Introdución al SQL

Este curso pretende ser una introducción al SQL, y en particular a la versión que utiliza el SGBD Oracle (7.3) llamado SQL*Plus. No se pretende realizar un estudio exhaustivo de todas las opciones, comandos y aspectos de almacenamiento y administración que se pueden considerar en SQL. Sólo se ha pretendido introducir y explicar los comandos más utilizados con sus opciones más útiles, dejando los detalles más específicos a los manuales de referencia.

Las razones que me han movido a construirlo en HTML y facilitar su acceso mediante Web son las posibilidades de interacción con el texto y de mayor difusión. Por ahora las tablas de ejemplo son estáticas y se encuentran en el texto mismo, pero mi intención es que se pueda interactuar con una base de datos en tiempo real. ¿Quizás más adelante?.

Si tienes cualquier sugerencia o encuentras una errata escondida dímelo.

Abril de 1998.

Jesús Vegas
Dpto. Informática
Universidad de Valladolid
jvegas@infor.uva.es

Índice

---

1 Terminología

SQL
: *Structured Query Language* o Lenguaje de Consultas Estructurado. Es el lenguaje que permite la comunicación con el Sistema Gestor de Bases de Datos (Oracle en nuestro caso).

El SQL es un lenguaje unificado
: Lo utilizan todo tipo de usuarios, desde el administrador de la base de datos, DBA, hasta el usuario final.

El SQL es un lenguaje no procedimental.
: El usuario especifica *Qué* quiere, no *Cómo* ni *Dónde* conseguirlo.

El SQL es relacionalmente completo.
: Permite la realización de cualquier consulta de datos.

SQL= DDL + DML
: Las sentencias del SQL se clasifican como parte del DDL o del DML.

Lenguaje de Definición de Datos, DDL
: sentencias del SQL que permiten definir los objetos de la Base de Datos (`create`, `revoke`, `grant`, `alter`, etc.). Cuando se definen dichos objetos se almacenan en el diccionario de datos.

Lenguaje de Manipulación de Datos, DML
: sentencias del SQL que se utilizan para manejar los datos de la base de datos (`select`, `insert`, `update`, `delete`, etc).

`commit`/`rollback`
: cada vez que se realiza alguna operación en la base de datos se realiza no sobre la tabla en sí, sino sobre una copia local de la misma. Así, si queremos que los resultados de la modificación se trasladen a la base de datos y perduren en el tiempo hay que confirmar dicha operación con el comando `commit`. También se puede impedir que los últimos cambios lleguen a efectuarse con `rollback`, aunque existen algunas sentencias SQL que se 'autoconfirman' y no se pueden volver atrás.

Diccionario de la Base de Datos

Guarda la definición de todos los objetos almacenados en la base de datos; sus características, restricciones, privilegios, relaciones entre ellos, etc.

2 Tipos de Sentencias

Las sentencias SQL pertenecen a dos categorías principales: Lenguaje de Definición de Datos, DDL y Lenguaje de Manipulación de Datos, DML. Estos dos lenguajes no son lenguajes en sí mismos, sino que es una forma de clasificar las sentencias de lenguaje SQL en función de su cometido. La diferencia principal reside en que el DDL crea objetos en la base de datos y sus efectos se pueden ver en el diccionario de la base de datos; mientras que el DML es el que permite consultar, insertar, modificar y eliminar la información almacenada en los objetos de la base de datos.

Cuando se ejecutan las sentencias DDL de SQL, el SGBD confirma la transacción actual antes y después de cada una de las sentencias DDL. En cambio, las sentencias DML no llevan implícito el `commit` y se pueden deshacer. Existe pues un problema al mezclar sentencias DML con DDL, ya que estas últimas pueden confirmar las primeras de manera involuntaria e implicita, lo que en ocasiones puede ser un problema.

A continuación se presenta una tabla con las sentencias SQL más comunes, clasificadas según el lenguaje al que pertenecen.

Sentencia DDL	***Objetivo***
Alter procedure	Recompilar un procedimiento almacenado.
Alter Table	Añadir o redefinir una columna, modificar la asignación de almacenamiento.
Analyze	Recoger estadísticas de rendimiento sobre los objetos de la BD para utilizarlas en el optimizador basado en costes.
Create Table	Crear una tabla.
Create Index	Crear un índice.
Drop Table	Eliminar una tabla.
Drop Index	Eliminar un índice.
Grant	Conceder privilegios o papeles, roles, a un usuario o a otro rol.
Truncate	Eliminar todas las filas de una tabla.
Revoke	Retirar los privilegios de un usuario o rol de la base de datos.
Sentencia DML	***Objetivo***
Insert	Añadir filas de datos a una tabla.
Delete	Eliminar filas de datos de una tabla.
Update	Modificar los datos de una tabla.
Select	Recuperar datos de una tabla.
Commit	Confirmar como permamentes las modificaciones realizadas.
Rollback	Deshacer todas las modificaciones realizadas desde la última confirmación.

3 SQL*Plus

La herramienta que nos proporciona ORACLE para interactuar con la base de datos se llama SQL*Plus. Básicamente, es un intérprete SQL con algunas opciones de edición y formateo de resultados.

Antes de ver la manera de conectarse a SQL*Plus, conviene tener claros algunos conceptos:

Usuario/Clave
: Para poder acceder a una base de datos gestionada por ORACLE debemos ser un usuario autorizado de la misma y conocer la palabra clave, *password*, asociada al usuario.

Variable de ambiente `ORACLE_SID`
: Indica la base de datos con la que vamos a trabajar.

3.1 Conexión

Para entar en SQL*Plus se debe ejecutar el comando

```
$ sqlplus usuario/passwd
```

donde le indicamos al SGBD Oracle quién somos y nuestra palabra clave.

Si la configuración del SGBD Oracle se corresponde a una configuración cliente-servidor asentada sobre una red (SQL*Net v2) deberemos indicar, además, el sevicio (o base de datos) con el que queremos contactar. Esto se hace colocando el símbolo @ antes del nombre del servicio como se indica a continuación:

```
$ sqlplus usuario/passwd@servicio
```

Otra circunstancia que hay que tener en cuenta a la hora de conectarnos a SQL*Plus es el modo establecido por el DBA para la autentificación del usuario de la base de datos. La primera posibilidad es que recaiga sobre el SGBD Oracle la autentificación de los usuarios, por lo que tendremos que darle nuestro nombre de usuario y la palabra de paso. Pero existe la posibilidad de que el SGBD Oracle deje en manos del Sistema Operativo esta responsabilidad. Así, no será necesario demostrarle al SGBD Oracle quién somos ya que el SO se ha encargado previamente de comprobar que todo es correcto. En este segundo caso, el comando de conexión con SQL*Plus debe omitir el nombre de usuario y la palabra clave, pero manteniendo el resto de esta manera:

```
$ sqlplus /@servicio
```

Una vez que hemos conseguido entrar en SQL*Plus nos presenta el *prompt* y espera la insercción de sentencias SQL. Todas las sentencias deben acabar con un ';'. Una sentencia puede continuar en varias líneas, que SQL*Plus va numerando. Si queremos anular la sentencia actual podemos hacerlo colocando un '.' como único carácter en una línea. Si queremos volver a ejecutar la última sentencia se puede hacer con el comando '/'. Si queremos ejecutar las sentencias que almacena un fichero .sql podemos hacerlo anteponiendo el símbolo '@' al nombre del fichero.

Para cerrar la sesión vale con teclear 'exit'.

3.2 Posibilidades de Edición

SQL*Plus almacena en un *buffer* la última sentencia SQL introducida. El *buffer* mantiene sólo una sentencia cada vez, y si se introduce una nueva sentencia se sobreescribe sobre la anterior.

La sentencia en el *buffer* puede ser recuperada para ejecutarla de nuevo con los comandos:

- `RUN` que visualiza la sentencia en el *buffer* antes de ejecutarla;
- `/` que ejecuta la sentencia sin visualizarla.

