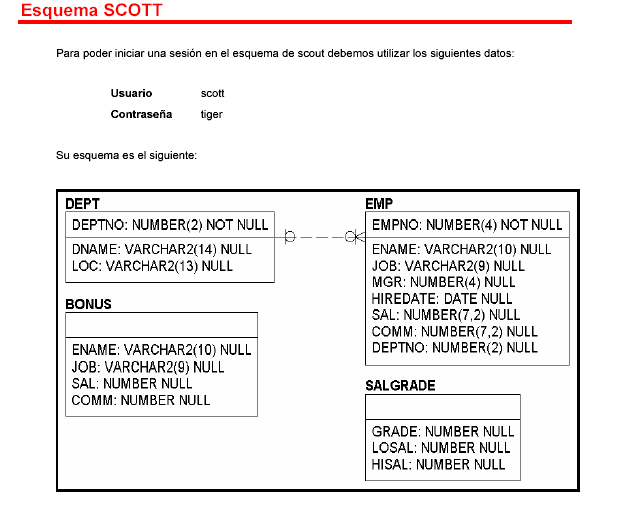
# UNIDAD DE TRABAJO 5: Consulta y Manipulación de Datos



## La sentencia SELECT

Una sentencia SELECT recupera información de la base de datos. Utilizando una sentencia SELECT, podemos hacer lo siguiente:

* *Selección*: podemos seleccionar los registros de la tabla que queramos, devueltos por una consulta. Podemos utilizar varios criterios para restringir de manera selectiva los registros que queremos ver.
* *Proyección*: podemos seleccionar las columnas de la tabla que deseemos, devueltas por una consulta. Podemos seleccionar tantas tablas como queramos.
* *Unión*: para extraer los datos almacenados en diferentes tablas mediante la creación de enlaces entre una columna y las dos tablas compartidas.

Sintaxis:

**SELECT** [**DISTINCT**] {\*, columna, … }

**FROM** tabla [,tabla …]

**[WHERE** condicion];

En la forma más simple una sentencia SELECT incluye las columnas a consultar (\* indica todas) y una cláusula FROM que especifica la tabla que contiene las columnas descritas en la cláusula SELECT. Adicionalmente, puede incluir una cláusula WHERE que filtra las filas o registros que devolverá la consulta. Si no hay cláusula WHERE, se mostrarán todas las filas de la tabla.

La opción DISTINCT sirve para eliminar filas duplicadas en el resultado de una consulta.

## Expresiones aritméticas, alias de columna, concatenación y strings

Puede ser necesario modificar la forma en que se visualizan los datos o se realizan los cálculos. Esto es posible utilizando expresiones aritméticas detrás de la cláusula SELECT. Una expresión aritmética puede contener nombres de columna, valores numéricos constantes y operadores aritméticos.

Los operadores aritméticos disponibles en SQL son: + - \* /

Ejemplo:

SELECT ename, sal, sal+300

FROM emp;

La precedencia de estos operadores es: \* / + - Podemos sobreescribir las reglas de precedencia utilizando paréntesis y así especificar el orden en que se ejecutan los operadores.

Cuando una expresión aritmética contiene un NULL se evalúa a NULL. Un NULL es un valor inaccesible o desconocido. No representa ni un cero ni un espacio en blanco. El cero es un número y el espacio es un carácter.

Si se desea renombrar el encabezamiento de una columna resultado se pueden utilizar alias. Es útil especialmente en cálculos. Sigue inmediatamente al nombre de la columna y opcionalmente puede situarse en medio la palabra clave AS. Si se desea que contenga espacios en blanco o caracteres especiales se encierra entre comillas dobles.

Ejemplo:

SELECT sal AS salario, ename “nombre del empleado”

FROM emp;

Con el operador de concatenación || se pueden concatenar columnas con otras columnas, expresiones aritméticas o valores constantes con el fin de crear una expresión de caracteres. Las columnas de cualquiera de los lados del operador se combinan para formar una sola columna resultante.

Ejemplo:

SELECT ename || job AS “Empleados”

FROM emp;

Un literal es un carácter, expresión o número incluido en la lista de la cláusula SELECT. Por cada registro devuelto se imprime una cadena de caracteres. Las cadenas de caracteres con formato de texto libre se pueden incluir en el resultado de la consulta y se tratan como una columna de la lista SELECT. Los valores literales de tipo fecha y carácter deben estar encerrados dentro de comillas simples ‘ ‘ mientras que los literales de tipo número no.

Ejemplo:

SELECT ename || ‘ es un ‘ || job AS Empleados

FROM emp;

En Oracle, para visualizar la estructura o definición de una tabla se utiliza el comando DESCRIBE.

Ejemplo:

DESCRIBE dept

## Restricción y clasificación de los datos: condiciones en la cláusula WHERE

Podemos restringir las filas recuperadas usando la cláusula WHERE. Una cláusula WHERE contiene una condición que se debe cumplir y se escribe a continuación de la cláusula FROM.

La cláusula WHERE puede comparar los valores en las columnas, valores literales, expresiones aritméticas o funciones. La cláusula WHERE consta de tres elementos:

* Nombre de la columna
* Operador de comparación
* Nombre de columna, constante o lista de valores

Ejemplo:

SELECT ename, job, deptno

FROM emp

WHERE job=’CLERK’;

Tabla resultado devuelta:

|  |  |  |
| --- | --- | --- |
| **ENAME** | **JOB** | **DEPTNO** |
| JAMES | CLERK | 30 |
| SMITH | CLERK | 20 |
| ADAMS | CLERK | 20 |
| MILLER | CLERK | 10 |

En el ejemplo, la sentencia SELECT recupera el nombre, oficio y número de departamento de todos los empleados cuyo oficio es CLERK.

Podemos observar que el oficio CLERK ha sido especificado en mayúsculas para asegurarse de que se corresponde con el valor de la columna JOB en la tabla EMP, que están todos en mayúsculas. Las cadenas de caracteres son, por tanto, “case sensitive”, es decir, distingue entre mayúsculas y minúsculas.

**Cadenas de caracteres y fechas**

Las cadenas de caracteres y las fechas se encierran entre comillas simples en la cláusula WHERE. Sin embargo, los valore numéricos no se encierran entre comillas.

Si la consulta anterior se hubiera formulado como:

SELECT ename, job, deptno

FROM emp

WHERE job=’clerck’;

No se habría devuelto ningún valor en la tabla resultado porque todos los valores están almacenados en mayúsculas.

En cuanto a las fechas, Oracle las almacena en un formato numérico interno, representado por el siglo, año, mes, día, hora, minutos y segundos. La fecha por defecto se visualiza como DD-MES-AA.

**Operadores de comparación**

Se utilizan en las condiciones en las que se compara una expresión con otra. Tienen el siguiente formato:

WHERE *expresion operador valor*

Los operadores de comparación clásicos son los siguientes:

= Igual a

> Mayor que

< Menor que

>= Mayor o igual que

<= Menor o igual que

<> Distinto o no igual a

Ejemplo:

SELECT ename, sal, comm

FROM emp

WHERE sal<=comm;

Otros operadores de comparación que pueden aparecer en una condición son:

* **BETWEEN … AND …**

Se utiliza para visualizar registros basados en un rango de valores. Los valores especificados como valor superior e inferior se incluyen en la operación.

Ejemplo:

SELECT ename, sal

FROM emp

WHERE sal BETWEEN 1000 AND 1500;

La cláusula WHERE se podría haber expresado también como:

WHERE sal >= 1000 AND sal <= 1500;

También se puede utilizar este operador de forma negativa precediéndola de NOT:

NOT BETWEEN … AND …

* **IN (lista …)**

Se utiliza este operador para localizar valores coincidentes con una determinada lista. Puede utilizarse con cualquier tipo de dato.

Ejemplos:

SELECT empno, ename, sal, mgr

FROM emp

WHERE mgr IN (7902, 7566, 7788);

SELECT empno, ename, mgr, deptno

FROM emp

WHERE ename IN (‘FORD’, ‘ALLEN’);

Admite la versión negativa del operador: NOT IN (lista ….)

* **LIKE**

Se utiliza para realizar búsquedas en cadenas de caracteres. No siempre se conoce el valor exacto a buscar. Se pueden seleccionar filas que coincidan con un patrón de caracteres usando el operador LIKE. La operación de coincidencia se conoce como una búsqueda que incluye *comodines*:

% representa cualquier secuencia de cero o más caracteres

\_ (guión bajo) representa un solo carácter

Ejemplos:

- Empleados cuyo nombre empieza por S:

SELECT ename

FROM emp

WHERE ename LIKE ‘S%’;

- Empleados que se contrataron en 1981:

SELECT ename, hiredate

FROM emp

WHERE hiredate LIKE ‘%1981’;

- Empleados cuyo nombre tenga una A como segunda letra:

SELECT ename

FROM emp

WHERE ename LIKE ‘\_A%’;

En el caso de necesitar una coincidencia exacta para los comodines, se utiliza la opción ESCAPE. Dicha opción especifica cual es el carácter ESCAPE. Si quisiéramos hacer una consulta sobre el departamento HEAD\_QUARTERS, lo haríamos de la siguiente manera:

SELECT \* FROM dept

WHERE dname LIKE ‘%\\_%’ ESCAPE ‘\’;

Devolverá la tabla resultado:

|  |  |  |
| --- | --- | --- |
| **DEPTNO** | **DNAME** | **LOC** |
| 50 | HEAD\_QUARTERS | ATLANTA |

La opción ESCAPE identifica la barra (\) como el carácter ESCAPE. En el patrón de búsqueda, el carácter ESCAPE precede al \_. Esto hace que Oracle interprete el \_ literalmente.

* **IS NULL**

Verifica la presencia de valores nulos. Un valor nulo significa que el valor es inaccesible, sin valor, desconocido o inaplicable. Por lo tanto, **no se puede comprobar con el símbolo =**, porque un valor nulo no puede ser igual o distinto a otro valor.

