materiales didácticos de aula
formación profesional específica

Desarrollo de Aplicaciones Informáticas

CICLO FORMATIVO DE GRADO SUPERIOR

FORMACIÓN PROFESIONAL A DISTANCIA

Unidad 3

SQL y el modelo relacional - LMD

MÓDULO

Desarrollo de Aplicaciones en Entornos de Cuarta Generación y con Herramientas CASE

FORMACIÓN PROFESIONAL
Principado de Asturias

Título del Ciclo: DESARROLLO DE APLICACIONES INFORMATICAS

Título del Módulo: DESARROLLO DE APLICACIONES EN ENTORNOS DE CUARTA GENERACIÓN Y CON HERRAMIENTAS CASE

Unidad 3: **SQL y el modelo relacional - LMD**

Dirección: Dirección General de Políticas Educativas, Ordenación Académica y Formación Profesional
Servicio de Formación Profesional Inicial y Aprendizaje Permanente

Dirección de la obra:
Alfonso Careaga Herrera
Antonio Reguera García
Arturo García Fernández
Ascensión Solís Fernández
Juan Carlos Quirós Quirós
Luís Mª Palacio Junquera
Yolanda Álvarez Granda

Coordinador de los contenidos:

Juan Manuel Fernández Gutiérrez

Autores:
Juan Manuel Fernández Gutiérrez
Rodrigo Fernández Martínez

Colección:
Materiales didácticos de aula

Serie:
Formación profesional especifica

Edita:
Consejería de Educación y Ciencia

ISBN: 978-84-692-3443-2

Deposito Legal: AS-3564-2009

Introducción

En las unidades didácticas anteriores hemos representado la información mediante el **modelo relacional**. ¿De qué manera podemos almacenar y tratar la información esquematizada en el modelo relacional por medio de un Sistema Gestor de Bases de Datos? La respuesta a esta pregunta se llama **SQL**.

En esta unidad se explican los fundamentos del SQL estándar, cómo utilizarlo de forma interactiva, y se sientan las bases para su utilización en PL/SQL. Es importante tener una cierta soltura en el manejo de SQL, pues nos permite obtener de forma rápida cualquier información que tengamos en las tablas de la base de datos, no solo en las tablas del usuario, sino también en el diccionario de datos.

El lenguaje SQL lo puede utilizar el usuario final como un recurso eficaz para obtener datos de diferentes tablas, el programador para confeccionar los módulos PL/SQL y el administrador para acceder a las tablas del diccionario y así poder tomar decisiones que optimicen la explotación de la base de datos.

Objetivos

- Conocer en qué consiste el estándar SQL como lenguaje para acceder a los datos de las tablas.
- Comprender los distintos objetos que constituyen el lenguaje SQL.
- Utilizar las sentencias, operadores y funciones que constituyen la base de SQL.
- Trabajar con múltiples tablas y vistas.

Contenidos Generales

1. Historia y evolución

Las primeras implementaciones en SQL se realizaron en 1975, con el nombre de SEQUEL. En 1979 ORACLE implementa el primer sistema comercial de gestión de bases de datos relacionales basado en SQL , posteriormente aparece otros sistemas también basados en SQL como DB2, SYBASE, INTERBASE, INFORMIX, etc, y otros productos que no lo poseían como lenguaje ofrecen un interface con el mismo, por ejemplo, INGRES, DATACOM, ADABAS, etc.

El comité de bases de datos X3H2 de ANSI, aprueba en 1986 una normalización del lenguaje pasando a denominarse SQL/ANS que también es aprobado al año siguiente como norma ISO. En un principio esta normalización es criticada por algunos autores, aduciendo que fallan numerosas características que los usuarios necesitan y que se contemplan en el modelo relacional.

En 1989 se publica una nueva versión del estándar ISO que añade ciertas reglas de claves ajenas que permiten definir la opción de modificación y borrado restringido. Por otra parte ANSI define ese mismo año un estándar para el lenguaje SQL embebido. En 1992 se aprueba una nueva versión de SQL conocida como SQL2 o SQL-92 , en la que se incrementa la capacidad semántica del esquema relacional y se añaden nuevos operadores, también se incluyen normas para el SQL embebido.

Aunque SQL-92 ha mejorado considerablemente la capacidad semántica (por ejemplo con la definición de dominios, como se describe en la Unidad 2) aún ha de evolucionar para adaptarse a las nuevas tendencias, como es la orientación a objetos. En este sentido, los comités ANSI/ISO están trabajando en la definición de una nueva versión del estándar SQL denominada SQL3 que amplia la capacidad semántica (por ejemplo, con la incorporación de disparadores definidos con la sentencia CREATE TRIGGER) e incluye bastantes de las características de los lenguajes orientados a objetos como son los tipos abstractos de datos, encapsulamiento, herencia, polimorfismo, métodos. Está previsto que SQL3 se normalice en 1999, y ya se está hablando de SQL4 que incluiría aquellas características del lenguaje que necesiten una mayor profundización.

SQL (Structured Query Language) es un lenguaje que permite realizar una serie de operaciones en bases de datos relacionales. Se compone de tres partes o sublenguajes, el LDD

o lenguaje de definición de datos, LMD o lenguaje de manipulación de datos, LCD o lenguaje de control de datos.

Como lenguaje de manipulación relacional es un lenguaje de especificación que permite seleccionar un conjunto de filas sobre las que se va a realizar una acción recuperación o actualización).

Su principal característica es la de ser un estándar en la manipulación y definición de las tablas de los sistemas gestores de bases de datos relacionales. No es un lenguaje en si mismo, en el sentido que se pueda adquirir independientemente, forma parte del núcleo del sistema gestor de la base de datos. Es utilizado por usuarios finales, interactivamente o bien a través de herramientas que lo incorporan, por programadores también de forma interactiva o embebido en lenguajes de tercera o cuarta generación, y por los administradores para crear los objetos de la base (tablas, usuarios, etc) conceder permisos, interrogar el diccionario de datos, etc.

Otra característica es su portabilidad a través de diferentes sistemas, incluso se puede utilizar como pasarela cuando en una red se dispone de diferentes productos de bases de datos.

En resumen el SQL estándar no es un lenguaje completo como el COBOL, PASCAL…., no dispone de sentencias propias de los lenguajes mencionados como IF, PERFORM, FOR …, pero si se le puede incorporar como lenguaje embebido a otro lenguaje denominado anfitrión para permitir el acceso a través de dichos lenguajes a las bases de datos. En algunos casos se le añaden las estructuras necesarias para convertirlo en un lenguaje específico de una herramienta como es el caso del PL/SQL.

Resumiendo, con SQL podemos:

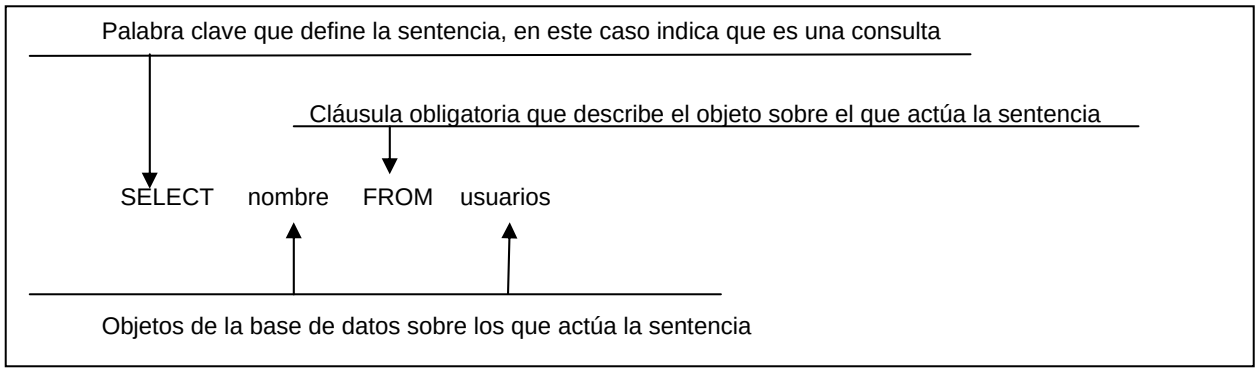
- Definir y eliminar objetos
- Conceder y denegar autorizaciones
- Consultar y actualizar datos.

