

ÍNDICE DE CONTENIDO

1	INTRODUCCIÓN a SQL desde la versión ORACLE8i	3
1.1	SQL	3
1.2	LAS RESPONSABILIDADES DEL ADMINISTRADOR	3
1.3	ROL DBA	4
1.4	USUARIOS SYS Y SYSTEM	4
1.5	PRIVILEGIOS SYSOPER Y SYSDBA	4
1.6	USUARIO INTERNAL	5
2	OBJETOS DE LA BASE DE DATOS	5
2.1	TABLAS	5
2.2	CONSTRAINTS O RESTRICCIONES	6
2.3	VISTAS	6
2.4	SECUENCIAS	6
2.5	ÍNDICES	6
2.6	CLUSTERS O AGRUPAMIENTOS	7
2.7	TRIGGERS O DISPARADORES	7
2.8	SNAPSHOTS O INSTANTÁNEAS	7
2.9	DB LINKS O ENLACES A OTRAS BASES DE DATOS	7
2.10	PROCEDIMIENTOS, FUNCIONES Y PAQUETES	7
2.11	SINÓNIMOS	7
2.12	USUARIOS	8
2.13	PERFILES	8
2.14	PRIVILEGIOS	8
2.15	ROLES	8
2.16	CONSULTAS AL DICCIONARIO DE DATOS	8
3	CREACIÓN DE OBJETOS DE LA BASE DE DATOS (DDL)	8
3.1	SQL*PLUS	10
3.2	USUARIOS	11
3.3	TABLAS	12
3.4	CONSTRAINTS O RESTRICCIONES	13
3.5	VISTAS	16
3.6	SECUENCIAS	17
3.7	ÍNDICES	19
3.8	SINÓNIMOS	19
3.9	EJERCICIOS	20
4	ACCESO A DATOS MEDIANTE SQL (SENTENCIA SELECT). FUNCIONES ESTÁNDAR.	20
4.1	SENTENCIA SELECT BÁSICA	20
4.1.1	Ejercicios	22
4.2	SELECCIÓN DE FILAS DE UNA TABLA	22

4.2.1	Ejercicios	25
4.3	CONTROL DEL ORDEN DE VISUALIZACIÓN DE LAS FILAS	25
4.3.1	Ejercicios	26
4.4	FUNCIONES SQL	26
4.4.1	LA TABLA DUAL	26
4.4.2	FUNCIONES DE CARÁCTER DE CONVERSIÓN	27
4.4.3	FUNCIONES DE CARÁCTER DE MANIPULACIÓN DE CARACTERES	27
4.4.4	FUNCIONES NUMÉRICAS	29
4.4.5	FUNCIONES DE FECHAS	29
4.4.6	FUNCIONES DE CONVERSIÓN	31
4.4.7	Ejercicios	33
4.5	SELECCIÓN DE DATOS DE VARIAS TABLAS	33
4.5.1	EQUIJOIN	33
4.5.2	NON-EQUIJOIN	34
4.5.3	OUTER JOIN	34
4.5.4	SELF JOIN	35
4.5.5	Ejercicios	35
4.6	DATOS AGREGADOS, FUNCIONES DE GRUPO PREFABRICADAS	35
4.6.1	FUNCIONES DE GRUPO	36
4.6.2	CLÁUSULA GROUP BY	37
4.6.3	CLÁUSULA HAVING	38
4.6.4	Ejercicios	38
4.7	SUBCONSULTAS	38
4.7.1	Ejercicios	40
4.8	OPERADORES DE CONJUNTOS	40
5	LENGUAJE DE MANIPULACIÓN DE DATOS (DML)	40
5.1	SENTENCIA INSERT	41
5.2	SENTENCIA UPDATE	41
5.3	SENTENCIA DELETE	42
5.3.1	Ejercicios	42
6	ANEXO. Soluciones a los ejercicios propuestos	42
7	Introducción a Consultas SQL anidadas	47

1 INTRODUCCIÓN a SQL desde la versión ORACLE8i

Nota Preliminar: En <http://www.oracle.com/technology/software/products/database/index.html> veremos toda la tecnología Oracle disponible. Actualmente, la versión más baja que podemos descargar es Oracle 9i (desde el sitio Web <http://www.oracle.com/technology/software/products/oracle9i/index.html>).

1.1 SQL

SQL es un lenguaje de consulta estructurado que permite comunicarnos con el servidor de Bases de Datos. Se distinguen los siguientes comandos:

- Recuperación de datos:
 - SELECT
- Lenguaje de Manipulación de Datos (DML):
 - INSERT
 - UPDATE
 - DELETE
- Lenguaje de Definición de Datos (DDL):
 - CREATE
 - DROP
 - ALTER
 - RENAME
 - TRUNCATE
- Control de transacciones, gestión de los cambios realizados por las sentencias DML:
 - COMMIT
 - ROLLBACK
 - SAVEPOINT
- Lenguaje de Control de Datos (DCL), para dar o quitar derechos de acceso sobre la BD y sus objetos:
 - GRANT
 - REVOKE

1.2 LAS RESPONSABILIDADES DEL ADMINISTRADOR

Una base de datos Oracle puede manejar gran cantidad de información y de usuarios. Por lo tanto, resulta imprescindible contar con una persona o grupo de personas, que controlen y dirijan el sistema. Esta importante figura de control, gestión y organización es conocida como **el administrador** de la base de datos.

Si la base de datos maneja mucha información y muchos usuarios, las tareas de administración pueden dividirse entre varios administradores. Las responsabilidades del administrador son:

- Instalar y actualizar el servidor Oracle y las herramientas de desarrollo.
- Controlar el almacenamiento de los datos y planificar el crecimiento de los mismos.
- Crear las estructuras primarias de almacenamiento de la base de Datos (tablespaces) hasta que los desarrolladores diseñen las aplicaciones.
- Crear los objetos primarios (tablas, vistas e índices) hasta que los desarrolladores comiencen a diseñar las aplicaciones.

Descargar Oracle 9i y SQL de Oracle8i

- Modificar las estructuras creadas, si es necesario, partiendo de la información que proporcionan los desarrolladores.
- Mejorar los tiempos de respuesta y controlar el rendimiento.
- Creación y mantenimiento de usuarios. Seguridad del sistema.
- Controlar y monitorizar el acceso de usuarios a la base de datos.
- Planificar la estrategia de Backup y recuperación.
- Realizar los backups y las recuperaciones.
- Contactar con el distribuidor para problemas técnicos que no pueda resolver.

El administrador(es) de la base de datos es un usuario Oracle con privilegio **DBA**.

Existen dos usuarios, creados al instalar la Base de Datos, que tienen concedido el rol DBA. Estos usuarios son:

- SYS cuya password inicial es CHANGE_ON_INSTALL. Todas las tablas y vistas del diccionario de datos son propiedad de este usuario. El mantenimiento de la integridad del diccionario lo gestiona internamente Oracle. En muchas bases de datos está prohibida la entrada a través de este usuario.
- SYSTEM cuya password inicial es MANAGER. Tiene todos los privilegios del sistema al igual que SYS. Bajo la identidad de este usuario se suelen instalar las tablas utilizadas por las herramientas Oracle, tales como Forms, Reports, Plus. No conviene que los usuarios mezclen tablas de aplicaciones con tablas de herramientas.

En cualquier caso, y para prevenir accesos no deseados al diccionario de datos o a las tablas de herramientas, es conveniente cambiar las passwords de estos usuarios.

1.3 ROL DBA

Un rol es un conjunto de privilegios. Existen privilegios del sistema (p.e. para crear una conexión) y sobre objetos (p.e. para leer datos de una tabla). EL rol DBA tiene garantizados todos los privilegios del sistema: crear y dar de baja usuarios, crear tablas, vistas, etc.

1.4 USUARIOS SYS Y SYSTEM

Cambio de la contraseña de estos usuarios:

```
SVRMGR> connect internal/oracle
ALTER USER SYS IDENTIFIED BY SYS;
ALTER USER SYSTEM IDENTIFIED BY SYSTEM;
```

1.5 PRIVILEGIOS SYSOPER Y SYSDBA

Son privilegios del sistema. SYSOPER (las operaciones de system) permite realizar backups, startup, shutdown, montar, desmontar, abrir, cerrar la base de datos.

SYSDBA = SYSOPER + "with admin option" + Crear BD.

Si un primer usuario garantiza un privilegio del sistema a un segundo usuario con la cláusula "with admin option", este segundo usuario puede -a su vez- garantizar ese privilegio a un tercero.

1.6 USUARIO INTERNAL

El usuario INTERNAL equivale al usuario SYS sin contraseña de la BD. El administrador se conecta con el usuario INTERNAL para realizar operaciones con la BD cerrada (p.e. alterar un espacio de tabla). También se emplea el usuario INTERNAL para crear una base de datos. Si el usuario INTERNAL requiere clave para conectarse, ésta se almacena en un fichero del sistema operativo:

```
<oracle_home>\database\pwdSID.orc
```

Si tenemos acceso a este fichero, podemos cambiar la clave de INTERNAL con la herramienta orapwd.

2 OBJETOS DE LA BASE DE DATOS

Los principales objetos de una Base de Datos son las tablas, las constrains o restricciones, las vistas, las secuencias, los índices, los cluster o agrupamientos, los triggers o disparadores, los snapshots o instantáneas, los DB Links o enlaces a otras bases de datos, los procedimientos, las funciones, los paquetes, los sinónimos, los usuarios, los perfiles, los privilegios y los roles.

2.1 TABLAS

La tabla es la unidad básica de almacenamiento. Se compone de registros o filas y de columnas o atributos. Cada columna es de un solo tipo. Los diferentes tipos son:

- **VARCHAR2(tamaño)** Dato de tipo carácter de longitud variable menor o igual que tamaño. (tamaño \leq 4000)
- **CHAR2(tamaño)** Dato de tipo carácter de longitud fija igual a tamaño. (tamaño \leq 255)
- **NUMBER(p, e)** Dato numérico de p dígitos decimales y e dígitos a la derecha del punto decimal.
- **DATE** Una fecha.
- **TIME** También almacena horas.
- **TIMESTAMP** La yuxtaposición de DATE y TIME.
- **LONG** Dato de tipo carácter de longitud variable hasta 2 Gbytes.
- **RAW(tamaño)** Dato binario de longitud inferior a un tamaño de 2000. (caracteres).
- **LONG RAW** Dato binario de longitud variable hasta 2 Gbytes.
- **LOB** (Large Object) hasta 4 Gbytes:
 - **BLOB** un LOB **B**inario (fotografías)
 - **CLOB** un LOB de tipo **C**arácter (para campos mayores de 4000 caracteres).
 - **NLOB** un LOB multibyte de ancho fijo.
 - **BFILE** representa un archivo binario almacenado fuera de la Base de Datos (películas)

Los tres primeros tipos de LOB se almacenan dentro de la BD, mientras que los BFILE se guardan en el sistema de ficheros, fuera de la BD.

Cuando necesitamos trabajar con atributos de tipo carácter y longitud superior a 4000 caracteres, se recomienda emplear los CLOB.

	LONG, LONG RAW	LOB
Atributos/tabla	SÓLO uno	varios
Tamaño	<2GB	<4GB
Búsqueda	<ul style="list-style-type: none">▪ No índice▪ No argumento de procedimiento.▪ No resultado función.▪ No WHERE▪ No ORDER BY▪ No GROUP BY▪ No CONNECT BY	A través de funciones de BD. Normal
Devuelve	Valores	Localizadores

2.2 CONSTRAINTS O RESTRICCIONES

Son reglas que siempre deben cumplir los valores de los atributos de las tablas. Se distinguen cinco tipos:

- NOT NULL, se aplica sobre un atributo de una tabla y fuerza a que éste no tome valores nulos.
- UNIQUE KEY, se aplica sobre uno o más atributos de una misma tabla, y garantiza que no se repite el valor del atributo, o el conjunto de valores de los atributos, en todos los registros de la tabla. (UK)
- PRIMARY KEY, se aplica sobre uno o más atributos de una tabla y permite que éste, o éstos, identifiquen de forma unívoca a las filas de la tabla. (PK)
- FOREIGN KEY, se aplica sobre un atributo de una tabla¹, de forma que bien toma el valor NULL, o bien toma uno de los valores del atributo de otra tabla al que apunta o referencia. El atributo foreign key nunca tomará un valor nulo que no exista entre los valores del atributo referenciado de la otra tabla.(FK)
- CHECK, se aplica sobre uno o más atributos y permite especificar condiciones. (CK)

2.3 VISTAS

Las vistas representan un subconjunto de una o más tablas. En la BD sólo se almacena la definición de la vista y no el resultado de la misma.

2.4 SECUENCIAS

Es un objeto que permite generar valores únicos. Se emplean para obtener las claves primarias de las tablas.

2.5 ÍNDICES

Los índices permiten que las filas de las consultas se recuperen más rápidamente, ayudan a buscar una palabra en menos tiempo, como ocurre con las palabras que aparecen en las esquinas superiores de un diccionario.

Oracle utiliza índices por defecto de tipo árbol B*. También existen índices de tipo BIT MAP que se basan en operaciones a nivel de bits.

2.6 CLUSTERS O AGRUPAMIENTOS

Son estructuras de almacenamiento que guardan cada fila de una tabla maestra junto a las correspondientes filas de otra tabla esclava. Agiliza determinados *joins* pero retarda las operaciones de DML. Los clusters sólo influyen a nivel físico de almacenamiento, el usuario sigue viendo igual sus tablas.

2.7 TRIGGERS O DISPARADORES

Son segmentos de código PL/SQL que se ejecutan cuando se producen operaciones de DML. Los triggers se estudian con detalle en el Tema IX, se definen a nivel de tabla y se clasifican según tres parámetros:

- Before/After, si se ejecuta antes o después de la sentencia.
- Insert/Update/Delete, según sea la sentencia que se ejecute.
- Each row/Statement dependiendo si se ejecuta para cada fila afectada en la sentencia, o bien una sola vez por sentencia con independencia de las filas de la tabla afectadas.

El código de los triggers no puede ejecutar ni commit ni rollback.

2.8 SNAPSHOTS O INSTANTÁNEAS

Si de las vistas almacenábamos su definición, de las instantáneas almacenamos el resultado de una consulta. Las instantáneas son fotografías de datos de una o más tablas tomadas en un instante determinado.

Existen dos métodos de refresco, el *Full refresh* que ejecuta y almacena la consulta de nuevo, y el *Fast refresh* que se apoya en ficheros de logs que le indican los cambios desde el último refresco.

2.9 DB LINKS O ENLACES A OTRAS BASES DE DATOS

Los DB Links permiten acceder a datos de tablas de una base de datos distinta a la base de datos desde la cual se ejecuta la consulta, todo ello dentro de una sesión.

2.10 PROCEDIMIENTOS, FUNCIONES Y PAQUETES

Oracle permite almacenar código PL/SQL en la Base de Datos: procedimientos, funciones y paquetes. Este código podrá ser ejecutado desde las sentencias SQL.