Ejemplos:

SELECT ename, mgr

FROM emp

WHERE mgr IS NULL;

SELECT ename, job, comm

FROM emp

WHERE comm IS NULL;

**Operadores lógicos**

Un operador lógico combina el resultado de dos condiciones para producir un único resultado o para invertir el resultado de una condición simple. Hay tres operadores lógicos disponibles en SQL:

AND devuelve TRUE si ambas condiciones son TRUE

OR devuelve TRUE si alguna de las condiciones es TRUE

NOT devuelve TRUE si la siguiente condición es FALSE

Los operadores lógicos AND y OR nos permiten utilizar varias condiciones en una cláusula WHERE.

Ejemplos:

SELECT empno, ename, job, sal

FROM emp

WHERE sal >= 1100 AND job = ’CLERK’;

SELECT empno, ename, job, sal

FROM emp

WHERE sal >= 1100 OR job = ’CLERK’;

SELECT ename, job

FROM emp

WHERE job NOT IN (‘CLERK’, MANAGER’, ANALYST’);

SELECT empno, ename, job,

FROM emp

WHERE comm IS NOT NULL;

**Reglas de Precedencia**

Cuando aparece más de un operador en una condición, se evalúan en el orden que digan las reglas de precedencia.

|  |  |
| --- | --- |
| **ORDEN EVALUADO** | **OPERADOR** |
| 1 | Todos los operadores de comparación |
| 2 | NOT |
| 3 | AND |
| 4 | OR |

Se pueden utilizar los paréntesis para modificar las reglas de precedencia.

Ejemplo:

SELECT ename, job, sal

FROM emp

WHERE job = ’SALESMAN’ OR job=’PRESIDENT’ AND sal>1500;

Hay 2 condiciones:

* La primera es que el oficio sea PRESIDENT y el salario mayor que 1500.
* La segunda es que el oficio sea SALESMAN.

Por lo tanto, la sentencia SELECT interpretará lo siguiente: “Seleccionar los registros de los empleados cuyo oficio sea PRESIDENT y gane más de 1500 o si su oficio es SALESMAN”.

En el caso de haber utilizado paréntesis como en el siguiente ejemplo, la condición sería totalmente distinta:

SELECT ename, job, sal

FROM emp

WHERE (job = ’SALESMAN’ OR job=’PRESIDENT’) AND sal>1500;

“Seleccionar los registros de los empleados cuyo oficio sea PRESIDENT o SALESMAN y ganen más de 1500”.

**La cláusula ORDER BY**

El orden de las filas recuperadas por el resultado de una consulta es indefinido. La cláusula ORDER BY se utiliza para ordenar las filas. Si se usa, será la última cláusula de la sentencia SELECT.

La ordenación se puede hacer de forma ascendente o de forma descendente, poniendo las palabras clave ASC (ascendente) o DESC (descendente) al final de la cláusula.

Ejemplos:

SELECT ename, job, deptno, hiredate

FROM emp

ORDER BY hiredate DESC;

Utilizando un alias de columna:

SELECT empno, ename, sal\*12 sal\_anual

FROM emp

ORDER BY sal\_anual;

Ordenando por más de una columna. Se puede ordenar por una columna no seleccionada:

SELECT ename, deptno, sal

FROM emp

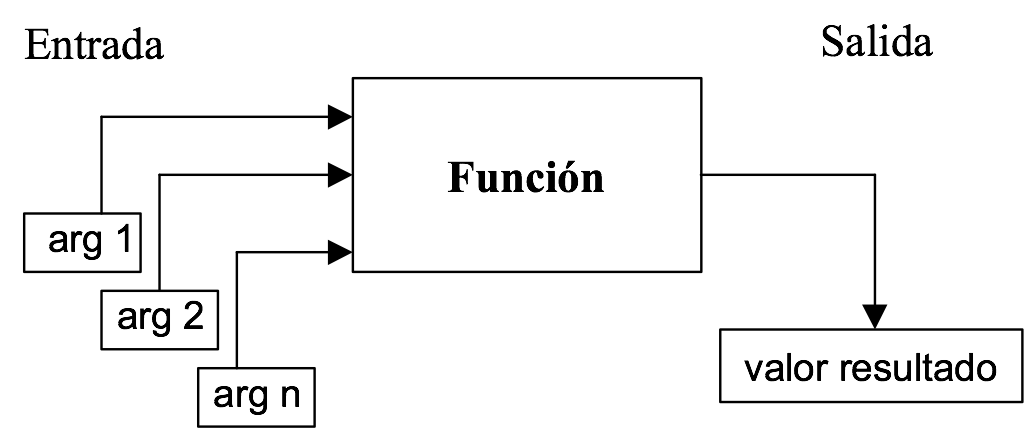
ORDER BY deptno, sal DESC;

## Funciones a nivel de fila

Las funciones constituyen una característica muy poderosa de SQL y se pueden utilizar para:

* Realizar cálculos sobre datos
* Modificar ítems de datos individuales
* Manipular la salida de grupos de filas
* Modificar formatos de los datos para su presentación
* Convertir los tipos de datos de las columnas

Las funciones SQL aceptan argumentos y devuelven un valor.



Existen 2 tipos distintos de funciones:

* A nivel de fila
* A nivel de grupos de filas

Las funciones a nivel de fila operan sólo sobre filas y devuelven un único resultado por cada una de ellas:

* Funciones de caracteres
* Funciones de número
* Funciones de fecha
* Funciones de conversión

Las funciones a nivel de fila se usan para manipular ítems de datos. Aceptan uno o más argumentos y devuelven un valor por cada fila que recupera la consulta. Un argumento puede ser:

* Una constante suministrada por el usuario
* Un valor de variable
* Un nombre de columna
* Una expresión

Las funciones a nivel de fila pueden actuar sobre cada fila recuperada por la consulta, devuelven un resultado por cada fila, pueden devolver un dato de diferente tipo que el referenciado, pueden aceptar uno o más argumentos del usuario, se pueden anidar y se pueden usar en las cláusulas SELECT, WHERE y ORDER BY.

Sintaxis:

*nombre\_de\_función (columna | expresión, [arg1, arg2,…])*

**Funciones de caracteres:**

a) LOWER (columna | expresión) -> Pasa a minúsculas el campo que se le pase.

b) UPPER (columna | expresión) -> Pasa a mayúsculas el campo.

c) INITCAP (columna | expresión) -> Pasa a mayúsculas la primera letra de cada palabra y deja las demás en minúsculas.

d) CONCAT (columna1 | expresión1, columna2 | expresión2) -> Concatena las 2 cadenas. Es equivalente al operador ||

e) SUBSTR (columna | expresión, m, [n]) -> devuelve la cadena de caracteres empezando por el carácter en la posición m hasta la n. Si n se omite (opcional), devuelve los caracteres hasta el final. Si m es negativo, el contador empieza desde el final de la cadena.

f) LENGTH (columna | expresión) -> longitud de la cadena que se le pase.

g) INSTR (columna | expresión, m) -> devuelve la posición del carácter m dentro de la cadena.

h) LPAD (columna | expresión, n, ‘string’) -> rellena la cadena por la derecha con el carácter ‘string’ hasta un total de n posiciones. Esto hace que los datos de una columna sean de tamaño fijo.

i) TRIM (parámetro, carácter\_a\_borrar FROM campo) -> sirve para eliminar caracteres del principio (leading), del final (trailing) o ambos (both).

El parámetro tiene tres valores posibles:

* leading: de una cadena de caracteres, borras el principio
* trailing: de una cadena de caracteres, borras el final
* both: de una cadena de caracteres, borras tantos el principio como el final.

Ejemplos (sacar de la tabla de empleados (emp) información como la siguiente):

* ‘El trabajo de’ nombre\_del\_empleado\_en\_mayúsculas ‘es’ ‘Puesto\_en\_minúsculas’

SELECT 'El trabajo de ' || INITCAP (ename) || ' es ' || LOWER (job) FROM emp;

* Visualizar el numero de empleado, nombre y departamento de blake:

SELECT ename, empno, deptno

FROM emp

WHERE ename = UPPER ('blake');

* Visualizar el nombre del empleado, nombre del empleado concatenado a su puesto de trabajo, longitud del nombre y posición de la letra “A” en el nombre:

SELECT ename, ename ||empno, LENGTH (ename), INSTR (ename, 'A')

FROM emp;

**Funciones numéricas:**

a) ROUND (columna | expresión, [n]) -> Redondea la columna, expresión o valor a n posiciones decimales. Si se omite n, no se redondea con lugares decimales. Si n es negativo, los números a la izquierda del punto decimal se redondean.

b) TRUNC (columna | expresión, [n]) -> Trunca la columna o valor en la n-ésima posición decimal. Si se omite n, sin lugares decimales. Si n es negativo, los números a la izquierda del punto decimal se truncan a cero.

c) MOD (m, n) -> Devuelve el resto de la división de m entre n.

Ejemplos:

ROUND (45.926, 2) ➔ 45.93

TRUNC (45.926, 2) ➔ 45.92

MOD (1600, 300) ➔ 100

* Calcular el ratio (resto) del salario respecto a la comisión de cada empleado, cuyo oficio sea “SALESMAN”:

SELECT ename, sal, comm, MOD (sal, comm)

FROM emp

WHERE job='SALESMAN';

* Visualizar el valor 45.923 redondeado a centenas, 0 y 10 posiciones decimales:

SELECT ROUND (45.923, 2), ROUND (45.923, 0), ROUND (45.923, -1)

FROM SYS.DUAL;

Tabla resultado devuelta:

|  |  |  |
| --- | --- | --- |
| **ROUND (45.923, 2)** | **ROUND (45.923, 0)** | **ROUND (45.923, -1)** |
| 45.92 | 46 | 50 |

* Visualizar el valor 45.923 truncado a centenas, 0 y 10 posiciones decimales:

SELECT TRUNC (45.923, 2), TRUNC (45.923, 0), TRUNC (45.923, -1)

FROM SYS.DUAL;

Tabla resultado devuelta:

|  |  |  |
| --- | --- | --- |
| **TRUNC (45.923, 2)** | **TRUNC (45.923, 0)** | **TRUNC (45.923, -1)** |
| 45.92 | 45 | 40 |

**Funciones de fecha:**

Internamente Oracle las almacena como números, que incluye: Siglo, año, mes, día, horas, minutos y segundos.