2. Características del lenguaje SQL

En lo que sigue vamos a describir los distintos elementos de SQL según el estándar ANSI/ISO, pero debemos advertir que los productos comerciales que implementan el estándar suelen incorporar algunas extensiones y variaciones con respecto a él, y que nosotros nos referiremos siempre al SQL implementado por Oracle.

2.1. Sentencias SQL

Las sentencias SQL se componen de una palabra clave, que también podemos llamar palabra reservada, que las definen y de una o varias cláusulas (algunas obligatorias otras opcionales) que túan sobre los objetos o elementos de la base de datos. Como en cualquier otro lenguaje se debe evitar utilizar estas palabras reservadas para nombrar otros objetos de la base de datos



Teniendo en cuenta las tres partes o sublenguajes de SQL podemos hablar de:

Sentencias de manipulación de datos.

Permiten manipular y consultar la información contenida en las tablas. Las consultas son las operaciones más realizadas a través del lenguaje SQL, muchos usuarios utilizan SQL únicamente como herramienta de consulta. (usuarios finales, programadores y administradores). Las siguientes sentencias pertenecen a este grupo:

SELECT	recupera datos
INSERT	añade nuevas filas
DELETE	elimina filas
UPDATE	modifica el contenido de las filas

Sentencias de definición de datos.

Permiten crear y borrar tablas y vistas así como modificar la estructura de las tablas. Estas sentencias, básicamente, las utilizan los administradores y los programadores. Son sentencias de este grupo:

CREATE	crea tablas y vistas
DROP	borra tablas, vistas, índices…
ALTER	modifica la estructura de una tabla

Sentencias de control de datos.

Se utilizan para gestionar la confidencialidad, por ejemplo las sentencias GRANT y REVOKE, y para gestionar las transacciones, como COMMIT y ROLLBACK.

Algunos autores incluyen las sentencias de control junto con las de definición de datos. No tiene mayor importancia que se incluyan en el grupo de definición o que constituyan un grupo aparte, es una mera clasificación.

2.2. Tipos de datos.

El estándar ANSI/ISO define una serie de tipos de datos que suelen ser un subconjunto de los que incorporan la mayoría de las bases de datos, en nuestro caso, enumeraremos los tipos de datos del estándar y luego trabajaremos con los tipos que incorpora Oracle.

Los tipos de datos los podemos agrupar en: numéricos y alfanuméricos.

Numéricos: Pueden contener números con o sin punto decimal y signo.

- Números enteros	INTEGER	
- Números enteros pequeños	SMALLINT	
- Números decimales	DECIMAL(p,s)	p=precisión s=escala
	NUMERIC(p,s)	
- Números en coma flotante	FLOAT(p)	

- De baja precisión REAL
- De alta precisión DOUBLE PRECISION

Alfanuméricos: Almacenan cadenas de caracteres.

- Cadena de longitud fija CHAR(n) n=longitud
  CHARACTER(n)
- Cadena de longitud variable CHARACTER VARYING(n)

Tipos de datos en Oracle

La mayor parte de las bases de datos tienen un conjunto de datos más amplio que los especificados por ANSI/ISO. En el siguiente cuadro definimos los tipos de datos de Oracle que vamos a utilizar en esta Unidad.

Tipo	Descripción
VARCHAR2(n)	Cadena de longitud variable.
CHAR(n)	Cadena de longitud fija, *n* es opcional si se omite el asume 1 byte. La máxima longitud es de 255 bytes
LONG	Cadena de longitud variable, 2 GB como máximo
RAW(n)	Información binaria, la longitud máxima de *n* es de 255 bytes
LONG RAW	Información binaria, hasta dos gigabytes, no se pueden indexar las columnas de este tipo. Se utiliza para almacenar información gráfica, sonido, etc
NUMBER(p,s)	Datos numéricos, donde *p* es la precisión con un rango desde 1 hasta 38 y *s* es la escala con rango desde –84 hasta 127
DATE	Almacena datos tipo fecha y hora
CLOB	Cadena de longitud variable, a partir de 4 GB como máximo
BLOB	Datos binarios no estructurados, 4 GB como máximo
BFILE	Datos binarios almacenados en archivos externos a la base, 4GB como máximo

2.3. Operadores, expresiones y funciones.

A continuación se describen los operadores aritméticos, de comparación, lógicos , de cadena y algunas funciones que iremos utilizando en el desarrollo de los siguientes puntos al tiempo que describimos las sentencias.

Si el lector está utilizando una base de datos que no sea Oracle, puede que algún operador o función de las que se describe no se ejecute correctamente; en este caso debe de consultar el manual del SQL para su base de datos.

Los operadores se usan para manipular datos y devolver un resultado. La combinación de operadores aritméticos, valores numéricos y funciones numéricas dan lugar a expresiones aritméticas. Si la combinación es de operadores de cadena, funciones de cadena y constantes o variables alfanuméricas, se obtienen expresiones de cadena o alfanuméricas. Por último si intervienen operadores de relación y/o lógicos, las expresiones son condicionales y el resultado es un único valor TRUE o FALSE.

Operadores aritméticos por orden de precedencia	**Descripción**	**Ejemplos**
+ -	Proporciona carácter positivo o negativo a una expresión	- 1
* /	Multiplicación y división	PTS * HORAS
+ -	Sumar y restar	SUELDO + COMISION

Operadores comparación	Descripción	Ejemplos
=	Comprobación de igualdad	SUELDO = 10000
!= <>	Distinto de…..	SUELDO <> 5000
>	Mayor que….	SUELDO > COMISION
<	Menor que…	COMISION < SUELDO
>=	Mayor o igual	SUELDO >= PRIMAS
<=	Menor o igual	SUELDO <= 400000
IN	Igual a cualquier elemento de la lista, si lo encuentra evalúa TRUE	**IN** ('ANALISTA','OPERADOR')
NOT IN	Evalúa FALSE si está en la lista	**NOT IN** (lista de valores)
ANY, SOME	Se compara con cada uno de los valores de una lista. Deben de ir precedidos de alguno de los siguientes operadores =, !=, >, <, <=, >= El operador IN equivale =ANY Evalúa TRUE cuando cumple la condición con cualquiera de los elementos de la lista.	SUELDO = ANY (100, 200) ó SUELDO = SOME(100, 200) La lista se puede sustituir por los valores devueltos por una consulta
ALL	Compara un valor con cada uno de la lista o los devueltos por una consulta. Debe ir precedido por uno de los siguiente operadores =, !=, >, <, <=, >=	SUELDO >= **ALL** (1000,2000)

	Evalúa TRUE cuando cumple la condición para todos y cada uno de los elementos de la lista.	
[NOT] BETWEEN ***A*** AND ***B***	Evalúa TRUE si el valor está comprendido entre A y B ambos inclusive	SUELDO **BETWEEN** 2000 **AND** 3000
EXISTS	Evalúa TRUE si devuelve alguna fila	Pendiente de ver las consultas **SELECT**
A [NOT] LIKE ***B***	Evalúa TRUE si A coincide con B, siendo B una constante que puede incorporar los caracteres comodin % que sustituye cualquier cadena y _ que sustituye un carácter	NOMBRE **LIKE** 'MAR%' Devuelve **TRUE** si NOMBRE contiene un valor que comience por MAR (MARIA, MARIANA, …)
IS [NOT] NULL	Testea si un item es o no nulo	DIRECCION **IS NULL**