SQL*Plus también nos permite editar la sentencia SQL alamacenada en el *buffer* mediante un sencillo (y limitado) editor en línea, cuyos comandos se enumeran a continuación:

Comando	***Abreviatura***	***Descripción***
APPEND texto	A texto	Añade texto al final de la línea.
CHANGE/fuente/destino	C/fuente/destino	Cambia el contenido 'fuente' por el 'destino'
CHANGE/texto	C/texto	Quita 'texto' de una línea.
CLEAR BUFFER	CL BUFF	Borra el *buffer*

DEL	DEL	Borra una línea.
INPUT	I	Inserta una o más líneas.
INPUT texto	I texto	Inserta una línea con 'texto'.
LIST	L	Lista las líneas del *buffer*
LIST n	L n ó n	Lista la línea n-ésima.
LIST *	L *	Lista la línea actual.
LIST LAST	L LAST	Lista la última línea.
LIST m n	L m n	Lista las líneas desde la m-ésima a la n-ésima.

Al contenido del *buffer* también se puede acceder desde el editor del Sistema Operativo. Así, el buffer podrá ser manipulado con las posibilidades del editor con el que estemos acostumbrados a trabajar. Al salir del editor se devuelve el control al SQL*Plus. Para conseguir trabajar con el editor del Sistema Operativo basta con colocar la variable `DEFINE_EDITOR` y luego llamar al editor.

```
SQL> define_editor=vi
SQL> edit
```

3.3 Utilización de Ficheros

SQL*Plus considera dos tipos de ficheros: de *spool* y de comandos.

Un **fichero de *spool*** almacena los resultados de una consulta (o varias) en un fichero con la extensión .lst (o lo manda a la impresora).

Los comandos asociados con los ficheros *spool* son

`SPOOL fichero`
: Manda el resultado de las consultas al fichero.

`SPOOL OUT`
: Manda el resultado de las consultas a la impresora.

`SPOOL OFF`
: Cierra el fichero de *spool*.

`EXIT`
: Al salir de SQL*Plus se cierran los ficheros de *spool*.

Los **archivos de comandos** almacenan comandos SQL y SQL*Plus para ser editado, almacenado y/o ejecutado; y tienen por defecto la extensión .sql :

- Para editarlo se puede utilizar el comando `edit fichero`.
- Para ejecutarlo se utilizará el comando `START fichero` o `@fichero`

El SQL*Plus nos proporciona más posibilidades en relación con los ficheros de comandos, la comunicación con el usuario final y la generación de informes. Pero antes de ver este tipo de cosas, es mejor que sigamos profundizando en el conocimiento del lenguaje SQL. Al final del curso se puede se encuentra un capítulo con algunas de las cosillas que quedan por contar del SQL*Plus.

---

4 Creación

La primera fase de cualquier base de datos comienza siempre con sentencias DDL, ya que antes de poder almacenar información debemos definir los objetos básicos donde agrupar la información. Los objetos básicos con que trabaja SQL son las tablas. Una tabla es un conjunto de celdas agrupadas en filas y columnas donde se almacenan elementos de información.

Antes de llevar a cabo la creación de una tabla conviene planificar:

- nombre de la tabla,
- nombre de cada columna,
- tipo y tamaño de los datos almacenados en cada columna,
- información adicional, restricciones, etc.

Hay que tener en cuenta también ciertas restricciones en la formación de los nombres de las tablas: longitud máxima de 30 caracteres, no puede haber nombres de tabla duplicados, deben comenzar con un carácter alfabético, permitir caracteres alfanuméricos y el guión bajo '_', y Oracle no distingue entre mayúsculas y minúsculas.

La sintaxis del comando que permite crear un tabla es la siguiente:

```
CREATE TABLE [esquema.]tabla ({columna tipoColumna [NOT NULL],}+,
    {CONSTRAINT nombreRestricción
      {UNIQUE ([column,]+)|
       DEFAULT expresion|
       CHECK (condicion)|
       PRIMARY KEY ([column,]+)|
       FOREIGN KEY (column) REFERENCES tabla(columna)},}*)
```

Del examen de la sintaxis de la sentencia `Create Table` se pueden concluir que necesitamos conocer los distintos tipos de columna y las distintas restricciones que se pueden imponer al contenido de las columnas. Vayamos por partes.

4.1 Tipos de Columnas

Existen varios tipos de datos en SQL. De esta manera, cada columna puede albergar una información de naturaleza distinta. Los tipos de datos más comunes y sus características se resumen en la siguiente tabla.

Tipo de Dato	***Descripción***
`VARCHAR2 (tamaño)`	Almacena datos de tipo carácter alfanumérico de longitud variable, con un tamaño máximo de 2.000.
`CHAR(tamaño)`	Almacena datos de tipo carácter alfanumérico de longitud fija, con un tamaño máximo de 255.
`LONG`	Almacena datos de tipo carácter alfanumérico de longitud variable con un tamaño máximo de hasta 2 Gb.
`NUMBER (dig,dec)`	Almacena datos numéricos de `dig` dígitos, de los cuales `dec` son decimales. El tamaño máximo es de 38 dígitos.
`DATE`	Almacena fechas desde el 1-Ene-4712 AC hasta el 31-Dic-4712 DC.

`RAW(tamaño)`	Almacena datos de longitud variable, con un tamaño máximo de 255 bytes.
`LONG RAW`	Almacena datos de longitud variable, con un tamaño máximo de 2 Gb.

4.2 Restricciones

Las restricciones de los datos se imponen para asegurarnos que los datos cumplen con una serie de condiciones predefinidas para cada tabla. Estas restricciones ayudan a conseguir la integridad de referencia: todas las referencias dentro de una BD son válidas y todas las restricciones se han cumplido.

Las restricciones se van a definir acompañadas por un nombre, lo que permitirá activarlas o desactivarlas según sea el caso; o también mezcladas en la definiciones de las columnas de la tabla. A continuación vamos a describir cada una de las restricciones mencionadas.

`NOT NULL`
: Establece la obligatoriedad de que esta columna tenga un valor no nulo. Se debe especificar junto a la columna a la que afecta. Los valores nulos no ocupan espacio, y son distintos a 0 y al espacio en blanco. Hay que tener cuidado con los valores nulos en las operaciones, ya que `1 * NULL` es igual a `NULL`.

`UNIQUE`
: Evita valores repetidos en una columna, admitiendo valores nulos. Oracle crea un índice automáticamente cuando se habilita esta restricción y lo borra al deshabilitarse.

`DEFAULT`
: Establece un valor por defecto para esa columna, si no se le asigna ninguno.

`CHECK`
: Comprueba que se cumpla una condición determinada al rellenar esa columna. Esta condición sólo debe estar construida con columnas de esta misma tabla.

`PRIMARY KEY`
: Establece el conjunto de columnas que forman la clave primaria de esa tabla. Se comporta como única y obligatoria sin necesidad de explicitarlo. Sólo puede existir una clave primaria por tabla. Puede ser referenciada como clave ajena por otras tablas. Crea un índice automáticamente cuando se habilita o se crea esta restricción. En Oracle, los índices son construidos sobre árboles B^+.

`FOREIGN KEY`
: Establece que el contenido de esta columna será uno de los valores contenidos en una columna de otra tabla maestra. Esta columna marcada como clave ajena puede ser NULL. No hay límite en el número de claves ajenas. La clave ajena puede ser otra columna de la misma tabla. Se puede forzar que cuando una fila de la tabla maestra sea borrada, todas las filas de la tabla detalle cuya clave ajena coincida con la clave borrada se borren también. Esto se consigue añadiendo la coletilla `ON DELETE CASCADE` en la definición de la clave ajena.