Va desde el 1 de enero de 4712 A.C. hasta el 31 de diciembre de 9999 D.C.

SELECT SYSDATE FROM SYS.DUAL; 🡪 para saber la fecha actual. DUAL es una tabla virtual de la BD .

Recordemos que el formato de fecha es DD-MES-AA.

A las fechas, se les puede aplicar operadores aritméticos (+, -):

* Fecha + Número ➔ Fecha (agrega una cantidad de días a la fecha)
* Fecha – Número ➔ Fecha (resta una cantidad de días a partir de una fecha)
* Fecha – Fecha ➔ Numero de días (resta una fecha de otra)
* Fecha + Numero/24 ➔ Fecha (agrega una cantidad de horas a la fecha)

Ejemplo: Número de semanas que llevan contratados los empleados.

SELECT ename, (SYSDATE - hiredate)/7

FROM emp;

Las funciones de fecha son las siguientes:

* MONTHS\_BETWEEN (‘fecha1’, ‘fecha2’): devuelve un número de meses entre dos fechas dadas.
* ADD\_MONTHS (‘fecha’, n): devuelve una fecha en la que se añadirá ‘n’ meses.
* NEXT\_DAY (‘fecha’, ‘caracteres’): calcula la fecha posterior a la dada. Los caracteres pueden ser un número representando un día o una cadena de caracteres Ejemplo: NEXT\_DAY (04-10-04, ‘MONDAY’) ➔ ‘08-10-04’
* LAST\_DAY (‘fecha’): te devuelve el último día del mes de la fecha especificada.
* ROUND (‘fecha’, [formato de fecha]): devuelve la fecha redondeada a la unidad especificada por el formato. Ejemplos de formato: ‘month’, ‘year’.
* TRUNC (‘fecha’, [formato de fecha]): devuelve la fecha con la porción del día truncado en la unidad especificada por el modelo de formato de fecha. Si se omite el formato, la fecha se trunca en el día más próximo.
* CEIL: que redondean hacia arriba Funcionan parecido a Round y Trunc.
* FLOOR: que redondean hacia abajo Funcionan parecido a Round y Trunc. <https://blogs.oracle.com/connect/post/from-floor-to-ceiling-and-other-functional-cases> (Ejemplos de Ceil, Floor, Round Trunc)

Ejemplos:

* Comparar las fechas de contratación de todos los empleados que empezaron en 1987. Visualizar el número de empleados, fecha de contratación y el mes en que el empezaron, utilizando funciones ROUND y TRUNC.

SELECT empno, hiredate, ROUND (hiredate, 'month'), TRUNC (hiredate, 'month')

FROM emp

WHERE hiredate LIKE '%87';

* Para todos los trabajadores empleados por menos de 200 meses, visualizar número de empleado, fecha de contratación, fecha de revisión de 6 meses, primer viernes después de la fecha de contratación y el mismo día del mes en que se contrataron.

SELECT empno, hiredate, ADD\_MONTHS (hiredate, 6),

NEXT\_DAY (hiredate, 'VIERNES'), LAST\_DAY (hiredate)

FROM emp

WHERE MONTHS\_BETWEEN (sysdate, hiredate) <300;

**Tipos y conversión de tipos:**

Recordemos los tipos de datos existentes en Oracle:

* NUMBER (n, d) 🡪 n: numero máximo de dígitos
* d: numero de decimales
* dígitos enteros = n – d
* VARCHAR2 (s) 🡪 s: cadena de caracteres de tamaño máximo s (tamaño variable)
* DATE
* CHAR (s) 🡪 s: cadena de caracteres de tamaño fijo s

Las conversiones de datos pueden realizarse de 2 maneras:

1. Implícitas: las hace Oracle, sin intervención del usuario.

***De A***

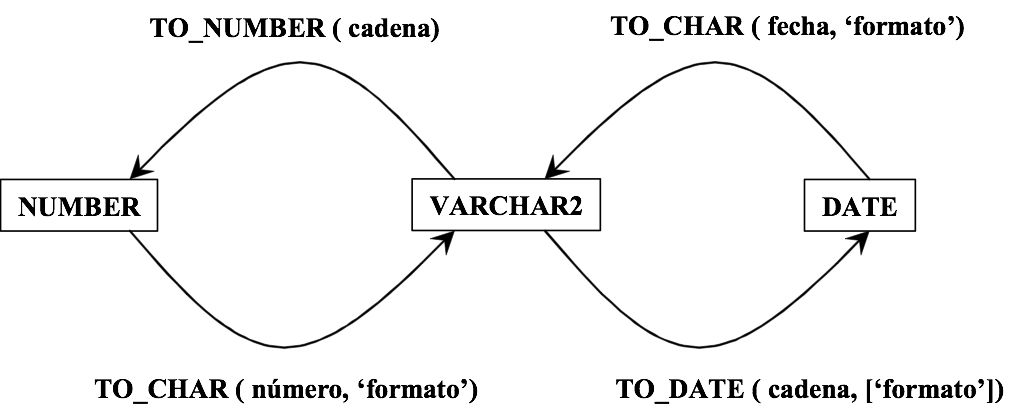
VARCHAR2 o CHAR NUMBER

VARCHAR2 o CHAR DATE

NUMBER VARCHAR2

DATE VARCHAR2

1. Explícitas: las que el usuario hace explícitamente.



TO\_CHAR

1. de fecha a caracteres.

2. de número a caracteres.

1. TO\_CHAR (fecha, ‘formato’): de fecha a caracteres

Ejemplo: TO\_CHAR (hiredate, ‘MM/YY’).

Formato:

* 1. Encerrado entre comillas simples
  2. Es case sensitive Y ≠ y
  3. Elementos del formato
     1. YYYY : año completo en numero
     2. YEAR : año en letras
     3. MM : número del mes con 2 dígitos
     4. MONTH : nombre completo del mes
     5. DY : abreviatura de 3 letras del día de la semana
     6. DAY : nombre completo del día
     7. S : siglo
     8. SCC : pone AC antes de fecha
     9. Q : pone el trimestre del año
     10. RM : pone el mes en números romanos
     11. J: día juliano (el numero de días desde el 31 de diciembre del 4712 a.c.)
     12. fm : quita el relleno en blanco o los 0 a la izquierda

2. TO\_CHAR (numero, ‘formato’): de número a caracteres

Ejemplo: SELECT TO\_CHAR (sal, ’99,999L’) FROM emp;

Nota: Mirar el libro McGRAW

9: Digito

TO\_NUMBER (cadena\_de\_caracteres): convierte de caracteres a números.

TO\_DATE (cadena\_de\_caracteres [,formato]): cambia de caracteres a fecha

Ej.: SELECT TO\_DATE (‘01111996’,’DDMMYYYY’)

FROM SYS.DUAL;

Hay un formato de fecha: RR (Como YY, pero sensible a cambios de siglo)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Fecha especificada (2 dígitos) | |
|  |  | 0-49 | 50-99 |
| Fecha actual  (2 dígitos) | 0-49 | Mismo siglo | Siglo anterior |
| 50-99 | Siglo posterior | Mismo siglo |

Algunos ejemplos que utilizan este formato RR comparado con el YY:

|  |  |  |  |
| --- | --- | --- | --- |
| **Año Actual** | **Fecha especificada** | **Formato RR** | **Formato YY** |
| 1995 | 27-OCT-95 | 1995 | 1995 |
| 1995 | 27-OCT-17 | 2017 | 1917 |
| 2001 | 27-OCT-17 | 2017 | 2017 |
| 2001 | 27-OCT-95 | 1995 | 2095 |

Funciones de Oracle

https://sites.google.com/site/josepando/home/funciones-sql/funciones-que-devuelven-una-valor-nico-para-cada-fila-de-una-consulta-o-vista/funciones-de-tipo-fecha

**Otras funciones de Oracle:**

Función NVL: Convierte un nulo a un valor concreto

* Es aplicable a cualquier tipo de datos: fechas, caracteres y números.
* Los tipos de datos de ambas expresiones deben coincidir.

Sintaxis: *NVL (expresion1, expresion2)*

Cada vez que la *expresión1* equivale a nulo, la sustituye por la *expresion2*.

Ejemplos:

* Numérico: NVL (comm, 0)
* Fecha: NVL (hiredate, ’01-ene-82’)
* Cadena de caracteres: NVL (job, ‘Sin trabajo’)
* Calcular la compensación anual de todos los empleados, es decir, multiplicar el salario mensual por 12 y añadirle la comisión:

SELECT ename, sal, comm, (sal\*12) + NVL (comm, 0)

FROM emp;

Función DECODE: hace las veces de sentencias CASE o IF-THEN-ELSE para facilitar consultas condicionales.

Esta función descifra una expresión después de compararla con cada valor de búsqueda. Si la expresión coincide con el valor de búsqueda, devuelve el resultado. Si se omite el valor por defecto (default), se devolverá un valor nulo en el caso de que la búsqueda no coincida con ninguno de los valores resultantes.

Sintaxis:

DECODE (columna | expresión, valor1, resultado1

[, valor2, resultado2, …, ]

[, default ]) ;

Interpretación:

IF (columna | expresión) = valor1

THEN resultado1

IF (columna | expresión) = valor2

THEN resultado2

IF (columna | expresión) = valor3

THEN resultado3

ELSE resultado 4

Ejemplo:

Mostrar el puesto de trabajo y el salario de los empleados de tal manera que si el puesto de trabajo (JOB) es ANALYST, el salario (SAL) se incrementa en un 10%; si JOB es CLERK, el salario se incrementa en un 15%; si JOB es MANAGER, el salario se incrementa en un 20%. El resto no percibe incremento de salario.

SELECT job, sal,

DECODE (job, 'ANALYST', sal \*1.1,

'CLERK', sal \*1.15,

'MANAGER', sal\*1.20,

sal)

FROM emp;

**Anidamiento de funciones:**

Las funciones a nivel de fila pueden ser anidadas hasta cualquier profundidad. Estas se evalúan desde el nivel más interno hasta el más externo.