Operadores lógicos	**Descripción**	**Ejemplos**
NOT	Evalúa TRUE si la condición es falsa	NOT (SUELDO > 10000)
AND	Evalúa TRUE si ambas condiciones son verdaderas	(SUELDO > 1000) **AND** (COMISION < 2000)
OR	Evalúa TRUE si al menos una de las condiciones es verdadera	(SUELDO > 1000) **AND** (COMISION < 2000)

Operador de cadenas	**Descripción**	**Ejemplos**
||	Concatena dos cadenas	'sema'||'na' → 'semana'

Funciones numéricas	Descripción	Ejemplos
ABS(n)	Valor absoluto de *n*	**ABS**(-15) → 15
ACOS, ASIN, ATAN, COS, LN, LOG, SIN, TAN	Funciones trigonométricas	
CEIL(n)	Devuelve el entero mas pequeño mayor o igual que *n*	**CEIL**(12.4) → 13
FLOOR	Devuelve el entero más grande menor o igual que *n*	**FLOOR**(12.4) → 12
MOD(m,n)	Devuelve el resto de dividir *m* por *n*	**MOD**(7,2) → 1
POWER(m,n)	Eleva *m* a *n*	**POWER**(4,2) → 16
ROUND (n[,m])	Redondea *n.* Si *m* no es cero redondea a *m* decimales	**ROUND**(12.19,1)→ 12.2
SIGN(n)	Devuelve –1 si *n*<0, 0 si *n*=0, 1 si *n*>0	**SIGN**(10) → 1
SQRT(n)	Devuelve la raíz cuadrada de *n*	**SQRT**(16) → 4
TRUNC(n[,m])	Trunca *n*. Mismo tratamiento que la función ROUNDED, salvo que no redondea	**TRUNC**(12.19,1) → 12.1
Funciones alfanuméricas	**Descripción**	**Ejemplos**
CHR(n)	Devuelve el carácter del código ASCII	**CHR**(65) → A
CONCAT(A, B)	Concatena las cadenas o variables A y B. Equivale a ||	
INITCAP(A)	Convierte a mayúsculas la	**INITCAP**('la casa') → La Casa

	inicial de cada palabra contenida en argumento	
LOWER(A)	Convierte en minúsculas el argumento	**LOWER**(‘ CASA’) → casa
LPAD(A,n [,B])	Devuelve A ajustada a la derecha con relleno por la izquierda con B hasta completar *n*	**LPAD**(‘casa’,7,’*’) → ***casa
LTRIM(A, B)	Elimina empezando por la izquierda del argumento A los caracteres que coincidan con el argumento B hasta que encuentre uno diferente.	**LTRIM**(‘ababcasa’,’ab’)→ casa
REPLACE(A,B[,C])	Reemplaza en la cadena A la cadena B por la cadena C	**REPLACE**(‘roma re’,’r’,’T’) → Toma Te
RPAD(A,n[,B])	Mismo tratamiento que LPAD pero por la derecha	**RPAD**(‘casa’,7,’*’) → casa***
RTRIM(A, B)	Mismo tratamiento que LTRIM por la derecha	
SUBSTR(A,m[,n])	Devuelve una subcadena de A de longitud *n* comenzando en la posición *m*, si *m* es negativo comienza por la derecha	**SUBSTR**(‘abcdef’,2,3) → bcd
UPPER(A)	Convierte a mayúsculas el argumento	**UPPER**(‘casa’) → CASA
ASCII(A)	Devuelve el código ascii	**ASCII**(‘A’) → 65
INSTR(A, B, [,n[,m]])	Busca en el argumento A la cadena B devolviendo la posición de comienzo. *m* y *n* por defecto valen 1,	**INSTR**(‘piso raso’,’so’,2,2) → 8

	de lo contrario comienza a buscar a partir de *n* y cuenta a partir de la ocurrencia *m*	
LENGTH(A)	Devuelve la longitud de una cadena	**LENGTH**('la casuca') → 9

3. SQL y el modelo relacional.

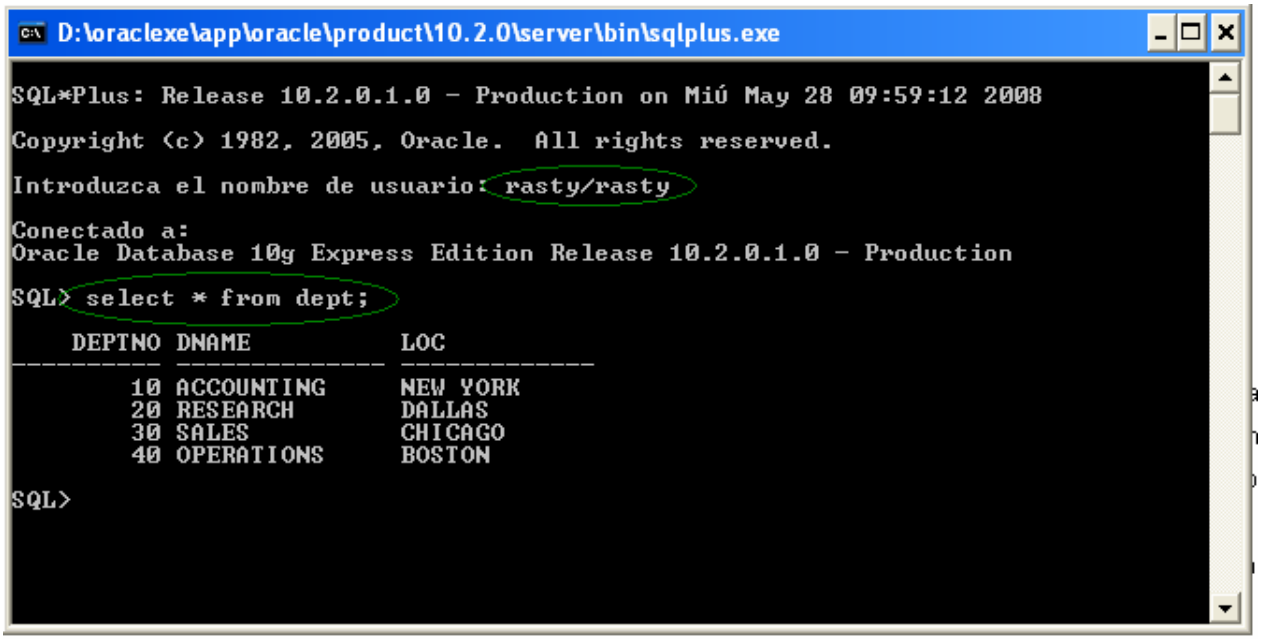
El lenguaje SQL es un lenguaje relacional, y aunque sea muy discutible la fidelidad con la que implementa los distintos elementos del modelo, es el lenguaje que incorporan la mayoría de los productos relacionales disponibles en el mercado.

En lo que sigue veremos las distintas sentencias SQL, relacionándolas con los operadores relacionales tratados en las unidades anteriores.

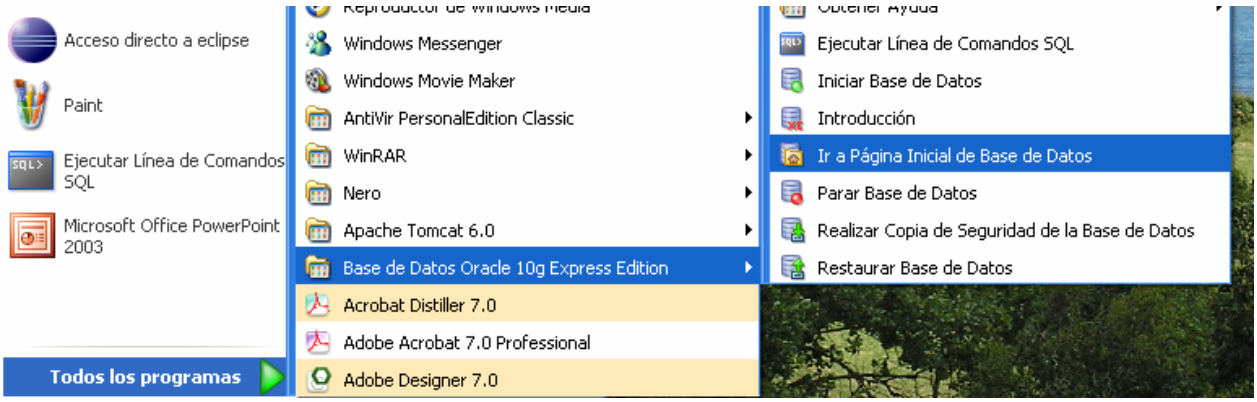
Comenzaremos con la sentencia SELECT, siguiendo siempre el método de ir descubriendo todas las posibilidades de cada sentencia, desde sus opciones más simples a las más complejas, con el apoyo continuo de ejemplos prácticos. A la vez que se explican las sentencias, en los ejemplos se irá introduciendo el uso de las funciones y operadores descritos anteriormente.

