ESCUELA SUPERIOR DE INGENIERÍA DE CÁDIZ

GRADO EN INGENIERÍA INFORMÁTICA

# PRÁCTICAS DE BASES DE DATOS

## Curso 2014–15

Prof. Dra. Mª Esther Gadeschi Díaz
Dpto. de Ingeniería Informática
Universidad de Cádiz

# Índice general

# Índice de figuras

# Índice de tablas

# Índice de ejemplos

Parte 1

# SGBD Oracle

# Capítulo 1

# SQL*Plus

Este capítulo nos permite conocer y familiarizarnos con el producto *SQL*Plus*. En especial con las siguientes características: identificación de usuario, introducción de órdenes, utilización de variables, entrada/salida de datos, personalización del entorno de trabajo y realización de informes.

## 1.1. Introducción

El producto de Oracle *SQL*Plus*[1] es un intérprete de línea de órdenes. Está constituido por las órdenes *SQL* del estándar *ANSI*, enfocados a la creación, comprobación y manipulación de la base de datos; y las órdenes *SQL*Plus* suministradas por Oracle, que constituyen un conjunto adicional al anterior, permitiendo un mejor acceso a la base de datos, presentación de resultados y control de transacciones. Cualquiera de los dos tipos de órdenes se pueden ejecutar directamente desde el indicador *SQL* o en segundo plano.

El *SQL*Plus* suministra al usuario un acceso directo a la base de datos, en función de los privilegios que se tengan asignados.

Este producto puede ser utilizado por muchos tipos de usuarios. Los programadores se basan en él para crear, mantener y manipular la base de datos. El administrador del sistema también se basa en él para mantener y hacer un seguimiento de la base de datos.

[1]En este libro se va a seguir la misma nomenclatura que se sigue en los manuales de Oracle. Los corchetes indica que lo que va dentro es opcional. Las llaves indican que es obligatorio especificar alguna opción. La barra vertical (|) separa a las distintas opciones. Las opciones establecidas por defecto están subrayadas. Esta nomenclatura se va a mantener en todo el texto.

Por último, el usuario final se basa en *SQL*Plus* para extraer información de la base de datos.

Como ya hemos visto, *SQL* constituye la esencia de una base de datos Oracle; de hecho, es la esencia de todas las bases de datos relacionales. *SQL*Plus* es la implementación específica que hace Oracle de *SQL* con características adicionales que permiten recuperar, dar formato y controlar los datos según las necesidades. Pensamos que, de entre todas las opciones que se ofrecen para extraer datos de una base de datos Oracle, *SQL*Plus* es una extraordinaria herramienta.

Todos las órdenes *SQL* manipulan la base de datos de una u otra forma comportándose de una manera similar, devolviendo un mensaje de estado que indica el éxito o el fracaso de la operación. La orden `select`[2] se comporta de forma diferente, no modificando en nada la base de datos y devolviendo una copia de la información que se haya solicitado. No existen ayudas de ningún tipo: menús, teclas especiales, etc., debiendo el usuario conocer las órdenes y teclearlas para obtener la información de la base de datos Oracle.

Las órdenes *SQL*Plus* permiten controlar el entorno y presentar de forma controlada las consultas realizadas a la base de datos. Normalmente se usan como descriptor de informes interactivo. Se pueden crear informes refinados y bien formateados, proporcionando un control sencillo sobre los títulos, cabeceras de columnas, subtotales y totales, reformas de números y textos, etc.

El uso más común de *SQL*Plus* es para consultas simples e impresión de informes. Se pueden formatear los informes de acuerdo con los gustos y necesidades del usuario usando sólo unas pocas órdenes, que son palabras reservadas.

Los informes se pueden escribir completamente mientras se trabaja interactivamente con este producto, esto es, se puede escribir mandatos que trabajen con cabeceras de páginas, títulos, columnas, formatos, cambios y sumas y otros, mientras se ejecuta una consulta *SQL*. *SQL*Plus* inmediatamente produce el informe formateado con las especificaciones dadas. Para responder a preguntas rápidas que probablemente no volverán a repetirse es una buena aproximación.

Más comunes son, sin embargo, los informes complejos que se necesitan producir periódicamente y que se quiere que se impriman en lugar de salir por la pantalla. Desafortunadamente, cuando se abandona el entorno de trabajo, éste olvida rápidamente todas las instrucciones que se le han dado. Si se estuviera obligado a usar *SQL*Plus* solamente de esta forma interactiva, cada vez que se quisiese obtener el mismo informe habría que introducir todos los mandatos nuevamente.

La alternativa es muy sencilla. Simplemente se escriben los mandatos, línea a línea, en un fichero. El producto *SQL*Plus* puede entonces leer este fichero y ejecutar las órdenes secuencialmente.

Cuando utilizamos el último método, el de introducir en un fichero tanto las órdenes de formateo del *SQL*Plus* como las consultas en *SQL*, la manera de distinguir entre las

[2] Ver Lenguaje de Manipulación de Datos.

órdenes de uno y las órdenes de otro es que las órdenes y las consultas de *SQL* terminan todas con un punto y coma (;) y las de *SQL*Plus* no.

Las órdenes *SQL*Plus* no se almacenan en un área temporal. Actúan directamente sobre el entorno de *SQL*Plus*. Cuando se ejecuta este producto, se selecciona un entorno determinado a través de una lista de parámetros de entorno que queda rellenado con unos valores por defecto. A través de las órdenes *SQL*Plus*, el usuario puede modificar alguno de estos parámetros, el cual permanecerá hasta que sufra una modificación posterior o hasta que abandone el entorno de trabajo. Resumiendo, las órdenes *SQL* afectan a la base de datos, y las órdenes *SQL*Plus* afectan al entorno de trabajo del uso de las órdenes *SQL*.

## 1.2. Identificación ante el *SQL*Plus*

El prefijo *SQL** seguido de un nombre de producto indica que dicho producto es una herramienta de desarrollo de aplicaciones[3], apoyándose en órdenes *SQL* que se ejecutan dentro de un marco específico, en donde cumplen su función.

El *SQL*Plus* permite el acceso directo a la base de datos a través del uso de órdenes de *SQL* y *SQL*Plus*.

Para conectar con la base de datos hay distintas maneras, vamos a ir viendo cada una de ellas y los pasos a seguir desde el indicador del sistema operativo:

1) Introducir sólo la orden a continuación del indicativo del sistema operativo y esperar a que *SQL*Plus* nos pregunte el identificativo de usuario y la contraseña, ver ejemplo 1.1.

2) Especificar el identificativo de usuario y dejar que sea el *SQL*Plus* el que pregunte por la contraseña, ver ejemplo 1.2.

3) Especificar en la línea de órdenes el identificativo y contraseña separados por un barra inclinada (/) en la línea de órdenes del sistema operativo cuando se invoca al producto *SQL*Plus*, ver ejemplo 1.3.

Por supuesto, la contraseña no aparecerá en pantalla cuando nos conectamos como en los ejemplos 1.1 y 1.2.

Para salir del producto *SQL*Plus* y devolver el control al sistema operativo basta con teclear una de las siguientes órdenes `exit` o `quit`, ver ejemplo 1.4.

[3]En versiones posteriores de Oracle existen productos que ya no cumplen esta norma del uso del asterisco.

**Ejemplo** 1.1 Identificación ante el producto

```
> sqlplus

SQL*Plus: Release 9.2.0.1.0 - Production on Tue Apr 13 16:28:13 2004

Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.

Enter user-name: nombre_usuario@juno
Enter password:
SQL>
```

**Ejemplo** 1.2 Identificación ante el producto

```
> sqlplus nombre_usuario@juno

SQL*Plus: Release 9.2.0.1.0 - Production on Tue Apr 13 16:35:01 2004

Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.

Enter password:
SQL>
```

**Ejemplo** 1.3 Identificación ante el producto

```
> sqlplus nombre_usuario@juno/clave

SQL*Plus: Release 9.2.0.1.0 - Production on Wed Apr 14 16:36:12 2004

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
JServer Release 9.2.0.1.0 - Production
SQL>
```

**Ejemplo** 1.4 Salida del producto

```
SQL> exit

Disconnected from Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
JServer Release 9.2.0.1.0 - Production
>
```

Cuando el *DBA* ha decidido que el nombre de usuario sea el mismo del sistema operativo, no hace falta introducir el indicativo de usuario ni la contraseña, ya que tomará por defecto el indicativo de la cuenta del sistema operativo y por contraseña, la misma que esté vigente en ese momento en el sistema. Estos identificativos de los usuarios son de la siguiente forma:

**Formato:**

```
ops$identificativo_S.O.
```

## 1.3. Conexión con otros usuarios

Cualquier usuario de Oracle puede conectarse con otro usuario, sólo tiene que conocer el identificativo del usuario y su contraseña. La conexión se puede hacer desde dentro del propio producto *SQL*Plus*. Si la conexión no se produce debido a cualquier error, Oracle nos desconecta del producto *SQL*Plus* y si queremos continuar trabajando, debemos volver a conectarnos.

**Formato:**

```
conn[ect] [nombre_usuario[/clave]]
```

Hay distintas maneras de conectarse:

1) Una forma de conectarnos es introduciendo el identificador y la clave separados por una barra. En este caso, la clave es visible por pantalla (ver ejemplo 1.5).

2) O bien, sólo introducir el identificativo de usuario y esperar que el producto *SQL*Plus* nos pida la clave (ver ejemplo 1.6).

3) O también, solamente dando la orden y esperando que *SQL*Plus* nos pida identificativo de usuario y clave (ver ejemplo 1.7).

**Ejemplo** 1.5 Conexión con usuario

```
SQL> conn nombre_usuario@juno/clave
Connected.
SQL>
```

**Ejemplo** 1.6 Conexión con usuario

```
SQL> conn nombre_usuario@juno
Introduzca la clave:
Connected.
SQL>
```

**Ejemplo** 1.7 Conexión con usuario

```
SQL> conn
Enter user-name: nombre_usuario@juno
Enter password:
Connected.
SQL>
```

La clave no aparecerá en pantalla en los apartados 2 y 3 anteriores.

Para desconectarnos de una cuenta, pero sin salirnos del *SQL*Plus*, tecleamos la orden `disconnect` (ver ejemplo 1.8). Si queremos volver al producto, tendremos que conectarnos de nuevo a una cuenta de usuario.

**Ejemplo** 1.8 Desconexión con usuario

```
SQL> disconnect
SQL>

Disconnected from Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
JServer Release 9.2.0.1.0 - Production
SQL>
```

## 1.4. Introducción de órdenes

El producto *SQL*Plus* reconoce la primera palabra de una sentencia *SQL* y pregunta por líneas adicionales de instrucciones numerando las líneas.

Las órdenes *SQL* se situan en el *buffer* por omisión del *SQL*. Por eso pueden ser editadas y reejecutadas sin tener que introducir nuevamente toda la sentencia *SQL*.

El formato de las órdenes *SQL* es libre, no se distingue entre mayúsculas y minúsculas, pudiéndose incluso partir palabras en líneas consecutivas. Esto permite la identación para permitir una fácil lectura.

Las órdenes *SQL* se ejecutan cuando se detecta:

- Un punto y coma (;) al final de la sentencia *SQL*.
- Una barra (/) en la línea de órdenes.
- Una línea en blanco y después introduciendo `r[un]` en la línea de órdenes *SQL*.

El punto y coma no se introduce en el *buffer* de edición y no puede ser editado.

El *SQL*Plus* reconoce la primera palabra de una orden de formateado y no pregunta nada más con líneas adicionales, en su lugar se establecerá el parámetro especificado.

Las órdenes de formateado del *SQL*Plus* no se sitúan así mismo en el *buffer* de edición.

## 1.5. Las órdenes de edición

Cuando una sentencia *SQL* se introduce en el entorno *SQL*Plus*, se sitúa en un *buffer*. Mediante unas órdenes, el *SQL*Plus* permitirá su modificación y posterior ejecución. Las órdenes de edición del producto *SQL*Plus* se pueden ver en la tabla 1.1.

Tenemos que tener en cuenta que en el *buffer* solamente se introduce la última consulta ejecutada.

| **Orden** | **Uso** |
|---|---|
| `a[ppend]` texto | Añade «texto» al final de la línea actual. |
| `c[hange]` /viejo/nuevo | Cambia «viejo» por «nuevo» en la línea actual. |
| `c[hange]` /texto | Borra «texto» de la línea actual. |
| `cl[ear] buff[er]` | Borra todas las líneas del *buffer*. |
| `cl[ear] scr[een]` | Borra la pantalla. |
| `del` | Borra la línea actual. |
| `i[nput]` | Añade líneas al *buffer*. |
| `i[nput]` texto | Añade una línea con el «texto» después de la línea actual. |
| `l[ist]` | Lista todas las líneas del *buffer*. |
| `[l[ist]] n` ó `n` | Lista la línea `n` del *buffer* haciendo que dicha línea pase a ser la línea actual del editor. |
| `l[ist] *` | Lista la línea actual. |
| `l[ist] n *` | Lista desde la línea `n` hasta la actual. |
| `r[un]` | Lista y ejecuta la orden del *buffer*. |
| / | Ejecuta el *buffer*. |

Tabla 1.1: Órdenes de edición

Las órdenes de edición afectan: a la línea actual o a la línea siguiente a la actual. Junto a los números de línea aparece un asterisco (*) que indica cuál es la línea actual, o

también para indicar, en caso de error, el lugar donde el intérprete ha encontrado el error sintáctico, ver ejemplo 1.12.