F3 (F2 (F1 (col, arg1), arg2), arg3)

paso1

paso2

paso3

Ejemplos:

SELECT ename, NVL (TO\_CHAR (mgr), ‘ Sin jefe’)

FROM emp

WHERE mgr IS NULL;

SELECT TO\_CHAR (NEXT\_DAY (ADD\_MONTHS (hiredate, 6), ‘VIERNES’),

‘fmDay, Month ddth, YYYY’) “Prox revision de 6 meses”

FROM emp

ORDER BY hiredate;

## 4. Obtención de datos de múltiples tablas

En ocasiones necesitamos utilizar datos de más de una tabla al mismo tiempo. Por ejemplo, podríamos necesitar datos que estén en la tabla EMP y en la tabla DEPT. Para obtenerlos necesitamos unir ambas tablas o, lo que es lo mismo, realizar una JOIN.

La unión o JOIN de dos tablas se realiza a través de valores comunes, esto es, con las columnas de clave extranjera y la de clave primaria. Para ello, se escribe una condición en la cláusula WHERE de la sentencia SELECT.

Sintaxis:

SELECT tabla1.columna1, tabla2.columna2, …

FROM tabla1, tabla2

WHERE tabla1.columnaFK = tabla2.columnaPK;

Para combinar tablas se necesita como mínimo una cantidad de condiciones de JOIN equivalentes a la cantidad de tablas menos uno. Por lo tanto, para combinar 4 tablas, se necesitarían un mínimo de 3 condiciones. Esta regla no se puede aplicar si la tabla tiene una clave primaria combinada, en cuyo caso se pueden requerir más de una columna para identificar unívocamente a cada fila.

Ejemplo: Supongamos el siguiente esquema relacional:

CUSTOMER (name, custid)

ORD (custid, ordid)

ITEM (ordid, itemid)

SELECT customer.name, ord.ordid, item.itemid

FROM customer, ord, item

WHERE customer.custid = ord.custid

AND ord.ordid = i.ordid;

Cuando una condición de JOIN no es válida o se omite completamente, el resultado es un producto cartesiano, en el cual se muestran las combinaciones de todas las filas. Se combinan todas las filas de la primera tabla con todas las filas de la segunda.

Un producto cartesiano tiende a generar una gran cantidad de filas y este tipo de resultado es raramente útil. Se debería incluir una condición de JOIN válida en la cláusula WHERE, a menos que exista la necesidad específica de combinar todas las filas de todas las tablas.

Ejemplo:

SELECT emp.empno, emp.ename, emp.deptno, dept.deptno, dept.loc

FROM emp, dept

WHERE emp.deptno = dept.deptno;

Para evitar la ambigüedad, es necesario calificar los nombres de las columnas en la cláusula WHERE usando los nombres de las tablas. Sin los prefijos, la columna deptno podría ser tanto de la tabla dept como de la tabla emp.

Si no existen nombres repetidos en las tablas que se unen, no es necesario calificar las columnas. Sin embargo, se ganará una mejora del rendimiento con el uso de prefijos, ya que de esta manera le decimos exactamente a Oracle a donde ir para encontrar las columnas.

Los requerimientos para calificar a los nombres de columna ambiguos también son aplicables para las columnas que también podrían ser ambiguas en otras cláusulas como SELECT u ORDER BY.

Además del JOIN, se pueden añadir criterios adicionales para la cláusula WHERE. Por ejemplo, para visualizar el número de empleado de King, el nombre, número de departamento y localidad, necesitamos una condición adicional en la cláusula WHERE.

SELECT empno, ename, emp.deptno, loc

FROM emp, dept

WHERE emp.deptno = dept.deptno

AND ename = UPPER (‘King’);

**Alias de tabla**

Calificar las tablas con los nombres de tabla puede llevar mucho tiempo, especialmente si los nombres son largos. Para ello existen los alias de tabla. Así como los alias de columna dan a una columna otro nombre, los alias de tablas dan a una tabla otro nombre. Los alias de tablas ayudan a mantener reducido el código SQL, utilizando por tanto menos memoria.

Ejemplo:

SELECT e.empno, e.ename, e.deptno, d.deptno, d.loc

FROM emp e, dept d

WHERE e.deptno = d.deptno;

Las reglas de utilización de los alias de tabla son las siguientes:

* Pueden tener hasta 30 caracteres de longitud pero es preferible que sean cortos
* Si se usa un alias para un nombre de tabla en particular en la cláusula FROM, entonces debe ser sustituida a través de toda la sentencia SELECT.
* Los alias de tablas deben ser significativos
* El alias de tabla es válido sólo en la sentencia SELECT actual.

**Otros tipos de JOINS (Mirar el anexo 1 de tipos de JOINS)**

A las JOINS realizadas utilizando las claves primarias y extranjeras vistas hasta ahora se las denomina JOINS simples (EQUIJOINS o INNER JOINS), en las que los valores de las columnas comparadas son iguales.

En algunos casos las columnas a unir pueden estar en la misma tabla, correspondiéndose con relaciones reflexivas. Por ejemplo, el jefe de un empleado (MGR) puede combinarse con el número de empleado (ENAME) que supervisa.

SELECT trabajador.ename || ‘ trabaja para ’ || jefe.ename as currantes

FROM emp trabajador, emp jefe

WHERE trabajador.mgr = jefe.empno;

Existen otros tipos de JOINS:

* NON-EQUIJOINS
* OUTER JOINS

NON-EQUIJOINS:

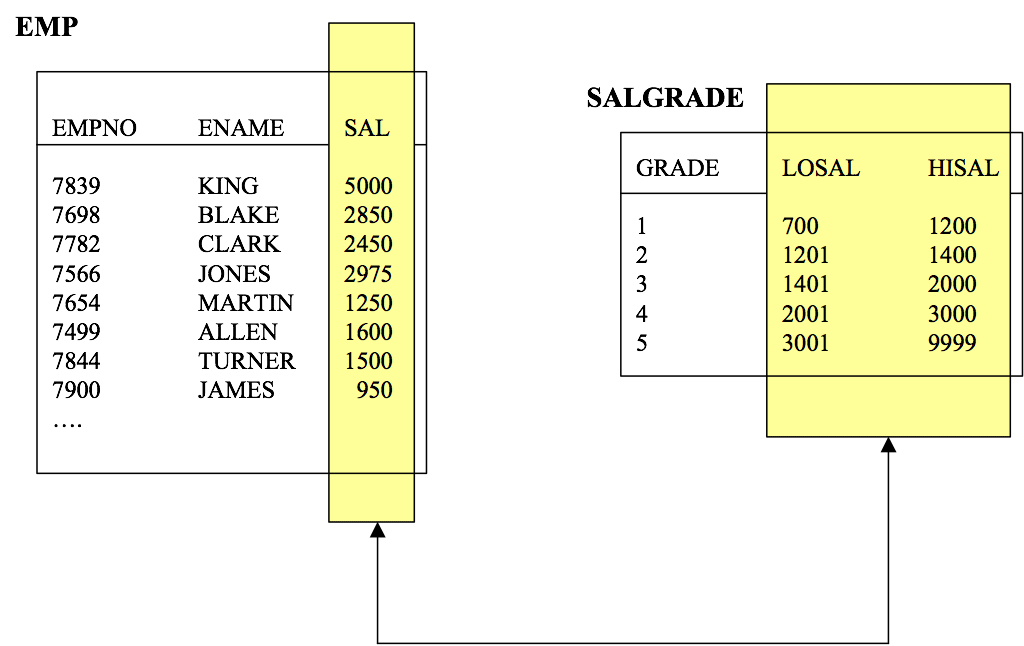
Se dan cuando no existe una correspondencia directa entre las tablas que se unen en el JOIN. La relación se obtiene utilizando un operador distinto del igual (=).

Ejemplo:

SELECT e.ename, e.sal, s.grade

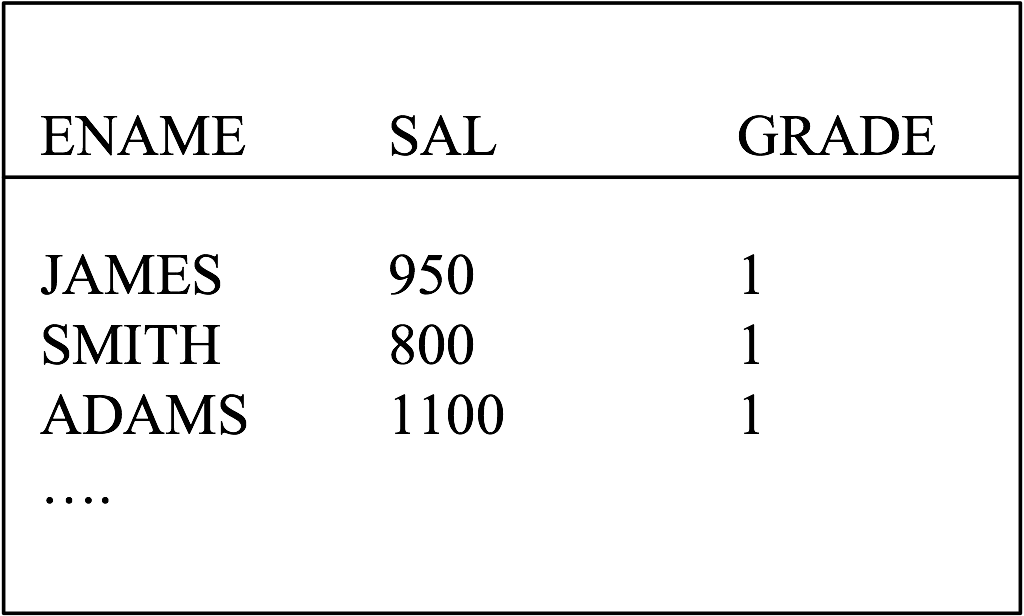
FROM emp e, salgrade s

WHERE e.sal BETWEEN s.losal AND s.hisal;



La relación entre las tablas EMP y SALGRADE es un NON-EQUIJOIN, lo cual significa que ninguna columna en la tabla EMP se corresponde directamente con una columna en la tabla SALGRADE. La relación entre las dos tablas es que la columna SAL en la tabla EMP está entre las columnas LOSAL y HISAL de la tabla SALGRADE.