Para poder realizar comprobaciones y resolver los ejercicios que se proponen hay que conectarse como usuario RASTY.

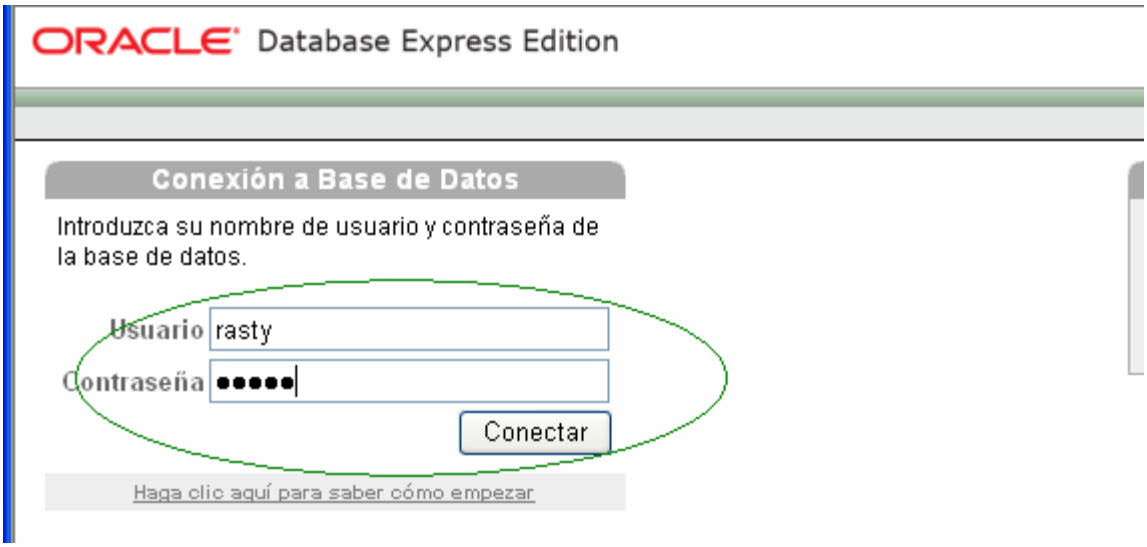
Podéis trabajar desde el entorno SQLPLUS. ***Inicio→ejecutar→sqlplus*** y conectarse como *rasty/rasty* . En la siguiente pantalla se muestra un ejemplo



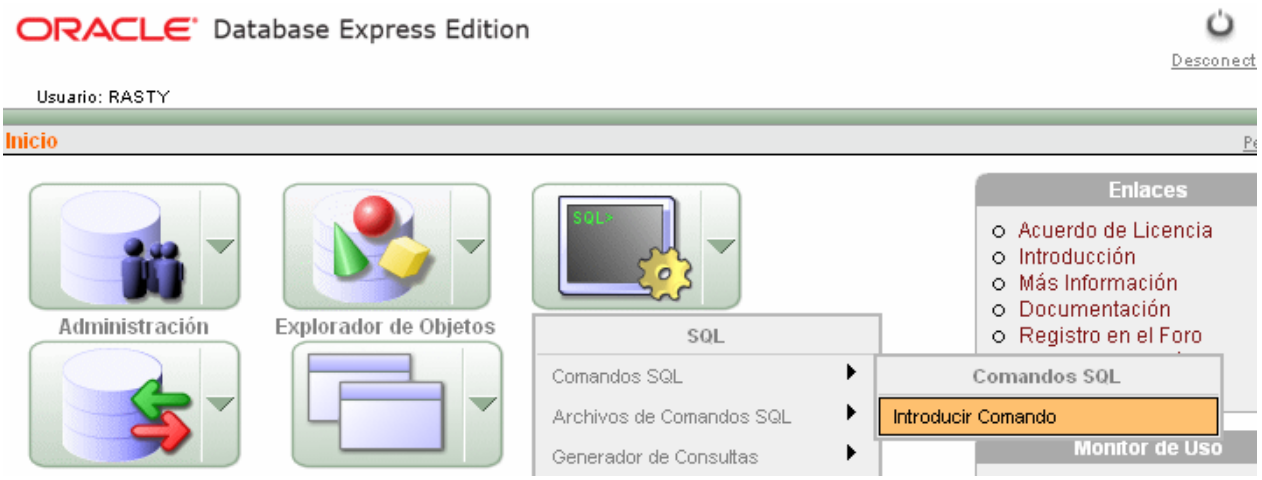
También se puede trabajar desde el asistente, la siguiente captura muestra como se inicia



Conectarse como *rasty*, contraseña *rasty*



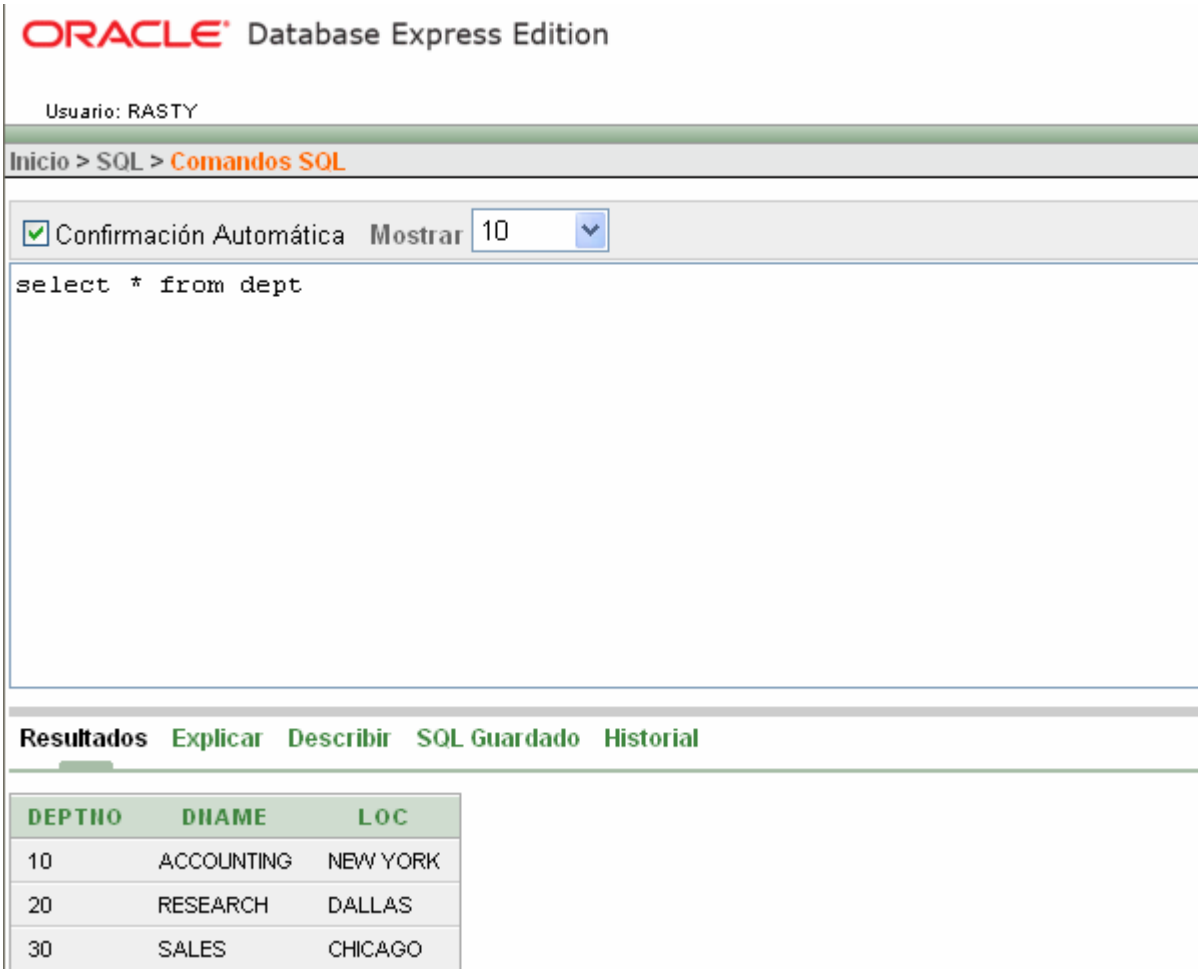
Seleccionar ***sql→comandos sql→ introducir comando***