2.11 SINÓNIMOS

Los sinónimos son alias de nombres de objetos. Normalmente, en un sistema existirá un usuario administrador propietario de todas las tablas, por ejemplo `pro_admin`. Para acceder a cualquier tabla de este usuario, conectados como otro usuario, tendríamos que escribir `pro_admin.tabla`. Resulta cómodo definir sinónimos públicos con el nombre de la tabla de forma que el resto de usuarios accedan a la tabla como `tabla`.

2.12 USUARIOS

Son los diferentes usuarios de la Base de Datos. Cada uno es propietario de sus objetos.

2.13 PERFILES

Los perfiles permiten definir limitaciones de recursos. Por ejemplo, podemos definir un perfil que limite el número de sesiones abiertas concurrentemente por un usuario cualquiera y, posteriormente, aplicar este perfil a uno o más usuarios concretos.

2.14 PRIVILEGIOS

Los privilegios pueden ser de dos tipos:

- Privilegios del sistema, como por ejemplo para crear tablas.
- Privilegios sobre objetos, como por ejemplo permiso de SELECT sobre una tabla concreta.

2.15 ROLES

Los roles son conjuntos de privilegios. Un rol puede tener garantizados una serie de privilegios tanto del sistema como sobre objetos, y a la vez puede tener garantizado otros roles.

2.16 CONSULTAS AL DICCIONARIO DE DATOS

Podemos visualizar los objetos de la BD consultando las siguientes vistas del Diccionario de datos:

Vistas del Diccionario de Datos

- **DBA_XXX** objetos de toda la BD.
 - **ALL_XXX** objetos a los que tiene acceso el usuario.
 - **USER_XXX** objetos que son propiedad del usuario.

Donde XXX son los distintos objetos: TABLES, VIEWS, etc.

Podrá consultar todos los objetos de la base de datos en la tabla DICTIONARY.

3 CREACIÓN DE OBJETOS DE LA BASE DE DATOS (DDL)

Antes de comenzar con la creación de objetos de la base de datos Oracle, debemos **instalar un SGBD Oracle y una base de datos.**

Descargar Oracle 9i y SQL de Oracle8i

Para ello, podemos dirigirnos a la web <http://technet.oracle.com> bajo la sección de productos, podemos proceder a la descarga de la versión Oracle 9i en sus distintas versiones (Personal, Estándar o Enterprise).

[Oracle9i Database Release 2 Enterprise/Standard/Personal/Client Edition for Windows XP 2003/Windows Server 2003 \(64-bit\)](http://technet.oracle.com/technet/b/cyberzone/library/qa/qasql/qasql9i.htm)

[Oracle9i Database Release 2 Enterprise/Standard Edition for HP-UX/IA64 \(v9.2.0.2\)](http://technet.oracle.com/technet/b/cyberzone/library/qa/qasql/qasql9i.htm)

[Oracle9i Database Release 2 Enterprise Edition for HP Alpha OpenVMS](http://technet.oracle.com/technet/b/cyberzone/library/qa/qasql/qasql9i.htm)

[Oracle9i Database Release 2 Enterprise/Standard Edition for Linux/IA64, Release 2 \(v9.2.0.2\)](http://technet.oracle.com/technet/b/cyberzone/library/qa/qasql/qasql9i.htm)

Dirección completa de descarga: <http://www.oracle.com/technology/software/products/oracle9i/index.html>

Para seleccionar la versión a descargar, debemos previamente leer la documentación incluida en las mismas páginas donde se indican los requisitos (tanto hardware como software) necesarios para soportar la instalación.

Una vez realizada la descarga, y siguiendo las instrucciones para proceder a su instalación, contenidas en el fichero "*readme.txt*" o dentro de la documentación incluida, podemos realizar la instalación del SGBD Oracle 9i. **Se recomienda la instalación típica, si se realiza por primera vez.** En la instalación típica, el asistente nos instalará todo el software necesario y nos creará y configurará una base de datos por defecto, preguntándonos antes el nombre que queremos dar a la misma.

Este nombre será utilizado posteriormente para configurar los ficheros del protocolo de comunicaciones de los clientes con la base de datos ("*listener.ora*" y "*tnsnames.ora*").

Por ejemplo, si el nombre que le hemos dado a la base de datos en la instalación es **PRUEBA**, en el fichero "*listener.ora*" situado dentro del subdirectorio "*network\admin*" del directorio que hemos indicado para la instalación (p.e. C:\ORANT) debe existir una entrada similar a la siguiente:

```

LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
      )
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = maquina)(PORT = 1521))
      )
    )
  )
)
)

SID_LIST_LISTENER =
  (SID_DESC =
    (GLOBAL_DBNAME = PRUEBA.WORLD)
    (ORACLE_HOME = c:\orant)
    (SID_NAME = PRUEBA)
  )
)

```

donde **maquina** indica el nombre o la IP de la máquina donde se ha instalado el SGBD Oracle8i.

Para poder realizar conexiones con los distintos clientes a la base de datos creada, debemos tener configurado una cadena de conexión en el fichero "*tnsnames.ora*" situado en el mismo directorio, cadena similar a la siguiente:

```
MIBASE.WORLD =  
(DESCRIPTION =  
  (ADDRESS_LIST =  
    (ADDRESS = (PROTOCOL = TCP)(HOST = maquina)(PORT = 1521))  
  )  
  (CONNECT_DATA =  
    (SERVICE_NAME = PRUEBA.WORLD)  
  )  
)
```

donde **MIBASE.WORLD** (o el string incluido por defecto con la instalación) será la cadena de conexión requerida por cada conexión que se realice desde las aplicaciones de este cliente con la base de datos.

En la instalación típica se crean unos servicios que se arrancan automáticamente con la máquina, dejando la base de datos abierta.

3.1 SQL*PLUS

SQL*PLUS es una herramienta de Oracle que reconoce y envía sentencias SQL al servidor. En la instalación típica del software Oracle 9i se instalan varias versiones de SQL*PLUS (plus80, plus80w, sqlplus). Podemos realizar la creación de objetos con cualquiera de ellas.

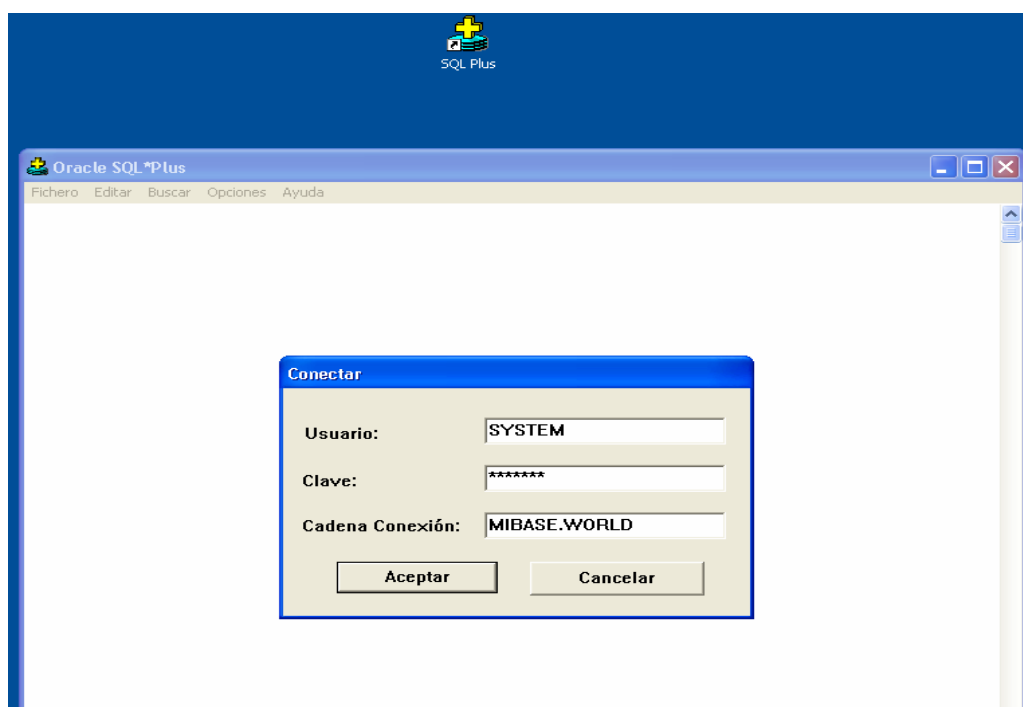
Para conectarnos a la herramienta sqlplus desde la línea de comandos:

```
sqlplus usuario[/contraseña[@cadena_conexion]]
```



donde *cadena_conexión* es la cadena de conexión definida en el fichero "**tnsnames.ora**"

Podemos utilizar también el acceso al programa instalado en el menú de windows bajo las opciones de Oracle. Si arrancamos este acceso directo, nos aparecerá una ventana de conexión con tres cajas de texto para indicar el usuario, contraseña y cadena de conexión respectivamente.



3.2 USUARIOS

Para crear un usuario nos conectamos a la base datos como administradores, por ejemplo como **system**, que tiene privilegios para crear usuarios.

Para crear un usuario ejecutamos:

```
CREATE USER nombre IDENTIFIED BY CONTRASEÑA;  
CREATE USER CURSO_GATE IDENTIFIED BY CURSO_GATE;
```

Si ahora intentamos conectarnos

```
CONNECT CURSO_GATE/CURSO_GATE
```

aparecerá un error debido a que todavía no tenemos permiso para crear una sesión.

Lo siguiente que debemos hacer es dotar al nuevo usuario de ese permiso. Para ello, nuevamente, desde un usuario administrador le garantizamos el rol CONNECT al nuevo usuario. Desde **sys** o **system** ejecutamos:

```
GRANT CONNECT TO CURSO_GATE;
```

Ahora ya podemos conectarnos. Si además el nuevo usuario va a crear objetos de la base de datos, le garantizaremos el rol RESOURCE

```
GRANT RESOURCE TO CURSO_GATE;
```

Para borrar un usuario ejecutamos la sentencia:

```
DROP USER usuario [CASCADE];
```

La cláusula CASCADE elimina los objetos de los que fuera propietario el usuario.

3.3 TABLAS

- Crear una tabla:

```
CREATE TABLE [schema.] tabla
    (columna tipo_dato [DEFAULT expr]
    [, columna tipo_dato [DEFAULT expr] ...]);
```

La expresión por defecto introduce ese valor al insertar un registro sin dar un valor a esa columna.

```
CREATE TABLE DEPARTAMENTOS (
    ID          NUMBER(10),
    NOMBRE      VARCHAR2(50));

DESC DEPARTAMENTOS;
```

También podemos crear una tabla por medio de subconsultas:

```
CREATE TABLE [schema.] tabla
    (columna [, columna ...])
AS subconsulta
```

```
CREATE TABLE DEPARTAMENTOS_AUX
AS SELECT * FROM DEPARTAMENTOS;
```

- Alterar una tabla:

- Añadir una nueva columna.
- Modificar una columna que ya existe
- Dar un valor por defecto a una nueva columna.

```
ALTER TABLE tabla
ADD (columna tipo_dato [DEFAULT expr]
    [, columna tipo_dato [DEFAULT expr] ...]);
ALTER TABLE tabla
MODIFY (columna tipo_dato [DEFAULT expr]
    [, columna tipo_dato [DEFAULT expr] ...]);
```

```
ALTER TABLE DEPARTAMENTOS
ADD LOCALIDAD VARCHAR2(10);
```

```
ALTER TABLE DEPARTAMENTOS
MODIFY LOCALIDAD VARCHAR2(50);
```

- Borrar una tabla:

```
DROP TABLE tabla [CASCADE CONSTRAINTS];
```

Borra la tabla, datos y estructura. La cláusula CASCADE CONSTRAINTS borra todas las referencias a la tabla, es decir elimina las foreign keys de otras tablas que la apuntan.

- Truncar una tabla:

```
TRUNC TABLE tabla;
```

Borra los datos de la tabla pero no la estructura. Equivale a realizar un DELETE tabla con la diferencia de que la sentencia TRUNC libera todo el espacio de almacenamiento ocupado por la tabla, mientras que DELETE no libera espacio.

3.4 CONSTRAINTS O RESTRICCIONES

Las restricciones o constraints se emplean para evitar la entrada de datos no íntegros en la base de datos. Conviene asignar un nombre a las constraints, de lo contrario el sistema las nombra como SYS_Cn, donde n es un número.

Se pueden definir restricciones a nivel de columna o a nivel de tabla.

- Crear una constraint:

- Podemos crear restricciones al crear la tabla.

```
CREATE TABLE [schema.] tabla
    (columna tipo_dato [DEFAULT expr]
    [restriccion_columna],
    ...
    [restriccion_tabla]);
```

- Restricciones a nivel de columna:

```
columna [CONSTRAINT restriccion] tipo_restriccion,
```

- Restricciones a nivel de tabla:

```
columna, ...
[CONSTRAINT restriccion] tipo_restriccion
(columna, ...),
```

No se puede definir una constraint NOT NULL a nivel de tabla.

- También podemos añadir restricciones de unicidad (UK), clave primaria (PK) o condición (CK) después de crear la tabla.

```
ALTER TABLE tabla
ADD [CONSTRAINT restriccion] tipo_restriccion (columna)
```

- Para añadir restricciones de clave externa (FK) empleamos la siguiente sentencia:

```
ALTER TABLE tabla1
ADD CONSTRAINT restriccion
FOREIGN KEY(columna1)
REFERENCES tabla2 (columna2)
[ON DELETE CASCADE];
```

Donde la columna1 de la tabla1 referencia valores de la columna2 de la tabla2. Cuando borramos una fila de la tabla padre (tabla2), la cláusula ON DELETE CASCADE se encarga de borrar las filas dependientes de la tabla hija (tabla1). Sin la opción ON DELETE CASCADE no se puede borrar una fila de la tabla padre que esté referenciada por filas de la tabla hija.

- Para añadir restricciones de no permitir valor nulo, se altera la tabla utilizando la cláusula MODIFY.

```
ALTER TABLE tabla  
MODIFY columna tipo_dato [DEFAULT expr] [NOT] NULL;
```

Tabla de departamentos añadiendo la clave primaria en la creación de la tabla, así como exigiendo la no nulidad de la clave:

```
CREATE TABLE DEPARTAMENTOS (  
  ID          NUMBER(10) NOT NULL,  
  NOMBRE      VARCHAR2(50),  
  LOCALIDAD  VARCHAR2(50),  
  CONSTRAINT DEPARTAMENTOS_PK  
  PRIMARY KEY (ID));
```

Normalmente, la clave primaria de las tablas se implementa a través de un identificador interno o ID que es totalmente transparente para el usuario final. El ejemplo anterior muestra la forma de definirla primary key. La clave primaria exige la unicidad y no nulidad. Oracle implementa las claves primarias y las unique key con índices únicos.