Seguidamente se presenta un ejemplo en el que se crean dos tablas, una de departamentos y otra de empleados:

```
REM
REM tabla departamento con un código de departamento, un nombre y una
REM localización.
REM
create table dep (
   cod_dep number(3),
   nombre varchar2(15) not null,
   loc varchar2(10),
```

```
  constraint dep_pk primary key (cod_dep),
  constraint dep_loc check
          (loc in ('Valladolid', 'Boecillo', 'Cigales'))
);

REM
REM tabla empleado con un código de empleado, un nombre, un oficio, un
REM jefe, una fecha de alta en la empresa, un salario mensual, una
REM comisión y el código del departamento donde trabaja.
REM
create table emp (
  cod_emp number(3),
  nombre varchar2(10) not null,
  oficio varchar2(11),
  jefe number(3),
  fecha_alta date,
  salario number(10),
  comision number(10),
  cod_dep number(3),
  constraint emp_pk primary key (cod_emp),
  constraint emp_fk foreign key (cod_dep) references dep(cod_dep)
          on delete cascade,
  constraint emp_ck check (salario > 0)
);
```

4.3 Comando `Describe`

Oracle nos proporciona un comando que resulta muy útil cuando queremos conocer la estructura de una tabla, las columnas que la forman y su tipo y restricciones. Este comando toma una mayor importancia según nos alejemos del momento de creación de una tabla.

La sintásis es la siguiente

```
DESCRIBE tabla
```

Y un ejemplo de su utilización se puede ver al describir la definición de las dos tablas creadas antes. Como no es una sentencia SQL no necesita el ';' al final. También se puede abreviar como `DESC`.

```
SQL> describe dep
 Name                          Null?    Type
 ------------------------------- -------- ----
 COD_DEP                         NOT NULL NUMBER(3)
 NOMBRE                          NOT NULL VARCHAR2(15)
 LOC                                      VARCHAR2(10)

SQL> desc emp
 Name                          Null?    Type
 ------------------------------- -------- ----
 COD_EMP                         NOT NULL NUMBER(4)
 NOMBRE                          NOT NULL VARCHAR2(10)
 OFICIO                                   VARCHAR2(10)
```

```
JEFE                    NUMBER(4)
FECHA_ALTA                DATE
SALARIO                 NUMBER(10)
COMISION                 NUMBER(10)
COD_DEP                  NUMBER(3)
```

5 Modificación

Después de crear una tabla, a veces nos encontramos con que se necesita añadir una columna adicional o modificar la definición de una columna existente. Esta operación se puede realizar con el comando `ALTER TABLE`.

```
ALTER TABLE tabla {ADD | MODIFY} ({columna tipoColumna [NOT NULL],}+);
```

Hay que tener en cuenta varios puntos:

- No es posible disminuir el tamaño de un columna.
- En las modificaciones, los tipos anterior y nuevo deben ser compatibles, o la tabla debe estar vacía.
- La opción `ADD ... NOT NULL` sólo será posible si la tabla está vacía.
- La opción `MODIFY ... NOT NULL` sólo podrá realizarse cuando la tabla no contenga ninguna fila con valor nulo en la columna en cuestión.

Por ejemplo la sentencia siguiente añade la fecha de nacimiento a la tabla de empleados.

```
SQL> alter table emp add (fecha_nac date not null);
```

También se puede querer modificar una tabla añadiendo o eliminando restricciones. En este caso el comando a utilizar será

```
ALTER TABLE tabla {ADD | DROP} CONSTRAINT restricción;
```

6 Inserción, Actualización y Borrado

Una vez que tenemos definida la estructura de una tabla se pueden insertan los datos, modificarlos o borrarlos de la tabla.

Esta tarea entra dentro de las operaciones que se realizan con el lenguaje DML. Este lenguaje permite manipular los objetos de la base de datos, insertando, modificando y/o borrando el contenido de las tablas. Hay que recordar que estas sentencias no son 'autoconfirmadas' y requieren de la sentencia `COMMIT` para que sus efectos perduren en el tiempo, o de la sentencia `ROLLBACK` para deshacer los cambios efectuados.

A continuación vamos a estudiar tres de las sentencias DML más comunes.

6.1 Inserción

El comando que permite insertar filas en las tablas es el siguiente.

INSERT INTO tabla [({columna,}*)] VALUES ({expresión,}+);

Sólo especificaremos las columnas donde insertar y su orden cuando no insertemos datos en todas ellas o no lo hagamos en el mismo orden en que definimos la tabla. La asociación columna-valor es posicional. Los valores deben cumplir con los tipos de datos definidos. Los valores de tipo caracter y fecha deben ir encerrados entre comillas simples, (').

A continuación se puede ver la inserción de filas en las tablas de ejemplo.

```
REM insertar filas en la tabla dep
insert into dep values (100,'Administracion','Valladolid');
insert into dep values (200,'I+D','Boecillo');
insert into dep values (300,'Produccion','Cigales');

REM insertar filas en la tabla emp
insert into emp values
      (101,'Cano','Presidente',null,'3-FEB-96',450000,null,100);
insert into emp values
      (102,'Roncal','Director',101,'3-FEB-96',350000,null,100);
insert into emp values
      (103,'Rueda','Secretario',102,'17-MAR-96',175000,null,100);
insert into emp values
      (104,'Martin','Contable',102,'17-MAR-96',235000,null,100);
insert into emp values
      (105,'Sanz','Comercial',101,'17-MAR-96',150000,10,100);
insert into emp values
      (106,'Lopez','Comercial',101,'21-MAR-96',150000,15,100);
insert into emp values
      (201,'Perez','Director',101,'4-JUN-96',350000,null,200);
insert into emp values
      (202,'Sastre','Analista',201,'8-JUN-96',300000,null,200);
insert into emp values
      (203,'Garcia','Programador',202,'8-JUN-96',225000,null,200);
insert into emp values
      (204,'Mateo','Programador',202,'8-JUN-96',200000,null,200);
insert into emp values
      (301,'Yuste','Director',101,'3-OCT-96',350000,null,300);
insert into emp values
      (302,'Recio','Analista',301,'4-FEB-97',300000,null,300);
insert into emp values
      (303,'Garcia','Programador',302,'4-FEB-97',210000,null,300);
insert into emp values
      (304,'Santana','Programador',302,'4-FEB-97',200000,null,300);
```

6.2 Actualización

Otra de las operaciones más comunes es la modificación de la información almacenada en las tablas. Para ello se utiliza el comando UPDATE cuya sintaxis se muestra a continuación.

UPDATE tabla SET {columna = expresión,}$^{+}$ [WHERE condición];

Se especificará en la cláusula SET las columnas que se actualizarán y con qué valores. La cláusula WHERE indica las filas con las que se va a trabajar. Si se omite la actualización afectará a todas las filas de la tabla.

6.3 Borrado

Con insertar y modificar, la otra operación que completa el trio es la de borrado de filas. La sintaxis es la que sigue:

DELETE FROM tabla [WHERE condición];

Borrará todas las filas que cumplan la condición especificada en la cláusula WHERE. Si esta cláusula no se fija, se borrarán todas las filas de la tabla. Aquí cabe decir que aunque con DELETE borremos todas las filas de una tabla, no borramos la definición de la tabla del diccionario y podemos insertar datos posteriormente en la tabla. Esta es una diferencia con la sentencia DROP TABLE, que produce la eliminación tanto del contenido de la tabla como de la definición de la misma.

---

7 Selección

La recuperación de los datos en el lenguaje SQL se realiza mediante la sentencia SELECT, seleccionar. Esta sentencia permite indicar al SGBD la información que se quiere recuperar. Esta es la sentencia SQL, con diferencia, más habitual. La sentencia SELECT consta de cuatro partes básicas:

- La cláusula SELECT seguida de la descripción de lo que se desea ver, los nombres de las columnas a seleccionar. Esta parte es obligatoria.
- La cláusula FROM seguida de la especificación de las tablas de las que se han de obtener los datos. Esta parte es obligatoria.
- La cláusula WHERE seguida por un criterio de selección, una condición. Esta parte es opcional.
- La cláusula ORDER BY seguida por el criterio de ordenación. Esta parte es opcional.