Ejecutamos la siguiente sentencia

```
SELECT * FROM DEPT
```

Observad que si la ejecutáis desde *sqlplus* termina en punto y coma. Desde el asistente no hace falta que termine en punto y coma.



El resultado es el mismo, solo cambia el entorno.

4. Lenguaje de manipulación de datos (LMD)

4.1. Recuperación de información. Sentencia SELECT

La sentencia SELECT permite recuperar información contenida en una o más tablas de la base de datos. Esta sentencia nos permite escribir una expresión relacional equivalente a las que escribíamos en la Unidad 2 usando el álgebra o el cálculo relacionales. Esta expresión define la información a recuperar.

El resultado de evaluar una expresión relacional es siempre una relación, luego el resultado de evaluar una sentencia SELECT es siempre una serie de datos en forma de tabla.

- **Selección de determinadas columnas (operación de proyección)**

La primera operación que vamos a realizar con la SELECT será la de consultar determinadas columnas de una tabla sin eliminar los duplicados.

Sintaxis simple de la sentencia SELECT

SELECT expresión FROM tabla;

Por el momento *expresión* describe las columnas que se desean recuperar de la tabla. Para seleccionar algunas columnas se las nombra separadas por comas

SELECT ename, sal FROM emp;

```
ENAME             SAL
---------- ----------
SMITH              800
ALLEN             1600
WARD              1250
JONES             2975
MARTIN            1250
BLAKE             2850
CLARK             2450
SCOTT             3000
KING              5000
TURNER            1500
ADAMS             1100
ENAME              SAL
JAMES              950
FORD              3000
MILLER            1300

14 filas seleccionadas.
```

Para seleccionar todas las columnas de una tabla se utiliza el asterisco (*).

SELECT * FROM tabla;

Este formato no se puede utilizar si la tabla tienen tipos de datos LONG RAW

- **Clausula WHERE (operación de restricción).**

A la SELECT le siguen las columnas que se quieren consultar, a FROM la tabla a consultar, y si se quiere establecer una condición de búsqueda (restricción sobre las filas, seleccionando las que satisfagan una determinada condición) se utiliza la cláusula WHERE .

Sintaxis con la cláusula WHERE

SELECT expresión FROM tabla WHERE condición;

Como se puede ver, el resultado de una consulta es una tabla, si la hacemos interactivamente la salida se visualiza de forma tabular siempre que quepa en la línea de la pantalla. También se puede enviar a impresora o a un fichero. Si la consulta se hace desde un programa la tabla se vuelca en el programa.

Vamos a hacer una restricción según una condición. Utilizamos la cláusula WHERE para seleccionar determinadas filas de la tabla consultada. La cláusula consta de la palabra WHERE seguida de una condición simple o compuesta. Si al evaluar la condición da como resultado *true* la fila se recupera, si *false* o *desconocido* no se recupera.

SELECT ename, sal FROM emp WHERE sal > 1500;

```
ENAME             SAL
---------- ----------
ALLEN            1600
JONES            2975
BLAKE            2850
CLARK            2450
SCOTT            3000
KING             5000
FORD             3000
ALLEN            1600
```

La operación anterior puede ser vista como una operación de cálculo relacional.

Las filas que se recuperan no están ordenadas, conforme a la teoría relacional, si se quieren ordenar, lo que va en contra del modelo relacional, pero que desde el punto de vista de explotación (facilidad para el usuario final) se puede considerar una operación necesaria, se utiliza la cláusula ORDER BY seguida por las columnas por las que se quiere que se ordenen. Si se quiere que se ordenen por más de una columna se deben separar por comas.

Sintaxis con ORDER BY

SELECT expresión FROM tabla
WHERE condición
ORER BY {expresión | posición}[{ASC | DESC}];

Ejemplo.

SELECT ename, sal FROM emp WHERE sal > 1500 ORDER BY ename;

```
ENAME             SAL
---------- ----------
ALLEN            1600
BLAKE            2850
CLARK            2450
FORD             3000
JONES            2975
KING             5000
SCOTT            3000
```

La ordenación puede ser ascendente (ASC) o descendente (DESC), por defecto es ASC.

También se puede ordenar por posición de la columna, por ejemplo, en el caso anterior podemos utilizar posición y descendente.

SELECT ename, sal FROM emp WHERE sal > 1500 ORDER BY 2 DESC;

```
ENAME             SAL
---------- ----------
KING             5000
SCOTT            3000
FORD             3000
JONES            2975
BLAKE            2850
CLARK            2450
ALLEN            1600
```

Este es un ejemplo más de la falta de fidelidad de SQL al modelo relacional, los atributos de una relación no tienen orden, luego no se puede hacer referencia a su posición en la cabecera.

- **Función *SUBSTR***

Con esta función se puede obtener parte del contenido de una columna.

SELECT SUBSTR(dname, 1, 2) FROM dept;

```
SU
--
AC
RE
SA
OP
```

Las cabeceras que visualizan las consultas son las expresiones que se colocan en la SELECT ajustadas al ancho del contenido que se recupera. Si queremos cambiarlas, colocamos las nuevas a continuación de cada expresión de la SELECT . Si la cabecera contiene espacio u otro carácter con significado especial, se debe colocar entre comillas, sino es opcional.

Realmente lo que estamos haciendo es nombrar los atributos de la relación resultado. Recuérdese el operador RENOMBRAR del álgebra relacional.

SELECT dname "Nombre del Departamento" FROM dept;

```
Nombre del Dep
--------------
ACCOUNTING
RESEARCH
SALES
OPERATIONS
```

- **Tabla *DUAL***

Lo habitual es trabajar con la SELECT accediendo a las tablas de la base de datos, pero si queremos realizar cualquier tipo de operación en la que no tengan que intervenir tablas del usuario podemos utilizar la tabla del sistema DUAL cuya estructura es la siguiente:

DESC DUAL

```
Nombre                          ¿Nulo?    Tipo
 ------------------------------ -------- -----
 DUMMY                                   VARCHAR2(1)
```

Como vemos es una tabla de una sola columna de un byte de longitud, y si ejecutamos

SELECT * FROM DUAL;

Veremos que tiene una sola fila.

DESCRIBE (abreviado **DESC)** es una sentencia de SQL*PLUS que nos permite ver la estructura de una tabla (se verá en la próxima Unidad de Trabajo).

Ejemplos de utilización de la tabla DUAL.

Empleando la función **ABS** para obtener el valor absoluto

SELECT ABS(-30) FROM DUAL;

```
     VALOR
----------
        30
```

Evaluando una expresión aritmética

SELECT 4 * 5 / 10 Resultado FROM DUAL;

```
RESULTADO
---------
        2
```

En términos relacionales, en todos estos ejemplos tenemos una proyección nula de la tabla DUAL seguida de una ampliación.