Tabla de empleados añadiendo la clave primaria:

```
CREATE TABLE EMPLEADOS (  
  ID          NUMBER(10) NOT NULL,  
  NOMBRE      VARCHAR2(30),  
  APELLIDOS   VARCHAR2(60),  
  CARGO       VARCHAR2(20),  
  JEFE_ID     NUMBER(10),  
  FECHA_ALTA  DATE,  
  SUELDO      NUMBER(9,2),  
  COMISION    NUMBER(9,2),  
  DEPARTAMENTO_ID NUMBER(10),  
  CONSTRAINT EMPLEADOS_PK  
  PRIMARY KEY (ID));
```

Obligar a que todo empleado trabaje en un departamento.

```
ALTER TABLE EMPLEADOS  
MODIFY DEPARTAMENTO_ID NOT NULL;
```

O bien al crear la tabla:

```
CREATE TABLE EMPLEADOS (  
  ID          NUMBER(10) NOT NULL,  
  NOMBRE      VARCHAR2(30),  
  APELLIDOS   VARCHAR2(60),  
  CARGO       VARCHAR2(20),  
  JEFE_ID     NUMBER(10),  
  FECHA_ALTA  DATE,  
  SUELDO      NUMBER(9,2),  
  COMISION    NUMBER(9,2),  
  DEPARTAMENTO_ID NUMBER(10) NOT NULL,  
  CONSTRAINT EMPLEADOS_PK  
  PRIMARY KEY (ID));
```

Añadir una FOREIGN KEY para que la columna DEPARTAMENTO_ID de la tabla EMPLEADOS referencie valores de la columna ID de la tabla DEPARTAMENTOS.

```
ALTER TABLE EMPLEADOS
ADD CONSTRAINT EMP_DEPARTAMENTO_ID_FK
  FOREIGN KEY (DEPARTAMENTO_ID)
  REFERENCES DEPARTAMENTOS (ID);
```

Añadir una FOREIGN KEY para que la columna JEFE_ID de la tabla EMPLEADOS referencie valores de la columna ID de la misma tabla.

```
ALTER TABLE EMPLEADOS
ADD CONSTRAINT EMP_JEFE_ID_FK
  FOREIGN KEY (JEFE_ID)
  REFERENCES EMPLEADOS (ID);
```

Exigir que el nombre de los departamentos sea único.

```
ALTER TABLE DEPARTAMENTOS
ADD CONSTRAINT DEP_NOMBRE_UK UNIQUE (NOMBRE);
```

Impedir que algún empleado tenga un sueldo mensual superior a 6000 €.

```
ALTER TABLE EMPLEADOS
ADD CONSTRAINT EMP_SUELDO_CK
  CHECK (SUELDO <= 6000);
```

- Borrar una restricción.

```
ALTER TABLE tabla
DROP PRIMARY KEY | UNIQUE (columna) | CONSTRAINT restriccion
  [CASCADE];
```

La opción CASCADE provoca que cualquier restricción dependiente también se borre.

```
ALTER TABLE EMPLEADOS
DROP CONSTRAINT EMP_JEFE_ID_FK;
```

```
ALTER TABLE EMPLEADOS
DROP CONSTRAINT EMP_SUELDO_CK;
```

```
ALTER TABLE EMPLEADOS
MODIFY DEPARTAMENTO_ID NULL;
```

```
ALTER TABLE DEPARTAMENTOS
DROP PRIMARY KEY CASCADE;
```

```
ALTER TABLE DEPARTAMENTOS
DROP UNIQUE(NOMBRE);
```

- Activar una restricción.

```
ALTER TABLE tabla
ENABLE CONSTRAINT restriccion;
```

- Desactivar una restricción.

```
ALTER TABLE tabla
DISABLE CONSTRAINT restriccion [CASCADE];
```

La opción CASCADE desactiva las restricciones de integridad dependientes.

- Algunas consultas al Diccionario de Datos:

```
SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, SEARCH_CONDITION
FROM USER_CONSTRAINTS
WHERE TABLE_NAME = 'EMPLEADOS';
```

La columna CONSTRAINT_TYPE toma los siguientes valores:

- C Check (la restricción NOT NULL realmente es una CHECK)
- P Primary key
- R Referencia de integridad, foreign key
- U Unique key

```
SELECT CONSTRAINT_NAME, COLUMN_NAME
FROM USER_CONS_COLUMNS
WHERE TABLE_NAME = 'EMPLEADOS';
```

3.5 VISTAS

Una vista es una tabla lógica basada en otra tabla o vista. Una vista es una ventana a través de la cual se pueden consultar datos y a veces modificarlos.

- Crear una vista.

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW vista
[(alias[, alias]...)]
AS subconsulta
[WITH CHECK OPTION [CONSTRAINT restriccion]]
[WITH READ ONLY]
```

- La subconsulta nunca debe llevar la cláusula ORDER BY.
- OR REPLACE recrea la vista si ya existe.
- FORCE crea la vista con independencia de que la tabla base exista o no.
- NOFORCE crea la vista sólo si la tabla base existe. Es la opción por defecto.
- alias es el nombre de las columnas de la vista.
- WITH CHECK OPTION garantiza que solamente las filas accesibles a la vista puedan ser insertadas o actualizadas.
- restriccion es el nombre de la restricción WITH CHECK OPTION.
- WITH READ ONLY impide realizar operaciones de DML sobre la vista.

Empleados del departamento de Informática.

```
CREATE OR REPLACE VIEW EMP_INFORMATICA_VIEW AS (
SELECT *
FROM EMPLEADOS
WHERE DEPARTAMENTO_ID = 1);
```

Empleados y sus departamentos.

```
CREATE OR REPLACE VIEW EMP_DEP_VIEW (EMPLEADO, DEPARTAMENTO) AS (
SELECT EMP.NOMBRE, DEP.NOMBRE
FROM EMPLEADOS EMP, DEPARTAMENTOS DEP
WHERE EMP.DEPARTAMENTO_ID = DEPARTAMENTOS.ID);
```

Sueldo mensual de los empleados

```
CREATE OR REPLACE VIEW EMP_SUELDO_VIEW AS (  
SELECT NOMBRE, APELLIDOS, SUELDO  
FROM EMPLEADOS);
```

Forzar la vista anterior para que sea de solo lectura.

```
CREATE OR REPLACE VIEW EMP_SUELDO_VIEW AS (  
SELECT NOMBRE, APELLIDOS, SUELDO  
FROM EMPLEADOS)  
WITH READ ONLY;
```

Para realizar DML sobre vistas se deben cumplir las siguientes condiciones:

- **UPDATE Y DELETE sólo si:**
 - Utiliza una sola tabla.
 - No utiliza la cláusula GROUP BY.
 - No utiliza la cláusula DISTINCT.
- **INSERT sólo si:**
 - Satisface los requisitos de UPDATE y DELETE
 - No contiene columnas definidas por expresiones.

```
SELECT *  
FROM EMP_INFORMATICA_VIEW;
```

```
UPDATE EMP_INFORMATICA_VIEW  
SET NOMBRE = INITCAP(LOWER(NOMBRE));
```

```
SELECT *  
FROM EMP_INFORMATICA_VIEW;
```

```
UPDATE EMP_INFORMATICA_VIEW  
SET DEPARTAMENTO_ID = 2;
```

```
SELECT *  
FROM EMP_INFORMATICA_VIEW;
```

Sea la consulta: ¿Dónde están los empleados de informática?

Para evitar este comportamiento creamos la vista con la cláusula WITH CHECK OPTION.

```
CREATE OR REPLACE VIEW EMP_INFORMATICA_VIEW AS (  
SELECT *  
FROM EMPLEADOS  
WHERE DEPARTAMENTO_ID = 1)  
WITH CHECK OPTION;
```

- Recuperar datos de una vista. Se consulta igual que si fuera una tabla.
- Modificar una vista. Se re-crea.

```
CREATE OR REPLACE VIEW ...
```

- Borra una vista.

```
DROP VIEW vista;
```

3.6 SECUENCIAS

Una secuencia es un objeto que genera números únicos de forma automática.

▪ Crear una secuencia.

```
CREATE SEQUENCE secuencia
  [INCREMENT BY n]
  [START WITH n]
  [{MAXVALUE n | NOMAXVALUE}]
  [{MINVALUE n | NOMINVALUE}]
  [{CYCLE | NOCYCLE}]
  [{CACHE n | NOCACHE}];
```

- Las opciones por defecto aparecen en negrita.
- INCREMENT BY n, especifica el intervalo (n) entre los números de la secuencia. n toma el valor 1 por defecto.
- START WITH n, comienza por n. Por defecto, n toma el valor 1
- NOMAXVALUE especifica un valor máximo de 10^{27} para secuencias ascendentes y -1 para descendentes.
- NOMINVALUE especifica un valor mínimo de 1 para secuencias ascendentes y $-(10^{27})$ para descendentes.
- CYCLE | NOCYCLE especifica si la secuencia continúa generando valores después de llegar a su valor máximo.
- CACHE n | NOCACHE especifica cuántos valores serán preasignados y mantenidos en la memoria del servidor. Por defecto es 20.

Secuencia para generar la clave primaria de los empleados.

```
CREATE SEQUENCE EMPLEADOS_SEQ;
```

▪ Recuperar valores de una secuencia.

Oracle proporciona dos pseudocolumnas para trabajar con las secuencias:

- CURRVAL, que devuelve el valor actual de la secuencia:

```
SELECT EMPLEADOS_SEQ.CURRVAL
FROM DUAL;
```

- NEXTVAL, incrementa la secuencia y devuelve el nuevo valor:

```
SELECT EMPLEADOS_SEQ.NEXTVAL
FROM DUAL;
```

▪ Modificar una secuencia.

```
ALTER SEQUENCE secuencia
  [INCREMENT BY n]
  [{MAXVALUE n | NOMAXVALUE}]
  [{MINVALUE n | NOMINVALUE}]
  [{CYCLE | NOCYCLE}]
  [{CACHE n | NOCACHE}];
```

▪ Borrar una secuencia.

```
DROP SEQUENCE secuencia;
```

```
DROP SEQUENCE EMPLEADOS_SEQ;
```

3.7 ÍNDICES

Los índices aceleran la recuperación de filas en las consultas. Oracle se encarga de usarlos y mantenerlos automáticamente.

- Crear un índice.

```
CREATE INDEX indice  
ON table (columna[, columna]...);
```

Siempre vamos a crear índices sobre las claves externas y primarias.

Podemos crear el índice de la clave primaria de una tabla al crear la tabla:

```
CREATE TABLE DEPARTAMENTOS (  
  ID          NUMBER(10) NOT NULL,  
  NOMBRE     VARCHAR2(50),  
  LOCALIDAD  VARCHAR2(50),  
  CONSTRAINT DEPARTAMENTOS_PK  
    PRIMARY KEY (ID)  
    USING INDEX);  
  
CREATE INDEX EMP_DEPARTAMENTO_ID_IDX  
ON EMPLEADOS (DEPARTAMENTO_ID);  
  
CREATE INDEX EMP_JEFE_ID_IDX  
ON EMPLEADOS (JEFE_ID);
```

- Borrar un índice.

```
DROP INDEX indice;  
  
DROP INDEX EMP_JEFE_ID_IDX;
```

3.8 SINÓNIMOS

Los sinónimos simplifican el acceso a los objetos. Por ejemplo, en lugar de referirnos a la tabla SYS.DUAL, nos referimos a la tabla DUAL, siendo DUAL un sinónimo de SYS.DUAL.

- Crear un sinónimo.

```
CREATE [PUBLIC] SYNONYM sinonimo FOR objeto;
```

PUBLIC crea un sinónimo accesible por todos los usuarios.

```
CREATE PUBLIC SYNONYM EMPLEADOS FOR CURSO_GATE.EMPLEADOS;
```

```
SELECT *  
FROM EMPLEADOS;
```

```
SELECT *  
FROM CURSO_GATE.EMPLEADOS;
```

- Borrar un sinónimo

```
DROP [PUBLIC] SYNONYM sinonimo;  
  
DROP PUBLIC SYNONYM EMPLEADOS;
```

3.9 EJERCICIOS

1. Borrar el usuario CURSO_GATE si existe eliminando todos sus objetos.
2. Crear el usuario CURSO_GATE, garantizarle los roles CONNECT y RESOURCE, y conectarse como este usuario.
3. Crear la tabla de Departamentos (DEPARTAMENTOS) con la siguiente estructura:

```
ID          NUMBER(10)
NOMBRE      VARCHAR2(50)
LOCALIDAD   VARCHAR2(50)
```

ID es la clave primaria, no nula y con índice

4. Crear la tabla de Empleados (EMPLEADOS) con la siguiente estructura:

```
ID          NUMBER(10)
NOMBRE      VARCHAR2(30)
APELLIDOS   VARCHAR2(60)
CARGO       VARCHAR2(20)
JEFE_ID     NUMBER(10)
FECHA_ALTA  DATE
SUELDO      NUMBER(9,2)
COMISION    NUMBER(9,2)
DEPARTAMENTO_ID NUMBER(10)
```

ID es la clave primaria, no nula y con índice. DEPARTAMENTO_ID no nulo.

5. Crear una foreign key del atributo DEPARTAMENTO_ID de la tabla EMPLEADOS referenciando la clave primaria de la tabla DEPARTAMENTOS.

Crear una foreign key del atributo JEFE_ID de la tabla EMPLEADOS referenciando la clave primaria de la misma tabla.

6. Crear índices sobre las claves externas creadas anteriormente.
7. Crear la tabla de Rangos de sueldos (RANGOS_SUELDO) con la siguiente estructura:

```
GRADO      NUMBER(2)
SUELDO_MIN NUMBER(9,2)
SUELDO_MAX NUMBER(9,2)
```

8. Insertar los datos de las tres con las sentencias INSERT del Anexo.

4 ACESO A DATOS MEDIANTE SQL (SENTENCIA SELECT). FUNCIONES ESTÁNDAR.

4.1 Sentencia SELECT Básica

Una SELECT recupera información de la BD. La definición básica es la siguiente:

```
SELECT [DISTINCT] {*, columna [[AS] alias]}
FROM tabla;
```

SELECT identifica qué columnas forman parte de la respuesta resultante
FROM identifica desde qué tablas se extraen los datos respuesta

- Palabra clave: hace referencia a un elemento SQL individual. ("SELECT", "FROM")
- Cláusula: es una parte de una sentencia SQL. ("SELECT ID, NOMBRE")
- Sentencia: combinación de dos o más cláusulas. ("SELECT * FROM DEPARTAMENTOS")

Seleccionar todas las columnas de una tabla

```
SELECT *  
FROM DEPARTAMENTOS ;
```

Seleccionar algunas columnas de una tabla

```
SELECT NOMBRE, APELLIDOS  
FROM EMPLEADOS ;
```

Operadores aritméticos

- + Suma
- - Resta
- * Multiplicación
- / División

```
SELECT NOMBRE, SUELDO, SUELDO + 35000  
FROM EMPLEADOS ;
```

Precedencia de los operadores:

- La multiplicación y división tienen prioridad sobre la suma y resta.
- Los operadores de igual prioridad se evalúan de izquierda a derecha.
- Los paréntesis permiten cambiar la prioridad.