Una primera aproximación a la sintaxis de la sentencia SELECT puede mostrarnos la siguiente expresión:

SELECT {* | {columna,}$^{+}$}
FROM {tabla,}$^{+}$
[WHERE condición]
[ORDER BY {expresiónColumna [ASC | DESC],}$^{+}$];

Como una primera utilización de la sentencia `SELECT` podemos utilizarla para ver todas las tablas que tenemos en la base de datos.

```
SQL> select table_name from user_tables;

TABLE_NAME
------------------------------
DEP
EMP
```

Un breve análisis de la sentencia anterior nos permite observar que hemos consultado sobre la columna llamada `table_name` almacenada en la tabla `user_tables`, que es la tabla que guarda la información sobre todas las tablas de cada usuario.

7.1 Selección de Columnas

Las columnas a seleccionar se enumeran sin más en la cláusula `SELECT`. Si se desea seleccionar todas las columnas de una tabla se puede hacer enumerando a todas las columnas o colocando un asterisco, `*`, en su lugar.

Cuando se consulta una base de datos, los nombres de las columnas se usan como cabeceras de presentación. Si éste resulta demasiado largo, corto o críptico, puede cambiarse con la misma sentencia SQL de consulta, creando un **alias de columna**.

```
SQL> select nombre "Departamento", loc "Está en" from dep;

Departamento    Esta en
--------------- ----------
Administracion  Valladolid
I+D             Boecillo
Produccion      Cigales
```

7.2 Cláusula `FROM`

La cláusula `FROM` define las tablas de las que se van a seleccionar las columnas.

Se puede añadir al nombre de las tablas el usuario propietario de las mismas de la forma `usuario.tabla`. De esta manera podemos distinguir entre las tablas de un usuario y otro. Oracle siempre considera como prefijo el nombre del propietario de las tablas, aunque no se lo indiquemos. De esta forma dos o más usuarios pueden tener tablas que se llamen igual sin que surjan conflictos. Si quisiéramos acceder a las filas de la tabla `dep` del usuario `jperez`, (ademas de tener privilegios de lectura sobre esa tabla) deberíamos escribir la siguiente sentencia SQL:

```
SQL> select * from jperez.dep;
```

También se puede asociar un alias a las tablas para abreviar los nombres de las tablas. Un ejemplo se puede ver en la sentencia SQL siguiente:

```
SQL> select d.nombre from dep d;
```

7.3 Cláusula `WHERE`

Hasta ahora hemos visto como puede utilizarse la sentencia `SELECT` para recuperar todas las columnas o un subconjunto de ellas de una tabla. Pero este efecto afecta a todas las filas de la tabla, a menos que especifiquemos algo más en la cláusula `WHERE`. Es aquí donde debemos proponer la condición que han de cumplir todas las filas para salir en el resultado de la consulta. La complejidad del criterio de búsqueda es prácticamente ilimitada, y en él se pueden conjugar operadores de diversos tipos con funciones de columnas, componiendo expresiones más o menos complejas.

Operadores de Comparación

Operador	***Operación***	***Ejemplo***
=	Igualdad	`select * from emp where cod_dep = 100;`
!=, <>, ^=	Desigualdad	`select * from emp where cod_dep != 100;`
<	Menor que	`select * from emp where cod_dep < 200;`
>	Mayor que	`select * from emp where cod_dep > 200;`
<=	Menor o igual que	`select * from emp where cod_dep <= 200;`
>=	Mayor o igual que	`select * from emp where cod_dep >= 200;`
in	Igual a cualquiera de los miembros entre paréntesis	`select * from emp where cod_dep in (100, 300);`
not in	Distinto a cualquiera de los miembros entre paréntesis	`select * from emp where cod_dep not in (200);`
between	Contenido en el rango	`select * from emp where cod_emp between 100 and 199;`
not between	Fuera del rango	`select * from emp where cod_emp not between 100 and 199;`
like '_abc%'	Contiene la cadena 'abc' a partir del segundo carácter y luego cualquier cadena de caracteres	`select * from emp where nombre like 'Ma%';`

Operadores de Aritméticos

Operador	***Operación***	***Ejemplo***
+	Suma	`select nombre, salario+comision from emp where oficio='VENDEDOR';`

-	Resta	`select nombre from emp where sysdate-fecha_alta > 365;`
*	Producto	`select nombre, salario*12 from emp;`
/	División	`select nombre, salario/31 from emp;`

Operadores de Cadenas de Caracteres

Operador	*Operación*	*Ejemplo*
||	Concatenación	`select nombre||oficio from emp;`

7.4 Cláusula `ORDER BY`

Se utiliza para especificar el criterio de ordenación de la respuesta a la consulta. Por defecto la ordenación es ascendente, aunque se puede especificar un orden descendente. La ordenación se puede establecer sobre el contenido de columnas o sobre expresiones con columnas. A continuación se puede ver un ejemplo de uso de la cláusula `ORDER BY` en la que quiere obtener un listado de los empleados ordenado de manera descendente por su salario y en caso de igualdad de salario, ordenado ascendentemente por su nombre.

```
SQL> select nombre, salario from emp order by salario desc, nombre;

NOMBRE       SALARIO
---------- ----------
Cano           450000
Perez          350000
Roncal         350000
Yuste          350000
Recio          300000
Sastre         300000
Martin         235000
Garcia         225000
Garcia         210000
Mateo          200000
Santana        200000
Rueda          175000
Lopez          150000
Sanz           150000

14 rows selected.
```

7.5 Cláusula `DISTINCT`

Cuando se realiza una consulta sobre una tabla en la que se extrae información de varias columnas, puede ocurrir que, si no incluimos la/s columna/s que forman la clave principal, obtengamos filas repetidas en la respuesta.

Si este comportamiento no nos resulta satisfactorio podemos utilizar la cláusula `DISTINCT` para eliminar las filas duplicadas obtenidas como respuesta a una consulta.

Podemos ver como funciona en el siguiente ejemplo, en el que preguntamos por los distintos oficios de nuestros empleados.

```
SQL> select oficio from emp;
```

Sin utilizar la cláusula `DISTINCT` obtendremos la siguiente respuesta

```
OFICIO
----------
Presidente
Director
Secretario
Contable
Comercial
Comercial
Director
Analista
Programador
Programador
Director
Analista
Programador
Programador

14 rows selected.
```

Pero si incluimos la cláusula `DISTINCT` la respuesta varía para adecuarse más a nuestras espectativas.

```
SQL> select distinct oficio from emp;

OFICIO
----------
Analista
Comercial
Contable
Director
Presidente
Programador
Secretario

7 rows selected.
```

7.6 Funciones

Existen en SQL muchas funciones que pueden complementar el manejo de los datos en las consultas. Se utilizan dentro de las expresiones y actuan con los valores de las columnas, variables o constantes.

Se pueden incluir en las clásulas `SELECT`, `WHERE` y `ORDER BY`.

Pueden anidarse funciones dentro de funciones. Y existe una gran variedad de funciones para cada tipo de datos:

- aritméticas,
- de cadenas de caracteres,
- de manejo de fechas,
- de conversión,
- otras,
- de grupo.