- **Cláusula *DISTINCT***

Realmente, todos los ejemplos de la SELECT vistos hasta ahora no implementan el operador relacional de proyección con fidelidad, puesto que permiten que la tabla resultado tenga filas duplicadas.

Como vimos en la Unidad 2, por definición, una relación no puede tener tuplas repetidas. Para que la SELECT genere un resultado sin filas repetidas es necesario incorporarle la cláusula DISTINCT

SELECT DISTINCT sal FROM EMP;

```
  SAL
------
   800
   950
  1100
  1250
  1300
  1500
  1600
  2450
  2850
  2975
  3000
  5000
```

- **Expresiones en la *SELECT***

Como ya vimos en los ejemplos anteriores en los que se utilizó la tabla DUAL, podemos utilizar expresiones que implican la aparición de columnas nuevas como consecuencia de la evaluación de la expresión. Por ejemplo, si se quiere obtener el nombre de los empleados con el 50% de su salario siempre que éste sea mayor de 1000 y que el departamento sea 10, se codificaría la siguiente sentencia:

SELECT ename, 0.5*sal Salario FROM emp WHERE sal > 1000 AND deptno = 10;

```
ENAME        SALARIO
---------- ----------
CLARK            1225
KING             2500
MILLER            650
```

- **DECODE**

Podemos utilizar DECODE, no existe en el SQL estándar, para obtener un dato de una expresión multievaluada. Tiene la siguiente sintaxis:

DECODE (exp0, exp1, res1, exp2, res2, ..., def)

DECODE compara el valor de exp0 con el valor de exp1 si coincide devuelve res1, si no coincide compara con exp2 si es igual devuelve res2, así sucesivamente hasta que encuentre uno que sea igual. Si no hay ninguno que coincida asume el valor def.

Ejemplo.

SELECT ename, DECODE(deptno, 10, 'contabilidad',30,'ventas', 'no hay') DPTO FROM emp;

```
ENAME      DPTO
---------- ------------
SMITH      no hay
ALLEN      ventas
WARD       ventas
JONES      no hay
MARTIN     ventas
BLAKE      ventas
CLARK      contabilidad
SCOTT      no hay
KING       contabilidad
TURNER     ventas
ADAMS      no hay
JAMES      ventas
FORD       no hay
MILLER     contabilidad
```

DECODE también se puede utilizar en las cláusulas WHERE y ORDER BY.

- ***El operador IS NULL***

Cuando se quiera controlar el contenido nulo o no nulo de una columna utilizaremos el operador IS [NOT] NULL. Por ejemplo, para consultar todos los clientes con nulos en la columna de *observaciones*, se escribiría la siguiente sentencia:

SELECT ename, sal FROM emp WHERE comm IS NULL;

El resultado siempre será *true* o *false* nunca podrá ser, como puede suceder con otras condiciones, *nulo.*

- ***El operador BETWEEN.***

Es un operador que testea rangos, es decir comprueba si un valor se encuentra entre otros dos valores especificados. Se puede sustituir por otros operadores de relación y lógicos, Por ejemplo la expresión *A BETWEEN B AND C* se puede sustituir por *(A >= B) AND (A =< C)*

Para obtener la relación de empleados cuyos saldos estén comprendidos entre 1000 y 2000 codificaríamos la siguiente sentencia:

SELECT ename, sal FROM emp WHERE sal BETWEEN 1000 AND 2000;

- ***El operador LIKE***

Con este operador seleccionamos aquellas filas en las que el contenido de una columna se corresponde con un patrón en el que interviene el comodín *%* que sustituye una subcadena o el carácter subrayado (_) que sustituye un carácter. En el patrón pueden intervenir más de un comodín.

En la tabla de *emp* existen los nombres *ALLEN* y *ADAMS,* si se quieren obtener dichos nombre deberíamos comparar la columna *nombre* con el patrón *A%,.*

SELECT ename FROM emp WHERE ename LIKE 'A%';

```
ENAME
-------
ALLEN
ADAMS
```

Para sustituir un único carácter usaremos el carácter subrayado (_)

SELECT ename FROM emp WHERE ename LIKE '_MIT_';

```
ENAME
----------
SMITH
```

- ***La función LPAD***

Se puede hacer que en una consulta el resultado ajuste a izquierda (por defecto) o derecha y que además rellene con una determinada cadena (uno o varios caracteres).

En el siguiente ejemplo hacemos que el nombre ajuste a la derecha y que rellene por la izquierda con * sobre un ancho de campo de 20 caracteres en total.

SELECT LPAD(ename,20,'*') "NOMBRE" FROM emp;

```
NOMBRE
--------------------
***************SMITH
***************ALLEN
***************WARD
***************JONES
**************MARTIN
**************BLAKE
**************CLARK
***************SCOTT
****************KING
**************TURNER
***************ADAMS
***************JAMES
****************FORD
**************MILLER
```

Si se desea completar por la derecha se empleará RPAD.

4.2. Operadores tradicionales de conjuntos (UNIÓN, INTERSECCIÓN, DIFERENCIA)

Hasta aquí hemos planteado casos de recuperación de información con la sentencia SELECT. Se pueden plantear recuperaciones en las que intervengan más de una SELECT utilizando los operadores UNION, INTERSECCION y DIFERENCIA. Se habla así de consultas compuestas.

Cada SELECT daría lugar a una tabla resultado, combinándose estas tablas, de acuerdo a los operadores anteriores presentes en la expresión, para dar lugar a una única tabla resultado. En una consulta compuesta, las expresiones que aparecen en las sentencias SELECT deben ser iguales en número y tipo.

Operador	Descripción
UNION	Selecciona las filas de ambas tabla, se eliminan las duplicadas.
UNION ALL	No elimina las duplicadas
INTERSECT	Selecciona únicamente filas que pertenezcan a ambas tablas
MINUS	Selecciona los las filas que pertenezcan a la primera pero no a la segunda.

4.3. Resumen o agrupación (GROUP BY)

La cláusula GROUP BY permite agrupar las filas recuperadas en una sentencia SELECT según un determinado criterio, devolviendo una fila por cada uno de los grupos formados. O sea, permite obtener información resumen de cada grupo formado por las filas que recuperaría la SELECT.

Formato con la cláusula GROUP BY

SELECT expresión FROM tabla GROUP BY expresión;

En los resúmenes se emplean las funciones de columna o funciones de grupo, funciones que actúan sobre un conjunto de filas y afectan a los valores de una columna devolviendo un único resultado.

Función	Descripción	Ejemplos
AVG([DISTINCT | ALL] n)	Calcula el promedio de *n*	SELECT AVG(saldo) FROM clientes
SUM([DISTINCT | ALL] n)	Calcula la suma total de *n*	SELECT SUM(saldo) FROM clientes
MIN([DISTINCT | ALL] n)	Devuelve el mínimo valor de *n*	SELECT MAX(saldo) FROM clientes
MAX([DISTINCT | ALL] n)	Devuelve el máximo valor de *n*	SELECT MIN(saldo) FROM clientes
COUNT([DISTINCT | ALL] n)	Cuenta el número de filas donde *n* no es nulo	SELECT COUNT(saldo) FROM clientes
COUNT(*)	Cuenta el número de filas consultadas	SELECT COUNT(*) FROM clientes

Con la opción DISTINCT la función tiene en cuenta únicamente los valores no repetidos. Con la opción ALL la función tiene en cuenta todos los valores, incluidos los duplicados. Por defecto asume ALL.

Todas las funciones de grupo excepto COUNT(*) ignoran los valores nulos.

Las funciones no sólo actúan sobre filas agrupadas, sino que pueden hacerlo sobre toda la tabla como vemos en los ejemplos del cuadro anterior

A continuación veremos ejemplos de agrupaciones o resúmenes:

En la tabla *emp* podemos sumar los salarios para un determinado departamento.

