

Considerando todavía el caso de una única expresión en la subconsulta, si esperamos que el resultado tenga más de una fila, no podemos usar un predicado de comparación, pero es frecuente, por ejemplo, la utilización de predicados de pertenencia a un conjunto. Pueden usarse también predicados con operadores cuantificados:

Operadores cuantificados

Se trata de los Operadores ANY o SOME y ALL, que comprueban una expresión escalar con el resultado del operador sobre una subconsulta.

```
... expresión_escalar operador_comparación { ALL | SOME | ANY } (subconsulta)
```

Operadores de comparación:`<` `<=` `=` `!=` `<>` `>=` `>`

`ANY` `SOME`: La expresión se compara con cada uno de los valores de la subconsulta y si para alguno es verdadera, el resultado es verdadero.

`ALL`: La expresión se compara con cada uno de los valores de la subconsulta y si para todos es verdadera, el resultado es verdadero.

```
SELECT *
FROM   emp
WHERE  sal = ANY (SELECT sal
                  FROM   emp
                  WHERE  deptno = 30)
```

```
SELECT *
FROM   emp
WHERE  sal > ALL (SELECT sal
                  FROM   emp
                  WHERE  deptno = 30)
```

Subconsultas (cont.)

Ya que una subconsulta puede devolver o no filas, existe un predicado para realizar dicha comprobación:

Predicado EXISTS

Comprueba si una subconsulta devuelve o no filas.

```
... [NOT]  EXISTS  subconsulta
```

Ej.: Si existen empleados del departamento 30, obtener los datos de todos los departamentos.

```
SELECT    *
FROM      dept d
WHERE     EXISTS (SELECT *
                  FROM   emp
                  WHERE  deptno = 30)
```

```
SELECT    *
FROM      dept d
WHERE     EXISTS (SELECT empno
                  FROM   emp
                  WHERE  deptno = 30)
```

- En la subconsulta que aparece en un predicado EXISTS pueden aparecer varias expresiones.

Este predicado requiere subconsultas más complejas para obtener toda su potencia.

Subconsultas correlacionadas o sincronizadas

Hasta ahora las subconsultas tenían todos los elementos para realizarlas, siendo por tanto consultas totalmente independientes de la consulta principal.

Ej.: Obtén los datos de los empleados cuyo salario supera la media de los salarios de la empresa.

```
SELECT    *
FROM      emp
WHERE     sal > (SELECT AVG(sal)
                 FROM   emp)
```

En ocasiones una subconsulta incluye referencias a alguna columna de una tabla que es recorrida por la consulta principal.

En estos casos decimos que la subconsulta es correlacionada o sincronizada, ya que para cada fila de la tabla que recorre la consulta principal, debe ejecutarse la subconsulta.

Suele necesitarse la incorporación de alias de las tablas.

Ej.: Obtén los datos de los empleados cuyo salario supera la media de los salarios de su departamento.

```
SELECT    *
FROM      emp a
WHERE     sal > (SELECT AVG(sal)
                 FROM   emp
                 WHERE  deptno = a.deptno)
```

Lenguaje SQL

JOIN

Permite obtener como resultado de una consulta datos de más de una tabla, vinculando las columnas de varias tablas mediante operadores de comparación.

Sintaxis:

```
SELECT *
FROM   t1, t2
WHERE  condición_join
...
```

```
SELECT *
FROM   t1 [INNER] JOIN t2
  ON   condición_join
WHERE  ...
```

La vinculación de las columnas se determina en la condición de join y usando el `INNER JOIN` aparecen las que cumplen esa condición.

Ejemplos:

```
SELECT *
FROM   emp, dept
WHERE  emp.deptno = dept.deptno
AND    ename LIKE 'A%'
```

```
SELECT *
FROM   emp INNER JOIN dept
  ON   emp.deptno = dept.deptno
WHERE  ename LIKE 'A%'
```

```
SELECT a.empno, a.ename, a.sal, a.comm
FROM   emp a  INNER JOIN  emp b
  ON   a.sal > b.sal
WHERE  b.empno = 7844
```

OUTER JOIN

Permite vincular las columnas de varias tablas mediante operadores de comparación, haciendo que aparezcan todas las columnas de una tabla, de la otra, o de ambas, cumplan o no la condición de join.

Sean dos relaciones `R` y `S`. Tenemos las operaciones:

- Left outer join de `R` y `S` ($R ⟕ S$): Todas las filas de la relación de la izquierda (`R`), rellenando con nulos en las filas que no se correspondan con las de `S`.
- Right outer join de `R` y `S` ($R ⟖ S$): Todas las filas de la relación de la derecha (`S`), rellenando con nulos en las filas que no se correspondan con las de `R`.
- Full outer join de `R` y `S` ($R ⟗ S$): Todas las filas de la relación de la izquierda (`R`), y todas las filas de la relación de la derecha (`S`), rellenando con nulos en las filas que no se correspondan con las de la otra.

```
SELECT *
FROM   r {LEFT | RIGHT | FULL} [OUTER] JOIN s
  ON   condición_join
```

```
SELECT *
FROM   r LEFT OUTER JOIN s
  ON   r.c2 = s.c2
```

Múltiples joins

El orden de ejecución es de izquierda a derecha (asociativos a la izquierda).