Veamos algunos ejemplos de cómo manejar estas las órdenes:

- Mostramos el contenido del *buffer*, donde cada línea está numerada y la línea actual además está marcada con un asterisco, ver ejemplo 1.9.
- Añadimos una línea al final de la línea actual y pasa a ser la nueva línea actual , ver ejemplo 1.10.
- Borrar la línea actual y cambiar el nombre de la tabla , ver ejemplo 1.11.

**Ejemplo** 1.9 Mostrar el contenido del *buffer*

```
SQL> l
  1  select *
  2* from proveedores
SQL>
```

**Ejemplo** 1.10 Añadir una línea a la orden del *buffer*

```
SQL> i where prv_num > 2
SQL> l
  1  select *
  2  from proveedores
  3* where prv_num > 2
SQL>
```

**Ejemplo** 1.11 Borrar línea y cambiar texto

```
SQL> del
SQL> l
  1  select *
  2* from proveedores
SQL> c/proveedores/tiendas
  2* from tiendas
SQL>
```

**Ejemplo** 1.12 Manera de mostrar un error

```
SQL> select art_num
  2  from proveedores;
select art_num
       *
ERROR en linea 1:
ORA-00904: "ART_NUM": identificador no valido

SQL>
```

## 1.6. Las órdenes de ficheros

El *SQL*Plus* también tiene órdenes para manejar ficheros. Podemos guardar el contenido del *buffer* en un fichero del directorio actual, teniendo por defecto la extensión `sql`. Asimismo, podemos recuperar un fichero del subdirectorio al *buffer*. Ver la tabla 1.2.

| **Orden** | **Uso** |
|---|---|
| `!`orden shell | Permite, sin salirse del *SQL*Plus*, ejecutar ciertas órdenes del shell. |
| `host` fichero | Permite, sin salirse del *SQL*Plus*, ejecutar proceso batch. |
| `edit` [fichero[.ext]] | Invoca al editor por defecto con el contenido del fichero o del *buffer*. |
| `get` fichero | Carga un fichero del sistema operativo en el *buffer*. |
| `save` fichero | Guarda el *buffer* en un fichero especificado del sistema operativo. Por defecto la extensión es `sql`. |
| `@`fichero[.ext] | Ejecuta el contenido del fichero. Por defecto la extensión es `sql`. |
| `spool` fichero | Almacena el resultado de una consulta en un fichero del sistema operativo con extensión `lst`. |
| `spool out\|off` | Manda el fichero anterior a la impresora por defecto, o no; en ambos casos se cierra el fichero. |
| `start` fichero | Ejecuta el contenido del fichero de órdenes especificado. Por defecto la extensión es `sql`. |

Tabla 1.2: Órdenes de ficheros

Veamos el ejemplo 1.13 donde combinaremos una serie de órdenes de ficheros:

1) Listaremos el contenido del *buffer*.

2) Dicho contenido se salvará en un fichero del sistema operativo, «fich-con», y cuya extensión por defecto será «sql».

3) Se ejecutará y se guardará el resultado de dicha consulta en un fichero del sistema operativo en el subdirectorio por defecto, denominado «resul.lst».

**Ejemplo** 1.13 Salvar el *buffer* y guardar la ejecución en un fichero

```
SQL> l
  1  select *
  2* from proveedores
SQL> save fich-con
Creado archivo fich-con.sql
SQL> spool resul
SQL> @fich-con

   PRV_NUM PRV_NOM
---------- -------------------------
         1 catio electronic
         2 estilograficas reunidas
         3 mecanica de precision
         4 sanjita
         5 electrolamp

5 filas seleccionadas.

SQL> spool off
SQL> !less resul.lst

SQL>
```

## 1.7. El editor del Sistema Operativo

Podemos introducir órdenes de *SQL* y/o también órdenes de *SQL*Plus* en un fichero y para ello usaremos un editor del sistema operativo[4]. También podemos definirnos nuestro propio editor con la orden `define`.

**Formato:**

```
DEF[INE]_EDITOR = nombre_editor
```

donde `nombre_editor` es el nombre de cualquier editor disponible en el sistema operativo.

El ejemplo 1.14 nos muestra como definirnos el editor `jed` como editor por defecto y que sea el que se invoque cuando tecleamos la orden `edit`.

[4]Podemos usar cualquier editor, o bien, el que sea en ese momento el editor por defecto del sistema operativo.

**Ejemplo** 1.14 Definición de editor

```
SQL> define_editor = jed
SQL>
```

Podemos editar con la orden `edit` cualquier fichero cuya extensión por defecto sea `sql`, o bien el contenido del *buffer* (`afiedt.buf`). Si no especificamos nada se editará, por defecto, el contenido del *buffer* y además, se guardará en un fichero denominado `afiedt.buf`.

**Formato:**

```
ED[IT] [nombre_fichero|afiedt]
```

En los ejemplos 1.15 y 1.16 observamos como se editar el fichero `consulta.sql` y el contenido del *buffer* `afiedt.buf`.

**Ejemplo** 1.15 Edición de fichero

```
SQL> edit consulta
```

**Ejemplo** 1.16 Edición del *buffer*

```
SQL> edit afiedt
```

o bien, simplemente:

```
SQL> edit
```

Cuando tenemos un fichero con órdenes *SQL*Plus* y sentencias *SQL*, puede ser recomendable introducir comentarios que nos permitan aclarar lo que hace la orden de cada línea, ver la tabla 1.3.

| **Orden** | **Uso** |
|---|---|
| `rem[ark]` texto | Se coloca al comienzo de la línea y sólo es válido para esa línea. |
| `/*` texto `*/` | Se coloca al comienzo del comentario y se cierra al final del texto. |

Tabla 1.3: Órdenes de comentarios

## 1.8. Ayudas

Existen dos órdenes que nos ayudan de distinta manera a la hora de manejar una base de datos. Son las órdenes `describe` y `help`.

- `describe`: Esta orden nos muestra información de las columnas de las tablas, de las vistas, o de los sinónimos[5], indicándonos número de columnas, nombre de las columnas y restricciones que deben cumplir las columnas (ver ejemplo 1.17).

  **Formato:**

  ```
  DESC[RIBE] nombre_objeto
  ```

  donde `nombre_objeto` es el nombre del objeto a describir.

**Ejemplo** 1.17 Descripción de un objeto

```
SQL> describe proveedores
 Nombre                                    ?Nulo?   Tipo
 ----------------------------------------- -------- -------------
 PRV_NUM                                   NOT NULL NUMBER(38)
 PRV_NOM                                   NOT NULL VARCHAR2(25)
```

Esta orden nos devuelve una tabla[6] que contiene tres columnas. El significado de cada una de ellas se muestra en la tabla 1.4.

| Nombre | Significado |
|---|---|
| Nombre | Muestra los nombres de las columnas en el mismo orden que se dieron en el momento de la creación de la tabla. |
| Nulo? | Nos dice si las correspondientes columnas pueden contener valores nulos o no. |
| Tipo | Muestra el tipo de dato asignado a las columnas en el momento de su creación. |

Tabla 1.4: Información de la orden `describe`

- `help`: Esta orden nos muestra información acerca de las órdenes de *SQL* y/o *SQL*Plus* especificadas[7] en `nombre_orden`.

  **Formato:**

  ```
  HELP [nombre_orden]
  ```

[5] Las tablas, las vistas y los sinónimos son objetos de la BD que se crean con el lenguaje DDL.

[6] Estamos manejando un *SGBD* relacional, por tanto toda la información que nos devuelve el sistema viene en forma de tabla.

[7] La respuesta de Oracle viene dada en inglés.

**Ejemplo** 1.18 Ayuda de una orden

```
SQL> help save

 SAVE
 ----

 Saves the contents of the SQL buffer in a host operating system
 script.
 In iSQL*Plus, click the Save Script button to save the Input area
 contents to a script.

 SAV[E] file_name[.ext] [CRE[ATE] | REP[LACE] | APP[END]]

 Not available in iSQL*Plus
```

Esta orden permite a los usuarios obtener una documentación de referencia del Oracle mientras se está trabajando con el *SQL*Plus*. Si se introduce únicamente una porción de la orden se mostrarán los formatos de todas las opciones de esa orden (ver ejemplo 1.18).

## 1.9. Formateado de consultas

Las órdenes de formateo se utilizan para formatear los resultados procedentes de una consulta *SQL* y producir informes simples.

Existen varias órdenes de formateo, veremos las más interesantes.

### 1.9.1. Cabeceras y pies de páginas

Existen dos órdenes que nos permiten colocar cabeceras y pies de páginas a los informes que podemos obtener como resultado de una consulta.

- `btitle`: Esta orden coloca un título, que puede estar compuesto de varias líneas, en la parte inferior de cada página del informe.

  **Formato:**

  ```
  BTI[TLE] [espc. [var | text] ...] | [OFF|ON]
  ```

- `ttitle`: Esta orden coloca un título en la parte superior de cada página, numera las páginas y coloca la fecha actual.

**Formato:**

```
TTI[TLE] [espc. [var | text] ...] | [off|on]
```

Las opciones de estas órdenes son:

**espc** Esta opción representa una o más de las cláusulas usadas para colocar y formatear el «texto», ver la tabla 1.5.

**text** Es el texto que queremos que aparezca en la cabecera o a pie de página y ha de colocarse entre comillas simples.

**var** Esta opción representa una variable del usuario o del sistema, ver la tabla 1.6.

**off|on** Por defecto la opción es `off`.

| Espc. | Uso |
|---|---|
| `s[kip] [n]` | Salta al comienzo de una nueva línea `n` veces. |
| `le[ft]` | Alinea el texto por la izquierda. |
| `ce[nter]` | Centra el texto. |
| `r[ight]` | Alinea el texto por la derecha. |

Tabla 1.5: Especificaciones para las órdenes `btitle` y `ttitle`

| Var. | Uso |
|---|---|
| `SQL.LNO` | Número de la línea actual. |
| `SQL.PNO` | Número de la página actual. |
| `SQL.RELEASE` | Número de versión actual de Oracle. |
| `SQL.SQLCODE` | Código de error actual. |
| `SQL.USER` | Identificativo del usuario. |

Tabla 1.6: Variables para las órdenes `btitle` y `ttitle`

En el ejemplo 1.19 hemos realizado una consulta y el resultado aparecera formateado de la siguiente manera: una cabecera centrada que nos indicará que corresponde con el informe del día y a pie de página el nombre de la cuenta de Oracle desde donde se está ejecutando dicha consulta.

**Ejemplo** 1.19 Título y pie de página

```
SQL> ttitle center 'Informe del dia'
SQL> btitle skip SQL.USER skip 'UCA'
SQL> select *
  2  from proveedores;

                                Informe del dia
   PRV_NUM PRV_NOM
---------- -------------------------
         1 catio electronic
         2 estilograficas reunidas
         3 mecanica de precision
         4 sanjita
         5 electrolamp


GADESCHI
UCA

5 filas seleccionadas.
```

## 1.9.2. Formateado de columnas

La orden `column` permite cambiar las cabeceras y formatos de cualquier columna de una sentencia `select`.

**Formato:**

```
COL[UMN] [{nom_columna|expresión} [opción ...]]
```

Donde «expresión» corresponde a columnas virtuales obtenidas por cálculo, función o expresión matemática[8].

En la tabla 1.7 se muestran algunas de las principales opciones de esta orden.

La opción `format`, a su vez, tiene una sintaxis propia para poder especificar los formatos, ver la tabla 1.8.

La orden `column` no cambia la definición de una columna en la tabla, sino que la modifica para obtener un informe con un formato dado. Se modifica la salida por pantalla o impresora de dicha columna.

En el ejemplo 1.20 se puede observar cómo formatear columnas.

1) Le asignamos un alias a la columna *art_nom* para luego poder utilizarlo con otras opciones de `column`.

[8] Ver Lenguaje de Manipulación de Datos.

2) Reducimos el ancho de la columna *art_nom* a 6 caracteres.

3) Por último, queremos que la columna *art_pv* tenga 5 dígitos solamente.

| Opción | Uso |
|---|---|
| `ali[as]` nombre | Asigna un alias a una columna que se podrá utilizar en las órdenes de columna. |
| `cle[ar]` | Elimina los valores de los atributos asignados a una columna. |
| `for[mat]` formato | Especifica los formatos de pantalla de las columnas. |
| `hea[ding]` texto | Define las cabeceras de las columnas. |
| `jus[tify]`{l\|c\|r} | Alinea las cabeceras de las columnas a la izquierda, al centro o a la derecha. Si no se usa esta orden, las columnas definidas como `number` se alinean a la derecha y los otros tipos de columna a la izquierda. |

Tabla 1.7: Opciones para la orden `column`

| Opción | Ejemplo | Uso |
|---|---|---|
| 9 | 999 | Determina el números de dígitos enteros de la columna especificada. |
| $ | $999 | Muestra un signo dolar al comienzo de cada fila de la columna indicada. |
| `An` | `A6` | Fija la anchura en caracteres de la columna especificada. |

Tabla 1.8: Opciones para la orden `format`

## 1.10. Variables del sistema

Las variables del sistema son como opciones de programas más que variables. Cada variable del sistema controla algunos aspectos de las operaciones del *SQL*Plus*. Las variables del sistema se establecen con la orden `set` y se comprueba su contenido con la orden `show`.

- `set`: establece un aspecto del entorno del *SQL*Plus* para la sesión actual. Para colocar una variable del sistema, se usa esta orden `set` seguido por el nombre de una variable del sistema y un valor para esa variable.