Se produce la siguiente tabla-resultado:



Se está evaluando el grado de salario de un empleado, el cual debe estar entre cualquier par de los rangos de salario alto y bajo.

Es importante destacar que todos los empleados aparecen exactamente una vez cuando se ejecuta la consulta. Ningún empleado se repite en la lista. Hay dos razones para esto:

* Ninguna de las filas en la tabla del grado de salario contiene grados que se solapen, lo que se traduce en que el valor del salario de un empleado sólo puede estar entre el salario más bajo y el más alto de una de las filas en la tabla del grado del salario
* Todos los salarios de los empleados están dentro de los límites proporcionados por la tabla del grado del salario, lo que significa que ningún empleado gana menos del valor más bajo contenido en la columna LOSAL ni más del valor más alto contenido en la columna HISAL.

OUTER-JOINS:

Si una fila no satisface una condición de JOIN, no aparecerá en la tabla-resultado de la consulta. Por ejemplo, si relacionamos las tablas EMP y SALGRADE por la columna DEPTNO, no aparecerá el departamento OPERATIONS porque nadie trabaja en él. La(s) filas que faltan pueden ser recuperadas si en la condición de JOIN se usa un operador de OUTER-JOIN: (+). Este operador se sitúa en el “lado” del JOIN que es deficiente en información. Tiene el efecto de crear una o más filas NULL para que aquellas filas de la tabla sin valores coincidentes en la otra puedan ser combinadas.

Sintaxis:

SELECT tabla1.columna, tabla1.columna2, … tabla2.columnax

FROM tabla1, tabla2

WHERE tabla1.columnai (+) = tabla2.columnaj;

SELECT tabla1.columna, tabla1.columna2, … tabla2.columnax

FROM tabla1, tabla2

WHERE tabla1.columnai = tabla2.columnaj (+);

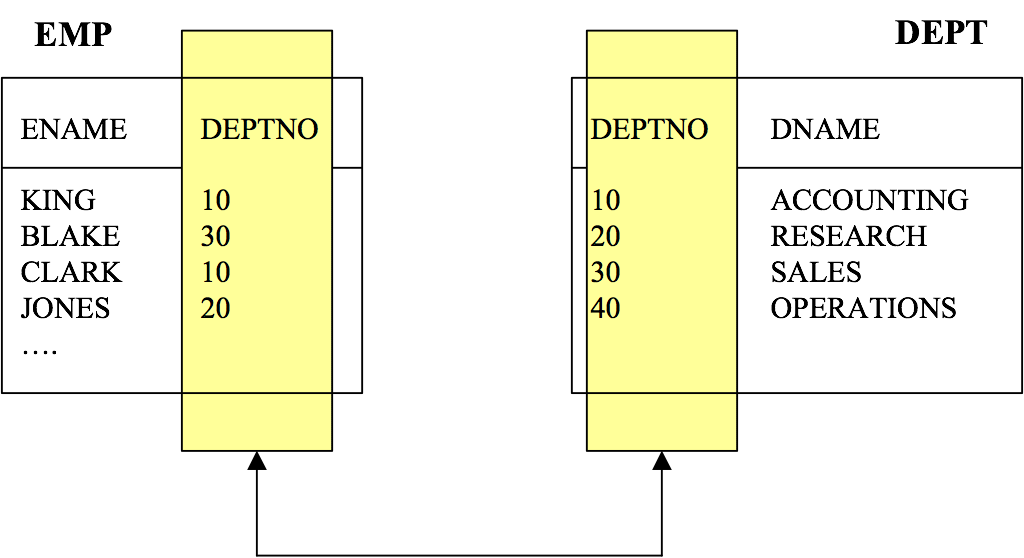
Ejemplo:

SELECT e.name, d.deptno, d.dname

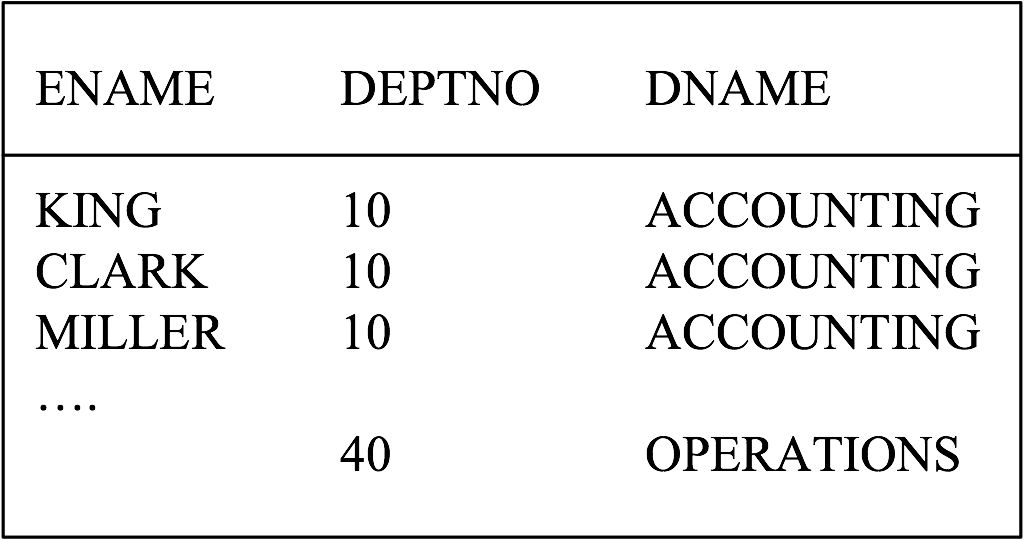
FROM emp e, dept d

WHERE e.deptno (+) = d.deptno

ORDER BY e.deptno;



Utilizando una OUTER-JOIN se produce la siguiente tabla resultado:



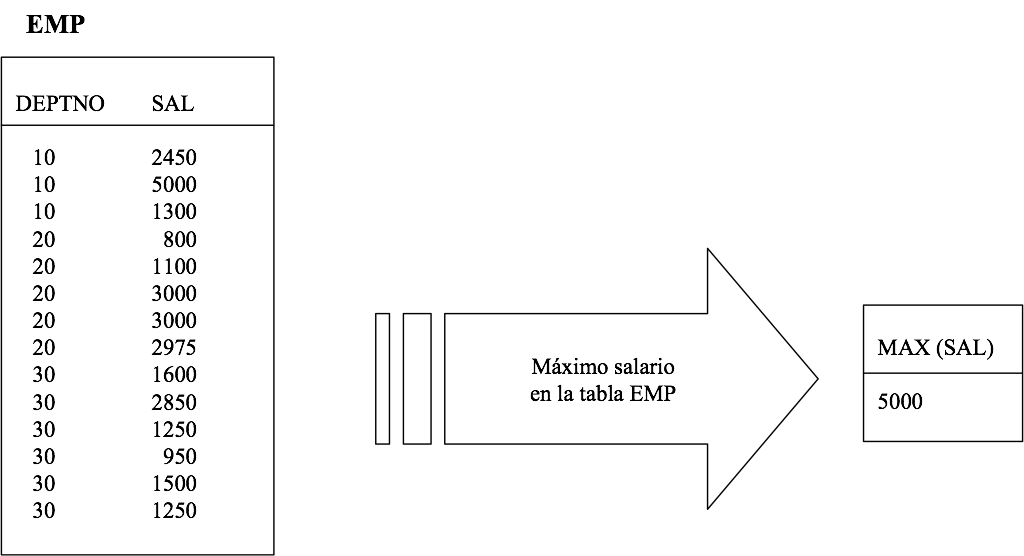
Como se observa, el departamento OPERATIONS, que no tiene ningún empleado, también se visualiza.

A la hora de utilizar una OUTER-JOIN hay que tener en cuenta las siguientes restricciones:

* El operador (+) puede aparecer sólo en un lado de la expresión, aquel lado al que le falta la información. Éste recupera aquellas filas de una tabla que no tiene correspondencia directa en otra tabla.
* Una condición que incluye un OUTER-JOIN no puede utilizar el operador IN o unirse a otra condición por el operador OR.

## Datos Agregados por medio de Funciones de Grupo

A diferencia de las funciones a nivel de fila estudiadas en apartados anteriores, las ***las funciones de grupo*** operan sobre conjuntos de filas para dar un resultado por cada uno de ellos. Dichos grupos pueden estar constituidos por la tabla entera o por partes de la misma.



**Funciones de grupo:**

Estas funciones aceptan un argumento y se clasifican en:

* **AVG ([DISTINCT|ALL] n)**: calcula el valor promedio del argumento n, ignorando los valores nulos.
* **COUNT (\* | [DISTINCT|ALL] expr)**: devuelve la cantidad de filas cuando expr no resulta en un valor nulo. Para contar las filas se usa \*, el mismo incluye filas duplicadas o aquellas que contienen valores nulos.
* **MAX ([DISTINCT|ALL] expr)**: calcula el valor máximo de la expr que se le pase.
* **MIN ([DISTINCT|ALL] expr)**: calcula el valor mínimo del argumento expr pasado.
* **SUM ([DISTINCT|ALL] n)**: devuelve la suma de los valores del argumento pasado, ignorando los nulos.
* **STDDEV ([DISTINCT|ALL] n):** calcula la desviación estándar estadística de n, ignorando los valores nulos.
* **VARIANCE ([DISTINCT|ALL] n)**: calcula la varianza estadística de n, ignorando los nulos.

**Uso de las Funciones de grupo:**

Sintaxis:

SELECT [columna, …] *función\_de\_grupo* (columna)

FROM tabla

[WHERE condición]

[GROUP BY columna1 [, columna2…]]

[ORDER BY columna1 [, columna2…]];

* El uso de DISTINCT, hace que la función no considere los valores duplicados. ALL provoca que se consideren todos los valores, incluyendo los duplicados. Por defecto, si no se especifica nada, se sobreentiende ALL.
* Los tipos de datos de los parámetros en pueden ser CHAR, VARCHAR2, NUMBER o DATE.
* Las funciones AVG, SUM, VARIANCE y STDDEV sólo pueden ser utilizadas con valores numéricos.
* Todas las funciones de grupo excepto COUNT(\*) ignoran los valores nulos. Para sustituir NULL por un valor concreto, use la función NVL.
* Oracle, por defecto, ordena el resultado en orden ascendente cuando se usa una cláusula GROUP BY. Se puede realizar la ordenación descendente incluyendo la opción DESC en la cláusula ORDER BY.