```
SELECT NOMBRE, SUELDO, 12 * SUELDO + 35000  
FROM EMPLEADOS ;
```

```
SELECT NOMBRE, SUELDO, 12 * (SUELDO + 35000)  
FROM EMPLEADOS ;
```

Valor NULL

Null es un valor inaccesible, no es un cero ni un espacio en blanco.

```
SELECT NOMBRE, COMISION  
FROM EMPLEADOS ;
```

Las expresiones aritméticas que contengan un Null se evalúan a Null.

```
SELECT NOMBRE, 12*SUELDO+COMISION  
FROM EMPLEADOS ;
```

Alias de columnas

Los alias de columna permiten cambiar los títulos de las columnas que visualizamos en una consulta.

Si el alias contiene espacios, caracteres especiales (+, *) o minúsculas debemos encerrarlo entre comillas dobles.

```
SELECT ID IDENTIFICADOR, NOMBRE "Nombre",  
       12*SUELDO+COMISION "Salario anual"  
FROM EMPLEADOS;
```

Operador de concatenación de caracteres ||

Este operador concatena columnas de tipo carácter o cadenas de caracteres con otras columnas de tipo carácter o cadenas de caracteres.

```
SELECT NOMBRE || ' ' || APELLIDOS "Nombre completo"  
FROM EMPLEADOS;
```

```
SELECT NOMBRE || ' ' || APELLIDOS || ' es ' || CARGO "Empleados"  
FROM EMPLEADOS;
```

Cómo evitar la selección de filas duplicadas

```
SELECT CARGO  
FROM EMPLEADOS;
```

Las filas duplicadas se eliminan usando DISTINCT en la cláusula SELECT.

```
SELECT DISTINCT CARGO  
FROM EMPLEADOS;
```

Estructura de una tabla

```
DESC[RIBE] tabla;  
DESC EMPLEADOS;
```

Ejecutar un archivo desde SQL*Plus

```
START nombre_fichero;  
@ nombre_fichero;
```

4.1.1 Ejercicios

9. ¿Qué errores presenta la siguiente sentencia?

```
SELECT NOMBRE, APELLIDOS  
       SUELDO x 12 Salario anual  
FROM EMPLEADOS;
```

10. Mostrar la estructura de la tabla DEPARTAMENTOS

11. Seleccionar a los empleados como sigue:

```
LOPEZ SEGUNDO, CARLOS: es PRESIDENTE
```

12. Consultar los diferentes cargos desempeñados en la empresa.

4.2 SELECCIÓN DE FILAS DE UNA TABLA

Para filtrar las filas que devuelve una consulta SELECT, añadimos la cláusula WHERE:

```
SELECT [DISTINCT] {*, columna [[AS] alias]}
FROM tabla
[WHERE condicion(es)];
```

Cada condición se compone de nombres de columnas, expresiones y operadores de comparación.

```
SELECT NOMBRE, APELLIDOS, CARGO
FROM EMPLEADOS
WHERE CARGO = 'COMERCIAL';
```

Operadores de comparación

- = Igual a
- > Mayor que
- >= Mayor que o igual a
- < Menor que
- <= Menor que o igual a
- <> Distinto de
- BETWEEN ... AND ... Entre dos valores (incluidos)
- IN (lista de valores) Coincide con algún valor de la lista
- LIKE 'patrón' Se ajusta a un patrón
- IS NULL Es un valor nulo

Las cadenas de caracteres se encierran entre comillas simples.

```
SELECT NOMBRE, APELLIDOS, CARGO
FROM EMPLEADOS
WHERE NOMBRE = 'INES';
```

```
SELECT NOMBRE, APELLIDOS, SUELDO
FROM EMPLEADOS
WHERE SUELDO > 200000;
```

```
SELECT NOMBRE, APELLIDOS, SUELDO
FROM EMPLEADOS
WHERE SUELDO BETWEEN 150000 AND 250000;
```

```
SELECT NOMBRE, APELLIDOS, SUELDO
FROM EMPLEADOS
WHERE CARGO IN ('COMERCIAL', 'PRESIDENTE');
```

El patrón del operador LIKE puede contener:

- “%” cualquier cadena de caracteres, incluso vacía.
- “_” un solo carácter.

P.e. los empleados cuyo nombre comienza por la letra 'C'

```
SELECT NOMBRE, APELLIDOS, SUELDO
FROM EMPLEADOS
WHERE NOMBRE LIKE 'C%';
```

```
SELECT NOMBRE||'|'|APELLIDOS "Empleados sin jefe"
FROM EMPLEADOS
WHERE JEFE_ID IS NULL;
```

```
SELECT NOMBRE||','||APELLIDOS "Empleados sin comisión"  
FROM EMPLEADOS  
WHERE COMISION IS NULL;
```

Operadores Lógicos

- **AND** Devuelve verdadero si ambas condiciones son verdaderas.
- **OR** Devuelve verdadero si alguna condición es verdadera.
- **NOT** Devuelve verdadero si la siguiente condición es falsa.

AND	V	F	NULL
V	V	F	NULL
F	F	F	F
NULL	NULL	F	NULL

OR	V	F	NULL
V	V	V	V
F	V	F	NULL
NULL	V	NULL	NULL

NOT	V	F	NULL
	F	V	NULL

Empleados con cargo comercial cuyo sueldo mensual es superior a 3100 €

```
SELECT NOMBRE, CARGO, SUELDO  
FROM EMPLEADOS  
WHERE (CARGO = 'COMERCIAL')  
AND (SUELDO > 3100);
```

Empleados con apellido LOPEZ o su nombre comience por A:

```
SELECT NOMBRE, APELLIDOS  
FROM EMPLEADOS  
WHERE (APELLIDOS LIKE '%LOPEZ%')  
OR (NOMBRE LIKE 'A%');
```

Empleados con comisión:

```
SELECT NOMBRE, APELLIDOS, COMISION  
FROM EMPLEADOS  
WHERE COMISION IS NOT NULL;
```

Empleados cuyo sueldo no esté en el rango de los 2000 a 3000 €.

```
SELECT NOMBRE, SUELDO  
FROM EMPLEADOS  
WHERE SUELDO NOT BETWEEN 2000 AND 3000;
```

Empleados cuyo nombre no empiece por C:

```
SELECT NOMBRE  
FROM EMPLEADOS  
WHERE NOMBRE NOT LIKE 'C%';
```

Empleados que no sean comerciales, ni presidentes:

```
SELECT NOMBRE, CARGO  
FROM EMPLEADOS  
WHERE CARGO NOT IN ('COMERCIAL', 'PRESIDENTE');
```

Precedencia de operadores:

Primero los de comparación, luego NOT, después AND y finalmente OR.

Se recomienda utilizar paréntesis para especificar las condiciones de forma más clara. Además, los paréntesis permiten modificar el orden de precedencia.

4.2.1 Ejercicios

13. Mostrar el nombre, apellidos, salario y comisión; de los empleados que tienen jefe, y su comisión es superior a 2500 €.
14. Mostrar el nombre, sueldo y sueldo anual de los empleados cuyo sueldo anual supere los 36000 €.
15. Mostrar la lista de los comerciales con su sueldo.
16. Mostrar a los empleados que no ejercen de comerciales cuyo nombre contiene la letra N.

4.3 CONTROL DEL ORDEN DE VISUALIZACIÓN DE LAS FILAS

Para mostrar las filas devueltas por una consulta ordenadas, utilizamos la cláusula ORDER BY:

```
SELECT [DISTINCT] {*, columna [[AS] alias]}
FROM tabla
[WHERE condicion(es)]
[ORDER BY {columna, expresion, alias} [ASC|DESC]];
```

Empleados ordenados por fecha de alta.

```
SELECT NOMBRE, APELLIDOS, FECHA_ALTA
FROM EMPLEADOS
ORDER BY FECHA_ALTA
```

Empleados ordenados por salario de mayor a menor

```
SELECT NOMBRE, SUELDO
FROM EMPLEADOS
ORDER BY SALARIO DESC;
```

Empleados ordenados por cargo y nombre:

```
SELECT CARGO, NOMBRE
FROM EMPLEADOS
ORDER BY CARGO, NOMBRE;
```

Para ordenar estructuras jerárquicas, Oracle proporciona las cláusulas CONNECT BY y START WITH.

```
SELECT [DISTINCT] {*, columna [[AS] alias]}
FROM tabla
[WHERE condicion(es)]
[CONNECT BY condicion(es)]
[START WITH condicion(es)]
```

Empleados ordenados jerárquicamente debajo de su jefe:

```
SELECT LPAD(' ', 2*LEVEL) || NOMBRE  
FROM EMPLEADOS  
CONNECT BY PRIOR ID = JEFE_ID  
START WITH ID=6;
```

La función LPAD añade un sangrado dependiendo del nivel de la jerarquía, el cual lo proporciona la pseudo columna LEVEL.

La cláusula CONNECT BY expresa las condiciones de ordenación de las filas. En este caso indicamos que el valor del atributo ID de la fila anterior (PRIOR ID) coincida con el valor del atributo JEFE_ID de la fila actual (= JEFE_ID). Es decir, estaré debajo de mi jefe.

Por último, la cláusula START WITH expresa las condiciones de comienzo de la ordenación. En este ejemplo, el presidente, la cabeza de la jerarquía tiene el ID = 6.

4.3.1 Ejercicios

17. Consultar los empleados que obtienen comisión, ordenados por su sueldo de mayor a menor. Mostrar el nombre del empleado, su sueldo y su comisión.

18. Ordenar los empleados por sus apellidos y nombres.

19. Obtener el sueldo anual de los empleados que no tienen comisión ordenados por fecha de alta en la empresa. Mostrar el nombre, sueldo anual y fecha de alta.

20. Mostrar el sueldo de los empleados incrementado en un 10%, ordenados por nombre y apellidos.

4.4 FUNCIONES SQL

Existen dos tipos de funciones:

- Funciones de fila: operan sólo sobre filas y devuelven un único resultado por cada una de ellas. Se dividen en cuatro subtipos:
 - De carácter
 - De conversión (**LOWER, UPPER, INITCAP**)
 - De manipulación de caracteres (**CONCAT, SUBSTR, LENGHT, INSTR, LPAD, RPAD, LTRIM, RTRIM**)
 - Numéricas (**ROUND, TRUNC, MOD**)
 - De fechas (**MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY, ROUND, TRUNC**)
 - De conversión (**TO_CHAR, TO_NUMBER, TO_DATE, NVL, DECODE**)
- Funciones de grupos: operan sobre grupos de filas y devuelven un resultado por cada grupo de filas (se verán posteriormente).

4.4.1 LA TABLA DUAL

El usuario SYS es propietario de la tabla DUAL, la cual sólo tiene una columna y una fila. Se emplea para ejecutar funciones de fila sobre expresiones, cadenas, números que no son columnas de una tabla. Al tener una sola fila la tabla, retorna una sola vez el valor deseado.

```
DESC DUAL;
```

```
SELECT *  
FROM DUAL;  
  
SELECT 'Hola'  
FROM DUAL;
```

4.4.2 FUNCIONES DE CARÁCTER DE CONVERSIÓN

- **LOWER(char)** devuelve la columna o cadena char con todas las letras en minúscula.

```
SELECT NOMBRE, LOWER(NOMBRE)  
FROM EMPLEADOS;  
SELECT NOMBRE  
FROM EMPLEADOS  
WHERE LOWER(CARGO) = 'comercial';
```

- **UPPER(char)** devuelve la columna o cadena char con todas las letras en mayúsculas.

```
SELECT NOMBRE, UPPER('en mayúsculas')  
FROM EMPLEADOS;
```

- **INITCAP(char)** Convierte la primera letra de cada palabra en mayúscula y el resto las deja como están.

```
SELECT NOMBRE, LOWER(NOMBRE), INITCAP(LOWER(NOMBRE))  
FROM EMPLEADOS;  
  
SELECT INITCAP(LOWER(NOMBRE || ' ' || APELLIDOS)) "Empleados"  
FROM EMPLEADOS;
```

4.4.3 FUNCIONES DE CARÁCTER DE MANIPULACIÓN DE CARACTERES

- **CONCAT(char1, char2)** devuelve char1 concatenado con char2.

```
SELECT CONCAT(NOMBRE, APELLIDOS)  
FROM EMPLEADOS;
```

Equivale al operador ||

```
SELECT NOMBRE || APELLIDOS  
FROM EMPLEADOS;
```

- **SUBSTR(char, M[, N])** devuelve una porción de char, comenzando en el caracter M, de N caracteres de longitud.
 - Si M es positivo, Oracle cuenta desde el principio de la cadena.
 - Si M es negativo, Oracle cuenta desde el final de la cadena.
 - Si se omite N, Oracle devuelve todos los caracteres hasta el final de char.

```
SELECT NOMBRE, SUBSTR(NOMBRE, 2), SUBSTR(NOMBRE, 2, 1),  
SUBSTR(NOMBRE, -2)  
FROM EMPLEADOS;
```

- **LENGTH(char)** devuelve número de caracteres de la char.

```
SELECT NOMBRE, LENGTH(NOMBRE) "Longitud del nombre"  
FROM EMPLEADOS  
ORDER BY "Longitud del nombre";
```

- **INSTR(char1, char2[, N[, M]])** Busca en la cadena char1 desde el carácter N la M-ésima ocurrencia de la cadena char2, y devuelve la posición del carácter de char1 que es el primero de esta ocurrencia. Si N es negativo, Oracle cuenta y busca hacia atrás desde el final de char1. N y M por defecto toman el valor 1.

```
SELECT CARGO, INSTR(CARGO, 'E'), INSTR(CARGO, 'E', 1, 2)  
FROM EMPLEADOS  
WHERE CARGO = 'PRESIDENTE'
```

```
SELECT APELLIDOS,  
SUBSTR(APELLIDOS, 1, INSTR(APELLIDOS, ' ')-1) "Primer apellido",  
SUBSTR(APELLIDOS, INSTR(APELLIDOS, ' ')+1) "Segundo apellido"  
FROM EMPLEADOS;
```

- **LPAD(char1, N[, char2])** Devuelve char1 relleno a la izquierda con la cadena char2 de forma que la cadena resultante presente una longitud de N caracteres. Por defecto, char2 es un espacio en blanco. Esta función se suele emplear para alinear a la derecha cadenas.

```
SELECT NOMBRE, LPAD(NOMBRE, 10), LPAD(NOMBRE, 10, '-')  
FROM EMPLEADOS;
```

- **RPAD(char1, N[, char2])** es análoga a LPAD pero rellena por la derecha. Esta función se suele emplear para alinear a la izquierda cadenas.