Equivalencias:

```
SELECT expresion, ...
FROM  t1  JOIN t2
  ON condición_join1
  JOIN t3
  ON condición_join2
  ...
WHERE predicado
```

```
SELECT expresion, ...
FROM  (t1  JOIN t2
  ON condición_join1)
  JOIN t3
  ON condición_join2
  ...
WHERE predicado
```

Modificación del orden:

```
SELECT expresion, ...
FROM  t1  JOIN
  (t2  JOIN tb3
  ON condición_join2)
  ON condición_join1
  ...
WHERE predicado
```

Expresiones con SELECT escalar

Un SELECT que obtenga una fila y una columna, denominado SELECT escalar, puede formar parte de una expresión.

Ejemplo:

```
SELECT *
FROM   emp
WHERE  (SELECT sal    FROM emp WHERE ename LIKE 'KING') - 3000  < sal

EMPNO ENAME      JOB         MGR HIREDATE     SAL    COMM DEPTNO
----- ---------- --------- ----- -------- ------- ------- ------
 7566 JONES      MANAGER    7839 02/04/81   2,975              20
 7698 BLAKE      MANAGER    7839 01/05/81   2,850              30
 7782 CLARK      MANAGER    7839 09/06/81   2,450              10
 7788 SCOTT      ANALYST    7566 09/12/82   3,000              20
 7839 KING       PRESIDENT       17/11/81   5,000              10
 7902 FORD       ANALYST    7566 03/12/81   3,000              20
```

Posibilidades:

- WHERE, HAVING
- SELECT

Efecto en cada caso

Expresión de consulta

Se denomina expresión de consulta o de tabla derivada o de tabla anidada a la utilización de una consulta en las cláusulas `SELECT` o `FROM` de otra.

Sintaxis:

```
SELECT expresión, ..., (SELECT ...)
FROM   t1, ...
 ...

SELECT expresión, ...
FROM   t1,  (SELECT ...)
...
```

Permite que en el resultado de una consulta aparezcan datos correspondientes a elementos diferentes.

Ejemplo:

```
SELECT empno, ename, sal, (SELECT MAX(sal) FROM emp) - sal "Diferencia"
FROM emp
```

```
EMPNO ENAME          SAL Diferencia
----- ---------- ------- ----------
 7369 SMITH          800       4200
 7499 ALLEN        1,600       3400
. . .
```

Expresión de consulta (cont.)

Ejemplos: Para los departamentos 10 y 20, hallar el valor medio de los salarios medios de cada departamento.

```
SELECT AVG(PREMED) AS MEDIA
FROM   (SELECT AVG(sal) AS PREMED
        FROM   emp
        WHERE  deptno IN (10,20)
        GROUP BY deptno)

    MEDIA
----------
2545,83333
```

Pero podría hacerse igualmente de la forma:

```
SELECT AVG(AVG(sal)) AS MEDIA
FROM   emp
WHERE  deptno IN (10,20)
GROUP BY deptno

    MEDIA
----------
2545,83333
```

Subconsultas de fila

Se trata de subconsultas que devuelven más de una columna.

Permite realizar operaciones de comparación simultáneamente sobre cada columna que aparece en su resultado.

Sintaxis:

```
SELECT expresión_a, ...
FROM tabla_1, ...
WHERE (expresión_p1, ..., expresión_pn) operador (SELECT expresión_s1, ..., expresión_sn
                                                  FROM ...
                                                  WHERE ...)
```

Operadores válidos: comparación, `IN`, `= SOME`, `>ALL`, ...

Operadores válidos en Oracle 9.2: `IN`, `= SOME`, `= ALL`

Ejemplo:

```
SELECT *
FROM   articulo
WHERE  (cd_articulo, precio_min) IN
                                    (SELECT cd_articulo, precio
                                     FROM   ventas)
```

Otras sentencias DML: INSERT

Permite introducir filas en una tabla. Evidentemente aquí *añadir* no significa colocar las nuevas final al final de las ya existentes.

Formato:

```
INSERT INTO <tabla> [(<columna1>, <columna2>, ...)]
{ VALUES  (<valor columna1>, <valor columna2>, ...) / <sentencia SELECT> }
```

Ejemplos:

1.– Añade una fila a tabla `emp` con datos en todas sus columnas, incluido un valor nulo.

```
INSERT INTO emp
VALUES  (7777, 'ANA', 'ANALISTA', 7369, '15/01/05', 2500, NULL, 10)
```

2.– Añade una fila a la tabla con datos en las columnas referenciadas en la lista.

```
INSERT INTO emp (empno, ename, sal)
VALUES  (8888, 'Juan', 2000)
```

3.– Añade filas con la sentencia `SELECT`:

```
INSERT INTO emp_new
   SELECT empno, ename, sal, comm
   FROM   emp_old
   WHERE  deptno = 10
   AND    sal    > 1500
```

Lenguaje SQL

Otras sentencias DML: UPDATE

Permite modificar datos de columnas en filas existentes.

Formato:

```
UPDATE <tabla>
SET <columna1> = <expresión1> [, <columna2> = <expresión2>, ...]
[ WHERE <predicado> ]
```

Ejemplos:

1.– Modifica el valor de la columna `sal` en todas las filas de la tabla.

```
UPDATE  emp
SET  sal = sal + 100
```

2.– Modifica el valor de varias columnas para las filas que cumplen la condición.

```
UPDATE  emp
SET     comm = 110, deptno = 10
WHERE   comm IS NULL
```

3.– Modificación usando un predicado con un `SELECT`.

```
UPDATE  emp
SET     sal = 2000
WHERE   sal = (SELECT MIN(sal)
               FROM   emp a
               WHERE  a.deptno = emp.deptno)
```

Otras sentencias DML: DELETE

Permite borrar filas de una tabla.