Ejemplos:

SELECT AVG(SAL), MAX(SAL), MIN(SAL), SUM(SAL)

FROM EMP

WHERE JOB LIKE ‘SALES%’;

SELECT MIN(hiredate), MAX(hiredate)

FROM EMP;

SELECT COUNT(\*)

FROM EMP

WHERE deptno = 30;

SELECT COUNT(comm)

FROM EMP

WHERE deptno = 30;

SELECT COUNT(deptno)

FROM EMP;

SELECT COUNT(DISTINCT(deptno))

FROM EMP;

SELECT AVG(comm)

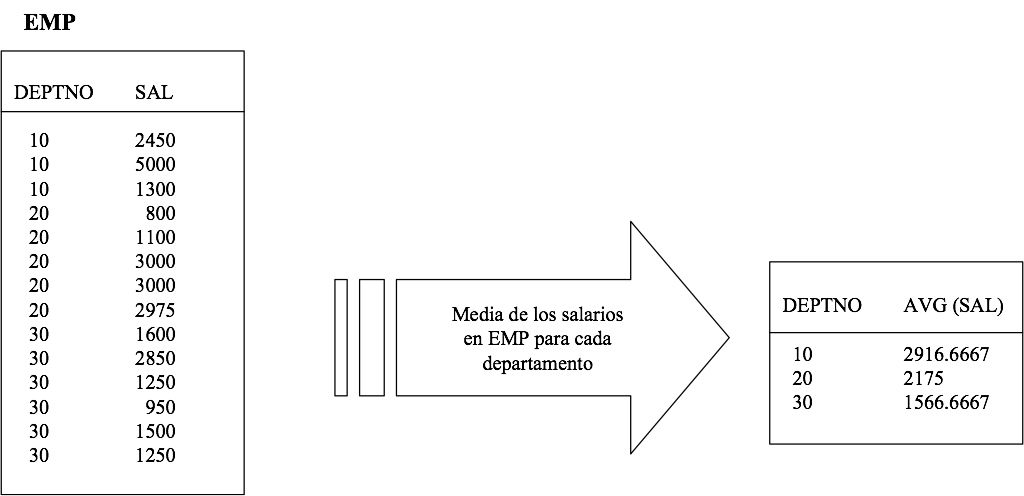
FROM EMP;

SELECT AVG(NVL(comm,0))

FROM EMP;

**La cláusula GROUP BY:**

Hasta ahora, todas las funciones de grupo trataban la tabla como un gran grupo de información. A veces, es necesario dividir la tabla de información en grupos más pequeños, asunto que se resuelve utilizando la cláusula GROUP BY.



La cláusula GROUP BY se utiliza con el propósito de repartir las filas de una tabla en grupos más pequeños. Las funciones de grupo se pueden utilizar para devolver información resumida para cada grupo.

Para utilizar esta cláusula hay que respetar unas reglas:

**Todas las columnas mendionadas en la SELECT que no son funciones de grupo, tienen que estar en la cláusula GROUP BY**

* Si se incluye una función de grupo en una cláusula SELECT, no se pueden seleccionar resultados individuales a menos que la columna aparezca en la cláusula GROUP BY. Si no se incluyen las columnas correspondientes en la lista, se recibirá un mensaje de error.
* Con el uso de la cláusula WHERE se pueden excluir filas antes de la división en grupos.
* No se pueden utilizar alias de columna en la cláusula GROUP BY.
* Por defecto, las filas se ordenan en forma ascendente de acuerdo a la lista GROUP BY. Puede modificarse con ORDER BY.

Ejemplo:

SELECT deptno, AVG (sal)

FROM emp

GROUP BY deptno;

SELECT AVG (sal)

FROM emp

GROUP BY deptno;

SELECT deptno, AVG (sal)

FROM emp

GROUP BY deptno

ORDER BY AVG(sal);

Algunas veces, es necesario ver los resultados de grupos dentro de grupos. Esto se realiza incluyendo más de una columna en la cláusula GROUP BY.

Ejemplo:

SELECT deptno, job, SUM (sal)

FROM emp

GROUP BY deptno, job;

Ejemplos de consultas no válidas que utilizan funciones de grupo:

SELECT deptno, COUNT (ename)

FROM emp;

ERROR >> columna no especificada en la cláusula GROUP BY

SELECT deptno, COUNT (ename)

FROM emp

GROUP BY deptno;

CORRECTO

SELECT deptno, AVG (sal)

FROM emp

WHERE AVG (sal) > 2000

GROUP BY deptno;

ERROR >> no puede usar la cláusula WHERE para restringir grupos

SELECT deptno, AVG (sal)

FROM emp

GROUP BY deptno

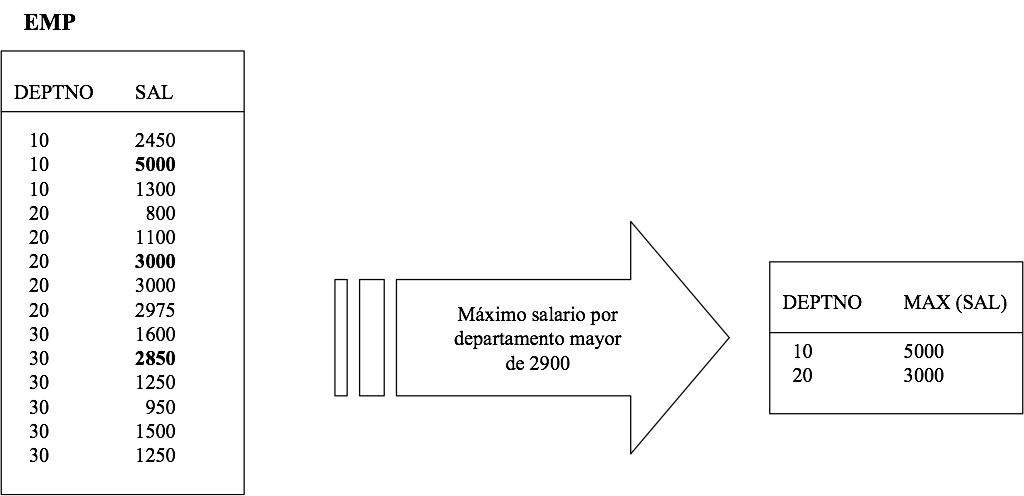
HAVING AVG (sal) > 2000;

CORRECTO

**Cláusula HAVING o Exclusión de resultados de un grupo:**

De la misma manera que se utiliza la cláusula WHERE para restringir los registros que se selecciona, se utiliza la cláusula HAVING para restringir grupos.

Por ejemplo, se desea localizar el máximo salario de cada departamento pero mostrar sólo aquellos cuyo máximo es superior a 2900. Para ello habrá que averiguar el máximo salario de cada departamento agrupando por nº de departamento y limitar los grupos a aquellos departamentos que superen 2900 su salario máximo.



Sintaxis:

SELECT [columna, …] *función\_de\_grupo* (columna)

FROM tabla

[WHERE condición]

[GROUP BY columna1 [, columna2…]]

**[HAVING condición\_de\_grupo]**

[ORDER BY columna1 [, columna2…]];

Ejemplos:

SELECT deptno, MAX (sal)

FROM emp

GROUP BY deptno

HAVING MAX (sal) > 2900;

SELECT job, SUM (sal)

FROM emp

WHERE job NOT LIKE ‘SALES%’

GROUP BY job

HAVING SUM (sal) > 5000

ORDER BY SUM (sal);

SELECT MAX (AVG (sal))

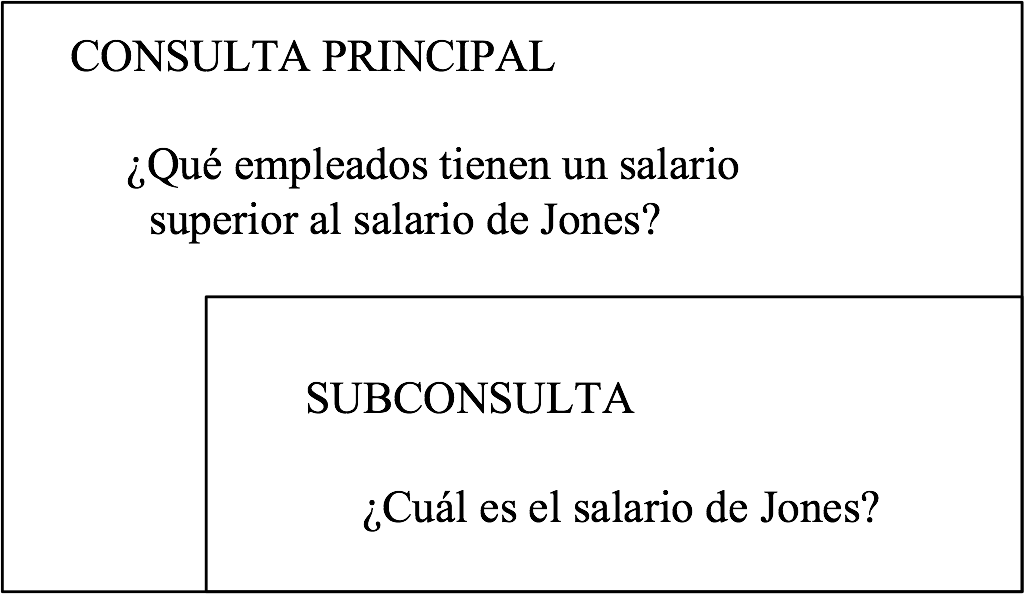
FROM emp

GROUP BY deptno;

## Subconsultas

Supongamos que se quiere escribir una consulta para localizar aquellos empleados que gana un salario superior al de Jones. Para realizar este problema, se necesitan 2 consultas:

* Una que averigüe el salario de Jones
* Otra que encuentre a los empleados que ganan más de esta cantidad



La consulta más interna o subconsulta devuelve una valor que será utilizado por la consulta principal (la consulta más externa). Usar una subconsulta es equivalente a realizar 2 consultas secuencialmente y utilizar el resultado de la primera que se ejecuta (subconsulta) como valor para la segunda consulta que se ejecuta o consulta principal.