Funciones Aritméticas

Función	***Cometido***	***Ejemplo***	***Resultado***
ABS(n)	Calcula el valor absoluto de *n*.	`select abs(-15) from dual;`	15
CEIL(n)	Calcula el valor entero inmediatamente superior o igual a *n*.	`select ceil(15.7) from dual;`	16
FLOOR (n)	Calcula el valor entero inmediatamante inferior o igual a *n*.	`select floor(15.7) from dual;`	15
MOD (m,n)	Calcula el resto resultante de dividir *m* entre *n*.	`select mod(11,4) from dual;`	3
POWER (m,n)	Calcula la potencia *n*-esima de *m*.	`select power(3,2) from dual;`	9
ROUND (m,n)	Calcula el redondeo de *m* a *n* decimales. Si *n*<0 el redondeo se efectua a por la izquierda del punto decimal.	`select round (123.456,1) from dual;`	123.5
SQRT(n)	Calcula la raíz cuadrada de *n*.	`select sqrt(4) from dual;`	2
TRUNC (m,n)	Calcula *m* truncado a *n* decimales (*n* puede ser negativo).	`select trunc (123.456,1) from dual;`	123.4
SIGN(n)	Calcula el signo de *n*, devolviendo -1 si *n*<0, 0 si *n*=0 y 1 si *n*>0.	`select sign(-12) from dual;`	-1

Funciones de Cadenas de Caracteres

Función	***Cometido***	***Ejemplo***	***Resultado***
CHR(n)	Devuelve el carácter cuyo valor codificado es *n*.	`select chr(65) from dual;`	A
ASCII(cad)	Devuelve el valor ascii de *cad*.	`select ascii('A') from dual;`	65
	Devuelve *cad1* concatenada con *cad2*. Esta función es		

	esquivalente al operador ||.		
LOWER(cad)	Devuelve la cadena *cad* con todas sus letras convertidas a minúsculas.	`select lower ('MinUsCulAs') from dual;`	minusculas
UPPER(cad)	Devuelve la cadena *cad* con todas sus letras convertidas a mayúsculas.	`select upper ('maYuSCulAs') from dual;`	MAYUSCULAS
INITCAP(cad)	Devuelve *cad* con el primer caracter en mayúsculas.	`select initcap ('isabel') from dual;`	Isabel
LPAD (cad1,n,cad2)	Devuelve *cad1* con longitud *n*, y ajustada a la derecha, rellenando por la izquierda con *cad2*.	`select lpad('P',5,'*') from dual;`	****P
RPAD (cad1,n,cad2)	Devuelve *cad1* con longitud *n*, y ajustada a la izquierda, rellenando por la derecha con *cad2*.	`select rpad('P',5,'*') from dual;`	P****
REPLACE (cad,ant,nue)	Devuelve *cad* en la que cada ocurrencia de la cadena *ant* ha sido sustituida por la cadena *nue*.	`select replace ('digo','i','ie') from dual;`	diego
SUBSTR (cad,m,n)	Devuelve la sudcadena de *cad* compuesta por *n* caracteres a partir de la posicion *m*.	`select substr ('ABCDEFG',3,2) from dual;`	CD
LENGTH(cad)	Devuelve la longitud de *cad*.	`select length('cadena') from dual;`	6

Funciones de Manejo de Fechas

Función	***Cometido***	***Ejemplo***	***Resultado***
SYSDATE	Devuelve la fecha y hora actuales.	`select sysdate from dual;`	14-MAR-97
ADD_ MONTHS(d,n)	Devuelve la fecha *d* incrementada en *n* meses.	`select add_months (sysdate,4) from dual;`	14-JUL-97
LAST_DAY(d)	Devuelve la fecha del último día del mes de *d*.	`select last_day (sysdate) from dual;`	31-MAR-97
MONTHS_ BETWEEN(d1, d2)	Devuelve la diferencia en meses entre las fechas *d1* y *d2*.	`select months_between (sysdate,'01-JAN-97') from dual;`	2.43409424

NEXT_DAY (d,cad)	Devuelve la fecha del primer día de la semana *cad* después de la fecha *d*.	`select next_day (sysdate, 'sunday') from dual;`	16-MAR-97

Funciones de Conversión de Tipos

Función	***Cometido***	***Ejemplo***	***Resultado***
TO_NUMBER (cad,fmto)	Convierte la cadena *cad* a un número, opcionalmente de acuerdo con el formato *fmto.*	`select to_number ('12345') from dual;`	124345
TO_CHAR(d, fmto)	Convierte la fecha *d* a una cadena de caracteres, opcionalmente de acuerdo con el formato *fmto.*	`select to_char (sysdate) from dual;`	'14-MAR-97'
TO_DATE (cad,fmto)	Convierte la cadena *cad* de tipo varchar2 a fecha, opcionalmente de acuerdo con el formato *fmto.*	`select to_date ('1-JAN-97') from dual;`	01-JAN-97

Con las fechas pueden utilizarse varios formatos. Estos formatos permiten modificar la presentación de una fecha. En la siguiente tabla se presentan algunos formatos de fecha y el resultado que generan.

Máscaras de Formato Numéricas

Formato	***Cometido***	***Ejemplo***	***Resultado***
cc ó scc	Valor del siglo.	`select to_char (sysdate,'cc') from dual;`	20
y,yyy ó sy,yyy	Año con coma, con o sin signo.	`select to_char (sysdate,'y,yyy') from dual;`	1,997
yyyy ó yyy ó yy ó y	Año sin signo con cuatro, tres, dos o un dígitos.	`select to_char (sysdate,'yyyy') from dual;`	1997
q	Trimestre.	`select to_char(sysdate,'q') from dual;`	1
ww ó w	Número de la semana del año o del mes.	`select to_char (sysdate,'ww') from dual;`	11
mm	Número del mes.	`select to_char (sysdate,'mm') from dual;`	03
ddd ó dd ó d	Número del día del año, del mes o de la semana.	`select to_char (sysdate,'ddd') from dual;`	073
hh ó hh12 ó hh24	La hora en formato 12h. o 24h.	`select to_char (sysdate,'hh') from dual;`	12
mi	Los minutos de la hora.	`select to_char (sysdate,'mi') from dual;`	15
ss ó sssss	Los segundos dentro del minuto, o desde las 0 horas.	`select to_char (sysdate,'sssss') from dual;`	44159

Máscaras de Formato de Caracteres

Formato	*Cometido*	*Ejemplo*	*Resultado*
syear ó year	Año en Inglés	`select to_char (sysdate,'syear) from dual;`	nineteen ninety-seven
month o mon	Nombre del mes o su abreviatura de tres letras.	`select to_char (sysdate,'month') from dual;`	march
day ó dy	Nombre del día de la semana o su abreviatura de tres letras.	`select to_char (sysdate,'day') from dual;`	friday
a.m. ó p.m.	El espacio del día.	`select to_char (sysdate,'a.m.') from dual;`	p.m.
b.c. ó a.d.	Indicador del año respecto al del nacimiento de Cristo.	`select to_char (sysdate,'b.c.') from dual;`	a.d.

Otras Funciones

Función	*Cometido*	*Ejemplo*	*Resultado*
DECODE(var, val1, cod1, val2, cod2, ..., defecto)	Convierte el valor de *var*, de acuerdo con la codificación.	`select decode(oficio, 'Presidente', 'P', 'Director', 'D', 'X') from emp;`	P, D, X, ...
GREATEST(exp1, exp2, ...)	Devuelve el mayor valor de una lista.	sin ejemplo.	sin ejemplo.
LEAST(cad,fmto)	Devuelve el menor valor de una lista.	sin ejemplo.	sin ejemplo.
NVL(val, exp)	Devuelve la expresión exp si *val* es NULL, y *val* si en otro caso.	`select salario+nvl (comision,0) from emp;`	450000, 350000, ...

7.7 Cláusula `GROUP BY`

SQL nos permite agrupar las filas resultado de una consulta en conjuntos y aplicar funciones sobre esos conjuntos de filas.

La sintaxis es la siguiente:

SELECT {* | {columna,}$^{+}$}
FROM {tabla,}$^{+}$
WHERE condición
GROUP BY {columna ,}$^{+}$

HAVING condición
ORDER BY {expresiónColumna [ASC | DESC],}$^{+}$;

En la cláusula `GROUP BY` se colocan las columnas por las que vamos a agrupar. Y en la cláusula `HAVING` se especifica la condición que han de cumplir los grupos para pasar al resultado.