```
SELECT NOMBRE, RPAD(NOMBRE, 10), RPAD(NOMBRE, 10, '-')  
FROM EMPLEADOS;
```

- **LTRIM(char1 [, char2])** Devuelve char1, quitando los caracteres de la izquierda contenidos en char2. Si no se especifica char2, por defecto toma el valor de un espacio en blanco. La base de datos analiza char1, comenzando desde la izquierda. Cuando se encuentra el primer carácter que no aparece char2, devuelve el resultado.

```
LTRIM(' Hola') → 'Hola'  
LTRIM('xxxHola', 'x') → 'Hola'  
LTRIM('xyxyxyHola', 'xy') → 'Hola'  
LTRIM('xyxyxxxxyHola', 'xy') → 'Hola'
```

- **RTRIM(char1 [, char2])** Devuelve char1, quitando los caracteres de la derecha contenidos en char2. Si no se especifica char2, por defecto toma el valor de un espacio en blanco. La base de datos analiza char1, comenzando por la derecha. Cuando se encuentra el primer carácter que no aparece char2, devuelve el resultado.

```
RTRIM('Hola ') → 'Hola'  
RTRIM('Holaxxx', 'x') → 'Hola'
```

```
RTRIM('HolaxxXXxx', 'x') → 'HolaxxXX'  
RTRIM('Hola a todos', 'todos') → 'Hola a '
```

4.4.4 FUNCIONES NUMÉRICAS

- **ROUND (N [, M])** Devuelve el número N redondeado a M posiciones decimales. Si se omite M no se redondea con lugares decimales. Si M es negativo, los números a la izquierda del punto decimal se redondean. N puede ser una columna numérica o una expresión numérica.

```
ROUND(386.126, 2) → 386.13  
ROUND(386.126, -2) → 400
```

```
SELECT (386.126, 2)  
FROM DUAL;
```

(Para el resto de ejemplos es igual.)

- **TRUNC (N [, M])** Trunca el número N en la M-ésima posición decimal. M por defecto es cero. M también puede ser negativo, en cuyo caso elimina la parte decimal y hace cero las M cifras a la izquierda del punto decimal.

```
TRUNC(386.126, 2) → 386.12  
TRUNC(386.126, -2) → 300
```

- **MOD (N , M)** Devuelve el resto de dividir N por M. Devuelve N si M es cero.

```
MOD(15, 2) → 1  
MOD(386.126, 300) → 86.126  
MOD(43, 0) → 43
```

4.4.5 FUNCIONES DE FECHAS

- **SYSDATE** es una función que devuelve la fecha y hora actual.

```
SELECT SYSDATE  
FROM DUAL;
```

- **fecha + N** devuelve fecha más N días.

```
SELECT SYSDATE, SYSDATE+2  
FROM DUAL;
```

- **fecha - N** devuelve fecha menos N días.

```
SELECT SYSDATE, SYSDATE-7  
FROM DUAL;
```

- **fecha1 – fecha2** devuelve el número de días entre ambas fechas.

```
SELECT SYSDATE-FECHA_ALTA,  
       TRUNC(SYSDATE-FECHA_ALTA) "Días en la empresa"  
FROM EMPLEADOS;
```

- **fecha + N/24** devuelve fecha más N horas.

```
SELECT SYSDATE, SYSDATE+2/24  
FROM DUAL;
```

- **MONTHS_BETWEEN (fecha1, fecha2)** Devuelve la cantidad de días entre fecha1 y fecha2. El resultado puede ser positivo o negativo. Si fecha1 es mayor, el resultado es positivo; en caso contrario, es negativo. La parte decimal representa la porción del mes.

```
SELECT SYSDATE+70, SYSDATE, C
FROM DUAL;
```

```
SELECT NOMBRE,
       TRUNC(MONTHS_BETWEEN(SYSDATE, FECHA_ALTA)) "Meses contratado"
FROM EMPLEADOS;
```

- **ADD_MONTHS (fecha, N)** Agrega N meses a fecha. N puede ser negativo.

```
SELECT SYSDATE, ADD_MONTHS(SYSDATE, 1) "Dentro de un mes",
       ADD_MONTHS(SYSDATE, -1) "Hace un mes"
FROM DUAL;
```

- **NEXT_DAY (fecha, 'carácter')** Devuelve la fecha del primer día de la semana indicado por carácter, posterior a la fecha. 'carácter' puede ser un día de la semana (depende del idioma) o el número de día de la semana.

```
SELECT SYSDATE,
       NEXT_DAY(SYSDATE, 'VIERNES'), NEXT_DAY(SYSDATE, 5)
FROM DUAL;
```

```
SELECT SYSDATE,
       NEXT_DAY(SYSDATE, 'FRIDAY'), NEXT_DAY(SYSDATE, 5)
FROM DUAL;
```

- **LAST_DAY (fecha)** Devuelve la fecha del último día del mes que contiene fecha.

```
SELECT SYSDATE, LAST_DAY(SYSDATE)
FROM DUAL;
```

- **ROUND (fecha [, 'fmt'])** Cuando no se especifica ningún formato fmt, devuelve la fecha del día más próximo. Si fmt= 'YEAR', devuelve el primer día del año. Si fmt = 'MONTH', redondea al primer día del mes que esté más próximo.

```
SELECT SYSDATE "Hoy",
       ROUND(SYSDATE, 'YEAR') "Primer día del año",
       ROUND(SYSDATE, 'MONTH') "Primer día del mes"
FROM DUAL;
```

- **TRUNC (fecha [, 'fmt'])** Devuelve la fecha truncada al primer día del mes si fmt = 'MONTH', al primer día del año si fmt = 'YEAR', etc. Si no se especifica el formato, la fecha se trunca en el día más próximo.

```
SELECT SYSDATE, SYSDATE+20,
       ROUND(SYSDATE, 'MONTH'), ROUND(SYSDATE+20, 'MONTH'),
       TRUNC(SYSDATE, 'MONTH'), TRUNC(SYSDATE+20, 'MONTH')
FROM DUAL;
```

4.4.6 FUNCIONES DE CONVERSIÓN

- **TO_CHAR(fecha, 'fmt')** convierte una fecha en una cadena de caracteres según el formato fmt.

```
SELECT TO_CHAR(SYSDATE, 'DD Month YYYY')
FROM DUAL;
```

```
SELECT TO_CHAR(SYSDATE, 'fmDD "de" Month "de" YYYY')
FROM DUAL;
```

```
SELECT TO_CHAR(SYSDATE, 'fmDD "de" fmMonth "de" YYYY')
FROM DUAL;
```

```
SELECT 'Hoy es '||TO_CHAR(SYSDATE, 'day')||', '||
      TO_CHAR(SYSDATE, 'fmDD')||' de '||
      TO_CHAR(SYSDATE, 'fmMonth "de" YYYY')
FROM DUAL;
```

```
SELECT RTRIM(TO_CHAR(SYSDATE, 'Month'))
      "Mes sin espacios a la derecha"
FROM DUAL;
```

```
SELECT TO_CHAR(SYSDATE, 'HH') "Hora",
      TO_CHAR(SYSDATE, 'MM') "Minuto",
      TO_CHAR(SYSDATE, 'SS') "Segundo"
FROM DUAL;
```

- **TO_DATE(char [, 'fmt'])** convierte la cadena char a una fecha según el formato fmt

```
SELECT TO_DATE('06/02/2001', 'DD/MM/YYYY') "Día / Mes / Año",
      TO_DATE('14:20:39', 'HH24:MI:SS') "Horas, Minutos, Segundos",
      TO_DATE('02:20:39 PM', 'HH:MI:SS PM') "Hs, Ms, Ss"
FROM DUAL;
```

```
SELECT TO_CHAR(TO_DATE('14:20:39', 'HH24:MI:SS'), 'HH:MI:SS AM')
FROM DUAL;
```

- **TO_CHAR(num, 'fmt')** convierte un número a una cadena de caracteres según el formato fmt.

fmt puede contener los siguientes elementos:

- 9 representa un número.
- 0 fuerza a que se muestre el cero.
- \$ signo del dólar.
- L usa el símbolo de moneda local.
- . (un punto) imprime el punto decimal.
- , (una coma) imprime el indicador de millar

```
SELECT NOMBRE, TO_CHAR(SUELDO, '999,999,999L')
FROM EMPLEADOS;
```

- **TO_NUMBER(char [, fmt])** convierte la cadena char en un número. El formato de la cadena char se puede indicar en fmt.

```
SELECT TO_NUMBER('843'), TO_NUMBER('843,57'),  
       TO_NUMBER('350,000', '999,999,999.00')  
FROM DUAL;
```

- **NVL(expr1, expr2)** Si el valor de expr1 es nulo, devuelve expr2. Si el valor de expr1 no es nulo devuelve expr1.

```
SELECT NOMBRE, NVL(TO_CHAR(JEFE_ID), 'Sin jefe')  
FROM EMPLEADOS;
```

```
SELECT NOMBRE, COMISION, NVL(COMISION, 0)  
FROM EMPLEADOS;
```

```
SELECT NOMBRE, SUELDO "Sueldo mensual",  
       NVL(COMISION, 0) "Comisión",  
       (12*SUELDO) + NVL(COMISION, 0) "Sueldo anual"  
FROM EMPLEADOS  
ORDER BY "Sueldo anual" DESC;
```

- **DECODE(expr, valor1, resul1,
 valor2, resul2,
 ...
 valorN, resulN,
 resulPorDefecto)**

Esta función equivale a evaluar una sentencia del tipo

```
IF expr = valor1 THEN  
    devuelve resul1;  
ELSIF expr = valor2 THEN  
    devuelve resul2;  
    ...  
ELSIF expr = valorN THEN  
    devuelve resulN;  
ELSE  
    devuelve resulPorDefecto  
END IF;
```

```
SELECT NOMBRE, CARGO, SUELDO,  
       'Aumento del ' ||  
       DECODE(CARGO, 'PRESIDENTE', '30%',  
               'DIRECTOR', '20%',  
               'COMERCIAL', '10%') "Revisión",  
       DECODE(CARGO, 'PRESIDENTE', 1.3,  
               'DIRECTOR', 1.2,  
               'COMERCIAL', 1.1)* SUELDO "Sueldo revisado"  
FROM EMPLEADOS;
```

Comentar con respecto a las conversiones, que Oracle en las asignaciones puede convertir implícitamente como sigue:

de VARCHAR2 o CHAR a NUMBER,

de VARCHAR2 o CHAR a DATE,
de NUMBER a VARCHAR2, y
de DATE a VARCHAR2.

4.4.7 Ejercicios

21. Mostrar la fecha en que comenzó a trabajar cada empleado de la forma:
Fecha de entrada (como título) 'Carlos Alonso García empezó a trabajar el 20 de febrero de 1991', ordenados por fecha entrada.
22. Incrementar el sueldo mensual de los comerciales así: Si su comisión es igual o inferior a 1000 €, aumentar un 25%; y si es superior a 1000 € aumentar un 12%.

4.5 SELECCIÓN DE DATOS DE VARIAS TABLAS

Cuando necesitamos consultar datos de más de una tabla empleamos una condición de JOIN.

```
SELECT tabla.columna, tabla.columna
FROM tabla1, tabla2
WHERE tabla1.columna = tabla2.columna;
```

```
SELECT EMPLEADOS.NOMBRE, DEPARTAMENTOS.LOCALIDAD
FROM EMPLEADOS, DEPARTAMENTOS
WHERE EMPLEADOS.DEPARTAMENTO_ID = DEPARTAMENTOS.ID;
```

Se establece un producto cartesiano si:

- Se omite una condición de join.
- Se define una condición de join inválida.
- Se combinan todas las filas de la primera tabla con todas las filas de la segunda.

Para evitar un producto cartesiano, se debe incluir siempre una condición de join válida en la cláusula WHERE.

```
SELECT EMPLEADOS.NOMBRE, DEPARTAMENTOS.LOCALIDAD
FROM EMPLEADOS, DEPARTAMENTOS;
```

Existen cuatro tipos de Join principales:

- Equijoin
- Non-equijoin
- Outer join
- Self join

4.5.1 EQUIJOIN

Los equijoin enlazan datos de más de una tabla a partir de igualdades (operador =) de valores de atributos de estas tablas. Normalmente se igualan los valores de una *Foreign Key* a los de la *Primary Key* a la que apuntan.

```
SELECT EMPLEADOS.NOMBRE "Empleado",
       DEPARTAMENTOS.NOMBRE "Departamento"
FROM EMPLEADOS, DEPARTAMENTOS
WHERE EMPLEADOS.DEPARTAMENTO_ID = DEPARTAMENTOS.ID;
```

En este caso recuperamos los nombres de los empleados, seguidos del nombre del departamento en el que trabajan (cláusula SELECT). En la cláusula FROM especificamos las tablas de las que vamos a extraer información. Por último, en la cláusula WHERE introducimos la condición de join. Al tratarse de un equijoin, no aparecerán los departamentos en los que no trabaja ningún empleado. Tampoco aparecerían los empleados que no trabajasen en algún departamento. (No es el caso debido a que el atributo DEPARTAMENTO_ID está definido como NOT NULL).

Cuando se trabaja con varias tablas, se recomienda utilizar alias de tablas y cualificar los atributos con el alias de la tabla a la que pertenecen.

```
SELECT EMP.NOMBRE "Empleado", DEP.NOMBRE "Departamento"
FROM EMPLEADOS EMP, DEPARTAMENTOS DEP
WHERE EMP.DEPARTAMENTO_ID = DEP.ID;
```

Además de las condiciones de join, también podemos añadir otras condiciones para filtrar.

```
SELECT EMP.NOMBRE "Empleado", DEP.NOMBRE "Departamento",
       DEP.LOCALIDAD "Localidad"
FROM EMPLEADOS EMP, DEPARTAMENTOS DEP
WHERE (EMP.DEPARTAMENTO_ID = DEP.ID)
AND (CARGO <> 'COMERCIAL');
```

También podemos combinar datos de más de dos tablas añadiendo nuevas condiciones de join. Siempre son necesarias al menos n-1 condiciones de join para combinar n tablas.

4.5.2 NON-EQUIJOIN

Los non-equijoin enlazan datos de más de una tabla sin utilizar el operador de igualdad (=).

```
SELECT EMP.NOMBRE, EMP.SUELDO, RAN.GRADO
FROM EMPLEADOS EMP, RANGOS_SUELDO RAN
WHERE EMP.SUELDO BETWEEN RAN.SUELDO_MIN AND RAN.SUELDO_MAX;
```

Obsérvese cómo ninguna columna de la tabla EMPLEADOS se corresponde directamente con una columna de la tabla RANGOS_SUELDO.

4.5.3 OUTER JOIN

Los outer join enlazan datos de más de una tabla que no cumplen la condición de join.

```
SELECT EMP.NOMBRE "Empleado", DEP.NOMBRE "Departamento"
FROM EMPLEADOS EMP, DEPARTAMENTOS DEP
WHERE EMP.DEPARTAMENTO_ID = DEP.ID;
```

No aparecen los departamentos en los que no trabajan empleados.