Formato:

```
DELETE FROM <tabla>
[ WHERE <predicado> ]
```

Ejemplos:

1.– Borra las filas que cumplen la condición indicada.

```
DELETE  FROM  emp
WHERE   sal > 2000
```

2.– Si no se pone ninguna condición, o sea, si no existe subcláusula `WHERE`, el borrado afecta a todas las filas de la tabla.

```
DELETE  FROM  emp
```

Tengamos en cuenta que al aparecer un predicado, puede ocurrir que éste contenga una sentencia `SELECT`, como en el caso anterior.

Lenguaje SQL

Sentencias DDL: CREATE TABLE

Crea la estructura de una tabla.

Formato simplificado:

```
CREATE TABLE <tabla>
 ( <definición de columna 1>,
   ...,
   <definición de columna n>,
   <definición de restricción 1>,
   ...,
   <definición de restricción n>)
```

En la definición del atributo o columna puede aparecer:

```
<nombre de columna> { <tipo de dato> / <dominio> } [<restricción de columna> ...]
```

Las restricciones se usan para declarar en el modelo relacional las condiciones que los datos cumplen en el mundo real.

Pueden ser de diverso tipo (valor defectivo, valor nulo, clave primaria, clave foránea o ajena, de unicidad, de comprobación de una condición, etc.):

- Valor defectivo de una columna. Se expresa con `DEFAULT <valor>`
- Valor nulo (defectivo) o no nulo de una columna. Se expresa con `[NOT] NULL`
  Ejemplos:

```
CREATE TABLE emp1
   (empno  NUMBER(4) NOT NULL,
    ename  VARCHAR2(15),
    mgr    NUMBER(4) DEFAULT 7500,
    deptno NUMBER(2))
```

Sentencias DDL: CREATE TABLE (cont.)

Otras restricciones:

- De clave primaria.
  Indica la condición de que un conjunto de columnas toma valores diferentes para cada fila y ninguno de ellos es nulo. Se expresa mediante `PRIMARY KEY`

```
CREATE TABLE emp2
   (empno  NUMBER(4) PRIMARY KEY,
    ename  VARCHAR2(15),
    mgr    NUMBER(4)
    deptno NUMBER(2))
```

"de columna"

```
CREATE TABLE emp2
  (empno  NUMBER(4),
   ename  VARCHAR2(15),
   mgr    NUMBER(4)
   deptno NUMBER(2),
   PRIMARY KEY (empno))
```

"de tabla"

  Toda restricción puede nombrarse incluyendo `CONSTRAINT <nombre de restricción>`

  Aquí sería anteponiéndolo a `PRIMARY KEY`:

```
CREATE TABLE emp2
   (empno  NUMBER(4)
      CONSTRAINT cp_emp2 PRIMARY KEY,
    ename  VARCHAR2(15),
    mgr    NUMBER(4)
    deptno NUMBER(2))
```

```
CREATE TABLE emp2
  (empno  NUMBER(4),
   ename  VARCHAR2(15),
   mgr    NUMBER(4),
   deptno NUMBER(2),
   CONSTRAINT cp_emp2 PRIMARY KEY (empno))
```

Sentencias DDL: CREATE TABLE (cont.)

Otras restricciones:

- De unicidad.
  Indica que un conjunto de atributos no puede tener valores iguales en filas distintas.

  Formato:

```
<definición de columna > UNIQUE,

UNIQUE (<columna 1>[, <columna 2>, ...])

[CONSTRAINT <nombre de restricción>] UNIQUE (<columna 1>[, ...])
```

  Ejemplo:

```
CREATE TABLE dept1
   (deptno NUMBER(2) PRIMARY KEY,
    dname  VARCHAR2(14) UNIQUE,
    loc    VARCHAR2(13))
```

```
CREATE TABLE dept1
  (deptno NUMBER(2) PRIMARY KEY,
   dname  VARCHAR2(14),
   loc    VARCHAR2(13),
   UNIQUE (dname))
```

- De comprobación o de CHECK.
  Permiten declarar una condición que debe cumplir uno o más atributos.

  Formato:

```
<definición de columna > [CONSTRAINT <nombre de restricción>] CHECK <condición>
[CONSTRAINT <nombre de restricción>] CHECK <condición>
```

  Ejemplo:

```
... SAL NUMBER(7,2) CHECK(sal > 750),
... , CHECK(mgr > 7050  OR  deptno = 10)
```

Sentencias DDL: CREATE TABLE (cont.)

Otras restricciones:

- De clave foránea o ajena.
  Un conjunto de atributos es una clave foránea si sus valores se corresponden con los de otro conjunto de atributos que es clave candidata en *otra* relación.

  Se dice que la clave ajena *referencia* a la clave primaria.