  **Formato:**

  ```
  SET variable_sistema valor
  ```

Esta orden tiene muchas opciones, algunas de ellas las podemos ver en la tabla 1.9[9]. Y en el ejemplo 1.21 observamos la manera de establecer un valor para estas opciones.

a) Establecer un máximo de 20 registros por página.

b) Muestra por pantalla las órdenes cuando las ejecutamos desde un fichero.

c) Desactiva la pausa.

**Ejemplo** 1.20 Formateo de columnas

```
SQL> column art_nom alias nombre
SQL> column nombre format a6
SQL> column art_pv format 99999
SQL> select *
  2  from articulos
  3  where art_num < 4;

   ART_NUM ART_NO   ART_PESO ART_COL     ART_PC ART_PV    ART_PRV
---------- ------ ---------- ------- ---------- ------ ----------
         1 impres        150 rojo           400    580          4
           ora

         2 calcul        150 negro         4000   4700          1
           adora

         3 calend        100 blanco         420    600          4
           ario

3 filas seleccionadas.
```

**Ejemplo** 1.21 Establecer variables

```
SQL> set pages 20
SQL> set echo on
SQL> set pause off
SQL>
```

- `show`: nos muestra el valor de una variable del sistema. Visualiza el valor de una característica de la orden `set`, o bien todas (`all`) las características de `set`, o de otras órdenes de *SQL*Plus*. Podemos colocar más de una variable del sistema después de la orden `show`, y cada una se visualiza en una línea separada.

[9] La opción subrayada es la opción por defecto.

**Formato:**

```
SHO[W] variable_sistema [,variable_sistema]
```

Esta orden también nos muestra los valores de otras variables como podemos ver en la tabla 1.10.

En el ejemplo 1.22 vemos la utilidad de esta orden. Primero nos muestra el número de registros que está establecido por páginas y después el nombre del esquema desde donde estamos trabajando.

**Ejemplo** 1.22 Valores de variables

```
SQL> show pages
pagesize 20
SQL> show user
USER es "GADESCHI"
SQL>
```

## 1.11. Entrada y salida de datos

Todo lenguaje de programación tiene un apartado dedicado a la entrada de datos tanto por teclado como por pantalla y lo mismo ocurre con la salida de los datos obtenidos como resultado de la ejecución de programas. El producto *SQL*Plus* también dispone de un conjunto de órdenes para regular la entrada y salida de datos.

### 1.11.1. Definición de variables

La definición de variables se realiza con la orden `define` que ya hemos usado para predefinir un editor de texto. Esta orden permite a los usuarios especificar variables y asignarles un valor de tipo carácter. O bien, listar el valor y el tipo de variable de una o de varias variables.

| Variable | Valor | Uso |
|---|---|---|
| `auto[commit]` | `off`\|`on`\|`imm[ediate]` | Controla cuándo Oracle realiza los cambios pendientes en la base de datos. `on` realiza los cambios pendientes después de cada orden *SQL*. `imm[ediate]` funciona igual que la opción `on`. |
| `define` | `&`\|`c`\|`off`\|`on` | Conjunto de caracteres usados para prefijar la sustitución de variables a `c`. `on` u `off` controla si se hace o no la sustitución de las variables. Ver el ejemplo 1.28. |

*continúa en la siguiente página*

*continuación de la página anterior*

| Variable | Valor | Uso |
|---|---|---|
| `echo` | `off`\|`on` | `on` nos mostraría por pantalla las órdenes cuando ejecutamos ordenes del *SQL*Plus* desde un fichero de arranque o bien desde el prompt del producto. `off` hace que este producto los ejecute sin visualizarlos. |
| `feed[back]` | 6\|n\|`off`\|`on` | Informa del nº de registros que devuelve una consulta. `off`\|`on` activa o desactiva la opción. |
| `lin[esize]` | 80\|n | Establece el número total de caracteres por línea. |
| `long` | 80\|n | Establece la anchura máxima en caracteres para mostrar y copiar valores `long`. |
| `newp[age]` | 1\|n | Estable el nº de líneas en blanco de cabecera y de pie de página. |
| `pages[ize]` | 14\|n | Establece el número de líneas por páginas. |
| `pau[se]` | `off`\|`on`\|`texto` | `on` hace que *SQL*Plus* espere a que se presione *return* después de visualizar cada página de la salida. `off` hace que no haya pausa en la visualización de páginas. «texto» es el mensaje que se visualizará en la parte inferior de la pantalla mientras se espera la pulsación de la tecla *return*. |
| `show[mode]` | `off`\|`on` | `on` hace que *SQL*Plus* visualice la disposición antigua y la nueva de una característica `set` y su valor cuando se cambia. `off` detiene la visualización de ambas. |
| `spa[ce]` | 1\|n | Número de espacios entre columnas. |
| `sqlp[rompt]` | *SQL*>\|texto | Sustituye el indicativo por defecto por «texto». |
| `ti[me]` | `off`\|`on` | `on` nos muestra la hora después de cada orden, `off` lo desactiva. |
| `verify` | `off`\|`on` | Controla que aparezca o no el valor anterior y posterior de la variable cuando hacemos una sustitución de variables con el operador &. |

Tabla 1.9: Variables del sistema

La sintaxis es la siguiente:

**Formato:**

```
DEF[INE] [nom_variable]|[nom_variable = 'texto']
```

**nom_variable** Es el nombre que le vamos a dar a la variable.

| Variable | Uso |
|---|---|
| `all` | Muestra el conjunto de todas las opciones. |
| `bti[tle]` | Muestra el título actual de pie de página. |
| `lno` | Nos indica el nº de la línea actual. |
| `parameters` | Muestra el valor actual de un o más parámetros de inicialización. Se puede poner una cadena de caracteres después de la orden para que nos muestre un subconjunto de parámetros cuyos nombres incluya esa cadena. |
| `pno` | Muestra el nº de la página actual. |
| `rel[ease]` | Muestra la versión actual de Oracle. |
| `spoo[l]` | Nos indica si la salida va a un fichero de `spool`. |
| `tti[tle]` | Muestra el título de cabecera de página actual. |
| `user` | Nos muestra el nombre del usuario bajo el cual estamos trabajando en ese momento. |

Tabla 1.10: Valor de variables

**texto** La variable queda definida con «texto».

En los ejemplos 1.23 y 1.24 vemos como asignarle el valor 20 de tipo caracter a la variable «var1» y como visualizar el contenido de dicha variable.

**Ejemplo** 1.23 — Asignar valor a variable

```
SQL> define var1=20
SQL>
```

**Ejemplo** 1.24 — Visualizar el valor de variable

```
SQL> define var1
DEFINE VAR1            = "20" (CHAR)
SQL>
```

Podemos borrar la definición de una variable con la orden `undefine` como sigue:

**Formato:**

```
UNDEF[INE] nom_variable
```

En el ejemplo 1.25 vemos cómo borrar la definición de la variable «var1» definida anteriormente y comprobar que no tiene valor.

**Ejemplo** 1.25 Eliminar valor de variable

```
SQL> undefine var1
SQL> define var1
SP2-0135: el simbolo var1 es UNDEFINED (INDEFINIDO)
SQL>
```

### 1.11.2. Uso de operadores con variables

Las variables también se pueden utilizar sin necesidad de definirlas previamente como hemos visto en el apartado anterior. Para ello utilizamos los operadores *ampersand* (&) y doble *ampersand* (&&).

Ambos operadores (& y &&) actuan de la misma manera sobre una variable aunque la semántica exacta es diferente dependiendo de si la variable tratada está definida previamente o no.

**Formato:**

```
&[&]nombre_variable
```

- Variable no definida previamente:
  - &var1: En una sentencia *SQL* es posible dejar indeterminado el nombre de una columna o de una tabla, y dejar que *SQL* nos solicite dicho valor antes de ejecutar dicha sentencia. Si la variable del sistema `verify`[10] está activada, *SQL* nos mostrará el valor anterior y el actual suministrado por el usuario. Para solicitar la sustitución de la variable por el valor dado por el usuario se emplea el operador *ampersand* (&) delante de la variable en la consulta *SQL*.

    En el ejemplo 1.26 podemos observar cómo se utiliza dicho operador y cómo le damos un valor a la variable.

    Dicho operador (&) está definido por defecto, pero podemos definirlo con cualquier otro caracter usando la orden de *SQL*Plus* `set` como podemos observar en el ejemplo 1.28.
  - &&var1: El sistema pide al usuario un valor para la variable «var1» que es colocado en el lugar de la instrucción donde aparece, quedando así definida la variable con ese valor a partir de ese momento.

    El ejemplo 1.31 nos muestra como se realiza esto. En principio, suponemos que la variable del sistema `verify` está activada. El sistema nos pide un valor para la variable «var1» y nos muestra el resultado de la consulta. Cuando damos la orden `run` ya no nos vuelve a pedir que introduzcamos el valor para dicha variable. Con el doble *ampersand* una vez que hemos introducido un valor ya

[10] Ver tabla 1.9.

queda fijado para toda la sesión a menos que utilicemos la orden `undefine` para redefinir el valor de la variable.

Tenemos que tener cuidado a la hora de definir variable usando el doble *ampersand* ya que la última orden del ejemplo 1.31 nos borrará todos los registros de la tabla mostrados anteriormente y no nos preguntará el nombre de la tabla ya que tomará como valor de la variable «var1» el que ya le habiamos dado antes.

- Variable definida previamente: Los dos operadores tienen el mismo funcionamiento en este caso. No se solicita al usuario ningún valor, usando el que tenga la variable actualmente.

Las variables pueden ser usadas en interactivo, o sea, en una consulta realizada desde el indicativo del *SQL*Plus*, o bien, en una consulta contenida en un fichero.

**Ejemplo** 1.26 Uso de variables con &

```
SQL> select *
  2  from &var1;
Introduzca un valor para var1: proveedores
antiguo   2: from &var1
nuevo   2: from proveedores

   PRV_NUM PRV_NOM
---------- -------------------------
         1 catio electronic
         2 estilograficas reunidas
         3 mecanica de precision
         4 sanjita
         5 electrolamp


5 filas seleccionadas.
```

## 1.11.3. Uso de variables con distintas órdenes

El operador *ampersand* (&) se puede utilizar para recibir distintos argumentos cuando utilizamos algunas de las órdenes de ficheros. Cuando usamos las órdenes `start` y @ es posible pasarles distintos argumentos al fichero que queremos ejecutar.

Tenemos dos formatos distintos para realizar la misma orden, como podemos ver a continuación:

**Formato:**

```
START nombre_fichero arg_1 arg_2

@nombre_fichero arg_1 arg_2
```

También podemos utilizar este operador con órdenes de *SQL*Plus* como podemos observar en el ejemplo 1.27.

En el ejemplo 1.29, &1 se va a sustituir por el valor de arg_1, mientras que &2 se va a sustituir por el valor de arg_2. En este caso los nombres de las variables a continuación del operador *ampersand* (&) deben ser númericos (1 y 2, en este caso). La sustitución se hace en el orden: el primer argumento va a la primera variable, el segundo a la segunda, y así sucesivamente.

**Ejemplo** 1.27 Uso de variables con & (cont.)

```
SQL> describe &tab
Introduzca un valor para tab: proveedores
 Nombre                ?Nulo?   Tipo
 --------------------- -------- ----------------------------
 PRV_NUM               NOT NULL NUMBER(38)
 PRV_NOM               NOT NULL VARCHAR2(25)
```

**Ejemplo** 1.28 Redefinición de operador

```
SQL> set define #
SQL> select *
  2  from #var1;
Introduzca un valor para var1: proveedores
antiguo   2: from #var1
nuevo   2: from proveedores

   PRV_NUM PRV_NOM
---------- -------------------------
         1 catio electronic
         2 estilograficas reunidas
         3 mecanica de precision
         4 sanjita
         5 electrolamp


5 filas seleccionadas.
```

**Ejemplo** 1.29 Contenido del fichero nombre_fichero

```
SELECT &1
FROM &2;
```

En el ejemplo 1.30 vemos cómo podemos utilizar el operador con órdenes de *SQL*Plus*. En este ejemplo nos decribirá la definición de la tabla que le indiquemos.

**Ejemplo** 1.30 Uso del operador & con órdenes de *SQL*Plus*

```
SQL> describe &var
Introduzca un valor para var: proveedores
 Nombre                ?Nulo?   Tipo
 -------------------- -------- --------------------
 PRV_NUM               NOT NULL NUMBER(38)
 PRV_NOM               NOT NULL VARCHAR2(25)
```

## 1.12. Ficheros de órdenes

Los ficheros de órdenes son ficheros del sistema operativo que contienen órdenes del producto *SQL*Plus*.

Cuando el usuario entra en el producto *SQL*Plus*, Oracle comprueba si existe en el subdirectorio actual un fichero especial llamado `login.sql`. Si existe este fichero, lo ejecuta, realizando las órdenes que tenga en su interior. Estas se ejecutan secuencialmente y en el mismo orden en que aparecen en dicho fichero.

Se pueden colocar en este fichero cualquier orden propia del producto *SQL*Plus* así como sentencias *SQL*; todas ellas se ejecutarán antes que el producto *SQL*Plus* proporcione el indicativo *SQL>*. Ésta puede ser una forma conveniente de crearse un entorno individual dentro del producto *SQL*Plus*.

El ejemplo 1.32 nos muestra un fichero de órdenes.