Sintaxis:

SELECT columna1, ..., columnax

FROM tabla

WHERE expresión operador (SELECT columna1, ..., columnaz

FROM tabla);

Una subconsulta es una sentencia SELECT que está incluida en una cláusula de otra sentencia SQL. Se pueden construir comandos simples y potentes usando subconsultas. Pueden ser útiles cuando se necesita seleccionar filas de una tabla con una condición que depende de los datos que están en la misma tabla.

Se puede poner una subconsulta en cláusulas WHERE, HAVING y FROM de una SELECT o un DELETE.

El operador que precede a la subconsulta es un operador de comparación.

Ejemplo:

SELECT ename

FROM emp

WHERE sal > (SELECT sal

FROM emp

WHERE empno = 7566);

Para usar subconsultas se deben contemplar unas reglas:

* Encierre las subconsultas entre paréntesis
* Coloque la subconsulta a la derecha del operador
* No agregar una cláusula ORDER BY a una subconsulta. Se puede tener solamente una cláusula ORDER BY para una sentencia SELECT y específicamente debe ser la última cláusula en la sentencia SELECT principal.
* Para las subconsultas se utilizan 2 clases de operadores: a nivel de fila para subconsultas que devuelven sólo una fila (Operadores mono-registro) y a nivel de grupo para subconsultas que devuelven más de una fila (Operadores multi-registro).

|  |  |
| --- | --- |
| **OPERADORES MONO-REGISTRO** | **OPERADORES MULTI-REGISTRO** |
| =  >  >=  <  <=  <> | IN: igual a los valores de cierta lista  ANY: compara los valores con cada valor devuelto por la subconsulta  ALL: compara los valores con cada uno de los valores devueltos por la subconsulta |

Ejemplos de subconsultas mono-registro:

SELECT ename, job

FROM emp

WHERE job = (SELECT job

FROM emp

WHERE empno = 7369);

SELECT ename, job

FROM emp

WHERE job = (SELECT job

FROM emp

WHERE empno = 7369)

AND sal > (SELECT sal

FROM emp

WHERE empno = 7876);

SELECT ename, job, sal

FROM emp

WHERE sal = (SELECT MIN(sal)

FROM emp);

SELECT deptno, MIN (sal)

FROM emp

GROUP BY deptno

HAVING MIN (sal) > (SELECT MIN (sal)

FROM emp

WHERE deptno = 20);

Puesto de trabajo con la media más baja de salarios:

SELECT job, AVG (sal)

FROM emp

GROUP BY job

HAVING AVG (sal) = (SELECT MIN (AVG (sal))

FROM emp

GROUP BY job);

Consultas erróneas:

SELECT empno, ename

FROM emp

WHERE sal = (SELECT MIN(sal)

FROM emp

GROUP BY deptno);

>> ERROR: la subconsulta devuelve más de una fila (el operador = sólo

admite un valor). Se resolvería si se utilizara IN (operador multiregistro).

SELECT ename, job

FROM emp

WHERE job = (SELECT job

FROM emp

GROUP BY ename = ‘SMYTHE’);

>> No se devuelve ninguna fila pues SMYTHE no existe. La consulta principal recibe un NULL de la subconsulta y no existe ningún empleado

que tenga empleo nulo.

Ejemplos de subconsultas multi-registro:

SELECT ename, sal, deptno

FROM emp

WHERE sal IN (SELECT MIN (sal)

FROM emp

GROUP BY deptno);

SELECT empno, ename, job

FROM emp

WHERE sal < ANY (SELECT sal

FROM emp

WHERE job = ‘CLERK’);

Nota:

< ANY significa menos que el máximo

> ANY significa más que el mínimo

= ANY es equivalente a IN

SELECT empno, ename, job

FROM emp

WHERE sal > ALL (SELECT AVG (sal)

FROM emp

GROUP BY deptno);

Nota:

> ALL significa más que el máximo

> ALL significa menos que el mínimo

El operador NOT puede ser utilizado con los operadores IN, ANY y ALL.

## 5. OPERADORES DE UNION, INTERSECT Y MINUS

* Son operadores de conjuntos , por tanto se utilizaran entre dos select que serán las encargadas de seleccionar los dos conjuntos

Select… from …where

Operador\_de\_conjunto

Select …from..where

* UNION: Combina los resultados de las dos consultas. Elimina los valores repetidos

Select… from …where

UNION

Select …from..where

Select nombre from alumnos UNION select nombre from nuevos;

* UNION ALL : Visualiza también las filas duplicadas
* INTERSECT: Devuelve las filas que sean comunes en ambas tablas

Select… from …where

INTERSECT

Select …from..where

* MINUS: Devuelve las filas que están en la primera select y no están en la segunda

Select… from …where

MINUS

Select …from..where

A veces los programadores prefieren utilizar los operadores IN,AND, OR en lugar de los vistos anteriormente.

Select nombre from alumnos where nombre in (select nombre from nuevos)

En este caso anterior estoy haciendo una intersección

* Reglas de utilización de operadores de conjuntos
  1. Las columnas de las dos consultas se relacionan en orden , de izquierda a derecha
  2. Los nombres de columna de la primera select no tiene porque ser los mismos que los nombres de la segunda select
  3. La select necesitan tener el mismo número de columnas
  4. Los tipos de datos deben coincidir, aunque la longitud no tiene porque ser la misma.

## 6.Manipulación de datos

El lenguaje de manipulación de datos o DML (*Data Manipulation Language*) es una parte esencial de SQL. Cuando se quiere agregar, actualizar o eliminar datos, se ejecuta una sentencia DML. Un conjunto de sentencias DML que aún no se han hecho permanentes se denomina **transacción** o unidad lógica de trabajo.

Imagine una base de datos de un banco. Cuando un cliente del banco transfiere dinero de una cuenta a otra, la transacción consiste en tres operaciones separadas: disminuir el saldo, incrementar el de la otra cuenta y registrar la transacción. Oracle debe garantizar que las tres sentencias SQL se ejecutan para mantener las cuentas con el saldo adecuado. Cuando una de las sentencias no puede ejecutarse, el resto de las sentencias deben deshacerse.

## La sentencia INSERT

Permite añadir nuevos registros a una tabla.

Sintaxis:

INSERT INTO tabla [columna [, columna ...])]

VALUES (valor [, valor ...]);

Mediante esta sintaxis, sólo se inserta un registro o fila al mismo tiempo.

Ejemplo:

INSERT INTO DEPT (deptno, dname, loc)

VALUES (50, ‘DESARROLLO’, ‘MADRID’);

Es preciso listar los valores en el orden por defecto de las columnas de la tabla.

Opcionalmente se pueden listar las columnas en las cláusula INSERT.

Si se desean insertar valores nulos, se pueden utilizar dos métodos:

* Método implícito: omitir la columna en la lista.

INSERT INTO DEPT (deptno, dname)

VALUES (60, ‘MI DEPARTAMENTO’);

* Método explícito: especificar la palabra clave NULL.

INSERT INTO DEPT

VALUES (60, ‘MI DEPARTAMENTO’, NULL);

Antes hay que asegurarse que la columna permite valores nulos (comprobarlo con DESCRIBE).

Ejemplos:

INSERT INTO EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

VALUES (7196, ‘HIGUERAS’, ‘SALESMAN’, 7782, SYSDATE, 2000, NULL, 10);

INSERT INTO EMP

VALUES (2296, ‘ORTEGA’, ‘SALESMAN’, 7782,

TO\_DATE (‘FEB 3, 97’, ‘MON DD, YY’), 1300, NULL, 10);

**Copia de registros de otra tabla**

Se pueden usar la sentencia INSERT para agregar filas a una tabla donde los valores se derivan de otras tablas ya existentes. En lugar de la cláusula VALUES, utilizamos una subconsulta.

INSERT INTO JEFES (id, nombre, salario, alta)

SELECT empno, ename, sal, hiredate FROM EMP WHERE job = ‘MANAGER’;

Cuidar la concordancia de columnas de la cláusula INSERT y las de la subconsulta.

## La sentencia UPDATE

Permite modificar los registros existentes. Puede afectar a más de un registro al mismo tiempo.

Sintaxis:

UPDATE tabla

SET columna = valor [, columna = valor]

[WHERE condición];

Los registros a modificar se especifican en la cláusula WHERE.

UPDATE EMP

SET deptno = 20

WHERE empno = 7782;

Si se omite la cláusula WHERE, se modificarán **todos** los registros de la tabla.

UPDATE EMPLEADOS

SET deptno = 20;

Ejemplos:

UPDATE EMP

SET (job, deptno) = (SELECT job, deptno

FROM EMP

WHERE empno =7499)

WHERE empno = 7698;

UPDATE EMP

SET deptno = 55

WHERE deptno = 10;

SQL>> ERROR: violación de restricción de integridad (el departamento 55 no existe en la tabla padre DEPT).

## La sentencia DELETE

Permite borrar registros de una tabla.

Sintaxis:

DELETE [FROM] tabla

[WHERE condición];

Los registros a eliminar se especifican en la cláusula WHERE.

DELETE FROM DEPARTAMENTOS

WHERE dname = ‘DEVELOPMENT’;

Si se omite la cláusula WHERE se borrarán **todos** los registros de la tabla.

DELETE FROM DEPARTAMENTOS;

Ejemplos:

DELETE FROM EMPLEADOS

WHERE deptno = (SELECT deptno

FROM DEPT

WHERE dname = ‘SALES’);

DELETE FROM DEPT

WHERE deptno = 10;

SQL>> ERROR: violación de restricción de integridad (no se puede eliminar un registro que contiene una clave primaria, usada como clave extranjera en otra tabla).

## 7.Transacciones en la base de datos

El servidor Oracle asegura la consistencia de los datos basándose en transacciones. Las transacciones otorgan más flexibilidad y control cuando cambian los datos y aseguran en los datos ante un fallo eventual en el proceso del usuario o un fallo del sistema.