```
SELECT tabla.columna, tabla.columna
FROM tabla1, tabla2
WHERE tabla1.columna(+) = tabla2.columna;
```

```
SELECT tabla.columna, tabla.columna
FROM tabla1, tabla2
WHERE tabla1.columna = tabla2.columna(+);
```

- Debe utilizar outer join para ver las filas que no cumplen la condición de join.
- El operador de un outer join es un signo más encerrado entre paréntesis: (+)
- Este operador 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.
- El operador (+) debe colocarse detrás de la tabla sin filas coincidentes.

```
SELECT EMP.NOMBRE "Empleado", DEP.NOMBRE "Departamento"
FROM EMPLEADOS EMP, DEPARTAMENTOS DEP
WHERE EMP.DEPARTAMENTO_ID(+) = DEP.ID;
```

EL operador outer join "crea" empleados nulos para que aparezcan los departamentos en los que no trabajan empleados.

```
SELECT EMP.NOMBRE "Empleado", DEP.NOMBRE "Departamento"
FROM EMPLEADOS EMP, DEPARTAMENTOS DEP
WHERE EMP.DEPARTAMENTO_ID = DEP.ID(+);
```

En este caso no obtenemos el efecto deseado puesto que no existe ningún empleado que no tenga asignado un departamento. Recordar siempre que el operador (+) debe ir detrás de la tabla en la que nos interesa que se creen filas NULL.

4.5.4 SELF JOIN

Un self join es una combinación de una tabla consigo misma. Emplearemos alias diferentes para la misma tabla.

```
SELECT JEF.NOMBRE "Jefe", EMP.NOMBRE "Empleado"
FROM EMPLEADOS JEF, EMPLEADOS EMP
WHERE JEF.ID = EMP.JEFE_ID;
```

```
SELECT JEF.NOMBRE "Jefe", EMP.NOMBRE "Empleado"
FROM EMPLEADOS JEF, EMPLEADOS EMP
WHERE JEF.ID(+) = EMP.JEFE_ID;
```

4.5.5 Ejercicios

23. Mostrar la localidad, nombre del empleado y sueldo, de los empleados cuyo sueldo mensual es de grado 2 o 3.
24. Mostrar el nombre de los departamentos cuyos empleados tienen comisión o su sueldo anual es superior a tres millones de pesetas.

4.6 DATOS AGREGADOS, FUNCIONES DE GRUPO PREFABRICADAS

En determinadas circunstancias nos interesa obtener datos que resultan de operar sobre grupos de valores. Para ello utilizamos las funciones de grupo. Para

especificar los grupos de registros empleamos la cláusula GROUP BY, y en tercer lugar, para filtrar los grupos de registros introducimos la cláusula HAVING.

4.6.1 FUNCIONES DE GRUPO

- **AVG ([DISTINCT | ALL] n)** devuelve la media aritmética.
- **COUNT ({ * | [DISTINCT | ALL] expr})** devuelve el número de filas.
- **MAX ([DISTINCT | ALL] expr)** devuelve el valor máximo.
- **MIN ([DISTINCT | ALL] expr)** devuelve el valor mínimo.
- **STDDEV ([DISTINCT | ALL] n)** devuelve la desviación estándar.
- **SUM ([DISTINCT | ALL] n)** devuelve la suma.
- **VARIANCE ([DISTINCT | ALL] n)** devuelve la varianza.
 - DISTINCT hace que la función considere sólo los valores no duplicados.
 - ALL incluye todos los valores. Por defecto se toma ALL por lo que no es necesario especificarlo.
 - Los tipos de datos de los argumentos de expr pueden ser CHAR, VARCHAR2, NUMBER o DATE.
 - Todas las funciones de grupo excepto COUNT(*) ignoran los valores nulos. Utilice la función NVL si es necesario

```
SELECT AVG(SUELDO) "Sueldo medio"
FROM EMPLEADOS;
```

```
SELECT COUNT(*) "Nº de empleados"
FROM EMPLEADOS;
```

```
SELECT COUNT(DISTINCT CARGO) "Nº de cargos"
FROM EMPLEADOS;
```

```
SELECT COUNT(COMISION) "Nº de empleados con comisión"
FROM EMPLEADOS;
```

COUNT(columna) devuelve el número de filas que tienen un valor no nulo en columna.

```
SELECT MAX(COMISION) "Comisión mayor"
FROM EMPLEADOS;
```

```
SELECT MIN(FECHA_ALTA) "Primer contrato"
FROM EMPLEADOS;
```

```
SELECT SUM(SUELDO) "Sueldos mensuales"
FROM EMPLEADOS;
```

```
SELECT AVG(COMISION) "Comisión media"
FROM EMPLEADOS;
```

AVG(COMISION) ignora los valores nulos.

```
SELECT AVG(COMISION) "Empleados con comisión",
       AVG(NVL(COMISION, 0)) "Todos los empleados"
FROM EMPLEADOS;
```

4.6.2 CLÁUSULA GROUP BY

La cláusula group by permite crear grupos de registros.

```
SELECT columna, funcion_grupo
FROM tabla
[WHERE condición(es)]
[GROUP BY expresion_group_by]
[ORDER BY columna];
```

- expresion_group_by especifica columnas cuyos valores determinan si éstos satisfacen la condición para agrupar registros.
- Si se incluye una función de grupo en una cláusula SELECT, no se puede seleccionar resultados individuales a menos que la columna aparezca en la cláusula GROUP BY.
- Se pueden excluir filas antes de la división en grupos, utilizando la cláusula WHERE.
- No se pueden utilizar alias de columna dentro de la cláusula GROUP BY.
- Por defecto las filas se ordenan en forma ascendente de acuerdo a la lista GROUP BY. Este orden se puede alterar con la cláusula ORDER BY.

```
SELECT AVG(SUELDO) "Sueldo medio"
FROM EMPLEADOS
GROUP BY DEPARTAMENTO_ID;

SELECT AVG(SUELDO) "Sueldo medio"
FROM EMPLEADOS
WHERE CARGO <> 'PRESIDENTE'
GROUP BY DEPARTAMENTO_ID;
```

```
SELECT DEPARTAMENTO_ID, AVG(SUELDO) "Sueldo medio"
FROM EMPLEADOS
GROUP BY DEPARTAMENTO_ID;
```

```
SELECT DEP.NOMBRE "Departamento", AVG(EMP.SUELDO) "Sueldo medio"
FROM DEPARTAMENTOS DEP, EMPLEADOS EMP
WHERE DEP.ID = EMP.DEPARTAMENTO_ID
GROUP BY DEP.NOMBRE;
```

```
SELECT DEP.NOMBRE "Departamento", EMP.CARGO "Cargo",
       AVG(EMP.SUELDO) "Sueldo medio"
FROM DEPARTAMENTOS DEP, EMPLEADOS EMP
WHERE DEP.ID = EMP.DEPARTAMENTO_ID
GROUP BY DEP.NOMBRE, EMP.CARGO;
```

- Cualquier columna o expresión en la SELECT que no sea una función de grupo tiene que ser especificada en la cláusula GROUP BY.

```
SELECT DEPARTAMENTO_ID, COUNT(SUELDO)
FROM EMPLEADOS;

SELECT DEPARTAMENTO_ID, COUNT(SUELDO)
FROM EMPLEADOS
GROUP BY DEPARTAMENTO_ID;
```

4.6.3 CLÁUSULA HAVING

La cláusula HAVING permite filtrar grupos. Primero se agrupan los registros, después se aplican las funciones de grupo, y en tercer lugar, sólo se devuelven los que cumplen la cláusula HAVING.

Identificadores de departamento y sueldo máximo de aquellos departamentos cuyo sueldo máximo sea superior a 290,000 pts.

```
SELECT DEPARTAMENTO_ID, MAX(SUELDO)
FROM EMPLEADOS
GROUP BY DEPARTAMENTO_ID
HAVING MAX(SUELDO) > 290000;
```

Cargo y salario mensual total, para cada cargo con cuyo saldo mensual total sea superior a 400,000 pts. Se excluyen a los comerciales y el resultado se ordena por el salario mensual total.

```
SELECT CARGO, SUM(SUELDO) "Salario mensual total"
FROM EMPLEADOS
WHERE CARGO <> 'COMERCIAL'
GROUP BY CARGO
HAVING SUM(SUELDO) > 400000
ORDER BY SUM(SUELDO)
```

- No puede utilizar una cláusula WHERE para filtrar grupos. Utilice la cláusula HAVING.

```
SELECT DEPARTAMENTO_ID, MAX(SUELDO)
FROM EMPLEADOS
WHERE MAX(SUELDO) > 290000
GROUP BY DEPARTAMENTO_ID;

SELECT DEPARTAMENTO_ID, MAX(SUELDO)
FROM EMPLEADOS
GROUP BY DEPARTAMENTO_ID
HAVING MAX(SUELDO) > 290000;
```

4.6.4 Ejercicios

25. Calcular el número de empleados del departamento de VENTAS.
26. Calcular el salario medio de cada cargo de la empresa.
27. Calcular la comisión media de los empleados, excluyendo al presidente y suponiendo que todos los empleados al menos cobran una comisión de 0 €.
28. Calcular el sueldo máximo de los empleados de cada departamento siempre que el mínimo sueldo del departamento sea superior a 430,00 €.

4.7 SUBCONSULTAS

Una subconsulta es una consulta escrita dentro de una cláusula de otra consulta. Por ejemplo, "Los empleados que ganan más que Tomás (nótese la subconsulta que va entre paréntesis y que nos devuelve el sueldo de Tomás).

```
SELECT NOMBRE, SUELDO
FROM EMPLEADOS
WHERE SUELDO > (SELECT SUELDO
                FROM EMPLEADOS
                WHERE NOMBRE = 'TOMAS');
```

Existen tres tipos de subconsultas:

- Mono-registro, que devuelven un único registro.
- Multi-registro, que devuelven más de un registro.
- Multi-columna, que devuelven más de una columna.

Consejos:

- Encierre las subconsultas entre paréntesis.
- Una subconsulta debe aparecer a la derecha del operador

```
SELECT columna
FROM tabla
WHERE expr operador (subconsulta);
```

- En las subconsultas mono-registro se utilizan operadores a nivel de fila:
 - = Igual a
 - > Mayor que
 - >= Mayor que o igual a
 - < Menor que
 - <= Menor que o igual a
 - <> Distinto de
- En las subconsultas multi-registro se introducen los operadores ANY o ALL entre el operador de nivel de fila y la subconsulta:
 - = ANY Igual a alguno
 - > ANY Mayor que alguno (más que el mínimo)
 - < ANY Menor que alguno (menos que el máximo)
 - = ALL Igual que todos
 - ...
 - = ANY es lo mismo que IN
 - <> ALL es lo mismo que NOT IN

Empleados con el mismo cargo que 'Carlos' y con sueldo menor que el de 'Tomás'.

```
SELECT NOMBRE, CARGO, SUELDO
FROM EMPLEADOS
WHERE (CARGO = (SELECT CARGO
                FROM EMPLEADOS
                WHERE NOMBRE = 'Carlos'))
AND (SUELDO < (SELECT SUELDO
                FROM EMPLEADOS
                WHERE NOMBRE = 'Tomás'));
```

Comerciales cuyo salario mensual junto a su comisión sea superior a algún salario de los directores.

```
SELECT NOMBRE, SUELDO
FROM EMPLEADOS
WHERE (CARGO = 'COMERCIAL')
AND ((SUELDO + COMISION) > ANY (SELECT SUELDO
                                FROM EMPLEADOS
                                WHERE CARGO = 'DIRECTOR'));
```

Empleados cuyo salario es inferior al salario medio de todos los departamentos.

```
SELECT NOMBRE, SUELDO
FROM EMPLEADOS
WHERE (SUELDO < ALL (SELECT AVG(SUELDO)
                    FROM EMPLEADOS
                    GROUP BY DEPARTAMENTO_ID));
```

Determinar los empleados cuyo salario y comisión se correspondan (ambos) con el salario y la comisión de cualquier empleado del departamento de VENTAS.

```
SELECT NOMBRE, SUELDO, COMISION
FROM EMPLEADOS
WHERE (SUELDO, NVL(COMISION, 0)) IN
      (SELECT EMP.SUELDO, NVL(EMP.COMISION, 0)
       FROM EMPLEADOS EMP, DEPARTAMENTOS DEP
       WHERE (EMP.DEPARTAMENTO_ID = DEP.ID)
       AND   (DEP.NOMBRE = 'VENTAS'));
```

- **EXISTS** Este operador seguido de una subconsulta devuelve verdadero si la subconsulta retorna al menos una fila.

Seleccione los departamentos en los que trabajen empleados.

```
SELECT DEP.NOMBRE
FROM DEPARTAMENTOS DEP
WHERE EXISTS (SELECT *
             FROM EMPLEADOS
             WHERE DEPARTAMENTO_ID = DEP.ID);
```

4.7.1 Ejercicios

29. Mostrar el nombre y fecha de entrada de todos los empleados que trabajan en el mismo departamento que 'Esther'.

4.8 OPERADORES DE CONJUNTOS

Existen una serie de operadores de conjunto que devuelve o eliminan filas de dos o más consultas, cada una de las cuales retorna valores del mismo tipo en la misma posición.

- **UNION** devuelve todas las filas que contengan los operadores excluyendo las repetidas.
- **UNION ALL** devuelve todas las filas que contengan los operadores incluyendo las repetidas.
- **INTERSECT** devuelve las filas que coinciden en ambos operadores.
- **MINUS** devuelve las filas del primer operador que no estén incluidas por el segundo.

5 LENGUAJE DE MANIPULACIÓN DE DATOS (DML)

El lenguaje de manipulación de datos cubre las sentencias que insertan nuevos registros en la Base de Datos (INSERT), las sentencias que modifican valores de la BD (UPDATE), y las encargadas de borrar registros (DELETE).

5.1 SENTENCIA INSERT

Para añadir nuevos registros a una tabla empleamos la sentencia INSERT

```
INSERT INTO tabla [(columna [, columna ...])]  
VALUES (valor [, valor ...]);
```

Insertar el departamento de Desarrollo de Sevilla

```
INSERT INTO DEPARTAMENTOS (ID, NOMBRE, LOCALIDAD)  
VALUES (5, 'DESARROLLO', 'SEVILLA');
```

Insertar al empleado 'Luis Barrios Gómez' del departamento de Desarrollo a partir de hoy, con un sueldo de 2650 € y sin comisión.

```
INSERT INTO EMPLEADOS (ID, NOMBRE, APELLIDOS, FECHA_ALTA  
SUELDO, COMISION, DEPARTAMENTO_ID)  
VALUES (8, 'Luis', 'Barrios Gómez', SYSDATE  
2650, NULL, 5);
```

o bien: (sin indicar la comisión)

```
INSERT INTO EMPLEADOS (ID, NOMBRE, APELLIDOS, FECHA_ALTA  
SUELDO, DEPARTAMENTO_ID)  
VALUES (8, 'LUIS', 'BARRIOS GOMEZ', SYSDATE  
2650, 5);
```

Insertar al empleado 'Raul' del departamento de Desarrollo a partir del 1 de febrero de 2000, con un sueldo de 3500 € y sin comisión.