La evaluación de las diferentes cláusulas en tiempo de ejecución se efectúa en el siguiente orden:

- `WHERE` filtra las filas
- `GROUP BY` crea una tabla de grupo nueva
- `HAVING` filtra los grupos
- `ORDER BY` clasifica la salida

Un ejemplo de utilización de la selección de grupos puede ser seleccionar los empleados agrupados por su oficio. Un primer intento de consulta es el siguiente:

```
SQL> select nombre, oficio from emp group by oficio;

select nombre, oficio from emp
       *
ERROR at line 1:
ORA-00979: not a GROUP BY expression
```

Se presenta un error debido a que cuando se utiliza `GROUP BY`, las columnas implicadas en el `SELECT` y que no aparezcan en la cláusula `GROUP BY` deben tener una función de agrupamiento. En otras palabras, la columna `nombre` debe tener una función de agrupamiento que actue sobre ella (max, min, sum, count, avg). Si no puede ser así, deberá llevar dicha columna a la cláusula `GROUP BY`.

De nuevo, el ejemplo quedará así:

```
SQL> select count(nombre), oficio from emp group by oficio;

COUNT(NOMBRE) OFICIO
------------- -----------
            2 Analista
            2 Comercial
            1 Contable
            3 Director
            1 Presidente
            4 Programador
            1 Secretario

7 rows selected.
```

Las funciones de agrupamiento que se pueden utilizar son las siguientes.

Funciones de Agrupamiento

Función	*Cometido*	*Ejemplo*

COUNT(col)	Cuenta el número de filas agrupadas.	`select count(nombre) ,oficio from emp group by oficio;`
AVG(col)	Calcula el valor medio de todos los valores de la columna *col*.	`select avg(salario),oficio from emp group by oficio;`
MAX(col)	Calcula el valor máximo de todos los valores de la columna *col*.	`select max(salario),oficio from emp group by oficio;`
MIN(col)	Calcula el valor mínimo de todos los valores de la columna *col*.	`select min(salario),oficio from emp group by oficio;`
SUM(col)	Calcula la suma de los valores de la columna *col*.	`select sum(salario), oficio from emp group by oficio;`
STDDEV (col)	Calcula la desviación típica de los valores de la columna *col* sin tener en cuenta los valores nulos.	`select stddev(salario), oficio from emp group by oficio;`
VARIANCE (col)	Calcula la varianza de los valores de la columna *col* sin tener en cuenta los valores nulos.	`select variance(salario), oficio from emp group by oficio;`

Hay que tener en cuenta que los valores nulos no participan en el cálculo de las funciones de conjuntos. Estas funciones se pueden utilizar con las cláusulas `DISTINCT` y `ALL`. También se pueden utilizar aunque no realicemos agrupación alguna en la consulta, considerando a toda la tabla como un grupo.

```
SQL> select count(*) from emp;

  COUNT(*)
----------
        14
```

7.8 Expresiones con Sentencias `Select`

El resultado de cada consulta es un conjunto de filas. Y con conjuntos se pueden realizar tres operaciones típicas: la unión, la intersección y la diferencia.

Unión, `UNION`
: Combina todas las filas del primer conjunto con todas las filas del segundo. Cualquier fila duplicada se reducirá a una sóla.

Intersección, `INTERSECT`
: Examinará las filas de los conjuntos de entrada y devolverá aquellas que aparezcan en ambos. Todas las filas duplicadas serán eliminadas antes de la generación del conjunto resultante.

Diferencia, `MINUS`
: Devuelve aquellas filas que están en el primer conjunto pero no en el segundo. Las filas duplicadas del primer conjunto se reducirán a una fila única antes de empezar la comparación con el segundo conjunto.

Reglas para el Manejo de los Operadores de Conjuntos:

- Pueden ser encadenados en cualquier combinación, siendo evaluados de izquierda a derecha.
- No existe jerarquía de precedencia en el uso de estos operadores, pero puede ser forzada mediante paréntesis.
- Pueden ser empleados con conjuntos de diferentes tablas siempre que se apliquen las siguientes reglas:
  - Las columnas son relacionadas en orden, de izquierda a derecha.
  - Los nombres de las columnas son irrelevantes.
  - Los tipos de datos deben coincidir.

Como ejemplo podemos consultar sobre todos los nombres de empleado que trabajan para los departamentos 100 o 300. Esto se consigue restando a todos los nombres de empleados, aquellos que están en el departamento 200.

```
SQL> select nombre from emp
  2  minus
  3  select nombre from emp where cod_dep=200;

NOMBRE
----------
Cano
Lopez
Martin
Recio
Roncal
Rueda
Santana
Sanz
Yuste

9 rows selected.
```

7.9 Combinaciones

Hasta ahora hemos construido consultas con una única tabla, pero esto no debe ser siempre así.

De hecho, sólo se alcanza la verdadera potencia del SQL cuando combinamos el contenido de más de una tabla.

Supongamos que queremos conseguir una lista con los empleados y los departamentos para los que trabajan. Esta información está repartida en las dos tablas que tenemos, `emp` y `dep`. Así, podríamos intentar una consulta que seleccionara el campo `nombre` de la tabla `emp` y el `nombre` del departamento. Y aquí surge el primer problema, ¿cómo distinguimos entre dos columnas que llamándose igual, pertenecen a tablas distintas? Para eso se utiliza como prefijo o el nombre de la tabla (`dep.nombre`) o un alias de tabla, un nombre que se asocia a cada tabla y se coloca como prefijo a la columna (`d.nombre`).

Realicemos la consulta ...

```
SQL> select e.nombre, d.nombre from emp e, dep d;

NOMBRE     NOMBRE
---------- --------------
Cano       Administracion
```

```
Roncal     Administracion
Rueda      Administracion
Martin     Administracion
Sanz       Administracion
Lopez      Administracion
Perez      Administracion
Sastre     Administracion
Garcia     Administracion
Mateo      Administracion
Yuste      Administracion
Recio      Administracion
Garcia     Administracion
Santana    Administracion
Cano       I+D
Roncal     I+D
Rueda      I+D
...
42 rows selected.
```

El resultado puede sorprender un poco. Lo que obtenemos es el producto cartesiano de todos los empleados por todos los departamentos. SQL ha cogido cada fila de la tabla `emp` y le ha asociado todos los `cod_dep` de la tabla `dep`.

Para conseguir lo que queremos tenemos que forzar que se asocie a un empleado con el nombre del departamento para el que trabaja. Y esto se puede hacer si añadimos la condición de que el `cod_dep` tenga el mismo valor en la fila de la tabla `emp` que en la fila escogida de la tabla `dep`:

```
SQL> select e.nombre, d.nombre from emp e, dep d
  2> where e.cod_dep = d.cod_dep;

NOMBRE     NOMBRE
---------- ---------------
Cano       Administracion
Roncal     Administracion
Rueda      Administracion
Martin     Administracion
Sanz       Administracion
Lopez      Administracion
Perez      I+D
Sastre     I+D
Garcia     I+D
Mateo      I+D
Yuste      Produccion
Recio      Produccion
Garcia     Produccion
Santana    Produccion

14 rows selected.
```

De la misma manera se pueden combinar más de dos tablas. Lo importante es emparejar los campos que han de tener valores iguales.

Reglas de Combinación:

- Pueden combinarse tantas tablas como se desee.
- El criterio de combinación puede estar formado por más de una pareja de columnas.
- En la cláusula `SELECT` pueden citarse columnas de ambas tablas, condicionen o no la combinación.
- Si hay columnas con el mismo nombre en las distintas tablas, deben identificarse especificando la tabla de procedencia o utilizando un alias de tabla.