**Ejemplo** 1.31 Uso de variables con & y &&

```
SQL> select *
  2  from &&var1;
Introduzca un valor para var1: proveedores
antiguo   2: from &&var1
nuevo   2: from proveedores

   PRV_NUM PRV_NOM
---------- -------------------------
         1 catio electronic
         2 estilograficas reunidas
         3 mecanica de precision
         4 sanjita
         5 electrolamp


5 filas seleccionadas.

SQL> run
  1  select *
  2* from &&var1
antiguo   2: from &&var1
nuevo   2: from proveedores

   PRV_NUM PRV_NOM
---------- -------------------------
         1 catio electronic
         2 estilograficas reunidas
         3 mecanica de precision
         4 sanjita
         5 electrolamp


5 filas seleccionadas.

SQL> delete from &var1;
SQL>
```

**Ejemplo** 1.32 Fichero de órdenes: `login.sql`

```
REM nos muestra el nombre del usuario
 SHOW user

REM establecemos a 16 el número máximo de registros por página
 SET pages 16

REM nos mostrará la versión del Oracle
 SHOW release

REM establecemos el editor por defecto
 DEFINE_EDITOR = jed

/* activamos la pausa y hacemos que nos muestre el mensaje
   'pulsa return ' después de cada página */
 SET PAUSE ON
 SET PAUSE 'pulsa return '

REM nos mostrará la fecha de hoy con formato por defecto
 SELECT sysdate
 FROM dual;
```

## 1.13. Resumen de órdenes de *SQL*Plus*

**Ejemplo** 1.33 Lista de todas las órdenes del `SQL*Plus`

```
SQL> help index

Enter Help [topic] for help.

 @                  DISCONNECT         RESERVED WORDS(SQL)
 @@                 EDIT               RESERVED WORDS (PL/SQL)
 /                  EXECUTE            RUN
 ACCEPT             EXIT               SAVE
 APPEND             GET                SET
 ARCHIVE LOG        HELP               SHOW
 ATTRIBUTE          HOST               SHUTDOWN
 BREAK              INPUT              SPOOL
 BTITLE             LIST               SQLPLUS
 CHANGE             PASSWORD           START
 CLEAR              PAUSE              STARTUP
 COLUMN             PRINT              STORE
 COMPUTE            PROMPT             TIMING
 CONNECT            QUIT               TTITLE
 COPY               RECOVER            UNDEFINE
 DEFINE             REMARK             VARIABLE
 DEL                REPFOOTER          WHENEVER OSERROR
 DESCRIBE           REPHEADER          WHENEVER SQLERROR
```

Parte 2

# Lenguaje de Manipulación de Datos

# Capítulo 2

# Manipulación de datos

Este capítulo estudia las cuatro órdenes para manipular la información de la Base de Datos. Podemos modificar la tabla resultante de una consulta para obtener un mejor resultado como puede ser eliminación de tuplas repetidas, clasificación de las tuplas o identificar a las columnas por un nuevo nombre.

## 2.1. Introducción

El lenguaje de manipulación de datos (*DML*) se utiliza para realizar las operaciones de mantenimiento y consulta de una base de datos. Está formado por cuatro órdenes y su explicación la podemos ver en la tabla 2.1.

| **Orden** | **Operación** |
|---|---|
| `select` | Muestra el contenido de una tabla. |
| `insert` | Introduce registros en una tabla. |
| `update` | Actualiza un atributo o campo de una tabla. |
| `delete` | Elimina uno o varios registros de una tabla. |

Tabla 2.1: Órdenes del Lenguaje de Manipulación de Datos

## 2.2. La orden `select`

La orden `select` se utiliza, principalmente, para realizar cualquier consulta a una o a varias tablas o vistas de nuestro o de otros esquemas siempre que tengamos privilegio para ello.

Esta orden no sólo es la que más se emplea en el *SQL*, sino que es la más versátil. Puede tomar un aspecto realmente complejo, incorporar varias consultas anidadas, afectar a múltiples tablas, usar funciones, etc.

Cualquier consulta a una o a varias tablas de la base de datos, deberá constar, como mínimo, de una sentencia formada por la orden `select` y la cláusula `from`.

La sintaxis de esta orden es bastante compleja. Nosotros iremos estudiando cada una de las cláusulas paso a paso. La estructura general de esta orden es la siguiente:

**Formato:**

```
SELECT nombre de las columnas
FROM nombre de las tablas
[WHERE condiciones de selección de tuplas]
[GROUP BY nombre de columnas]
[HAVING condiciones de selección de grupos]
[ORDER BY números o nombre de columnas];
```

### 2.2.1. Proyección de una tabla

La operación de proyección es una operación del álgebra relacional[1] que consiste en seleccionar una o varias columnas o atributos de una o varias tablas, indicando el orden de aparición de izquierda a derecha de dichas columnas.

La lista de las columnas que queremos proyectar aparecerán después de la orden `select` separadas por comas y en el orden en que queremos que aparezcan en el resultado. Mientras que en la cláusula `from` indicaremos los nombres de las tablas cuyas columnas queremos seleccionar, separadas también por comas (ver ejemplo 2.1).

Un caso especial es cuando queremos proyectar todas las columnas de una tabla dada. Para ello utilizamos un asterisco (*) en vez de los nombres de las columnas, como podemos observar en el ejemplo 2.2.

[1] Corresponde con la operación de proyección del álgebra relacional representada por la letra griega Π.

**Ejemplo** 2.1 Proyección de datos de los proveedores

```
SQL> select prv_num, prv_nom
  2  from proveedores;

   PRV_NUM PRV_NOM
---------- -------------------------
         1 catio electronic
         2 estilograficas reunidas
         3 mecanica de precision
         4 sanjita
         5 electrolamp

5 filas seleccionadas.
```

**Ejemplo** 2.2 Proyección de todos los datos de los proveedores

```
SQL> select *
  2  from proveedores;

   PRV_NUM PRV_NOM
---------- -------------------------
         1 catio electronic
         2 estilograficas reunidas
         3 mecanica de precision
         4 sanjita
         5 electrolamp

5 filas seleccionadas.
```

En esta consulta, el orden de aparición de las columnas, de izquierda a derecha, es el mismo orden en que fueron creadas las columnas, cuando se creó la tabla con la orden `create table` del lenguaje DDL. El resultado es otra tabla, obtenida mediante la proyección de todas las columnas de la tabla de partida.

### 2.2.2. Selección de filas de una tabla

Una tabla está formada por un conjunto de filas, tuplas o registros. Cuando realizamos una consulta, podemos proyectar una serie de columnas en un orden específico, pero también podemos seleccionar un conjunto de tuplas y no todas las que forman la tabla[2]. Este subconjunto está formado por las tuplas que cumplan unas determinadas condiciones.

[2]Esta operación se corresponde con la operación de selección ($\sigma$) del álgebra relacional.

Las condiciones de selección pueden tener criterios diferentes y se especifican en la cláusula `where`. Esta cláusula es opcional y se coloca a continuación de la cláusula `from`.

Esta cláusula tiene el siguiente formato:

**Formato:**

```
SELECT colum1, colum3, colum4
FROM nom_tabla
WHERE criterio de selección de filas;
```

El criterio en una cláusula `where` se establece mediante una expresión lógica compuesta por una serie de condiciones, cada una de las cuales toma el valor «verdadero» o «falso», combinado con los operadores lógicos `not` (negación), `and` (*Y* lógico) y `or` (*O* lógico). Si una fila cumple la condición, hará que el valor de la expresión lógica sea «verdadero», y formará parte del resultado.

Las condiciones pueden utilizar los siguientes operadores:

- Comparación con un valor[3] ( `=, <>, !=, <, >, <=, >=` ).
- Comparación con un intervalo de valores (`between`).
- Comparación con una lista de valores (`in`).
- Comparación con un patrón (`like`).
- Test sobre la indeterminación de un valor (`is null`).
- Test «todos» o «al menos uno» (`all`, `any`).
- Test de existencia (`exists`).

#### 2.2.2.1. Uso de los operadores lógicos

Una expresión lógica puede contener un número cualquiera de operadores lógicos.

- *Operador* `and`: Para que una fila sea seleccionada debe cumplirse a la vez las dos condiciones, que se unen mediante dicho operador.
- *Operador* `or`: La fila se selecciona si se cumple, por lo menos, una de las condiciones unidas por este operador.

[3]Los operadores de comparación `<>y !=` realizan las mismas funciones, por lo tanto podemos utiliza uno y otro indistintamente.

- *Prioridad de los operadores lógicos*: Si el número de expresiones lógicas es importante, el resultado depende del orden en que se efectúen las operaciones elementales. Este orden se define por la prioridad de los operadores. El *SQL* efectúa primero las comparaciones, después las negaciones con `not`, después los *Y* logicos con `and` y, finalmente, los *O* logicos con `or`. Conviene, para mayor seguridad, emplear paréntesis que agrupen a las expresiones.

- *Operador* `not`: este operador utilizado junto con otro operador invierte el valor del resultado.

#### 2.2.2.2. Selección por comparación con un valor

- *Comparación con una constante*: El caso más sencillo consiste en comparar el valor de una columna con una constante. La constante debe ser compatible con el tipo de columna considerada. El *SQL* admite dos tipos de constantes:

  - Las constantes de tipo cadena de caracteres.
  - Las constantes de tipo numéricas.

  Siempre que utilicemos cadenas de caracteres en una consulta *SQL*, ésta irá siempre entre comillas simples.

  La sintaxis es la siguiente:

  **Formato:**

  ```
  WHERE nom_columna operador_de_comparacion constante
  ```

- *Comparación usando expresiones aritméticas*: Una expresión aritmética puede incluir operadores aritméticos (+, -, *, /), nombres de columnas y constantes. El * y el / tienen el nivel máximo de prioridad. El valor de una expresión aritmética es indeterminado si contiene una columna cuyo valor es indeterminado; la comparación con esta expresión no se ejecuta nunca.

En el ejemplo 2.3 podemos comprobar cómo se utilizan los operadores lógicos para unir dos expresiones de comparación.

**Ejemplo** 2.3 Artículos que pesan más de 100g ó el número de su proveedor es igual a 5

```
SQL> select art_num, art_nom
  2  from articulos
  3  where art_peso > 100 and art_prv = 5;

   ART_NUM ART_NOM
---------- --------------------
         4 lampara
         5 lampara
         6 lampara
         7 lampara

4 filas seleccionadas.
```

#### 2.2.2.3. Selección por comparación con un intervalo

El operador `between` permite seleccionar las filas cuya columna especificada contenga un valor que se encuentre dentro de un intervalo determinado, incluyendo a los propios valores que determinan el intervalo.

**Formato:**

```
WHERE nom_columna BETWEEN valor1 AND valor2
```

En realidad, el operador `between` equivale a combinar dos comparaciones con un *Y* lógico.

Para seleccionar las filas cuya columna contiene un valor externo a un intervalo determinado, se empleará el operador `not between`.

**Ejemplo** 2.4 Los proveedores cuyo número esté entre 2 y 5

```
SQL> select *
  2  from proveedores
  3  where prv_num between 2 and 5;

   PRV_NUM PRV_NOM
---------- -------------------------
         2 estilograficas reunidas
         3 mecanica de precision
         4 sanjita
         5 electrolamp

4 filas seleccionadas.
```

Podemos comprobar que en el ejemplo 2.4 aparecen los proveedores que cumplen la condición de selección. Observamos que también aparecen los datos de los proveedores cuyos números son el límite inferior y el superior del intervalo.

#### 2.2.2.4. Selección por comparación con una lista

El operador `in` permite seleccionar las filas para las cuales se cumple que una de sus columnas contiene un valor incluido en una lista de valores dada.

**Formato:**

```
WHERE nom_columna IN (valor1, valor2, ...)
```

El empleo de este operador equivale a combinar comparaciones con un *O* lógico.

Para seleccionar las filas cuya columna contiene un valor distinto del existente en una lista, es preciso emplear el operador `not in` como podemos observa en el ejemplo 2.5.

**Ejemplo** 2.5 Datos de las tiendas que no están en las ciudades indicadas

```
SQL> select *
  2  from tiendas
  3  where tda_pob not in
  4       ('paris', 'barcelona', 'palencia', 'lyon');

   TDA_NUM TDA_POB             TDA_GER
---------- ------------------- -------------------------
         1 madrid-batan        contesfosques, jordi
         2 madrid-centro       martinez, juan
         3 pamplona            dominguez, julian
         5 trujillo            mendez, pedro
         6 jaen                marin, raquel
         7 valencia            petit, joan
         8 requena             marcos, pilar
        10 gerona              gomez, gabriel

8 filas seleccionadas.
```

#### 2.2.2.5. Selección por comparación con un patrón

Las consultas de correspondencia con un patrón recuperan filas para las que el contenido de una columna de texto se corresponde con un texto dado, es decir, comprueban si el valor de un columna se ajusta a un patrón especificado

El operador `like` permite seleccionar las filas que contienen en la columna indicada (de tipo alfanumérico) un valor coincidente con el patrón dado.

Con los operadores `=` o `in`, el valor de la columna debe coincidir exactamente con la constante que sigue. Si se produce un error de codificación, o se ha olvidado el valor exacto, no es posible encontrar la fila buscada por este medio. Por lo tanto tendremos que recurrir a la búsqueda con un patrón.

Para definir el patrón disponemos de dos caracteres especiales que, en el momento de la comparación con el valor de la columna correspondiente, representan:

- % cualquier secuencia de ninguno o de varios caracteres.
- _ (subrayado) un carácter cualquiera, sólo uno.

**Formato:**

```
WHERE nom_columna LIKE 'patrón'
```

Para seleccionar las filas cuya columna contiene un valor que no debe coincidir con el patrón, bastará emplear el operador `not like`.