  Oracle permite la definiciónd e claves ajenas sobre conjuntos de atributos que referencian a una clave primaria o a un conunto de atributos con valor único.

```
Tabla EMP2
empno  ename      mgr   deptno
-----  --------   ----  ------
 7500  Ana                 20
 7710  Juan       7500      10
 7520  Alvaro     7710      10
```

```
CREATE TABLE emp2
   (empno  NUMBER(4)
      CONSTRAINT cp_emp2 PRIMARY KEY,
    ename  VARCHAR2(15),
    mgr    NUMBER(4),
    deptno NUMBER(2)
      REFERENCES dept2(deptno))
```

```
Tabla DEPT2
deptno  dname          loc
------  ------------   ---------
    10  Ventas         Barcelona
    20  Contabilidad   Valencia
    30  Publicidad     Madrid
```

```
CREATE TABLE emp2
  (empno  NUMBER(4),
   ename  VARCHAR2(15),
   mgr    NUMBER(4),
   deptno NUMBER(2),
   CONSTRAINT cp_emp2 PRIMARY KEY (empno),
   FOREIGN KEY (deptno)
     REFERENCES dept2(deptno))
```

Sentencias DDL: CREATE TABLE (cont.)

Otras restricciones (continuación de clave foránea):

- De clave foránea o ajena (cont.)

  Para nombrarla, antes de `REFERENCES` y de `FOREIGN KEY`, respectivamente, se coloca
  `CONSTRAINT <nombre de restricción>`

```
deptno NUMBER(2) CONSTRAINT cf_emp2_deptno  REFERENCES dept2(deptno)

CONSTRAINT cf_emp2_deptno  FOREIGN KEY (deptno) REFERENCES dept2(deptno)
```

  Las restricciones de clave foránea permiten implementar la integridad referencial mediante las acciones referenciales, que determinan el comportamiento que una clave foránea tiene ante la modificación y el borrado de los valores de los atributos a los que referencia.

  Oracle permite las acciones referenciales siguientes:
  - `NO ACTION` No realizar ninguna acción si existen filas de la clave foránea con el valor afectado por la modificación. Es la defectiva y ya no hay que declararla.
  - `CASCADE` Trasladar la modificación a los valores de la clave foránea.
  - `SET NULL` Hacer que la clave foránea tome valores nulos para los valores afectados por la modificación.

  Se indica con `ON DELETE <acción>` ya que para la actualización sólo permite la defectiva.

```
... FOREIGN KEY (deptno)
      REFERENCES dept2(deptno)  ON DELETE SET NULL
```

Otras sentencias DDL

Eliminación de tablas

En SQL hay que distinguir entre la estructura de una tabla y sus datos. Una tabla existe porque posee una estructura, aunque pueda tener o no filas.

Para eliminar la estructura se usa la sentencia `DROP TABLE`

```
DROP TABLE <tabla> [CASCADE CONSTRAINTS]

DROP TABLE emp2
```

Con `CASCADE CONSTRAINTS` se eliminan todas las restricciones asociadas a ella.

Modificación de tablas

Para modificar la estructura de una tabla se usa `ALTER TABLE` que permite muchas posibilidades.

Para añadir una columna a la tabla:

```
ALTER TABLE <tabla>  ADD <definición de columna nueva>

ALTER TABLE emp2     ADD  job VARCHAR2(9)
```

Para modificar una columna existente:

```
ALTER TABLE <tabla>  MODIFY <nueva definición de columna>

ALTER TABLE emp2     MODIFY  job VARCHAR2(12)      (Pero no VARCHAR2(5) !)
```

Otras sentencias DDL (cont.)

Modificación de tablas (cont.)

Para modificar una columna existente (cont.):

```
ALTER TABLE emp2     MODIFY  job VARCHAR2(12) DEFAULT NULL
```

Para eliminar una columna de la tabla:

```
ALTER TABLE <tabla>  DROP   (<columna>) CASCADE CONSTRAINTS

ALTER TABLE emp2     DROP   (job)
```

Para añadir una restricción a una tabla existente:

```
ALTER TABLE <tabla>  ADD <definición de restricción>

ALTER TABLE emp2     ADD CONSTRAINT ck_emp2_sal  CHECK (sal > 900)
```

Para eliminar una restricción de una tabla:

```
ALTER TABLE <tabla>  DROP CONSTRAINT <nombre de restricción>

ALTER TABLE emp2     DROP CONSTRAINT ck_emp2_sal
```

Lenguaje SQL

Vistas

Una vista es un tipo especial de tabla en la que sus datos se derivan de otras tablas, mediante la evaluación dinámica de una consulta.

No posee datos propios ni por tanto zona de almacenamiento para ellos.

Elementos:

- Nombre. Unico en el esquema relacional.
- Atributos. Forman las columnas de la vista.
- Definición. Consiste en una sentencia `SELECT` sobre otras tablas o vistas.

Una vista tiene un comportamiento similar al de una tabla base, excepto en las limitaciones de actualización: hay vistas actualizables y no actualizables, dependiendo de su definición.

Formato para crear una vista (Oracle):

```
CREATE [OR REPLACE] VIEW <vista> [(<columna 1>, ..., <columna n>)]
  AS  <sentencia SELECT>
```

Ejemplo:

```
CREATE VIEW  empleado
  AS  SELECT  empno, ename, mgr, job, deptno
      FROM    emp
      WHERE   deptno = 10

CREATE VIEW  departamento (cdgo, nombre, num_emp)
  AS  SELECT   a.deptno, b.dname, count(*)
      FROM     emp a, dept b
      WHERE    a.deptno = b.deptno
      GROUP BY a.deptno, b.dname
```

Vistas (cont.)

La cláusula `ORDER BY` no tiene sentido.

Oracle incluye `OR REPLACE` para poder redefinir una vista ya existente.