SELECT SUM(sal) FROM emp WHERE deptno= '10';

```
SUM(SAL)
---------
     8750
```

Pero si se quiere consultar subtotales para cada departamento tenemos que utilizar la cláusula

GROUP BY

SELECT deptno, SUM(sal) FROM emp GROUP BY deptno;

```
  DEPTNO   SUM(SAL)
-------- ----------
      10       8750
      20      10875
      30       9400
```

No se permiten incluir columnas de detalle en las agrupaciones, por ejemplo, en este caso no se podría incluir el nombre del empleado. En general en la selección siempre se incluirá una columna de agrupación y una función de columna. Por otra parte las columnas de totalización NULL se realizan dentro del mismo grupo.

- ***Cláusula HAVING***

Se pueden excluir filas agrupadas, para lo cual utilizaremos la cláusula HAVING cuyo comportamiento es análogo al de la WHERE, con la salvedad que la cláusula HAVING actúa sobre las filas que previamente han sido agrupadas con la cláusula GROUP.

Sintaxis de la cláusula HAVING

SELECT expresión FROM tabla GROUP BY expression HAVING condición;

Ejemplo.

Excluir aquellas filas cuya suma de salarios por departamento no superen los 2000

SELECT deptno, SUM(sal) FROM emp GROUP BY deptno HAVING SUM(sal) <= 9500;

```
DEPTNO   SUM(SAL)
------- ----------
     10       8750
     30       9400
```

La cláusula WHERE también se puede utilizar en agrupamientos, pero hay que tener en cuenta que actúa sobre filas individuales, en cambio la cláusula HAVING actúa sobre filas agrupadas, por lo tanto las expresiones que contenga actuaran sobre grupos de filas, siempre incluirá una función de columna. Dicho de otra manera WHERE interviene en el proceso de selección, en cambio HAVING interviene una vez que las filas están agrupadas. En el siguiente caso vamos obtener las salarios por departamento que superen los 5000 solamente de los empleados que tengan comisión.

SELECT deptno, SUM(sal) FROM emp WHERE comm IS NOT NULL
GROUP BY deptno HAVING SUM(sal) > 5000;

```
DEPTNO   SUM(SAL)
------ ----------
    30       5600
```

Si se quiere realizar varios niveles de totalización, por ejemplo totalizar por departamento y dentro de estos por categoria, incluiríamos ambas columnas de agrupamiento en la selección (no obligatorio) y en la GROUP BY.

SELECT deptno, job, SUM(sal) FROM emp GROUP by deptno, job;

```
DEPTNO JOB         SUM(SAL)
------ --------- ----------
    10 CLERK           1300
    10 MANAGER         2450
    10 PRESIDENT       5000
    20 CLERK           1900
    20 ANALYST         6000
    20 MANAGER         2975
    30 CLERK            950
    30 MANAGER         2850
    30 SALESMAN        5600
```

La cláusula GROUP BY ordena los datos en ascendente y del nivel más externo al más interno.

4.4. Consultas en múltiples tablas (REUNIÓN O COMBINACIÓN)

La combinación consiste en concatenar las filas de una tabla con las de la otra.

En general se pueden recuperar datos de varias tablas obteniendo una única tabla compuesta por datos de las tablas accedidas.

Las consultas en más de una tabla se suelen hacer cuando nos encontramos con tablas relacionadas, tablas que tienen alguna columna en común, de forma que queremos obtener información de ambas tablas o de una de ellas, a través de la relación. En las tablas que manejamos hay una muy sencilla que es la relación *empleados/departamentos,* supongamos que queremos hacer una consulta para visualizar los nombres de los departamento con sus correspondientes empleados.

SELECT dname, ename FROM emp, dept WHERE emp.deptno=dept.deptno;

```
DNAME          ENAME
-------------- ----------
RESEARCH       SMITH
SALES          ALLEN
SALES          WARD
RESEARCH       JONES
SALES          MARTIN
SALES          BLAKE
ACCOUNTING     CLARK
RESEARCH       SCOTT
ACCOUNTING     KING
SALES          TURNER
RESEARCH       ADAMS
SALES          JAMES
RESEARCH       FORD
ACCOUNTING     MILLER
```

Cuando en la cláusula FROM hay más de una tabla, cualquier referencia a una columna debería ir calificada con el nombre de la tabla, por ejemplo, *emp.deptno*, para referirse a la columna código del departamento de la tabla empleados. La calificación sólo es obligatoria en caso de ambigüedad, o sea, cuando nos referimos a un nombre de columna que se repita en más de una tabla. En la consulta anterior las columnas que relaciona ambas tablas (deptno) se llaman igual, para distinguirlas hay que anteponer el nombre de la tabla a la que pertenecen, empleados o departamentos (emp.deptno, dep.deptno).

4.5. Reuniones externas

Cuando se incluyen en una selección atributos de una tabla que no tienen contrapartida en la relacionada, se dice que es una reunión externa.

La reunión externa se identifica con el signo **+** asociado al atributo que interviene en la relación, puede ser reunión externa izquierda o bien reunión externa derecha. En el siguiente ejemplo la relación se establece entre los códigos de departamento de las tablas de empleados *(emp)* y departamentos
(dep)

SELECT dname, ename FROM emp, dept WHERE emp.deptno(+) = dept.deptno;

```
DNAME          ENAME
-------------- ----------
ACCOUNTING     CLARK
ACCOUNTING     KING
ACCOUNTING     MILLER
RESEARCH       SMITH
RESEARCH       ADAMS
RESEARCH       FORD
RESEARCH       SCOTT
RESEARCH       JONES
SALES          ALLEN
SALES          BLAKE
SALES          MARTIN
SALES          JAMES
SALES          TURNER
SALES          WARD
OPERATIONS
```

Como se puede observar el departamento OPERATIONS no tiene empleados, es decir se obtiene una relación con todos los departamentos que tienen empleados, incluyen los departamento que no tienen correspondencia en la tabla *emp.*

4.6. Los ALIAS (variables de filas explícitas)

Como vimos en las unidades anteriores, en las expresiones de cálculo relacional se utilizan variables de tupla. El mecanismo de calificación de columnas utilizado anteriormenete no es ni más ni menos que el uso que de este concepto hace la sentencia SELECT. En los ejemplos vistos se usó como variable de fila el nombre de la tabla. Se dice que estas son variables de fila implícitas.

En la cláusula FROM se pueden definir alias de los nombres de tabla y estos alias pueden usarse como variables de fila. Los alias son variables de fila explícitas. Con el objeto de clarificar el código en una consulta donde intervengan muchas tablas podemos utilizar variables de filas explícitas, o sea alias, acortándose el nombre de las tablas. En el siguiente ejemplo se pone el alias ***e*** a la tabla *emp* y a la tabla *dep* se pone ***d***

```
SELECT dname, ename  FROM   emp  e,   dept  d   WHERE    e.deptno =  d.deptno;
```

Una vez establecidas las bases de acceso a varias tablas, se pueden restringir las búsquedas tanto como se quiera, por ejemplo, podemos realizar una consulta para visualizar los nombres de los empleados con los correspondientes departamentos, pero sólo de aquellos que tengan comisión.

```
SELECT dname, ename  FROM   emp  e,   dept  d   WHERE    e.deptno =  d.deptno
                                                AND comm IS NULL;
```

```
DNAME          ENAME
-------------- ----------
RESEARCH       SMITH
RESEARCH       JONES
SALES          BLAKE
ACCOUNTING     CLARK
RESEARCH       SCOTT
ACCOUNTING     KING
RESEARCH       ADAMS
SALES          JAMES
RESEARCH       FORD
ACCOUNTING     MILLER
```

Como norma aconsejamos que si se manejan varias tablas se utilicen los alias tanto si existe ambigüedad como si no, es decir se aplicará el siguiente formato:

```
SELECT a.columna, b.columna….
FROM tabla1 a, tabla2 b, ….
WHERE condición;
```

4.7. Producto cartesiano generalizado

Como se explicó en la Unidad 2, el producto cartesiano generalizado de dos tablas da lugar a otra cuyas filas resultan de la concatenación de cada fila de una tabla con todas las filas de la otra. La operación es semejante a una combinación en la que se omite la cláusula WHERE.

```
SELECT ename, dname  FROM   emp,  dept ;
```

También se puede combinar la tabla consigo misma

```
SELECT a.ename, b.ename     FROM  emp  a,  emp  b;
```

La estructura anterior puede parecer que carece de sentido, sin embargo es importante recordarla, en su momento se propondrá un ejercicio en el que es imprescindible aplicarla.