El patrón debe de ser del tipo de dato `varchar2` por eso coloca entre comillas simples.

**Ejemplo** 2.6 Datos de los gerentes de las tiendas de Madrid

```
SQL> select tda_num, tda_ger
  2  from tiendas
  3  where tda_pob like 'madrid%';

   TDA_NUM TDA_GER
---------- -------------------------
         1 contesfosques, jordi
         2 martinez, juan

2 filas seleccionadas.
```

### 2.2.2.6. Selección con un valor indeterminado

Si el valor de una columna no ha sido inicializado en una determinada fila, o sea, tiene valor `null`, ésta no interviene jamás en una selección por comparación de valores. Ahora bien, es posible comprobar si el valor no ha sido inicializado gracias al operador `is null`, o al operador opuesto `is not null`.

**Formato:**

```
WHERE nom_columna IS NULL
```

Éste es el único test de comprobación que permite seleccionar una fila cuando el valor de una columna es indeterminado.

**Ejemplo** 2.7 Datos de los artículos que no tienen definido el color

```
SQL> select art_num, art_nom
  2  from articulos
  3  where art_col is null;

   ART_NUM ART_NOM
---------- --------------------
         8 pesacartas 1-500
         9 pesacartas 1-1000

2 filas seleccionadas.
```

#### 2.2.2.7. Resumen

Notas sobre el uso de la orden `select`:

- Las cláusulas `select` y `from` son obligatorias en cada consulta *SQL*.
- La cláusula `from` deberá ir a continuación de la cláusula `select`.
- No es necesario seleccionar o proyectar todas las columnas de una tabla.
- Las columnas se muestran de izquierda a derecha en el orden especificado en la cláusula `select`.
- Si queremos proyectar todas las columnas de una tabla usaremos el asterisco (*) y éstas aparecerán en el mismo orden en que fueron creadas.
- Las filas se muestran en el orden en que están almacenadas.
- Los usuarios pueden obtener información de sus propias tablas, o bien de aquellas a las que se les haya dado permiso de acceso.
- Las cláusulas `select` y `from` aparecerán precediendo a cualquier otra cláusula.

Notas sobre el empleo de la cláusula `where`:

- Las columnas especificadas en la cláusula `where` deberán ser parte de la tabla especificada en la cláusula `from`.
- Las columnas utilizadas en la cláusula `where` no tienen que estar necesariamente en la lista de la orden `select`.

- Las comparaciones de series de caracteres en la cláusula `where` requieren que éstas estén acotadas entre comillas simples, en las comparaciones numéricas no es necesario.

- La cláusula `where` deberá especificar los mismos caracteres, mayúsculas o minúsculas, que hubiera en las tablas de la base de datos.

## 2.3. Eliminación de registros repetidos

El *SQL* visualiza por defecto el resultado de una consulta, sin eliminar las repeticiones de las ocurrencias devueltas. Se puede obtener el resultado de una consulta sin dichas repeticiones introduciendo la cláusula `distinct` después de la orden `select`.

El ejemplo 2.8 nos muestra los distintos colores que tienen los artículos. Hemos eliminado las tuplas que no tienen definido el color. También podemos observar que la tabla resultante nos muestra los colores ordenados alfabéticamente.

## 2.4. Renombrar columnas

Los nombres de las columnas, expresiones y funciones que aparecen en la cláusula `select` de una consulta pueden cambiarse temporalmente, incluyendo el nuevo nombre en la cláusula `select` después del nombre de la columna, separado por un espacio.

El nuevo nombre o alias irá entre comillas dobles si se compone de dos o más palabras, o si queremos que aparezca tal cual lo hemos escrito en mayúsculas o en minúsculas, como se muestra en el ejemplo 2.9. El cambio de nombres únicamente mantiene su efecto durante el tiempo de la consulta; los nombres de las columnas de las tablas permanecen y no les afecta el cambio.

**Ejemplo** 2.8 Mostrar todos los colores de los artículos

```
SQL> select distinct art_col
  2  from articulos
  3  where art_col is not null;

ART_COL
-------
azul
blanco
negro
rojo
verde

5 filas seleccionadas.
```

**Ejemplo** 2.9 Mostrar todos los colores de los artículos

```
SQL> select distinct art_col "Colores"
  2  from articulos
  3  where art_col is not null;

Colores
-------
azul
blanco
negro
rojo
verde

5 filas seleccionadas.
```

## 2.5. Clasificación de filas

Cuando no se determina el orden en que se desean obtener las filas, puede suceder que se obtengan resultados distintos en dos ejecuciones consecutivas de una misma consulta[4], a causa de esta misma indeterminación. Se puede solicitar la clasificación de las filas seleccionadas según el valor de una o varias columnas. Una clasificación no necesita la existencia de un índice; el *SQL* crea un índice temporal si ello le sirve para optimizar la consulta.

Para ello, es preciso añadir la cláusula `order by`, seguida del nombre de las columnas sobre las cuales se desea elaborar la clasificación separadas por comas. Si se añade `asc` o `desc` se está precisando el orden ascendente o descendente con el que se realizará la misma. Por defecto se hace en orden ascendente.

Los valores indeterminados (`null`) se visualizan agrupados, al principio de la tabla si el orden es descendente y al final si es ascendente.

**Formato:**

```
ORDER BY {nom_columna | entero | alias} [asc|desc], ...
```

Ésta es la única instrucción en la que el *SQL* tiene en cuenta el orden de las columnas. Así, nosotros podemos especificar el nombre de la columna por la que queremos ordenar o bien la posición que ocupa dicha columna en la lista de la orden `select`.

La columna sobre la que se realiza la clasificación debe, obligatoriamente, formar parte de la lista de columnas incluidas en la cláusula `select`, o bien, pertenecer a una de las tablas incluidas en la cláusula `from`. Si utilizamos la cláusula `distinct` forzozamente la

[4] El *SQL* no tiene predefinido un orden de aparición de las tuplas, en cambio, Oracle las mostrará en el mismo orden en que fueron creadas.

columna de ordenación tiene que estar en la lista de columnas a proyectar por la orden `select`, ver el ejemplo 2.10.

**Ejemplo** 2.10 Datos de los artículos ordenados ascendente por su precio de compra

```
SQL> select art_num Numero, art_nom Nombre
  2  from articulos
  3  where art_num < 10
  4  order by art_pc;

    NUMERO NOMBRE
---------- --------------------
         1 impresora
         3 calendario
         5 lampara
         4 lampara
         7 lampara
         6 lampara
         8 pesacartas 1-500
         9 pesacartas 1-1000
         2 calculadora

9 filas seleccionadas.

SQL> select distinct art_nom Nombre
  2  from articulos
  3  where art_num < 10
  4  order by art_pc;
order by art_pc
         *
ERROR en linea 4:
ORA-01791: no es una expresion obtenida bajo SELECT
```

### 2.5.1. Resumen

- La cláusula `order by` deberá ser la última cláusula en aparecer en la consulta *SQL*.
- Esta cláusula es el único sistema existente para ordenar los datos según el criterio que se desee, ya que las tablas almacenan las filas conforme éstas han sido introducidas.
- La cláusula `order by` únicamente clasifica las filas obtenidas en la consulta.
- Por omisión, la secuencia será ascendente.
- Para ordenaciones descendentes se deberá especificar explícitamente.

- Puede mezclarse ordenaciones ascendentes y descendentes en la misma cláusula `order by`.
- Puede clasificarse por columnas que no formen parte de la consulta, siempre que formen parte de alguna de las tablas especificadas en la cláusula `from`.
- La secuencia de las columnas especificadas en la cláusula `order by` determina la secuencia de las claves de clasificación, esto es, los valores de una columna serán clasificados con valores iguales de las columnas precedentes, y así sucesivamente.
- Las columnas con valores nulos (`null`) se evaluarán al final de todas si el orden es `asc` y al principio si son `desc`.

## 2.6. La orden `insert`

Esta orden se emplea para añadir nuevas filas a una tablaya existente. Su formulación más sencilla sirve para añadir una fila cada vez. Para ello, se citará el nombre de la tabla, el nombre de las columnas que hay que inicializar y el valor que se les desea dar. Si no se indica el nombre, se supone que las columnas que deben recibir los valores son todas las de la tabla, en el orden de izquierda a derecha tal como fueron creadas dichas columnas.

El formato general para esta orden es:

**Formato:**

```
INSERT INTO nom_tabla [(nom_column1, nom_column2, ...)]
VALUES (valor1, valor2, ...);
```

### 2.6.1. Inserción de todas las columnas de un registro

Cuando damos un valor para cada una de las columnas de un nuevo registro, se puede simplificar el formato de la sentencia `insert`.

**Formato:**

```
INSERT INTO nom_tabla
VALUES (valor1, valor2, ...);
```

El orden de los valores situados entre paréntesis deberá corresponder exactamente con el orden que se les dio a las columnas cuando fueron definidas las tablas en su creación.

Si se desea dejar una columna sin valor, debe indicarse explícitamente mediante el operador `null`, situado en la posición que tiene la columna que queremos dejar con valor nulo.

### 2.6.2. Inserción parcial de columnas en un registro

Para insertar únicamente unos cuantos valores en un registro, incluyendo un valor para cada columna declarada como `no null`, explícitamente se fijará qué columnas deberán recibir los valores especificados. Para este tipo de inserción, los nombres de las columnas en la cláusula `insert` que van situados entre paréntesis no necesitan estar situados en un orden determinado, pero deberán corresponderse directamente con los correspondientes valores situados también entre paréntesis en la cláusula `values`.

**Formato:**

```
INSERT INTO nom_tabla (nom_column3, nom_column2, ...)
VALUES (val3, val2, ...);
```

En el ejemplo 2.11 hemos insertado una tupla en la relación *Tiendas* de varias formas. En el último caso, hemos dejado el nombre del gerente sin especificar.

### 2.6.3. Insertar datos desde otra tabla

Pueden insertarse datos en una tabla, usando los registros de una o varias tablas, siempre que asociemos a la orden `insert` una consulta `select` en vez de una cláusula `values`. La cláusula `select` anidada obtendría los datos deseados de una tabla y la cláusula `insert` los situaría en otra tabla. Si dos tablas tuvieran columnas con idéntica definición, la siguiente proposición podría insertar todos los datos desde la tabla $y$ a la tabla $x$.

**Ejemplo** 2.11 Inserción de tuplas en una tabla

```
SQL> insert into tiendas
  2  values (16, 'Pto. Real', 'Martin, Luis');

1 fila creada.
...
SQL> insert into tiendas (tda_num, tda_ger, tda_pob)
  2  values (16,' Martin, Luis', 'Pto. Real');

1 fila creada.
...
SQL> insert into tiendas
  2  values (16, 'Pto. Real', null);

1 fila creada.
```

**Formato:**

```
INSERT INTO x
SELECT *
FROM y;
```

O bien, se puede especificar qué columnas de la tabla $x$ recibirán datos desde las columnas que se deseen de la tabla $y$.

**Formato:**

```
INSERT INTO x [(nom_column1, nom_column3, ...)]
SELECT {nom_columna | expresión}
FROM y
[WHERE criterio de selección];
```

La tabla emisora, $y$, debe existir con anterioridad y los tipos de datos deben corresponderse plenamente o, al menos, ser del mismo tipo.

La cláusula `select` anidada en la `insert` no se incluye entre paréntesis, dado que no se trata, en realidad, de una consulta anidada. La cláusula `order by` no puede intervenir en una consulta asociada a `insert`. No es necesario que las correspondientes columnas de las dos tablas tengan el mismo nombre. El orden de columnas en la cláusula `select` deberá corresponder con el orden de columnas especificado en la cláusula `insert`. Las expresiones y funciones pueden ser usadas en la cláusula `select` anidada para insertar datos procedentes de cálculos matemáticos.

Es posible crear una tabla e insertar filas en ella con una sola operación con la orden `create` del Lenguaje de Definición de Datos.

### 2.6.4. Resumen

- Una tabla deberá estar creada antes de poder insertar datos en ella.
- La lista de nombres de columnas se utiliza cuando:
  - Se insertan unas cuantas columnas que existen en la definición de la tabla.
  - Se introducen columnas en una secuencia diferente en la que fueron creadas.
- Las columnas de la tabla que no aparezcan en la orden `insert` tomarán como valor `null`.
- En la cláusula `values` pueden especificarse valores `null`, siempre y cuando en la definición de restricción de integridad correspondiente no aparezca especificado el valor `not null`.

- Los valores deberán colocarse en el mismo orden en el que aparecen sus correspondientes columnas en la cláusula `insert`.
- Si se omite la lista de columnas, deberá especificarse un valor para cada una de las columnas de la tabla.
- Si se omiten las columnas, los valores a insertar deberán seguir la misma secuencia seguida por las correspondientes columnas cuando se creó la tabla.
- Los valores insertados deberán corresponder con el tipo de dato de la columna correspondiente.
- Los valores tipo carácter deberán ir entre comillas simples.
- La especificación de las filas a copiar tiene la misma sintaxis que la de las consultas *SQL*. Podrá ser tan compleja como se desee con excepción de la cláusula `order by` que no puede ser utilizada.
- Las columnas que aparezcan en la cláusula `insert` son receptoras de los datos, mientras que las que aparecen en la `select` son columnas emisoras.
- La secuencia y tipo de datos de las columnas que aparecen en la cláusula `insert` deberán corresponder con la secuencia y tipo de datos de las columnas obtenidas por la subconsulta.