Los alias de las expresiones del `SELECT` serán las columnas de la vista si no se especifican de otra forma.

Una vez creada una vista sus datos pueden consultarse como si fuese una tabla real.

Formato para eliminar una vista:

```
DROP VIEW <vista>                Ejemplo:  DROP VIEW departamento
```

Utilidad de las vistas

- Independencia lógica de los datos
  Consiste en que los programas y la visión que de los datos tienen los usuarios, son inmunes a los cambios en la estructura lógica.
  - Crecimiento
  - Reestructuración
- Definición de esquemas externos
  Permiten presentar los datos acordes a las visiones que tienen los distintos grupos de usuarios de ellos.
- Simplificación de las consultas
  Permite asignar un nombre a una consulta que puede tener cierto grado de dificultad, que queda oculta por la propia vista.
- Establecer condiciones de seguridad
  El uso adecuado de vistas puede proporcionar la ocultación de datos a los usuarios.

Vistas (cont.)

Actualización de una vista

Supone la ejecución de cualquiera de las sentencias `UPDATE`, `INSERT` o `DELETE` sobre la vista, lo que desencadena dichas operaciones sobre las tablas o vistas que definen la propia vista.

Dependiendo que cómo se definió la vista, ésta será o no actualizable. No podremos actualizar una vista si esto supone más de una forma de actualizar las tablas base que la definen.

Una vista es actualizable, de acuerdo a SQL92, si se dan todas las condiciones siguientes:

- En el sentencia `SELECT` no aparece ni `UNION`, ni `INTERSECT`, ni `EXCEPT`, ni `GROUP BY`, ni `HAVING`.
- La condición de definición de la vista no es un join.
- La cláusula `SELECT` no incluye el término `DISTINCT`, ni puede incluir expresiones.
- La cláusula `FROM` incluye sólo la referencia de una tabla.
- Si en la definición de la vista aparece en el `WHERE` una subconsulta, en ella no puede aparecer una referencia en el `FROM` a la misma tabla que se referencia en la consulta principal que define la vista.

Además de las anteriores:

- Si una columna de una vista está definida como una expresión, la columna no permite `UPDATE` ni la vista `INSERT`.
- Si una vista no incluye alguna columna existente en la tabla definida con la propiedad de `NOT NULL`, entonces no es posible un `INSERT` en dicha vista.

Vistas (cont.)

Actualización de una vista (cont.)

Como se comentó, la regla general de actualización de una vista es que no podrá actualizarse si implica más de una forma de actualizar las tablas base que la definen.

Ejemplos:

1.– Una vista sobre todas las filas y sólo unas columnas. Es el resultado de una proyección y se conoce a veces como *vista vertical*.

```
CREATE VIEW   empleado1
  AS  SELECT  empno, ename, mgr, job, deptno
      FROM    emp
```

Suponiendo que `empno` es la clave primaria, la vista sería actualizable.

Si `empno` no formase parte de la vista, se trataría de una columna con valor no nulo que no aparece en la vista que, si no tuviese definido un valor defectivo, implicaría que la vista no acepta un `INSERT`.

2.– Vista sobre todas las columnas pero seleccionando las filas que cumplen una condición. A veces se dice que se trata de una *vista horizontal*.

```
CREATE VIEW   empleado2
  AS  SELECT  *
      FROM    emp
      WHERE   deptno = 10
```

Vistas (cont.)

Actualización de una vista (cont.)

3.– Vista con expresiones en la cláusula `SELECT`.

```
CREATE VIEW   empleado3
  AS  SELECT  empno, ename, sal + comm AS salario_total, deptno
      FROM    emp
```

4.– Vista con `DISTINCT`.

```
CREATE VIEW   empleado4
  AS  SELECT  DISTINCT job, sal
      FROM    emp
      WHERE   sal > 1000
```

5.– Vista sobre varias tablas.

```
CREATE VIEW  departamento (cdgo, nombre, num_emp)
  AS  SELECT   a.deptno, b.dname, count(*)
      FROM     emp a, dept b
      WHERE    a.deptno = b.deptno
      GROUP BY a.deptno, b.dname
```

Lenguaje SQL

Indices

Los índices son estructuras auxiliares cuyo objetivo es facilitar las operaciones de consulta. Se crean sobre una o más columnas de una tabla.

Formato:

```
CREATE [UNIQUE] INDEX <índice> ON <tabla> ( <columna 1> [, <columna 2>, ...] )
```

Ejemplos:

```
CREATE INDEX in_emp_sal  ON emp (sal)

CREATE UNIQUE INDEX in_dept_dname  ON dept (dname)

CREATE INDEX in_emp_jobsal  ON emp (job, sal)
```

La especificación de `UNIQUE` puede hacerse sobre columnas con valores únicos para cada fila.

La existencia de índices tiene una gran repercusión en la tarea de optimización del SGBD.

Por ese motivo los SGBD suelen crear de forma automática índices al declarar diversas restricciones de los datos. Oracle lo hace así con las restricciones de clave primaria y de unicidad.

Oracle dispone de varios tipos de índices. El formato general sería:

```
CREATE <tipo de índice>  INDEX <índice> ON <tabla> ( <columna 1> [, <columna 2>, ...] )
```

Seguridad

Se entiende por seguridad la protección de la BD frente a los accesos no autorizados, tanto intencionados como accidentales.