4.8. SELECT anidadas (subconsultas)

La capacidad de utilizar SELECT anidadas nos permite utilizar el resultado de una consulta como entrada para otra consulta. Hay algunas restricciones al utilizar las SELECT anidadas, por ejemplo no se pueden seleccionar una lista de valores si no vamos a utilizar cuantificadores (***some, all, any, in***). En general, puesto que el resultado de una subconsulta va a formar parte de la condición de búsqueda de otra consulta debe concordar con la misma. Se utilizarán en las cláusulas WHERE y HAVING con los operadores relacionales descritos anteriormente y también se puede utilizar en la cláusula FROM en lugar de una tabla.

En el siguiente ejemplo no hace falta utilizar cuantificadores ya que la subconsulta devuelve un único valor.

Obtener el nombre de los empleados cuyo salario sea mayor que el del empleado Jones

```
SELECT ename FROM emp WHERE sal >
        (SELECT sal FROM emp WHERE ename='JONES');
```

```
ENAME
------
SCOTT
KING
FORD
```

Con este otro ejemplo la subconsultas devuelve una lista de valores, luego hay que utilizar un cuantificador.

Obtener el nombre de los empleados que tienen un salario igual a cualquiera de los salarios mínimos de cada uno los departamentos.

```
SELECT ename FROM emp WHERE sal =
        ANY (SELECT min(sal) FROM emp GROUP BY deptno);
```

```
ENAME
------
SMITH
JAMES
MILLER
```

- ***Subconsulta en la FROM***

Al utilizar la SELECT en la FROM las consultas las hacemos contra la vista que se obtiene y no contra la tabla.

La siguiente sentencia selecciona el nombre de la tabla *emp* de aquellas filas que cumplan la condición que se establece.

SELECT ename FROM emp WHERE comm IS NULL;

La siguiente consulta es equivalente a la anterior.

SELECT ename FROM (SELECT * FROM emp) WHERE comm IS NULL;

En el ejemplo anterior la consulta se realiza sobre una vista, resultado de utilizar una subconsulta en la FROM.

4.9. Añadiendo filas. INSERT

Para añadir filas a las tablas se utiliza la sentencia INSERT. Se pueden añadir las filas completando todas las columnas de la tabla con algún valor o con nulos (NULL). También se pueden añadir filas completando solo algunas columnas de la tabla, en este caso hay que nombrarlas.

Sintaxis simple de la sentencia INSERT

INSERT INTO nombre_tabla [(columnas…)] VALUES (valores…)

Vamos añadir una fila a la tabla de *dept* completando todas las columnas de la tabla con algún valor:

INSERT INTO dept VALUES (50,'CONTABILIDAD','AVILES');

Se puede prescindir de algún valor sustituyéndolo por NULL.

INSERT INTO dept VALUES (60,'PERSONAL',NULL);

En el caso de que se quiera prescindir de varias columnas se puede determinar en cuáles vamos introducir información en vez de sustituir los valores por NULL.

En la tabla *emp* vamos añadir una fila dando valores a determinadas columnas, en este caso debemos especificar las columnas en las que se quiere introducir datos.

INSERT INTO emp (empno, ename, sal) VALUES(9999,'JOSE MARIA',3000);

La fila se insertará con valores nulos en las columnas no especificadas. Si alguna de estas columnas fue definida con un valor por defecto (DEFAULT), en ella se insertará dicho valor en lugar de NULL. Si no se definió con valor por defecto y tiene la restricción NOT NULL, la fila será rechazada por el SGBD, y no se insertará ante el intento de violación de la restricción. En un próximo apartado trataremos las restricciones.

Otro formato de la INSERT es el que permite recuperar datos de una tabla para añadirlos a otra, sustituyendo la cláusula VALUES por una subconsulta. En este caso las columnas de la subconsulta deben de coincidir en número y tipo con las especificadas en la INSERT.

```
INSERT  INTO dept2 SELECT * FROM dept;
```

4.10. Actualizar columnas. UPDATE

Con la sentencia UPDATE se modifican los valores, contenidos, de las columnas de las filas seleccionadas de una sola tabla.

Sintaxis simple de la UPDATE

```
UPDATE  Nombre_de_la _tabla
        SET columna= expresión [, columna = expresión].....
            [WHERE condición]
```

La WHERE selecciona las filas a actualizar. Sigue las mismas reglas que las empleadas en las sentencia SELECT, pudiendo incluso utilizar SELECT subordinadas.

En la cláusula SET se incluyen expresiones que generen un valor que esté en concordancia con el tipo de dato definido para la columna. No se pueden incluir funciones de columna ni subconsultas. Si se pueden incluir datos de la fila que está seleccionada.

Un ejemplo sencillo para actualizar una tabla puede consistir en incrementar el salario de todos los empleados en un 10% siempre que su comisión sea nula

```
UPDATE emp SET sal = sal * 1.10  WHERE comm IS NULL;
```

Para actualizar más de una columna en la misma UPDATE separarlas con una coma.

En el caso de que actualicemos una columna C1 con el valor de otra columna C2 (siempre el de la fila seleccionada) que a su vez se actualiza con el valor de una expresión, C1 toma el valor que tenia C2 antes de actualizarse.

Se pueden utilizar subconsultas en la cláusula WHERE, lo cual es interesante cuando hay que actualizar columnas a partir de valores contenidos en otras tablas. El tratamiento descrito en el apartado de SELECT ANIDADAS es aplicable en este punto.

4.11. Borrar filas DELETE

Con esta sentencia podemos borrar una serie de filas de la tabla accedida. Las filas que se borran se seleccionan con la cláusula WHERE cuyas condiciones de búsqueda son las mismas que las vistas en la sentencia SELECT.

Sintaxis simple de la sentencia DELETE

DELETE FROM nombre_de_la_tabla [WHERE condición]

Si quisiéramos borrar todos los empleados del departamento 10, codificaríamos la siguiente sentencia.

DELETE FROM emp WHERE deptno = 10;

Si no se utiliza la cláusula WHERE se borran todas las filas de la tabla, quedando la tabla vacía. No se borra la estructura.

DELETE FROM emp;

Con la sentencia anterior borraríamos todas las filas de la tabla *emp.*

No vamos a entrar en este momento a comentar que pasaría si la tablas estuvieran definidas con sus correspondientes ***constraints***, estamos suponiendo que no se han declarado los controles de integridad, este caso lo veremos en otro apartado.

En la cláusula WHERE podemos incluir subconsulta para poder borrar filas en base a datos de otras tablas. Al igual que se describió en el apartado de actualizaciones, la subconsulta permite referencias externas y diferentes niveles de anidamiento.

Ejemplo

Borrar los empleados del departamento ubicado en DALLAS

```
DELETE FROM emp WHERE deptno IN
  (SELECT deptno FROM dept WHERE loc LIKE '%DALLAS%');
```

Si queremos eliminar la estructura de una tabla, es decir eliminar realmente la tabla, no sólo el contenido, se utilizará la sentencia DROP TABLE. Se verá en el siguiente epígrafe.

Desarrollo de Aplicaciones Informáticas

materiales didácticos de aula

UNIÓN EUROPEA
Fondo Social Europeo

GOBIERNO DEL PRINCIPADO DE ASTURIAS
CONSEJERÍA DE EDUCACIÓN Y CIENCIA

FORMACIÓN PROFESIONAL
Principado de Asturias