## 2.7. La orden `update`

Esta orden cambia o actualiza los datos existentes de una tabla, especificando sus nuevos valores en las columnas seleccionadas. Pueden cambiarse todos los valores de una columna para todos los registros, o bien únicamente algunos valores para registros específicos. La cláusula `where` es usada para seleccionar las filas que serán actualizadas.

La modificación a realizar es una expresión que admite constantes, nombres de columnas e incluso consultas anidadas. La formación general de la orden es, pues, multifila. Para modificar una sola fila, basta escribir un criterio de tal modo que se seleccione sólo esa fila. Para modificar todas las filas de una tabla, no es necesario incluir una cláusula `where`.

**Formato:**

```
UPDATE nom_tabla
SET nom_columna = nuevovalor [,nom_columna = nuevovalor]
[WHERE criterio de selección de los registros];
```

### 2.7.1. Cláusula set

Pueden emplearse consultas anidadas en la cláusula `set` de una sentencia `update`, al igual que en la cláusula `where`.

**Formato:**

```
UPDATE nom_tabla
SET nom_columna = (SELECT nom_columna FROM ...), ...
[WHERE condición];
```

Si la consulta anidada asociada a la cláusula `set` no devuelve ningún valor y la columna admite nulos, dejará dicha columna vacía para todas las tuplas de la tabla.

La cláusula `where` de una orden `update` también puede tener una sentencia `select` anidada.

Hay que tener en cuenta que si la cláusula `set` lleva varias columnas separadas por comas igualadas a una consulta anidada, estas columnas van entre paréntesis.

**Formato:**

```
SET (nom_columna1, nom_columna2) = (consulta anidada)
```

**Ejemplo** 2.12 Actualización del peso de un artículo

```
SQL> update articulos
  2  set art_peso = (select art_peso
  3                  from articulos
  4                  where art_num = 27)
  5 where art_num = 24;

1 fila actualizada.
```

En el ejemplo 2.12, el peso del artículo nº 24 se ha actualizado pasando a tener el mismo peso que tiene el artículo nº 27.

### 2.7.2. Resumen

- `update` opera en todas las filas que cumplen la condición `where`.
- Si la cláusula `where` es omitida, todas las filas serán actualizadas.
- Puede establecerse una columna a valor `null` si la definición de restricción de integridad para esa columna en la tabla lo permite.

- Puede establecerse una columna igual a una expresión aritmética compuesta de columnas y constantes (+, -, *, ó /).

- También pueden utilizarse funciones Oracle que operen en una única columna en la cláusula `set`.

- Si el valor de la cláusula `set` proviene de una consulta anidada, deberá haber exactamente las mismas columnas que las especificadas en la subconsulta y, además, si son más de una columna deberán ir entre paréntesis.

- Cada subconsulta deberá ser subconsulta a una única fila para cada registro a actualizar.

## 2.8. La orden `delete`

Los registros de una tabla pueden borrarse con la orden `delete`. Podemos borrar todos los registros o bien algunos según un criterio de selección.

**Formato:**

```
DELETE [FROM] tabla
[WHERE criterio de selección de un registro];
```

Si una orden `delete` no contiene una cláusula `where`, todas las filas de la tabla serán borradas. Sin embargo, la tabla continúa existiendo en la base de datos, y podrán insertarse nuevas columnas en ella usando la orden `insert`. Los registros no pueden borrarse parcialmente, ya que la orden `delete` afecta al registro completo.

La formulación general de la orden es multifila. Para suprimir una sóla fila basta escribir un criterio que permita seleccionar sólo esa. Para suprimir todos los registros de una tabla no hace falta incluir una cláusula `where`.

El criterio de selección de tuplas puede obtenerse a partir del resultado de una consulta anidada en la cláusula `where`.

**Ejemplo** 2.13 Eliminación de algunas tuplas de la tabla art_2

```
SQL> delete from art_2
  2  where art_num > 20;

6 filas suprimidas.
```

## 2.9. Diferencias entre `drop`, `delete` y `truncate`

La diferencia entre las órdenes `delete`, `drop`[5] y `truncate`[6] consiste en lo siguiente:

- La orden `delete`, sin un criterio de selección, hace que todos los datos de la tabla se borren, pero la definición de dicha tabla permanece en el diccionario de datos.
- La orden `drop` elimina, además, la definición de la tabla en el Diccionario de Datos, liberando por ello el espacio ocupado en la base de datos.
- La orden `truncate` borra todas las tuplas de una tabla pero no borra la definición de la tabla del Diccionario de Datos y, además, puede liberar el espacio ocupado por ella y así Oracle puede disponer de dicho espacio para otros objetos.

[5] Orden del Lenguaje de Definición de Datos.
[6] Orden del Lenguaje de Definición de Datos.

# Capítulo 3

# Funciones y expresiones

Este capítulo nos muestra las distintas funciones de las que disponemos así como la manera de utilizarlas. Se introduce el concepto de grupo de tuplas analizando las ventajas y los problemas que presenta su uso.

## 3.1. Introducción

Una base de datos, normalmente, está formada por toda una serie de estructuras de datos que contienen tipos de datos diferentes. A estos datos se les pueden aplicar los operadores aritméticos y las funciones, tanto numéricas como de carácter. Dichas funciones se pueden utilizar tanto con registros como con grupos de registros que cumplan una determinada condición.

## 3.2. Expresiones

Las expresiones aritméticas nos permiten realizar cálculos numéricos con los datos de la base de datos. De hecho, es posible combinar el valor de diferentes columnas entre sí o bien con constantes numéricas, utilizando los cuatro operadores aritméticos (+, -, *, /).

Si se trata de una operación de visualización, basta con escribir la expresión aritmética en la cláusula `select` como si fuera una columna. Lo mismo sucede en una cláusula `where` u `order by`.

**Ejemplo** 3.1 Beneficio de los artículos

```
SQL> select art_num Numero, art_nom Nombre,
  2         (art_pv - art_pc) Beneficio
  3  from articulos;

    NUMERO NOMBRE                BENEFICIO
---------- -------------------- ----------
         3 calendario                  180
         2 calculadora                 700
         1 impresora                   180
         4 lampara                     880
         5 lampara                     900
         6 lampara                     880
         7 lampara                     880
         8 pesacartas 1-500           1600
         9 pesacartas 1-1000          2000
        10 boligrafo                    20
        11 boligrafo                    20
        12 boligrafo lujo               40
        13 boligrafo lujo               40
        14 boligrafo lujo               40
        15 boligrafo lujo               40

15 filas seleccionadas.
```

## 3.3. La tabla *dual*

La tabla *dual* es una tabla pequeña pero útil. Proporcionada por Oracle, se emplea para probar funciones o realizar cálculos rápidos. Se crea en el momento de la instalación y creación de la base de datos y sólo tiene una columna y una fila.

En el ejemplo 3.2 podemos ver la descripción de esta pequeña tabla.

**Ejemplo** 3.2 Descripción de la tabla `dual`

```
SQL> describe dual
 Nombre                            ?Nulo?   Tipo
 --------------------------------- -------- ------------------------
 DUMMY                                      VARCHAR2(1)
```

## 3.4. Funciones

Una función manipula conjuntos de datos y devuelve un resultado. Las funciones pueden operar con cero, uno, dos, o más argumentos. El formato es el siguiente:

**Formato:**

```
nom_función(arg1, arg2, ...)
```

Las funciones se dividen en dos tipos:

- Funciones de registros únicos (o escalares).
  - Funciones numéricas.
  - Funciones de caracteres: devuelven un valor de tipo carácter o bien devuelven un valor de tipo numérico.
  - Funciones de fechas.
  - Funciones de conversión.
  - Otras funciones.
- Funciones de funciones de grupos.

Estas funciones difieren en el número de registros sobre los cuales actúan. Una función de registro único devuelve un único registro como resultado por cada registro de la tabla o vista consultado, mientras que las funciones de grupos devuelven un único registro como resultado por cada grupo de registros consultados.

Las funciones de registros aparecen en la lista de la orden `select` (suponiendo que la consulta no lleve la cláusula `group by`) y en la cláusula `where`.

Las funciones de grupos aparecen en la lista de la orden `select` y en la cláusula `having`, siempre que la consulta tenga la cláusula `group by`.

### 3.4.1. Funciones numéricas

Las funciones numéricas pueden aparecer en la cláusula `select` de una sentencia *SQL*, en la cláusula `where`, o bien en la cláusula `order by`. Las distintas funciones que podemos aplicar a los datos de tipo numérico se pueden ver en la tabla 3.1.

Los ejemplos 3.3 y 3.4 nos muestran cómo usar estas funciones para realizar cálculos númericos.

### 3.4.2. Funciones de caracteres

A las series de caracteres también se les puede aplicar funciones especiales para manipularlas. Podemos realizar toda una serie de operaciones con estas cadenas.

En la tabla 3.2 se indica las funciones de caracteres que devuelven un valor de tipo carácter y cómo podemos usarlas, mientras que en la tabla 3.3 vemos algunas de las funciones que se aplican a cadenas de caracteres y que nos devuelven un valor numérico como resultado.

En los ejemplos 3.5 y 3.6 hacemos uso de dos tipos distintos de estas funciones. En uno cambiamos el aspecto de salida de los datos de una columna, mientras que en el otro contamos para conocer quién es el proveedor con el nombre más largo.

**Ejemplo** 3.3 Raíz cuadrada de 27

```
SQL> select sqrt(27) "raiz de 27"
  2  from dual;

raiz de 27
-----------
 5,19615242

1 fila seleccionada.
```

**Ejemplo** 3.4 3 elevado al cubo

```
SQL> select power(3,3) "3 elevado al cubo"
  2  from dual;

3 elevado al cubo
-----------------
               27

1 fila seleccionada.
```

### 3.4.3. Funciones de conversión

Estas funciones convierten un valor de un tipo de dato en otro. La forma general es:

**Formato:**

```
nom_función(tipo_dato1,tipo_dato2)
```

**Ejemplo** 3.5 Inicial de los nombres de clientes en mayúscula

```
SQL> select initcap(clt_nom) "Nombre"
  2  from clientes;

Nombre
--------------------
Margarita
Miguel
Jean
Michel
Antoni
Marcel
Pablo
Gerad
Consuelo
Pau
Jorge
Pablo
Diego
Joaquin
Jacinto
Pedro

16 filas seleccionadas.
```

La tabla 3.4 muestra algunas de las funciones más usadas de conversión de datos.

Estas funciones normalmente se utilizan cuando queremos dar formato a columnas de tipo fecha[1].

### 3.4.4. Otras funciones

Existen otras funciones que no podemos incluirlas en ninguno de los grupos en que hemos dividido las funciones. Dichas funciones aparecen en la tabla 3.5 junto con sus usos.

En el ejemplo 3.7 hemos aplicado una de estas funciones con la tabla `dual` para saber el nombre de la cuenta de usuario desde donde estamos trabajando.

[1] Ver capítulo 6.

**Ejemplo** 3.6 ¿Qué proveedor tiene el nombre más largo?

```
SQL> select prv_num numero, length(prv_nom) nombre
  2  from proveedores;

    NUMERO     NOMBRE
---------- ----------
         1         16
         2         23
         3         21
         4          7
         5         11

5 filas seleccionadas.
```

**Ejemplo** 3.7 ¿Quién soy?

```
SQL> select user
  2  from dual;

USER
------------------------------
GADESCHI

1 fila seleccionada.
```

### 3.4.5. Funciones de grupo

Al igual que las funciones anteriores se aplican al valor de una columna, las funciones de grupo efectúan operaciones sobre un conjunto de valores de una columna dentro de un grupo de filas.

Un grupo es «un subconjunto de filas de una tabla en la que el valor de una columna es constante». Los grupos se especifican mediante la cláusula `group by`, seguida del nombre de la columna sobre la cual se efectúa la agrupación. Cuando no existe la cláusula `group by`, el grupo está formado por todo el conjunto de filas seleccionadas.

Las funciones de grupo se muestran en la tabla 3.6; algunas de ellas aceptan las opciones `distinct` y `all`.

El argumento de la función contiene el nombre de la columna sobre el cual debe ejercerse el cálculo. Si el argumento viene precedido de la opción `distinct`, se eliminarán antes las repeticiones de valores. Los valores indeterminados o `null` no se tienen en cuenta nunca. Si la función se incluye en la cláusula `select`, se visualizará como una columna o una expresión. De hecho, el resultado de una búsqueda que se realiza sobre una función es una tabla que contiene una sola columna, el valor de la función, y una sola fila que resume los valores de la columna de todas las filas.

Dentro de una cláusula `select`, podemos incluir varias funciones y combinarlas con los operadores aritméticos.

## 3.5. Consultas por grupos

La noción de grupo introduce dos nuevos tipos de cláusulas: `group by` para formar los grupos con los elementos que tenga los mismos valores en la columnas indicadas y `having` para seleccionar solamente los grupos que cumplan las condiciones impuestas.

### 3.5.1. La cláusula `group by`

La cláusula `group by` reordena la tabla que resulta de una consulta `select` en un número mínimo de grupos tales que, en el interior de cada uno, la columna especificada tenga el mismo valor para cada fila. Este tratamiento no afecta a la organización física de la tabla.

Cuando se emplea una cláusula `group by`, las funciones de grupo se calculan para cada grupo. Esta cláusula permite, generalmente, la visualización del valor de la columna común, seguida de aquellos valores de las funciones aplicadas a cada grupo.

La cláusula `group by` se utiliza para definir múltiples grupos de filas dentro de una consulta con la orden `select`.