Abarca varios paspectos pero mencionamos sólo los referentes a:

- El control de acceso al SGBD
- El acceso selectivo a los datos

El control de acceso al SGBD

Consiste en un mecanismo que permite a los usuarios autorizados acceder al sistema y evita que accedan los usuarios no autorizados.

Se basa en cuentas de usuario propias del SGBD con contraseñas asociadas.

Las contraseñas pueden ser propias del SGBD o estar estar vinculadas al sistema operativo.

```
CREATE USER usuario IDENTIFIED  {BY contraseña1 | EXTERNALLY};

ALTER  USER usuario IDENTIFIED   BY contraseña2;
```

Seguridad (cont.)

El acceso selectivo a los datos

Permite que los usuarios registrados accedan a los datos a los que están autorizados, y a la vez evita que accedan a los restantes.

Basado en el control de acceso discrecional, que asocia privilegios a los objetos, y permite conceder y revocar los privilegios a identificadores de autorización.

- Identificador de autorización (a lo que se le puede otorgar y revocar privilegios):
  - Cuenta de usuario (estará vinculado a ese usuario):

```
CREATE USER usuario1 IDENTIFIED BY contraseña1;
```

  - Rol (estará vinculado a un grupo de usuarios):

```
CREATE ROLE rola;   GRANT rola TO usuario1;   GRANT rola TO usuario2;
CREATE ROLE rolb;   GRANT rolb TO usuario3;
CREATE ROLE rolc;   GRANT rolc TO usuario4;   GRANT rolc TO usuario5;
CREATE ROLE rolx;   GRANT rolx TO rola;
CREATE ROLE roly;   GRANT roly TO rolb;       GRANT roly TO rolc;
GRANT rola TO rolb;
```

    Los roles se asocian a los diferentes grupos de usuarios de la BD; éstos últimos se determinan por las *funciones* que los usuarios realizan sobre los datos.
    Permiten agrupar a los usuarios de cualquier forma que podamos imaginar, debido al encadenamiento entre roles.

Seguridad (cont.)

El acceso selectivo a los datos (cont.)

- Privilegios. Dos niveles de asignación:
  - **Nivel del SGBD o del sistema**
    Son privilegios asociados a una cuenta de usuario o a un rol que indican funciones que se pueden realizar en el SGBD.
    No están recogido en el SQL estándar, dejándose como parte de la implementación.
    Suelen hacer referencia a la posibilidad de conexión, la potestad de crear objetos, la de ser DBA, la de crear esquemas, etc. En Oracle hay muchos, entre ellos:

```
CREATE TABLE       Crear tablas e índices en el esquema propio.
CREATE ANY TABLE   Crear tablas en cualquier esquema.
DROP ANY TABLE     Eliminar cualquier tabla en cualquier esquema.
SELECT ANY TABLE   Consultar cualquier tabla en cualquier esquema.
DELETE ANY TABLE   Eliminar filas de cualquier tabla en cualquier esquema.
CREATE VIEW        Crear una vista en el esquema propio.
CREATE ANY INDEX   Crear un índice en cualquier esquema.
AUDIT ANY          Auditar cualquier objeto en la BD.
...
```

  - **Nivel de objeto (tabla, vista, ...)**
    Son privilegios asociados a cada objeto de la BD, que permiten a la cuenta de usuario o al rol que los posee realizar acciones sobre el objeto.

```
SELECT
DELETE
INSERT [lista_columnas]
UPDATE [lista_columnas]
REFERENCES [lista_columnas]
```

Seguridad (cont.)

El acceso selectivo a los datos (cont.)

En Oracle hay varios roles predefinidos referentes al sistema, entre ellos:

- `CONNECT` Permite conectarse al SGBD.
- `RESOURCE` Permite crear objetos en el SGBD.
- `DBA` Para tener acceso a las funciones de administración.

También hay un grupo de usuarios denominado `PUBLIC` que hace referencia a todos los usuarios del SGBD.

La forma de otorgar privilegios globales o a nivel del SGBD es:

```
GRANT { privilegio | rol } [, ...]
   TO { usuario | rol | PUBLIC } [, ...]
   [WITH ADMIN OPTION]
```

Con `WITH ADMIN OPTION` a quien se le otorga un privilegio, puede a su vez concederlo a terceros.

Ejemplo:

```
GRANT   CONNECT, RESOURCE   TO  usuario1
```

La forma de revocar privilegios globales es:

```
REVOKE  { privilegio | rol | ALL_PRIVILEGES } [, ...]
   FROM { usuario | rol | PUBLIC } [, ...]
```

Ejemplo:

```
REVOKE RESOURCE   FROM  PUBLIC
```

Seguridad (cont.)

El acceso selectivo a los datos (cont.)

Para conceder un privilegio referente a un objeto:

```
GRANT { privilegio | ALL } [, ...]
   ON objeto
   TO { usuario | rol | PUBLIC } [, ...]
   [WITH GRANT OPTION]
```

Con `WITH GRANT OPTION` a quien se le otorga un privilegio, puede a su vez concederlo a terceros.

Ejemplo:

```
GRANT   SELECT, INSERT, UPDATE(sal)  ON tabla1  TO usuario1
```

La forma de revocarlos es mediante:

```
REVOKE  { privilegio | ALL } [, ...]
   ON   objeto
   FROM { usuario | rol | PUBLIC } [, ...]
```

Ejemplo:

```
REVOKE  DELETE  ON  vista1  FROM  usuario2
```

Si se revoca un privilegio concedido con `WITH GRANT OPTION` a un usuario, todos a los que se lo había traspasado también lo pierden, salvo que lo tuviesen por otra línea de concesión.