Existe un tipo especial de combinación llamada **Combinación Externa**. Suponga que se crea un nuevo departamento, (`insert into dep values (400,'Distribucion','Valladolid');`) pero todavía no hemos asignado personal al mismo. Si realizamos la consulta anterior, el nuevo departamento no aparecerá en la respuesta. Pero esto se puede evitar si señalamos en la cláusula `WHERE` la posibilidad de que en la tabla de empleados no exista alguno de los códigos de departamento que si exista en la tabla de departamentos. Esto se hace colocando un `(+)` de la siguiente manera:

```
SQL>  select e.nombre, d.nombre
  2  from emp e, dep d
  3  where e.cod_dep(+)=d.cod_dep;

NOMBRE     NOMBRE
---------- ---------------
Cano       Administracion
Roncal     Administracion
Rueda      Administracion
Martin     Administracion
Sanz       Administracion
Lopez      Administracion
Perez      I+D
Sastre     I+D
Garcia     I+D
Mateo      I+D
Yuste      Produccion
Recio      Produccion
Garcia     Produccion
Santana    Produccion
           Distribucion

15 rows selected.
```

7.10 Subconsultas

A veces se han de utilizar en una consulta los resultados de otra consulta, llamada subconsulta.

Un ejemplo de esto ocurre cuando queremos conocer los nombres de los empleados cuyo salario está por encima de la media:

```
SQL> select nombre from emp
  2  where salario > (select avg(salario) from emp);

NOMBRE
----------
```

```
Cano
Roncal
Perez
Sastre
Yuste
Recio

6 rows selected.
```

La consulta más interna calcula el salario medio, y la consulta más externa lo utiliza para seleccionar los nombres que ganan más que la media.

El valor de comparación puede ser un valor simple, como en el ejemplo anterior, o un conjunto de valores. Hay que tener en cuenta este detalle ya que el tipo de operador a utilizar varía. En el primer caso se puede utilizar un operador de comparación de carácter aritmético (<, >, etc.). Y en el segundo uno de tipo lógico (`IN`).

Las subconsultas pueden devolver más de una columna, y se habrán de comparar de manera consecuente:

- Las columnas de la clausula `WHERE` de la consulta principal deben estár agrupadas por parentesis.
- Las columnas encerradas entre paréntesis deben coincidir en número y tipo de datos con los datos que devuelve la subconsulta.

El nivel de anidamiento de subconsultas es ilimitado.

Se puede utilizar una subconsulta para insertar valores en una tabla en el momento de la creación de la misma con la cláusula `AS`. Si quisieramos crear una tabla con los datos de los empleados del departamento 200 lo podríamos hacer de la siguiente manera:

```
SQL> create table dep200 (nombre, oficio)
  2  as select nombre,oficio from emp
  3  where cod_dep=200;

Table created.
```

No es necesario especificar tipos ni tamaños de las columnas, ya que vienen determinados por los tipos y tamaños de las columnas recuperadas en la subconsulta.

---

8 Eliminación

Cuando una tabla ya no es útil y no vamos a volver a necesitarla debe ser borrada. Esta operación se puede realizar con el comando `DROP TABLE`.

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

Se borra la tabla de la base de datos, borrando toda la información contenida en la tabla, es decir, todas las filas. También se borrará toda la información que sobre la tabla existiera en el diccionario.

Puede que si alguna columna de esta tabla a borrar sirva como clave ajena de alguna tabla detalle, impida la eliminación de la tabla, ya que existe una restricción que requiere de la existencia de la tabla maestra. Esto se puede areglar colocando la coletilla `CASCADE CONSTRAINTS`. Esto produce que las restricciones de la tabla detalle se borren antes de borrar la tabla maestra.

La siguiente sentencia produce la eliminación de la tabla de departamentos.

```
SQL> drop table dep cascade constraints;

Table dropped.
```

---

9 Vistas

Una vista es como una ventana a través de la cual se puede consultar o cambiar información de la tabla a la que está asociada.

Las vistas tienen la misma estructura que una tabla: filas y columnas. La única diferencia es que sólo se almacena de ellas la definición, no los datos. Los datos que se recuperan mediante una consulta a una vista se presentarán igual que los de una tabla. De hecho, si no se sabe que se está trabajando con una vista, nada hace suponer que es así. Al igual que sucede con una tabla, se pueden insertar, actualizar, borrar y seleccionar datos en una vista. Aunque siempre es posible seleccionar datos de una vista, en algunas condiciones existen restricciones para realizar el resto de las operaciones sobre vistas.

¿Por qué utilizar vistas?

- Las vistas pueden proporcionar un nivel adicional de seguridad. Por ejemplo, en la tabla de empleados, cada responsable de departamento sólo tendrá acceso a la información de sus empleados. La siguiente sentencia produce la creación de la vista de los empleados del departamento de administración (`cod_dep=100`).

  ```
  SQL> create view ampAdmin as
    2  select * from ep where cod_dep=100;
  View created.
  ```

- Las vistas permiten ocultar la complejidad de los datos. Una BD se compone de muchas tablas. La información de dos o más tablas puede recperarse utilizando una combinación de dos o más tablas, y estas combinaciones pueden llegar a ser muy confusas. Creando una vista como resultado de la combinación se puede ocultar la complejidad al usuario.

- Las vistas ayudan a mantener unos nombres razonables.

Creación de una Vista

```
CREATE VIEW vista [({columna ,}+] AS consulta ;
```

La vista se crea con las columnas que devuelve una consulta. Si no nos importa que las columnas de la vista hereden los nombres de las columnas recuperadas en la consulta no tenemos que especificarlos.

Borrado de una Vista

```
DROP VIEW vista ;
```

9.1 Operaciones sobre Vistas

Consultas

La consultas sobre las vistas se tratan de igual modo que sobre las tablas.

Actualizaciones

La información puede ser actualizada en las vistas directamente o a través de las tablas sobre las que se definen.

Esisten algunas restricciones:

Borrado de filas de una tabla a través de una vista
: La vista se debe crear con filas de una sola tabla; sin utilizar las cláusulas `GROUP BY` y `DISTINCT`; y sin utilizar funciones de grupo o referencias a pseudocolumnas (ROWNUM).

Actualización de filas a través de una vista
: La vista ha de estar definida según las restricciones anteriores y además ninguna de las columnas a actualizar debe haber sido definida como una expresión.

Inserción de filas en una tabla a través de una vista
: Todas las restricciones y además todas las columnas obligatorias de la tabla asociada deben estar presentes en la vista.

9.2 Vistas de más de una Tabla

Se pueden definir vistas sobre más de una tabla. Por ejemplo, sobre la combinación de dos tablas.

Podemos querer ver todos los datos de los empleados del departamento Administración.

```
SQL> create view depAdmin (cod_emp, nombre_emp, nombre_dep, dir)
  2  as select e.cod_emp, e.nombre, d.nombre, d.loc
  3  from emp e, dep d
  4  where e.cod_dep=d.cod_dep and d.nombre='Administracion';

SQL> select * from depAdmin;

   COD_EMP NOMBRE_EMP NOMBRE_DEP      DIR
---------- ---------- --------------- ----------
       101 Cano       Administracion  Valladolid
       102 Roncal     Administracion  Valladolid
       103 Rueda      Administracion  Valladolid
       104 Martin     Administracion  Valladolid
       105 Sanz       Administracion  Valladolid
       106 Lopez      Administracion  Valladolid
```

6 rows selected.

---

10 Jugando con los Nombres

A continuación dos comandos que permiten jugar con los nombres de los objetos en SQL: `SINONYM` y `RENAME`.

10.1 Sinónimos

SQL permite crear un sinónimo para una tabla o vista. Esto supone que pueden utilizarse dos nombres diferentes para un mismo objeto.

Creación de un Sinónimo

```
CREATE SYNONYM sinonimo FOR [usuario.]{tabla | vista} ;
```

Borrado de un Sinónimo

```
DROP SYNONYM sinonimo ;
```

Una primera utilidad de los sinónimos es la posibilidad de independizar las aplicaciones de los nombres físicos de las tablas que manejan. Asi, las aplicaciones harán referencia a un sinónimo de tabla, que en cada caso puede estar asociado a una tabla distinta.