```
INSERT INTO EMPLEADOS (ID, NOMBRE, FECHA_ALTA  
SUELDO, DEPARTAMENTO_ID)  
VALUES (9, 'RAUL', TO_DATE('01/02/2000', 'DD/MM/YYYY')  
3500, 5);
```

También podemos insertar registros en una tabla a partir de una subconsulta:

```
INSERT INTO tabla [(columna [, columna ...])]  
subconsulta;
```

La subconsulta debe devolver filas tales que el tipo de dato y orden de los valores devueltos coincida con el tipo de dato y orden de las columnas de la tabla sobre la que vamos a insertar.

5.2 SENTENCIA UPDATE

Para modificar valores de uno o más registros de una tabla empleamos la sentencia UPDATE

```
UPDATE tabla  
SET columna = valor [, columna = valor]  
[WHERE condición(es)];
```

Actualizar todos los nombres y apellidos de empleados dejando la primera letra en mayúsculas y el resto en minúsculas.

```
UPDATE EMPLEADOS  
SET NOMBRE = INITCAP(LOWER(NOMBRE)),  
APELLIDOS = INITCAP(LOWER(APELLIDOS));
```

Aumentar la comisión de los comerciales en un 5%;

```
UPDATE EMPLEADOS
SET COMISION = COMISION*1.05
WHERE CARGO = 'COMERCIAL';
```

5.3 SENTENCIA DELETE

Para borrar registros de una tabla empleamos la sentencia DELETE

```
DELETE [FROM] tabla
[WHERE condición(es)];
```

Borra a los empleados del departamento de Desarrollo

```
DELETE EMPLEADOS
WHERE DEPARTAMENTO_ID IN (SELECT ID
                        FROM DEPARTAMENTOS
                        WHERE NOMBRE = 'DESARROLLO');
```

5.3.1 Ejercicios

30. Actualiza los nombres y apellidos de los empleados corrigiendo los acentos.

6 ANEXO. Soluciones a los ejercicios propuestos

1. Borrar el usuario CURSO_GATE si existe eliminando todos sus objetos.

```
DROP USER CURSO_GATE CASCADE;
```

2. Crear el usuario CURSO_GATE, garantizarle los roles CONNECT y RESOURCE, y conectarse como este usuario.

```
CREATE USER CURSO_GATE IDENTIFIED BY CURSO_GATE;
GRANT CONNECT, RESOURCE TO CURSO_GATE;
CONNECT CURSO_GATE/CURSO_GATE
```

3. Crear la tabla de Departamentos (DEPARTAMENTOS) con la siguiente estructura:

```
ID          NUMBER(10)
NOMBRE      VARCHAR2(50)
LOCALIDAD  VARCHAR2(50)
```

ID es la clave primaria, no nula y con índice

```
DROP TABLE DEPARTAMENTOS CASCADE CONSTRAINTS;
```

```
CREATE TABLE DEPARTAMENTOS (
  ID          NUMBER(10) NOT NULL,
  NOMBRE      VARCHAR2(50),
  LOCALIDAD  VARCHAR2(50),
  CONSTRAINT DEPARTAMENTOS_PK
    PRIMARY KEY (ID)
    USING INDEX);
```

La borramos primero, por si ya existiera.

4. Crear la tabla de Empleados (EMPLEADOS) con la siguiente estructura:

```
ID          NUMBER(10)
NOMBRE      VARCHAR2(30)
APELLIDOS   VARCHAR2(60)
CARGO       VARCHAR2(20)
```

```
JEFE_ID          NUMBER(10)
FECHA_ALTA      DATE
SUELDO          NUMBER(9,2)
COMISION        NUMBER(9,2)
DEPARTAMENTO_ID NUMBER(10)
```

ID es la clave primaria, no nula y con índice. DEPARTAMENTO_ID no nulo.

```
DROP TABLE EMPLEADOS CASCADE CONSTRAINTS ;
```

```
CREATE TABLE EMPLEADOS (
  ID          NUMBER(10) NOT NULL,
  NOMBRE      VARCHAR2(30),
  APELLIDOS   VARCHAR2(60),
  CARGO       VARCHAR2(20),
  JEFE_ID     NUMBER(10),
  FECHA_ALTA  DATE,
  SUELDO      NUMBER(9,2),
  COMISION    NUMBER(9,2),
  DEPARTAMENTO_ID NUMBER(10) NOT NULL,
  CONSTRAINT EMPLEADOS_PK
    PRIMARY KEY (ID)
    USING INDEX);
```

- 5. Crear una foreign key del atributo DEPARTAMENTO_ID de la tabla EMPLEADOS referenciando la clave primaria de la tabla DEPARTAMENTOS.**

```
ALTER TABLE EMPLEADOS
ADD CONSTRAINT EMP_DEPARTAMENTO_ID_FK
  FOREIGN KEY (DEPARTAMENTO_ID)
  REFERENCES DEPARTAMENTOS (ID);
```

- 6. Crear una foreign key del atributo JEFE_ID de la tabla EMPLEADOS referenciando la clave primaria de la misma tabla.**

```
ALTER TABLE EMPLEADOS
ADD CONSTRAINT EMP_JEFE_ID_FK
  FOREIGN KEY (JEFE_ID)
  REFERENCES EMPLEADOS (ID);
```

- 7. Crear índices sobre las claves externas creadas anteriormente.**

```
CREATE INDEX EMP_DEPARTAMENTO_ID_IDX
ON EMPLEADOS (DEPARTAMENTO_ID);
```

```
CREATE INDEX EMP_JEFE_ID_IDX
ON EMPLEADOS (JEFE_ID);
```

- 8. Crear la tabla de Rangos de sueldos (RANGOS_SUELDO) con la siguiente estructura:**

```
GRADO          NUMBER(2)
SUELDO_MIN     NUMBER(9,2)
SUELDO_MAX     NUMBER(9,2)
```

```
DROP TABLE RANGOS_SUELDO CASCADE CONSTRAINTS;
```

```
CREATE TABLE RANGOS_SUELDO (
  GRADO          NUMBER(2),
  SUELDO_MIN     NUMBER(9,2),
  SUELDO_MAX     NUMBER(9,2));
```

- 9. Insertar los datos de las tres con las sentencias INSERT del Anexo1.**

```
INSERT INTO DEPARTAMENTOS (ID, NOMBRE, LOCALIDAD )
VALUES (1, 'INFORMATICA', 'BILBAO');

INSERT INTO DEPARTAMENTOS (ID, NOMBRE, LOCALIDAD )
VALUES (2, 'INVESTIGACION', 'MADRID');

INSERT INTO DEPARTAMENTOS (ID, NOMBRE, LOCALIDAD )
VALUES (3, 'VENTAS', 'BARCELONA');

INSERT INTO DEPARTAMENTOS (ID, NOMBRE, LOCALIDAD )
VALUES (4, 'PRODUCCION', 'VALENCIA');
commit;

INSERT INTO EMPLEADOS (ID, NOMBRE, APELLIDOS, CARGO, JEFE_ID,
FECHA_ALTA, SUELDO, COMISION, DEPARTAMENTO_ID )
VALUES (6, 'INES', 'FERNANDEZ SANCHEZ', 'PRESIDENTE', NULL,
TO_Date( '17/11/1991', 'DD/MM/YYYY'), 500000, NULL, 1);

INSERT INTO EMPLEADOS (ID, NOMBRE, APELLIDOS, CARGO, JEFE_ID,
FECHA_ALTA, SUELDO, COMISION, DEPARTAMENTO_ID )
VALUES (4, 'Esther', 'PEÑA MATA', 'DIRECTOR', 6,
TO_Date( '01/05/1991', 'DD/MM/YYYY'), 285000, NULL, 3);

INSERT INTO EMPLEADOS (ID, NOMBRE, APELLIDOS, CARGO, JEFE_ID,
FECHA_ALTA, SUELDO, COMISION, DEPARTAMENTO_ID )
VALUES (5, 'DANIEL', 'LOPEZ LEON', 'DIRECTOR', 6,
TO_Date( '09/06/1991', 'DD/MM/YYYY'), 245000, NULL, 1);

INSERT INTO EMPLEADOS (ID, NOMBRE, APELLIDOS, CARGO, JEFE_ID,
FECHA_ALTA, SUELDO, COMISION, DEPARTAMENTO_ID )
VALUES (1, 'CARLOS', 'ALONSO GARCIA', 'COMERCIAL', 4,
TO_Date('20/02/1991', 'DD/MM/YYYY'), 160000, 30000, 3);

INSERT INTO EMPLEADOS (ID, NOMBRE, APELLIDOS, CARGO, JEFE_ID,
FECHA_ALTA, SUELDO, COMISION, DEPARTAMENTO_ID )
VALUES (2, 'ANDRES', 'GOMEZ SEGUNDO', 'COMERCIAL', 4,
TO_Date( '22/02/1991', 'DD/MM/YYYY'), 125000, 50000, 3);

INSERT INTO EMPLEADOS ( ID, NOMBRE, APELLIDOS, CARGO, JEFE_ID,
FECHA_ALTA, SUELDO, COMISION, DEPARTAMENTO_ID )
VALUES (3, 'ANA', 'HERRANZ CIFUENTES', 'COMERCIAL', 4,
TO_Date( '28/09/1991', 'DD/MM/YYYY'), 125000, 140000, 3);

INSERT INTO EMPLEADOS ( ID, NOMBRE, APELLIDOS, CARGO, JEFE_ID,
FECHA_ALTA, SUELDO, COMISION, DEPARTAMENTO_ID )
VALUES (7, 'TOMAS', 'CALLEJA ARCAZ', 'COMERCIAL', 4,
TO_Date( '08/09/1991', 'DD/MM/YYYY'), 150000, 0, 3);
commit;

INSERT INTO RANGOS_SUELDO ( GRADO, SUELDO_MIN, SUELDO_MAX )
VALUES (1, 70000, 120000);
INSERT INTO RANGOS_SUELDO ( GRADO, SUELDO_MIN, SUELDO_MAX )
VALUES (2, 120001, 140000);
INSERT INTO RANGOS_SUELDO ( GRADO, SUELDO_MIN, SUELDO_MAX )
VALUES (3, 140001, 200000);
INSERT INTO RANGOS_SUELDO ( GRADO, SUELDO_MIN, SUELDO_MAX )
VALUES (4, 200001, 300000);
INSERT INTO RANGOS_SUELDO ( GRADO, SUELDO_MIN, SUELDO_MAX )
```

```
VALUES (5, 300001, 999999);  
commit;
```

10. ¿Qué errores presenta la siguiente sentencia?

```
SELECT NOMBRE, APELLIDOS  
       SUELDO x 12 Salario anual  
FROM EMPLEADOS;
```

Falta la coma después de APELLIDOS, x debe ser *, y el alias Salario anual debe ir entre comillas dobles.

11. Mostrar la estructura de la tabla DEPARTAMENTOS

```
DESC EMPLEADOS;
```

12. Seleccionar a los empleados como sigue:

```
LOPEZ SEGUNDO, CARLOS: es PRESIDENTE
```

```
SELECT APELLIDOS||', '||NOMBRE||': es '||CARGO "Empleados"  
FROM EMPLEADOS;
```

13. Consultar los diferentes cargos desempeñados en la empresa.

```
SELECT DISTINCT CARGO  
FROM EMPLEADOS;
```

14. Mostrar el nombre, apellidos, sueldo y comisión; de los empleados que tienen jefe, y su comisión es superior a 2500 €.

```
SELECT NOMBRE, APELLIDOS, SUELDO, COMISION  
FROM EMPLEADOS  
WHERE (JEFE_ID IS NOT NULL)  
AND (COMISION IS NOT NULL)  
AND (COMISION > 2500);
```

o también:

```
SELECT NOMBRE, APELLIDOS, SUELDO, COMISION  
FROM EMPLEADOS  
WHERE (JEFE_ID IS NOT NULL)  
AND (COMISION > 2500);
```

15. Mostrar el nombre, sueldo y sueldo anual de los empleados cuyo sueldo anual supere los 36000 €.

```
SELECT NOMBRE, SUELDO, 12*SUELDO "SUELDO ANUAL"  
FROM EMPLEADOS  
WHERE (12*SUELDO) > 36000;
```

16. Mostrar la lista de los comerciales con su sueldo.

```
SELECT NOMBRE, CARGO, SUELDO  
FROM EMPLEADOS  
WHERE CARGO = 'COMERCIAL';
```

17. Mostrar a los empleados que no ejercen de comerciales cuyo nombre contiene la letra N.

```
SELECT NOMBRE, CARGO  
FROM EMPLEADOS  
WHERE (CARGO <> 'COMERCIAL')  
AND (NOMBRE LIKE '%N%');
```

- 18.** Consultar los empleados que obtienen comisión, ordenados por su sueldo de mayor a menor. Mostrar el nombre del empleado, su sueldo y su comisión.

```
SELECT NOMBRE, SUELDO, COMISION
FROM EMPLEADOS
WHERE COMISION IS NOT NULL
ORDER BY SUELDO DESC;
```

- 19.** Ordenar los empleados por sus apellidos y nombres.

```
SELECT APELLIDOS, NOMBRE
FROM EMPLEADOS
ORDER BY APELLIDOS, NOMBRE;
```

o también:

```
SELECT APELLIDOS, NOMBRE
FROM EMPLEADOS
ORDER BY APELLIDOS ASC, NOMBRE ASC;
```

- 20.** Obtener el sueldo anual de los empleados que no tienen comisión ordenados por fecha de alta en la empresa. Mostrar el nombre, sueldo anual y fecha de alta.

```
SELECT NOMBRE, 12*SUELDO "SUELDO ANUAL", FECHA_ALTA
FROM EMPLEADOS
WHERE COMISION IS NULL
ORDER BY FECHA_ALTA;
```

- 21.** Mostrar el sueldo de los empleados incrementado en un 10%, ordenados por nombre y apellidos.

```
SELECT NOMBRE, SUELDO, 1.1*SUELDO "SUELDO + 10%"
FROM EMPLEADOS
ORDER BY NOMBRE, APELLIDOS;
```

- 22.** Mostrar la fecha en que comenzó a trabajar cada empleado de la forma:

Fecha de entrada (como título)

'Carlos Alonso García empezó a trabajar el 20 de febrero de 1991'
ordenados por fecha entrada.

```
SELECT INITCAP(LOWER(NOMBRE))||' '||INITCAP(LOWER(APELLIDOS))||
' empezó a trabajar el '||TO_CHAR(FECHA_ALTA, 'fmDD "de "')||
TO_CHAR(FECHA_ALTA, 'fmmonth "de" YYYY') "Fecha de entrada"
FROM EMPLEADOS
ORDER BY FECHA_ALTA;
```

- 23.** Incrementar el sueldo mensual de los comerciales como sigue:

Si su comisión es igual o inferior a 1000 €, aumentarlo en un 25%; y si es superior a 1000 €, aumentarlo en un 12%.