Lenguaje SQL

Seguridad (cont.)

Ejemplos:

```
CONNECT us1/s1                                    -> Conectado como US1
SELECT *  FROM e;                                 -> Consulta la tabla E de US1

CONNECT us2/s2                                    -> Conectado como US2
SELECT *  FROM e;                                 -> Consulta la tabla E de US2
  ERROR en línea 1:
  ORA-00942: la tabla o vista no existe
SELECT *  FROM us1.e;                             -> Consulta la tabla E de US1
  ERROR en línea 1:
  ORA-00942: la tabla o vista no existe

CONNECT us1/s1                                    -> Conectado como US1
GRANT SELECT, INSERT ON e TO us2;                 -> Otorga privilegios en E a US2
GRANT SELECT, INSERT ON t TO us2 WITH GRANT OPTION;
                -> Otorga privilegios en T a US2 con la potestad de traspasarlos

CONNECT us2/s2                                    -> Conectado como US2
SELECT *  FROM us1.e;                             -> Consulta la tabla E de US1
SELECT *  FROM us1.t;                             -> Consulta la tabla T de US1

CONNECT us3/s3                                    -> Conectado como US3
SELECT *  FROM us1.t;                             -> Consulta la tabla T de US1
  ERROR en línea 1:
  ORA-00942: la tabla o vista no existe
```

Seguridad (cont.)

Ejemplos (cont.):

```
CONNECT us2/s2                                  -> Conectado como US2
GRANT SELECT ON us1.t TO us3;                   -> Traspasa privilegios a US3

CONNECT us3/s3                                  -> Conectado como US3
SELECT *  FROM us1.t;                           -> Consulta la tabla T de US1

CONNECT us1/s1                                  -> Conectado como US1
REVOKE SELECT, INSERT ON t FROM us2;
                -> Elimina privilegios en T a US2 que a su vez US2 había traspasado a US3

CONNECT us2/s2                                  -> Conectado como US2
SELECT *  FROM us1.e;                           -> Consulta la tabla E de US1
SELECT *  FROM us1.t;                           -> Consulta la tabla T de US1
  ERROR en línea 1:
  ORA-00942: la tabla o vista no existe

CONNECT us3/s3                                  -> Conectado como US3
SELECT *  FROM us1.t;                           -> Consulta la tabla T de US1
  ERROR en línea 1:
  ORA-00942: la tabla o vista no existe
```

Transacciones

Para que las modificaciones que realiza un SGBD sean consistentes y no se vean afectadas ante cualquier tipo de error del sistema, debe realizarlas mediante transacciones.

Una transacción es una secuencia de acciones que o se ejecutan todas, o el SGBD garantiza que el efecto es como si no se ejecutase ninguna.

Finaliza con la posibilidad de confirmación (`COMMIT`) o de anulación (`ROLLBACK`) de los cambios.

Si el SGBD tiene la posibilidad de definir transacciones y que cumplan estas características, se dice transaccional.

La ejecución de un programa habitualmente supone una secuencia de transacciones. No hay una analogía entre programa y transacción.

En Oracle todas las modificaciones de datos realizadas por sentencias DML se realizan dentro de una transacción; no así las DDL.

Si deseamos confirmar las modificaciones, acabamos la transacción con `COMMIT`. Si deseamos anularlas, acabamos con `ROLLBACK`.

Además cuando se sale del entorno de SQL*Plus sin error, se produce un `COMMIT` implícito. Si se sale anormalmente, se origina un `ROLLBACK` implícito.

La sentencia `ROLLBACK` anula las modificaciones realizadas desde el principio de la transacción.

Si en el curso de una transacción se definió un punto de restauración o de salvado con:

```
SAVEPOINT <punto de salvado>
```

puede hacerse que se anulen los cambios posteriores a ese momento y que la transacción continúe mediante:

```
ROLLBACK TO <punto de salvado>
```

Transacciones (cont.)

Ejemplos:

- Transacción 1:

```
SELECT *  FROM miemp  WHERE empno IN (7900, 7369);  -> sal 7900:  950;  sal 7369:  800
UPDATE miemp  SET sal = 1000  WHERE empno = 7900;   -> sal 7900: 1000;  sal 7369:  800
UPDATE miemp  SET sal = 2000  WHERE empno = 7369;   -> sal 7900: 1000;  sal 7369: 2000
ROLLBACK;                                           -> sal 7900:  950;  sal 7369:  800
                    transacción terminada anulando sus cambios
```

- Transacción 2:

```
SELECT *  FROM miemp  WHERE empno IN (7900, 7369);  -> sal 7900:  950;  sal 7369:  800
UPDATE miemp  SET sal = 1000  WHERE empno = 7900;   -> sal 7900: 1000;  sal 7369:  800
SAVEPOINT sp1;
UPDATE miemp  SET sal = 2000  WHERE empno = 7369;   -> sal 7900: 1000;  sal 7369: 2000
ROLLBACK TO SAVEPOINT sp1;                          -> sal 7900: 1000;  sal 7369:  800
                    transacción  que anula parcialmente sus cambios
                    pero que sigue su curso

UPDATE miemp  SET sal = 1500  WHERE empno = 7369;   -> sal 7900: 1000;  sal 7369: 1500
COMMIT;                                             -> sal 7900: 1000;  sal 7369: 1500
                    transacción terminada confirmando sus cambios
```

Catálogo

Con el término catálogo del sistema de un SGBD relacional se designa el conjunto de tablas y vistas del sistema de una base de datos relacional en las que se almacenan los metadatos de la misma.