**Formato:**

```
GROUP BY nom_columna [,nom_columna,...]
```

También puede añadirse a la consulta una cláusula `where`, para establecer un criterio en la selección de las filas, o bien, la cláusula `order by` para ordenar por distintas columnas la consulta resultante.

En el ejemplo 3.8 primero hemos eliminado los artículos que no tienen definido el color, después hemos formado grupos por colores y dentro de cada grupo hemos hallado el precio de compra más caro. En el ejemplo 3.9 hemos formado directamente los grupos dandole un nombre al grupo de los artículos que no tienen definido el color, y posteriormente hemos hallado el precio máximo de compra. Se puede observar que la cláusula `group by` ordena las tuplas ascendentemente por el atributo por el cual formamos los grupos.

### 3.5.2. Resumen

- Un grupo se define como las filas que tienen un valor común en una o más columnas. Estas columnas se muestran en la cláusula `group by`.

- Se devuelve una fila por cada grupo formado. Por ejemplo, si se agrupa por colores, se obtendrá una fila resumen de información por cada uno de los colores diferentes que hubiera.
- Todos los ítems de la lista especificada en la cláusula `select` deberán ser del mismo nivel de agrupación.
- Las columnas que no aparezcan en la cláusula `group by` no podrán aparecer en la cláusula `select`, excepto como argumentos para agrupar funciones.

**Ejemplo** 3.8 Precio máximo de los artículos por color

```
SQL> select art_col Color, max(art_pc) "maximo precio compra"
  2  from articulos
  3  where art_col is not null
  4  group by art_col;

COLOR   maximo precio compra
------- --------------------
azul                    2100
blanco                  2000
negro                   4000
rojo                    2100
verde                   2100

5 filas seleccionadas.
```

**Ejemplo** 3.9 Precio máximo de los artículos por color, contando a los indefinidos

```
SQL> select nvl(art_col,'indefinido') Color,
  2         max(art_pc) "maximo precio compra"
  3  from articulos
  4  group by art_col;

COLOR      maximo precio compra
---------- --------------------
azul                       2100
blanco                     2000
negro                      4000
rojo                       2100
verde                      2100
indefinido                 3000

6 filas seleccionadas.
```

Las consultas `select` pueden agrupar información según el valor de más de una columna al mismo tiempo.

### 3.5.3. La cláusula having

La cláusula **having** equivale a la cláusula **where** pero aplicada a grupos. Esta cláusula generalmente no puede emplearse si antes no ha sido especificada la cláusula **group by**. De hecho, el criterio especificado en la cláusula **having** afecta al valor de una función calculada sobre un grupo. La cláusula **having** se utiliza para restringir los grupos seleccionados en el resultado de la consulta una vez ejecutada ésta con la cláusula **group by**.

Los grupos que no cumplen el criterio especificado en la cláusula **having** no se incluyen en el resultado.

**Formato:**

```
HAVING nom_función o expresión
```

**Ejemplo** 3.10 ¿Qué color lo llevan más de dos artículos?

```
SQL> select nvl(art_col,'indefinido') Color,
  2         count(*) "Numero de elementos"
  3  from articulos
  4  group by art_col
  5  having count(*) > 2;

COLOR      Numero de elementos
---------- -------------------
azul                         3
rojo                         4

2 filas seleccionadas.
```

En el ejemplo 3.10 primero hemos formado grupos por colores pero solamente hemos mostrado aquellos grupos que tienen más de dos elementos.

## 3.6. Los valores null y la función nvl

El valor **null** es un valor indeterminado; por tanto, los valores nulos no se utilizan cuando se evalúan expresiones o funciones. Si un valor **null** se emplea, por ejemplo, en un cálculo aritmético, el resultado será siempre nulo (**null**).

Algunas veces se podría desear sustituir temporalmente el valor nulo por otro valor sobre todo cuando se realizan cálculos aritméticos. Para sustituir temporalmente el valor nulo por cualquier otro se deberá usar la función **nvl** (ver tabla 3.5).

Vamos a ver dos ejemplos del uso de esta función, una con valores de tipo caracter y otra con valores de tipo numérico.

Compárese el ejemplo 3.11 con el 2.9. En los dos hemos mostrado los colores distintos que tienen los artículos. En el ejemplo 2.9 hemos eliminado las tuplas que no tienen color, con lo cual la consulta devuelve solamente 5 filas, mientras que en el ejemplo 3.11 a los artículos que no tienen color le hemos asignado la cadena de caracteres «indefinido» y la consulta nos dice que devuelve 6 filas en vez de 5.

**Ejemplo** 3.11 Mostrar todos los colores de los artículos, incluido los nulos

```
SQL> select distinct nvl(art_col,'indefinido') "Colores"
  2  from articulos;

Colores
----------
azul
blanco
indefinido
negro
rojo
verde

6 filas seleccionadas.
```

Los ejemplos 3.12 y 3.13 muestra el uso de esta función con columnas de tipo numéricas. En el ejemplo 3.12 hemos usado la función `nvl` para asignarle un valor momentáneo al peso y luego calcular la media aritmetica con la función `avg`. En el ejemplo 3.13 no hemos usado la función `nvl` con lo cual las tuplas que no tienen definido el peso no se eligen para calcular la media aritmetica. Vemos que en este segundo ejemplo, la media de pesos es mayor que en el primer caso, ya que se divide por un número menor de elementos.

**Ejemplo** 3.12 Peso medio de los artículos, forma1

```
SQL> select avg(nvl(art_peso,0)) "Peso medio"
  2  from articulos;

Peso medio
----------
181,333333

1 fila seleccionada.
```

**Ejemplo** 3.13 Peso medio de los artículos, forma 2

```
SQL> select avg(art_peso) "Peso medio"
  2  from articulos;

Peso medio
----------
209,230769

1 fila seleccionada.
```

| Función | Uso |
|---|---|
| `abs(n)` | Retorna el valor absoluto del número $n$. |
| `ceil(n)` | Retorna el menor entero mayor o igual a $n$. |
| `cos(n)` | Retorna el coseno de $n$, expresado en radianes. |
| `cosh(n)` | Retorna el coseno hiperbólico de $n$, expresado en radianes. |
| `exp(n)` | Retorna $e$ elevado a la $n$-*ésima* potencia. |
| `floor(n)` | Retorna el mayor entero menor que o igual a $n$. |
| `ln(n)` | Retorna el logaritmo natural de $n$, si $n$ es mayor que 0. |
| `log(m,n)` | Retorna el logaritmo, base $m$, de $n$. |
| `mod(m,n)` | Retorna el resto de $m$ dividido por $n$. Devuelve $m$, si $n$ es 0. |
| `power(m,n)` | Retorna $m$ elevado a la $n$-*esima* potencia. |
| `round(n[,m])` | Retorna $n$ redondeado a $m$ lugares a la derecha del punto decimal; si $m$ es omitido, a 0 lugares. Si $m$ es negativo redondea a la izquierda del punto decimal. |
| `sign(n)` | Si $n<0$, la función retorna -1; si $n=0$, la función retorna 0; si $n>0$, la función retorna 1. |
| `sin(n)` | Retorna el seno de $n$, expresado en radianes. |
| `sinh(n)` | Retorna el seno hiperbólico de $n$, expresado en radianes. |
| `sqrt(n)` | Retorna la raíz cuadrada de $n$. El valor de $n$ no puede ser negativo. Devuelve un resultado «real». |
| `tan(n)` | Retorna la tangente de $n$, expresado en radianes. |
| `tanh(n)` | Retorna la tangente hiperbólica de $n$, expresado en radianes. |
| `trunc`(n[,m]) | Retorna $n$ truncado a $m$ posiciones decimales; si $m$ es omitido, a 0 lugares. |

Tabla 3.1: Funciones numéricas

| Función | Uso |
|---|---|
| `chr(n)` | Retorna el caracter que tiene equivalente binario a *n* en el conjunto de caracteres de la base de datos. |
| `concat(serie1,serie2)` | Retorna *serie1* concatenada con *serie2*. Equivale a usar «\|\|». |
| `initcap(serie)` | Pone en mayúsculas la letra inicial de *serie*. |
| `lower(serie)` | Pone *serie* con todas las letras en minúsculas. |
| `lpad(serie1,n [,serie2])` | Retorna *serie1*, rellenada a la izquierda con una longitud *n* con la secuencia de caracteres de *serie2*. |
| `ltrim(serie, [conjunto])` | Elimina caracteres desde la izquierda de *serie*, que comiencen con el carácter a eliminar hasta el primer carácter no incluido en *conjunto*. |
| `rpad(serie1, n [,serie2])` | Igual que `lpad` pero el relleno se efectúa a la derecha de *serie1*. |
| `rtrim(serie1, [conjunto])` | Igual que `ltrim` pero a partir de la derecha de la *serie1*. |
| `soundex(serie)` | Retorna una cadena de caracteres conteniendo la representacion fonética de *serie*. |
| `substr(serie,m [,n])` | Extrae un trozo de *serie* comenzando en la posición *m* y de *n* caracteres de longitud. |
| `upper(serie)` | Retorna *serie* con todos los caracteres en mayúsculas. |

Tabla 3.2: Funciones carácter que devuelven carácter

| Función | Uso |
|---|---|
| `ascii(serie)` | Retorna la representación decimal en el conjunto de caracteres de la base de datos del primer *byte* de *serie*. |
| `length(serie)` | Retorna el número de caracteres de *serie*. |

Tabla 3.3: Funciones carácter que devuelven un número

| Función | Uso |
|---|---|
| `to_char(d[,fmt[,'ndl']])` | Convierte *d* de tipo de datos fecha a un valor de tipo de dato `varchar2` en el formato especificado por el formato de fecha *fmt*. Si se omite el formato, *d* es convertido a un valor de `varchar2` con formato por defecto. *ndl* especifica el lenguaje en el que se devolverán los nombres del día, del mes y sus abreviaturas. |
| `to_char(n[,fmt])` | Convierte *n* de tipo de datos `number` a un valor de tipo de dato `varchar2`, usando el formato de número opcional *fmt*. |
| `to_date(ch[,fmt[,'ndl']])` | Convierte *ch* de tipo `char` o `varchar2` a tipo de datos fecha. *ndl* tiene el mismo significado que en la función `to_char`. |
| `to_number(ch[,fmt])` | Convierte *ch*, de tipo `char` o `varchar2` a tipo `number` (entero). |

Tabla 3.4: Funciones de conversión de tipos de datos

| Función | Uso |
|---|---|
| `nvl(exp1,exp2)` | Si *exp1* es nulo devuelve *exp2*; si *exp1* no es nulo devuelve *exp1*. Los argumentos pueden ser de cualquier tipo de dato. Si los tipos de datos son diferentes Oracle convierte *exp2* al tipo de datos de *exp1*. |
| `uid` | Retorna un entero como identificador único del usuario actual. |
| `user` | Retorna el nombre del usuario actual de Oracle. |

Tabla 3.5: Otras funciones

| Función | Uso |
|---|---|
| `avg([distinct\|all] n)` | Calcula la media aritmética de *n*. |
| `count(*\|[distinct\|all] expr)` | Retorna el número de registros de la consulta. Si se especifica *expr*, devuelve registros cuando *expr* es `not null`. |
| `max([distinct\|all] expr)` | Devuelve el valor máximo de *expr*. |
| `min([distinct\|all] expr)` | Devuelve el valor mínimo de *expr*. |
| `stddev([distinct\|all] x)` | Devuelve la desviación estándar de *x*, un número. |
| `sum([distinct\|all] n)` | Devuelve la suma de valores de *n*. |
| `variance([distinct\|all] x)` | Devuelve la varianza de *x*, un número. |

Tabla 3.6: Funciones de grupo

# Capítulo 4

# Consultas anidadas

En este capítulo podremos comprobar cómo relizar una consulta cuando los criterios de selección los obtenemos del resultado de una consulta, llamada anidada. También se estudiará los diferentes operadores de que disponemos.

## 4.1. Introducción

Hemos visto que la información que devuelve una consulta `select` puede filtrarse según un criterio definido en una cláusula `where` o `having`. Puede ocurrir, y de hecho ocurre, que los criterios de selección de tuplas sean el resultado, a su vez, de una consulta `select`. Para hacer esto, debemos anidar las consultas, donde el resultado de una constituye el punto de partida para la ejecución de la del nivel inmediatamente superior. Con esto conseguimos que el criterio de selección de tuplas sea el resultado de una consulta a la base de datos.

Teóricamente, no hay límite para el número de niveles de anidación. Debemos tener en cuenta que las consultas de nivel inferior deben ir entre paréntesis.

A veces se puede expresar el mismo tipo de consulta mediante una unión de tablas[1]. El lenguaje *SQL* ofrece a menudo la posibilidad de escribir de distintos modos una misma consulta, pero el tiempo invertido en la obtención del resultado varía según la expresión y también según la lógica seguida por el optimizador para organizar la consulta. Por tanto, no es posible decir «a priori» cuál es la formulación más eficaz.

[1]Esta operación corresponde al producto natural o *join*. Ver capítulo 5.

En *SQL*, las consultas cuyo criterio de selección depende del resultado devuelto por otra consulta, se denomina «consultas anidadas». La subconsulta se evalúa en primer lugar y el resultado se sustituye dinámicamente en la consulta de nivel superior.

Las consultas anidadas también se pueden usar en la cláusula `having`, produciendo el mismo efecto que en la cláusula `where`.

La columna de la cláusula `where` de la consulta de primer nivel, debe ser del mismo tipo que los valores devueltos por la subconsulta. Las expresiones y las funciones pueden incluirse como columnas de las consultas anidadas.