Otra utilidad es la posibilidad de que un usuario acceda a las tablas de otro usuario como si fueran suyas, siempre que tenga permiso para hacerlo, si al definir el sinónimo incluye el nombre del usuario en la denominación de la tabla. Así si el usuarioA tiene permiso para leer el contenido de la tabla `emp` del usuarioB, entonces desde la ejecución de la sentencia `CREATE SYNONYM plantilla FOR usuarioB.emp` verá la tabla `usuarioB.emp` como `plantilla`.

10.2 Comando `RENAME`

El comando `RENAME` se utiliza para modificar el nombre de una tabla, vista o sinónimo.

La sintásis es la siguiente

```
RENAME {tabla | vista | sinonimo} to nuevoNombre ;
```

Esta sentencia cambiará el nombre antiguo por el nuevo, y a partir de este momento cualquier acceso al objeto por el nombre antiguo será respondido con un mensaje de error.

Conviene resaltar la diferencia entre el comando SYNONYM y el comando RENAME. Mientras que el primero mantiene el nombre original para acceder al objeto, el segundo elimina ese primer nombre sustituyendolo por el nuevo.

---

11 Lo Más SQL*Plus

En este apartado vamos a profundizar un poco en las otras posibilidades que nos brinda SQL*Plus en los:

- ficheros de comandos, y
- generación de informes,

11.1 Ficheros de Comandos

Aunque ya vimos una introducción a los ficheros de comandos en anteriormente, vamos ahora a profundizar un poco en las posibilidades que nos ofrece SQL*Plus.

En un fichero de comandos se pueden incluir:

- líneas de comentarios,
- líneas de ejecución,
- líneas de comandos SQL, y
- líneas de comandos SQL*Plus.

Líneas de Comentarios

Se pueden introducir comentarios en una archivo de comandos de tres maneras:

- Utilizando del comando REM del SQL*Plus.
- Utilizando los delimitadores de comentario de SQL /* y */.
- Utilizando los símbolos de comentario PL/SQL "__".

Líneas de Ejecución

Constan de una única barra inclinada, "/", y se introducen a continuación de cada sentencia SQL indicando su ejecución.

Sustituyen al punto y coma, ";" al final de las sentencias SQL.

Líneas de Comandos SQL

Se puede introducir cualquiera de los comandos SQL enumerados en este curso, y se ejecutarán de manera secuencial.

Se permite el anidamiento de los ficheros de comandos.

Líneas de Comandos SQL*Plus

SQL*Plus aporta una serie de posibilidades al lenguaje SQL que le acerca un poco mas a lo que entendemos como un lenguaje de programación.

Se pueden definir constantes y variables, capturar datos del teclado, introducir parámetros en la llamada de un archivo de comandos, y alguna cosa más.

Variables de Usuario

Se pueden definir Variables de usuario con el comando `DEFINE`

```
DEFINE Variable = valor
```

Para borrar una variable se utiliza el comando `UNDEFINE`

```
UNDEFINE variable
```

Como ejemplo se puede definir la variable `OFICIO`

```
SQL> define oficio=analista
```

Variables de Sustitución

Las variables de sustitución son un nombre de variable de usuario con el símbolo & delante. Cuando SQL*Plus detecta una variable de sustitución en un comando, ejecuta el comando tomando el valor de la variable.

Esto se puede ver en el ejemplo, donde preguntamos por los empleados que son analistas:

```
SQL> define oficio=Analista
SQL> define tabla=emp
SQL> select nombre, oficio from &tabla where oficio='&oficio';
old   1: select nombre, oficio from &tabla where oficio='&oficio'
new   1: select nombre, oficio from emp where oficio='Analista'

NOMBRE     OFICIO
---------- ----------
Sastre     Analista
Recio      Analista
```

Captura de Datos desde el Terminal

En muchas ocasiones es necesario recoger datos desde un terminal, que luego serán utilizados en el archivo de comandos. Para realizarlo se pueden utilizar dos medios: las variables de sustitución o los parámetros en la línea de comandos.

Cuando SQL*Plus reconoce una variable de sustitución sin valor asignado se lo pide al usuario:

```
SQL> select * from dep where nombre='&nombredep';
Enter value for nombredep: I+D
old   1: select * from dep where nombre='&nombredep'
new   1: select * from dep where nombre='I+D'
```

```
  COD_DEP NOMBRE         LOC
--------- -------------- ----------
      200 I+D            Boecillo
```

Si se desea que SQL*Plus pregunte por el valor de la variable al usuario sólo la primera vez que se encuentra con ella, se colocará "&&" delante de la variable de usuario.

También se pueden utilizar hasta nueve parámetros en la línea de comandos cuando se llama a la ejecución de un archivo de comandos.

En el archivo de comandos nos referiremos a los parámetros con las variables `&1`, `&2`, ... ,`&9` que se corresponden posicionalmente con ellos.

Desde el archivo de comandos se puede hacer referencia a los parámetros cualquier número de veces y en cualquier orden.

Comandos de Comunicación con el Usuario

Los siguientes comandos proporcionan un medio de comunicación con el usuario:

- PROMPT: presenta un mensaje en la pantalla.
- ACCEPT: Solicita un valor y lo almacena en la variable de usuario que se especifíque.
- PAUSE: Obliga al usuario a pulsar *Return* despué de leer un mensaje.

Para ver cómo funcionan sirve el siguiente ejemplo:

```
prompt Buscar los datos de un empleado.
pause Pulse Return.
accept nombremp prompt 'Empleado? '
select * from emp where nombre='&nombremp';
```

Otros Comandos

Los siguientes comandos también se pueden incluir en un archivo de comandos:

- CONNECT: para conectarse como otro usuario.
- HELP: para obtener ayuda en línea.
- EXIT: para dejar SQL*PLus y salir al Sistema Operativo.
- DESCRIBE ó DESC: para obtener información sobre la estructura de una tabla.
- HOST o !: para ejecutar un comando del Sistema Operativo.

11.2 Generación de Informes

Con SQL*Plus podemos dar forma a los resultados de las consultas para producir un informe. Podremos:

- Cambiar las cabeceras de las columnas.
- Dar forma a las columnas de tipo `number`, `varchar2`, `date` y `long`.
- Copiar y listar atributos de presentación de las columnas.
- Suprimir valores duplicados e introducir espacios para mejorar la presentación.
- Realizar y mostrar cálculos (totales, medias, mínimos, máximos, etc.).

- Definir las dimensiones de las páginas.
- Ubicar títulos en la cabecera y pie de las páginas.
- Introducir la fecha o el número de página en los títulos.

Pero de todo esto sólo vamos a ver el modo de realizar las operaciones más comunes y sencillas.

Básicamente, el formato con el que se van a presentar los resultados de las cosultas dependen de unos parámetros y de unos comandos.

Parámetros

- SET LINESIZE: pone el número máximo de caracteres por línea. Por defecto vale 80 y el máximo es 999.
- SET PAGESIZE: pone el número de filas de la salida antes de empezar una nueva página. Por defecto es 25. Incluye el título y las líneas de pausa.
- SET HEADING [ON | OFF]: Activa/desactiva la utilización de encabezados de columnas. Por defecto está activado.
- SET NULL texto: Indica la cadena de caracteres que hay que colocar en sustitución de los valores NULL. Por defecto es "".
- SET ECHO [ON | OFF]: Activa/desactiva la visualización de los comandos que SQL*Plus ejecuta según van siendo tratados. Por defecto está desactivada.
- SET FEEDBACK [n | ON | OFF]: Muestra el número de registros recuperados en cada consulta cuando se recuperan n o más registros. ON se pueden considerar como n=1, y OFF como n=0.
- SET VERIFY [ON | OFF]: Controla la salida de confirmación para los valores de las variables de sustitución. Por defecto está activado.

Comandos

- TTITLE: formación del encabezado de página.
- BTITLE: formación del pie de página.
- COLUMN: formatear cada columna.
- BREAK: puntos de ruptura en los listados.
- COMPUTE: realizar cálculos con las columnas.