Los SGBD emplean la información almacenada en ese conjunto de tablas, llamadas frecuentemente tablas del sistema para distinguirlas de las ordinarias que almacenan datos de los usuarios, para realizar sus funciones.

Estas tablas del sistema las crea y mantiene el propio SGBD, y son de sólo lectura para los usuarios.

Los datos de estas tablas son críticos para el buen funcionamiento del SGBD.

El catálogo de Oracle: diccionario de datos (tablas base y vistas)

Los usuarios acceden a las vistas del catálogo:

`USER`	Lo que el usuario ha creado.
`ALL`	A lo que el usuario tiene acceso, esto es, lo que ha creado y a lo que le han otorgado acceso.
`DBA`	A lo que el DBA puede acceder.

Lenguaje SQL

Catálogo (cont.)

Algunas vistas del catálogo de Oracle:

```
USER_TABLES   ALL_TABLES   DBA_TABLES   TAB   TABS   DICT
```

USER_TABLES	Tablas
USER_INDEXES	Indices
USER_OBJECTS	Objetos
USER_CONSTRAINTS	Restricciones
USER_VIEWS	Vistas
USER_SEQUENCES	Secuencias
USER_TAB_COLUMNS	Columnas de tablas y de vistas
USER_CATALOG	Tablas, vistas y sinónimos
USER_TRIGGERS	Disparadores
USER_USERS	Usuarios
...	

Ejemplos:

```
DESCRIBE  TAB                          -> Estructura de TAB
SELECT  *  FROM  TAB;                  -> Tablas y vistas del usuario
SELECT  *  FROM  TABS;                 -> Tablas y vistas del usuario
SELECT  *  FROM  DICT;                 -> Vistas del catálogo
SELECT  *  FROM  USER_VIEWS;           -> Vistas del usuario
SELECT  *  FROM  USER_CONSTRAINTS;     -> Restricciones del usuario
```

Lenguaje SQL

Catálogo (cont.)

Ejemplos (cont):

```
     CONNECT us2/s2                                  -> Conectado como US2
     SELECT  *  FROM  USER_CONSTRAINTS;              -> Restricciones del usuario

OWNER                          CONSTRAINT_NAME                C
------------------------------ ------------------------------ -
TABLE_NAME
------------------------------
SEARCH_CONDITION
--------------------------------------------------------------------------------
R_OWNER                        R_CONSTRAINT_NAME              DELETE_RU STATUS
------------------------------ ------------------------------ --------- --------
DEFERRABLE     DEFERRED  VALIDATED     GENERATED      BAD RELY LAST_CHA
-------------- --------- ------------- -------------- --- ---- --------
INDEX_OWNER                    INDEX_NAME                     INVALID
------------------------------ ------------------------------ -------
VIEW_RELATED
--------------
US2                            SYS_C00200812                  C
EMPI
"EMPNO" IS NOT NULL
                                                                      ENABLED
NOT DEFERRABLE IMMEDIATE VALIDATED     GENERATED NAME          03/09/04


1 fila seleccionada.
```

Catálogo (cont.)

Ejemplos (cont):

La tercera columna de `USER_CONSTRAINTS` es `CONSTRAINT_TYPE` que indica el tipo de restricción mediante un código.

Su significado es:

```
C   de check => la condición aparece expresada en SEARCH_CONDITION
P   primary key
U   unique
R   integridad referencial
V   vista con check option
O   vista con read only
```

Recuérdese que Oracle trata a las restricciones de no nulo como de tipo `CHECK`.

Las búsquedas en `USER_CONSTRAINTS` se realizan normalmente por el nombre de la tabla (`TABLE_NAME`) y por el nombre de la restricción (`CONSTRAINT_NAME`).

Por ese motivo las restricciones no sólo deben nombrarse, sino que el nombre debe ser descriptivo para deducir la tabla a la que afecta, el tipo de restricción y las columnas implicadas, evitando así que el SGBD le asigne un nombre críptico.

Bibliografía

[CeCa03] Celma, M.; Casamayor, J. C.; Mota, L.: **Bases de Datos Relacionales**. Prentice Hall, 2003.

[GrWe02] Groff, J.; Weinberg, P. N.: **SQL: The Complete Reference** (2^{nd} edition). McGraw-Hill, 2002. (Traducción: **SQL. Manual de referencia**. McGraw-Hill, 2003.)

[GuPe99] Gulutzan, P.; Pelzer, T.: **SQL-99 Complete, Really**. R&B Books, 1999.

[MeSi02] Melton, J.; Simon, A.: **SQL:1999 - Understanding Relational Language Components**. Morgan Kaufmann, 2002.

[Para02] Paramá, J. R.: Apuntes de BD1, 2002.

[RiMa02] Rivero, E; Martínez, L.; Reina, L.; Benavides, J.; Olaizola, J.: **Introducción al SQL para usuarios y programadores**. Thomson- Paraninfo, 2002.

[SiKo02] Silberschatz, A.; Korth, H.; Sudarshan, S.: **Database System Concepts** (4^{th} edition). McGraw-Hill, 2002.