```
SELECT NOMBRE, COMISION, TRUNC(COMISION/1000), SUELDO,
DECODE(TRUNC(COMISION/1000), 0, 1.25, 1, 1.12),
DECODE(TRUNC(COMISION/1000), 0, 1.25, 1, 1.12)*SUELDO
FROM EMPLEADOS
WHERE COMISION IS NOT NULL;
```

- 24.** Mostrar la localidad, nombre del empleado y sueldo, de los empleados cuyo sueldo mensual es de grado 2 o 3.

```
SELECT DEP.LOCALIDAD "Localidad", EMP.NOMBRE "Empleado",
EMP.SUELDO "Sueldo", RAN.GRADO "Grado"
FROM DEPARTAMENTOS DEP, EMPLEADOS EMP, RANGOS_SUELDO RAN
```

```
WHERE (DEP.ID = EMP.DEPARTAMENTO_ID)
AND (RAN.GRADO IN (2, 3))
AND (EMP.SUELDO BETWEEN RAN.SUELDO_MIN AND RAN.SUELDO_MAX);
```

- 25.** Mostrar el nombre de los departamentos cuyos empleados tienen comisión o su sueldo anual es superior a 36000 €.

```
SELECT DEP.NOMBRE "Departamento", EMP.NOMBRE "Empleado",
EMP.SUELDO*12 "Sueldo anual", EMP.COMISION "Comisión"
FROM DEPARTAMENTOS DEP, EMPLEADOS EMP
WHERE (DEP.ID = EMP.DEPARTAMENTO_ID)
AND ( (EMP.COMISION IS NOT NULL)
OR (EMP.SUELDO*12 > 36000));
```

- 26.** Calcular el número de empleados del departamento de VENTAS.

```
SELECT COUNT(*)
FROM EMPLEADOS EMP, DEPARTAMENTOS DEP
WHERE (EMP.DEPARTAMENTO_ID = DEP.ID)
AND (DEP.NOMBRE = 'VENTAS')
```

- 27.** Calcular el salario medio de cada cargo de la empresa.

```
SELECT CARGO, AVG(SUELDO)
FROM EMPLEADOS
GROUP BY CARGO;
```

- 28.** Calcular la comisión media de los empleados, excluyendo al presidente y suponiendo que todos los empleados al menos cobran una comisión de 0 €.

```
SELECT AVG(NVL(COMISION, 0))
FROM EMPLEADOS
WHERE CARGO <> 'PRESIDENTE';
```

- 29.** Calcular el sueldo máximo de los empleados de cada departamento siempre que el mínimo sueldo del departamento sea superior a 430 €.

```
SELECT DEP.NOMBRE "DEPARTAMENTO", MAX(EMP.SUELDO)
FROM DEPARTAMENTOS DEP, EMPLEADOS EMP
WHERE DEP.ID = EMP.DEPARTAMENTO_ID
GROUP BY DEP.NOMBRE
HAVING MIN(SUELDO) > 430;
```

- 30.** Mostrar el nombre y fecha de entrada de todos los empleados que trabajan en el mismo departamento que 'Esther'.

```
SELECT NOMBRE, FECHA_ALTA
FROM EMPLEADOS
WHERE DEPARTAMENTO_ID IN (SELECT DEPARTAMENTO_ID
FROM EMPLEADOS
WHERE NOMBRE = 'Esther');
```

- 31.** Actualizar los nombres y apellidos de los empleados corrigiendo los acentos.

```
UPDATE EMPLEADOS
SET NOMBRE = 'Inés', APELLIDOS = 'González Sánchez'
WHERE ID = 6;
```

7 Introducción a Consultas SQL anidadas

SQL es el lenguaje estándar para el acceso y manipulación de los datos de bases de datos relacionales, objeto-relacionales y XML-objeto-relacionales.

Una de las características más potentes de SQL son las consultas anidadas. Una *consulta anidada* (*nested query*) es aquella que tiene otra consulta embebida en ella. La consulta embebida puede aparecer en la cláusula FROM, en cuyo caso se llama una *tabla derivada* (*derived table*), o en la cláusula WHERE o en la cláusula HAVING, en cuyo caso se llama una *subconsulta* (*subquery*). La consulta que contiene una subconsulta se llama una *consulta externa* (*outer query*). Teóricamente, una consulta puede tener un número arbitrario de subconsultas anidadas en ella. Las subconsultas pueden ser agregadas o no-agregadas.

Una *subconsulta agregada* (*aggregate subquery*) contiene una función agregada (MIN, MAX, COUNT, SUM, y AVG) en su cláusula SELECT; y devuelve siempre como resultado un solo valor (puede que nulo). Una *subconsulta no-agregada* puede estar enlazada en la cláusula WHERE a su *consulta externa*, mediante uno de los siguientes operadores: EXISTS, NOT EXISTS, IN, NOT IN, θ ANY (o θ SOME, SOME es un sinónimo de ANY), y θ ALL, donde $\theta \in \{<, \leq, >, \geq, =, \neq\}$; el resultado es un conjunto de valores (puede que vacío). Una *subconsulta* puede ser *independiente* de la consulta externa o estar *correlacionada* a la consulta externa.

Una *subconsulta correlacionada* (*correlated subquery*) contiene un predicado (llamado *correlated predicate*) que referencia la relación en la consulta externa. Por tanto, el resultado de la subconsulta depende de cada tupla de la consulta externa. La consulta que contiene una *subconsulta correlacionada* se llama *consulta correlacionada* (*correlated query*).

Se ha estudiado profusamente tanto el modelo relacional anidado como el álgebra relacional anidada [Makinouchi 1977; Jaeschke y Schek 1982; Abiteboul y Bidoit 1984; Thomas y Fischer 1986; Schek y Scholl 1986; Ozsoyoglu et al. 1987; Van Gucht 1987; Gyssens y Van Gucht 1988, 1989; Roth et al. 1988; Colby 1989; Paredaens et al. 1989; Vossen 1991; Levene y Loizou 1993, 1994]. Como una extensión del tradicional modelo relacional [Codd 1970], el modelo relacional anidado permite que haya atributos en una relación anidada con valores no-atómicos. Adicionalmente, para incrementar la potencia de las bases de datos relacionales, tales extensiones pueden soportar los requisitos introducidos por las aplicaciones que involucran objetos complejos. Pero, tales extensiones también complican más la definición del álgebra relacional anidada. Por ejemplo, en las relaciones anidadas el operador de *join* tiene *seis casos* posibles ya que los atributos involucrados en una operación de *join* pueden estar en diferentes subrelaciones y en diferentes niveles de anidamiento [Garani and Johnson 2000]. Además, la optimización del operador relacional anidado no es tan sencilla como la optimización de los operadores relacionales. Un ejemplo típico es la característica básica de la propiedad conmutativa entre los operadores de selección y proyección algebraicos que no siempre se va a satisfacer por los operadores relacionales anidados.

La estrategia de evaluación tradicional de las *consultas correlacionadas* es la *iteración anidada* [Selinger et al. 1979]. Esto es, por cada tupla en la consulta externa, se procesa la subconsulta cada vez. Y, puesto que la iteración anidada puede ser muy costosa, se ha propuesto la consulta no-anidada, es decir, se propone la reescritura de consultas de forma plana [Kim 1982; Ganski y Wong 1987; Dayal 1987; Muralikrishna 1989, 1992; Seshadri et al. 1996b].

Casi todos los enfoques propuestos se centran en las subconsultas agregadas; de manera que las subconsultas no-agregadas se han tenido que transformar en subconsultas agregadas antes de la evaluación (e.g., Ganski y Wong [1987], Galindo-Legaria y Joshi [2001], y Akinde y Bohlen [2003]). Existen pocos enfoques propuestos solamente para subconsultas no-agregadas (e.g., Dayal [1987], Baekgaard y Mark [1995], y Badia [2003b]); pero estos enfoques tienen aún limitaciones, especialmente en las consultas con múltiples subconsultas y valores nulos. Más comúnmente, las consultas con subconsultas no-agregadas no pueden ser des-anidadas directamente; por eso, se requiere una transformación SQL a SQL. Sin embargo, la transformación de subconsultas ALL o NOT IN es más complicada debido a los valores nulos. Por tanto, no es posible aplicar un enfoque unificado a todos los tipos de subconsultas, y cada subconsulta es evaluada por su propia estrategia. Así, el rendimiento al procesar diferentes subconsultas puede diferir grandemente. Finalmente, la mayoría de los enfoques producen re-escrituras bastante complejas, y no se consideran las consultas con múltiples subconsultas.

Otros trabajos (Cao and Badia [2007]) se centran en consultas con subconsultas no-agregadas que aparecen en la cláusula WHERE, ya que la optimización de las subconsultas agregadas está bien establecida y la optimización de subconsultas no-agregadas aún tiene algunas limitaciones. Y se propone un enfoque uniforme y eficiente, el enfoque *relacional anidado* (*nested relational approach*), para

evaluar consultas con subconsultas no-agregadas. Conceptualmente, este enfoque consta de dos pasos: primero, el des-anidamiento de la consulta desde arriba hacia abajo; Segundo, el procesamiento de subconsulta de abajo hacia arriba. El primer paso es similar al des-anidamiento de subconsultas agregadas, por lo que existen técnicas (e.g., Dayal [1987]) que se pueden aplicar. El segundo paso necesita obtener el resultado de la subconsulta, que es un conjunto de valores (puede que vacío), así como comparar un atributo de valor atómico (*atómico-valuado*) en la consulta externa con el resultado de la subconsulta. Estas operaciones no existen en el álgebra relacional ni están aún soportadas. Puesto que el resultado de una subconsulta no-agregada es un conjunto de valores, que pueden ser modelados como un atributo *conjunto-valuado* (*set-valued*) en el modelo relacional anidado, se propone la utilización del *álgebra relacional anidado* para el procesamiento de consultas con subconsultas no-agregadas.

REFERENCIAS sobre Consultas Anidadas

- ABITEBOUL, S. AND BIDOIT, N. 1984. Non first normal form relations to represent hierarchically organized data. *Proc. PODS*, 191–200.
- AKINDE, M. AND BOHLEN, M. 2003. Efficient computation of subqueries in complex OLAP. *Proc. IEEE Int. Conf. on Data Engineering*. IEEE Computer Society Press, Los Alamitos CA, 163–174.
- BADIA, A. 2003b. Computing SQL subqueries with Boolean aggregates. In *Proceedings of the DAWAK Conference*. 391–400.
- BAEKGAARD, L. AND MARK, L. 1995. Incremental computation of nested relational query expressions. *ACM Trans. Datab. Syst.* 20, 2, 111–148.
- Cao, B. AND BADIA, A. 2007. SQL Query Optimization through Nested Relational Algebra. *ACM Trans. Database Systems*, 32(3), Article 18, 1-46.
- CODD, E. F. 1970. A relational model of data for large shared data banks. *Commun. ACM* 13, 6, 377–387.
- COLBY, L. S. 1989. A recursive algebra and query optimization for nested relations. *Proc. SIGMOD Conf. ACM*, 273–283.
- DAYAL, U. 1987. Of nests and trees: A unified approach to processing queries that contain nested subqueries, aggregates, and quantifiers. In *Proceedings of the Conference on Very Large Data Bases*. 197–208.
- GALINDO-LEGARIA, C. A. AND JOSHI, M. M. 2001. Orthogonal optimization of subqueries and aggregation. In *Proceedings of the ACM SIGMOD Conference*. ACM, New York, 571–581.
- GANSKI, R. A. AND WONG, H. K. T. 1987. Optimization of nested SQL queries revisited. In *Proceedings of the ACM SIGMOD Conference*. ACM, New York, 23–33.
- GARANI, G. AND JOHNSON, R. 2000. Joining nested relations and subrelations. *Inf. Syst.* 25, 4, 287–307.
- JAESCHKE, G. AND SCHEK, H. J. 1982. Remarks on the algebra of non first normal form relations. In *Proceedings of the PODS Conference*. ACM, New York, 124–138.
- KIM, W. 1982. On optimizing an SQL-like nested query. *ACM Trans. Datab. Syst.* 7, 3, 443–469.
- LEVENE, M. AND LOIZOU, G. 1993. Semantics for null extended nested relations. *ACM Trans. Datab. Syst.* 18, 3, 414–459.
- LEVENE, M. AND LOIZOU, G. 1994. The nested universal relation data model. *J. Comput. Syst. Sci.* 49, 3, 683–717.
- MAKINOCHI, A. 1977. A consideration on normal form of not-necessarily-normalized relation in the relational data model. In *Proceedings of the Conference on Very Large Data Bases*. 447–453.
- MURALIKRISHNA, M. 1992. Improved unnesting algorithms for join aggregate SQL queries. In *Proceedings of the Conference on Very Large Data Bases*. 91–102.
- OZSOYOGLU, G., OZSOYOGLU, Z. M., AND MATOS, V. 1987. Extending relational algebra and relational calculus with set-valued attributes and aggregate functions. *ACM Trans. Datab. Syst.* 12, 4, 566–592.
- PAREDAENS, J., DE BRA, P., GYSSENS, M., AND VAN GUCHT, D. 1989. *The Structure of the Relational Model*. Springer-Verlag, New York.
- ROTH, M. A., KORTH, H. F., AND SILBERSCHATZ, A. 1988. Extended relational algebra and calculus for nested relational databases. *ACM Trans. Datab. Syst.* 13, 4, 389–417.
- SCHEK, H. J. AND SCHOLL, M. H. 1986. The relational model with relation-valued attributes. *Inf. Syst.* 11, 2, 137–147.
- SELINGER, P. G., ASTRAHAN, M. M., CHAMBERLIN, D. D., LORIE, R. A., AND PRICE, T. G. 1979. Access path selection in a relational database management system. In *Proceedings of the ACM SIGMOD Conference*. ACM, New York, 23–34.
- SESHADRI, P., HELLERSTEIN, J. M., PIRAHESH, H., LEUNG, T. Y. C., RAMAKRISHNAN, R., SRIVASTAVA, R., STUCKEY, P. J., AND SUDARSHAN, S. 1996a. Cost-based optimization for magic: Algebra and implementation. In *Proceedings of the ACM SIGMOD Conference*. ACM, New York, 435–446.
- THOMAS, S. J. AND FISCHER, P. C. 1986. Nested relational structures. *Adv. Comput. Res.* 3, 269–307. TRANSACTION PROCESSING PERFORMANCE COUNCIL. The TPC-H benchmark. <http://www.tpc.org/tpch>.
- VAN GUCHT, D. 1987. On the expressive power of the extended relational algebra for the unnormalized relational model. In *Proceedings of the PODS Conference*. 302–312.
- VAN GUCHT, D., AND FISCHER, P. C. 1986. Some classes of multilevel relational structures. *Proc. of the PODS Conference*. 60–69.
- VOSSEN, G. 1991. Data Models, Database Language and Database Management Systems. Addison-Wesley, Reading, MA.