Las transacciones consisten de comandos DML que llevan a cabo un cambio en los datos de manera consistente. Por ejemplo, una transferencia de fondos entre dos cuentas, debiera incluir un crédito en una cuenta y un débito en la otra por la misma cantidad. Ambas acciones debieran tener éxito o fallar como una sola operación. El crédito no debería confirmarse sin el débito correspondientes.

Tipos de transacciones:

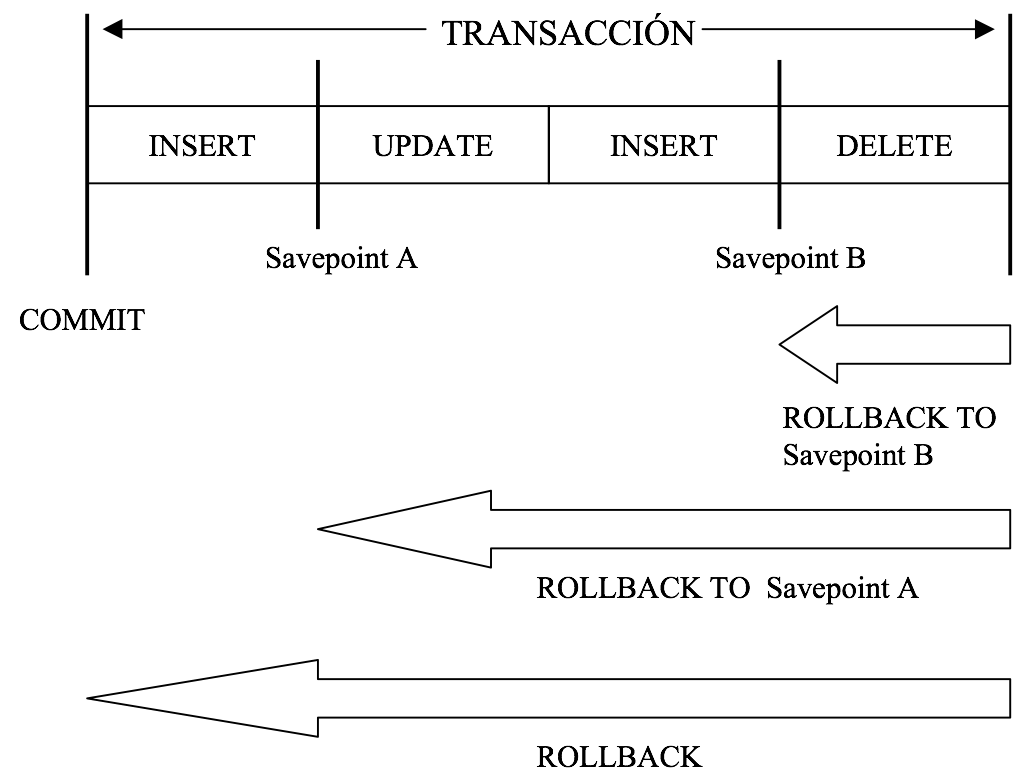
* Un conjunto de sentencias DML que Oracle tratará como una unidad simple
* Una única sentencia DDL
* Una única sentencia DCL

Las transacciones comienzan cuando se ejecuta el primer comando SQL.

Las transacciones finalizan con uno de los siguientes eventos:

* COMMIT o ROLLBACK
* Ejecución de un comando DCL o DCL (COMMIT automático)
* Salida del usuario
* Caída del sistema

Las sentencias COMMIT y ROLLBACK aseguran la consistencia de los datos, permiten visualizar los cambios sobre los datos antes de hacerlos permanentes y agrupan lógicamente tareas relacionadas entre sí.



* *COMMIT*: finaliza la transacción actual haciendo permanente todos los cambios
* *SAVEPOINT nombre*: Marca un Savepoint en la transacción actual
* *ROLLBACK [TO SAVEPOINT nombre]*: ROLLBACK finaliza la transacción actual deshaciendo todos los cambios pendientes. ROLLBACK TO SAVEPOINT deshace la transacción actual hasta el Savepoint especificado, deshaciendo por lo tanto el Savepoint y todos los cambios posteriores. Si se omite esta cláusula, la sentencia ROLLBACK deshace la transacción entera.

Se produce un COMMIT automático cuando se aplica un comando DDL o DCL o una salida normal desde SQL\*Plus (sin realizar un COMMIT o ROLLBACK explícitamente) con el comando quit.

SQL\*Plus dispone de un tercer comando: AUTOCOMMIT, que puede tomar valores ON u OFF. Si es ON, cada sentencia DML individual es confirmada (se le hace COMMIT) tan rápido como es ejecutada. Los cambios no son reversibles (sin ROLLBACK).

Se produce un ROLLBACK automático cuando ocurre una terminación anormal de SQL\*Plus o un fallo del sistema. Esto previene que el error cause cambios no deseados a los datos y devuelve a las tablas su estado inicial desde el último COMMIT. De esta manera, SQL\*Plus protege la integridad de las tablas.

**Estado de los datos antes de COMMIT o ROLLBACK**

* El estado previo de los datos puede ser recuperado.
* El usuario actual puede revisar los resultados de sus operaciones DML usando la sentencia SELECT.
* Otros usuarios no pueden ver los resultados de las sentencias DML ejecutadas por el usuario actual.
* Las filas afectadas son bloqueadas; otros usuarios no pueden cambiar los datos pertenecientes a esas filas.

**Estado de los datos después de COMMIT**

* Los cambios se hacen permanentes en la base de datos.
* Los datos anteriores se pierden definitivamente.
* Todos los usuarios pueden ver los resultados.
* Se liberan los bloqueos aplicados a las filas afectadas; esas filas están ahora disponibles para que otros usuarios las manipulen.
* Se borran todos los SAVEPOINTS.

Ejemplos:

UPDATE EMP SET deptno = 10 WHERE empno = 7782;

INSERT INTO DEPT VALUES (60, ‘MARKETING’, ‘BARCELONA’);

COMMIT;

DELETE FROM TEST;

ROLLBACK;

DELETE FROM TEST WHERE ID = 100;

SELECT \* FROM TEST WHERE ID = 100;

COMMIT;

**Hacer ROLLBACK hasta una marca**

Se puede crear una marca en una transacción usando el SAVEPOINT. En consecuencia, la transacción puede dividirse en dos secciones más pequeñas. Esto permite que se descarten los cambios hechos hasta la marca usando la sentencia ROLLBACK TO SAVEPOINT.

Si se crea un segundo SAVEPOINT con el mismo nombre que un SAVEPOINT anterior, este último es eliminado.

Ejemplo:

UPDATE DEPARTAMENTOS SET loc = ‘MIAMI’ WHERE deptno = 40;

SAVEPOINT update\_hecho;

INSERT INTO DEPARTAMENTOS VALUES (80, ‘PUBLICIDAD’, ‘PARIS’);

ROLLBACK TO update\_hecho;

* Si una única sentencia DML falla durante su ejecución, entonces se hace ROLLBACK sobre esa sentencia solamente.
* El usuario debería terminar explícitamente las transacciones usando una sentencia COMMIT o ROLLBACK.

**Consistencia en Lectura**

Los usuarios de la base de datos realizan dos tipos de acceso a la base de datos:

* Operaciones de lectura (sentencia SELECT)
* Operaciones de escritura (sentencias INSERT, UPDATE, DELETE)

La consistencia en lectura es necesaria para que:

* Las lecturas y escrituras a la base de datos aseguren una vista consistente de los datos
* Las lecturas no vean datos que están en proceso de cambio
* Las escrituras aseguren que los cambios a la base de datos se hacen de forma consistente
* Los cambios realizados por una escritura no crean conflictos con los cambios que otra escritura está realizando

El propósito de la consistencia en lectura es asegurar que cada usuario ve los datos tal como están desde el último COMMIT, antes de que comience una operación DML.

**Implementación de la consistencia en lectura**

La consistencia en lectura es una implementación automática. Mantiene una copia parcial de la base de datos en segmentos de rollback. Cuando una operación DML se realiza contra la base de datos, el servidor Oracle hace una copia de los datos antes de su cambio y la escribe en un segmento de rollback.

Todas las lecturas, excepto aquella que realiza el cambio, ven la base de datos tal como estaba; ven la imagen de los segmentos de rollback, que es como una foto de los datos.

Antes de hacer el COMMIT de los cambios en la base de datos, sólo el usuario que modifica los datos ve los cambios; el resto lee la imagen del segmento de rollback. Esto garantiza que las lecturas son consistentes. Al hacer COMMIT, el cambio realizado a la base de datos se convierte en visible a cualquiera que ejecute una sentencia SELECT.

Si la transacción experimenta un rollback, los cambios “se deshacen”:

* La versión original, antigua de los datos, en el segmento de rollback se escriben de nuevo a la tabla
* Todos los usuarios ven la base de datos tal como estaba antes de comenzar la transacción.

**Bloqueos**

Los bloqueos son mecanismos que previenen conflictos entre transacciones, que acceden a los mismos recursos, bien sea un objeto de usuario (como tablas o registros), o un objeto del sistema no visible a los usuarios (como estructuras de datos compartidas y registros del diccionario de datos).

El bloqueo en Oracle es completamente automático y no requiere acción por parte del usuario. El bloqueo implícito ocurre para todas las sentencias SQL. El bloqueo por defecto de Oracle es un mecanismo automático que utiliza el nivel más bajo aplicable de restricción, portano, ofrece el grado más alto de concurrencia y máxima integridad de datos. Oracle también permite que el usuario bloquee los datos manualmente.

Oracle utiliza dos modelos de bloqueo:

* Exclusivo: previene la compartición de un recurso. La primera transacción bloquea un recurso exclusivamente, es la única que puede alterarlo, hasta liberar el recurso.
* Compartido: permite la compartición de un recurso. Múltiples usuarios leyendo datos pueden compartir los datos, manteniendo bloqueos para prevenir acceso concurrente por una escritura (que necesita un bloqueo exclusivo). Varias transacciones pueden adquirir bloqueos compartidos sobre el mismo recurso.