Hemos de señalar que una consulta anidada no puede contener la cláusula `order by`, ya que ésta se utiliza para ordenar el resultado de una consulta principal.

**Formato:**

```
SELECT nom_columna, expresiones, etc.
FROM nom_tabla
WHERE nom_columna operador_de_comparación
   (SELECT nom_columna(s), expresiones, etc.
   FROM nom_tabla
   [WHERE criterio de selección de las filas]
   [GROUP BY nom_columna, ...]
   [HAVING nom_columna, expresiones, etc.]...);
```

## 4.2. Devolución de un solo valor

Una subconsulta puede devolver un único valor que sirve como valor de comparación de la consulta de la cláusula `where` principal.

Una primera consulta determinaría el valor del criterio de selección, y la segunda filtraría la tabla en base a este valor.

La consulta interior puede obtener valores desde otra tabla diferente a la referenciada en la cláusula `select` exterior.

El ejemplo 4.1 nos muestra cómo utilizamos el resultado de una consulta, llamada consulta anidada, como criterio de selección de una consulta principal.

## 4.3. Combinación con operadores lógicos

Podemos combinar dos consultas anidadas mediante los operadores lógicos `or` y `and`. Las columnas devueltas por una consulta interior deberán corresponderse con las columnas utilizadas en la cláusula `where` de la consulta exterior, tanto en su número como en el

tipo de datos.

**Ejemplo** 4.1 Clientes que viven en el mismo pais que el nº 3

```
SQL> select clt_num numero, clt_nom nombre, clt_apell apellido
  2  from clientes
  3  where clt_pais =
  4        (select clt_pais
  5        from clientes
  6        where clt_num = 3);

    NUMERO NOMBRE               APELLIDO
---------- -------------------- -------------------------
         3 jean                 dupont
         4 michel               dupret
         6 marcel               souris
         8 gerad                courbon

4 filas seleccionadas.
```

### Formato:

```
WHERE nom_columna operador comparativo (consulta anidada)
                operador_lógico
      nom_columna operador comparativo (consulta anidada)
```

Los ejemplos 4.2 y 4.3 tienen dos criterios de selección en la cláusula `where` y cada uno de ellos se obtiene a partir del resultado de una consulta anidada. Vemos que hemos empleado diferentes operadores de comparación.

**Ejemplo** 4.2 Articulos con el mismo color que el nº 15 o el peso igual al nº 3

```
SQL> select art_num numero, art_nom nombre
  2  from articulos
  3  where art_col =
  4        (select art_col
  5        from articulos
  6        where art_num = 15)
  7  or art_peso =
  8        (select art_peso
  9        from articulos
 10        where art_num = 3);

    NUMERO NOMBRE
---------- --------------------
         2 calculadora
         3 calendario
        15 boligrafo lujo

3 filas seleccionadas.
```

**Ejemplo** 4.3 Articulos con el mismo color que el nº 15 o un peso superior al del nº 3

```
SQL> select art_num numero, art_nom nombre
  2  from articulos
  3  where art_col =
  4        (select art_col
  5        from articulos
  6        where art_num = 15)
  7  or art_peso >
  8        (select art_peso
  9        from articulos
 10        where art_num = 3);

    NUMERO NOMBRE
---------- --------------------
         1 impresora
         2 calculadora
         4 lampara
         5 lampara
         6 lampara
         7 lampara
        15 boligrafo lujo

7 filas seleccionadas.
```

## 4.4. Devolución de múltiples filas

La consulta anidada debe incluirse en una cláusula `where` que tenga un operador de lista (`in`) o uno de los operadores `all` (todos) o `any` (al menos uno).

Ciertas subconsultas pueden retornar una lista de valores, en tales consultas deberá usarse el operador `in` o `not in` para unir la consulta del primer nivel con la lista de valores obtenidos por la subconsulta.

El operador de comparación « = » es más eficiente cuando se conoce que la subconsulta devolverá una única fila. Esto ocurre siempre que utilicemos este operador con la clave primaria de una tabla.

Si observamos los ejemplos 4.2 y 4.3, veremos que hemos utilizado el operador de igualdad cuando teníamos la certeza de que la consulta anidada solamente podía devolvernos un único valor y hemos utilizado el operador mayor que cuando había posibilidad de que la consulta anidada devolviera más de una tupla.

## 4.5. Devolución de múltiples columnas

Puede darse también el caso de que las subconsultas devuelvan más de una columna. En tales subconsultas, el orden de las columnas expresadas en la cláusula `where` de la consulta de primer nivel, deberá corresponder al orden de columnas seleccionadas por la consulta de nivel inferior. Las columnas de la cláusula `where` del nivel superior deberán estar situadas entre paréntesis.

El ejemplo 4.4 vemos que la consulta anidad devuelve una úncia tupla que contiene dos columnas. Dichas columnas se corresponden plenamente con las expresadas en la cláusula `where` de la consulta principal.

**Formato:**

```
WHERE (nom_columna1, nom_columna2)
operador (consulta anidada)
```

**Ejemplo** 4.4 Articulos que tengan el mismo color y el mismo peso que el nº 10

```
SQL> select art_num numero, art_nom nombre
  2  from articulos
  3  where (art_col,art_peso) =
  4                (select art_col, art_peso
  5                from articulos
  6                where art_num = 10);

    NUMERO NOMBRE
---------- --------------------
        10 boligrafo
        12 boligrafo lujo

2 filas seleccionadas.
```

## 4.6. Subconsultas correlacionadas

Cuando se realiza una consulta anidada, ésta se evalúa totalmente y su resultado, una tabla temporal, se utiliza como criterio de selección en la consulta inmediata superior. El nombre de las columnas en la cláusula `select` de la consulta anidada se relaciona con la tabla definida en la cláusula `from` de dicha consulta.

Esto no siempre es tan sencillo, ya que a veces la consulta superior y la anidada correspondiente pueden estar correlacionadas. Es decir, la consulta anidada no se evalúa totalmente devolviendo una tabla temporal, sino que para cada valor de la consulta

superior realiza una evaluación de la consulta anidada.

Esto significa que un valor obtenido mediante una subconsulta depende de una variable que recibe un valor desde la consulta de nivel superior.

Las consultas anidadas correlacionadas nos pueden obligar a usar «alias» para las tablas definidas en las cláusulas `from` correspondientes.

**Formato:**

```
SELECT alias1.nom_columna1, alias1.nom_columna4
FROM nom_tabla1 alias1
WHERE criterio_de_selección
    (SELECT alias2.nom_columna1
    FROM nom_tabla1 alias2
    WHERE alias1.nom_columna3=alias2.nom_columna5);
```

Donde *nom_ tabla1* es la misma tabla en las dos consultas pero le hemos asignado diferentes «alias» para identificar qué columnas pertenecen a la tabla de la consulta anidada y qué columnas corresponden a la tabla de la consulta principal. Esta consulta se realiza tomando cada tupla de la consulta externa y entrando en la interna para evaluar su contenido y devolver o no el valor a la cláusula `where` de la consulta superior.

La cláusula `where` de la consulta anidada comprueba que el valor que viene de la consulta externa sea igual al valor en la consulta interna, y así evaluarla y devolver el, o los valores a la cláusula `where` externa.

El ejemplo 4.5 nos muestra una consulta correlacionada donde la tabla de la consulta anidada y de la consulta principal es la misma. Esto hace que tengamos que utilizar alias para las tablas y así poder identificar a los atributos implicados en dicha consulta.

## 4.7. Operadores

Hasta este momento hemos visto los operadores de comparación, los operadores lógicos, los patrones, el operador de intervalo y nos queda por ver los operadores «todo», «al menos uno» y el operador de existencia.

### 4.7.1. Los operadores `any` y `all`

La lista obtenida por una subconsulta puede tratarse mediante los operadores «todos» `all` o «al menos uno» `any`. La subconsulta debe colocarse en una cláusula `where` compuesta por un operador comparativo (`=`, `<>`, `!=`, `<`, `>`, `<=`, `>=`) seguido por uno de los operadores `all` o `any` y, tras éste, por la consulta anidada.

**Ejemplo** 4.5 Datos de los artículos que tienen el mismo color de los que pesan más de 10g

```
SQL> select art_num numero, art_nom nombre
  2  from articulos t1
  3  where art_num in
  4     (select art_num
  5     from articulos t2
  6    where t1.art_col = t2.art_col and t2.art_peso > 10);

    NUMERO NOMBRE
---------- --------------------
         1 impresora
         2 calculadora
         3 calendario
         4 lampara
         5 lampara
         6 lampara
         7 lampara
        10 boligrafo
        11 boligrafo
        12 boligrafo lujo
        13 boligrafo lujo
        14 boligrafo lujo
        15 boligrafo lujo

13 filas seleccionadas.
```

### Formato:

```
SELECT nom_columna, expresiones, etc.
FROM tabla
WHERE columna operador_de_comparación ANY ó ALL
   (Subconsulta que retorna múltiples valores);
```

El operador `any` comprueba que al menos una columna de la tabla de la consulta principal cumpla la condición de la consulta anidada para que sea verdadero el criterio de selección de la cláusula `where`.

El operador `all` comprueba que la condición de la consulta anidada se cumpla para todas las tuplas de la consulta anidada y así el criterio de la cláusula `where` será verdadero.

El ejemplo 4.6 nos muestra el uso del operador `any`. Se desea encontrar todos los artículos cuyos pesos sean mayores que el peso de cualquiera de los artículos de color blanco.

### 4.7.2. Resumen

- Con el operador `all` la condición se cumple si la comparación se verifica para todos los valores invocados por la consulta anidada.
- Con el operador `any` la condición se cumple si la comparación se verifica para uno al menos de los valores invocados por la consulta anidada.
- El empleo de `any` y/o `all` es bastante delicado y puede llevar fácilmente a cometer errores. Es preferible emplear en su lugar consultas que hagan uso de funciones sobre los grupos o sobre las consultas de existencia.
- El operador `in` es equivalente a «`= any`».
- El operador `not in` es equivalente a «`!= all`» o «`<>all`».

**Ejemplo** 4.6 Artículos que pesen más que cualquiera de los de color blanco

```
SQL> select art_num numero, art_nom nombre
  2  from articulos
  3  where art_peso > any
  4     (select art_peso
  5     from articulos
  6    where art_col = 'blanco');

   NUMERO NOMBRE
---------- --------------------
         3 calendario
         2 calculadora
         1 impresora
         4 lampara
         5 lampara
         6 lampara
         7 lampara
        10 boligrafo
        11 boligrafo
        12 boligrafo lujo
        13 boligrafo lujo
        14 boligrafo lujo
        15 boligrafo lujo

13 filas seleccionadas.
```

### 4.7.3. El operador `exists`

El operador existencial `exists` se emplea para realizar consultas de existencia, consultas en que necesitemos utilizar el cuantificador universal «para todo» o la operación de división del álgebra relacional.

La subconsultas de existencia debe colocarse dentro de una cláusula `where` y después de la palabra clave `exists`. La condición se cumple si la consulta anidada da un resultado de una fila por lo menos.

**Formato:**

```
SELECT columnas
FROM tabla
WHERE EXISTS
   (Subconsulta que retorna uno o múltiples valores);
```

La existencia de filas de la consulta interna se puede utilizar para cualificar filas en una consulta exterior. La consulta anidada será cierta si encuentra una o más filas. En caso contrario, la cláusula `where` no sería cierta y no se ejecutaría la consulta superior.

La cláusula `select` de la consulta anidada no tiene necesariamente que llevar especificado los nombres de las columnas, ya que lo único que se chequea es si existe al menos una fila que cumpla la condición, luego debemos utilizar el asterisco (*). En el ejemplo 4.7 observamos que se nos mostrará la tabla *Proveedores* siempre que exista el proveedor *Sanjita*.

## 4.8. Consulta anidada en una cláusula `having`

Una cláusula `having` permite especificar un criterio de selección que debe verificar los grupos. Este criterio puede, a su vez, depender del resultado de una consulta anidada.

**Formato:**

```
HAVING {columna | función} {operador_de_comparación | lista}
{(subconsulta anidada) | constante | columna | función}
```

El ejemplo 4.8 nos muestra cómo hacemos una selección de entre todos los grupos obtenidos. La selección se hace en base al criterio expresado en la cláusula `having`.

**Ejemplo** 4.7 Si tenemos un proveedor que se llame «sanjita» muestra todos los proveedores

```
SQL> select prv_num numero, prv_nom nombre
  2  from proveedores
  3  where exists
  4              (select *
  5              from proveedores
  6              where prv_nom = 'sanjita');

    NUMERO NOMBRE
---------- -------------------------
         1 catio electronic
         2 estilograficas reunidas
         3 mecanica de precision
         4 sanjita
         5 electrolamp

5 filas seleccionadas.
```

**Ejemplo** 4.8 Colores de los artículos cuya media de pesos es superior a la media de todos los artículos

```
SQL> select art_col color
  2  from articulos
  3  group by art_col
  4  having avg(art_peso) >=
  5         (select avg(art_peso)
  6         from articulos);

COLOR
-------
blanco
verde

2 filas seleccionadas.
```

# Capítulo 5

# Consultas a múltiples tablas

Presentamos en este capítulo cómo podemos hacer uso de los operadores de reunión y de los tradicionales de conjuntos para obtener información procedente de varias tablas. También se muestran unas recomendaciones para optimizar nuestras consultas.

## 5.1. Búsquedas multitablas

### 5.1.1. Introducción

Las consultas que realizamos normalmente requieren información que están repartida o contenida en más de una tabla. Para poder hacer esto, necesitamos que una columna contenga la misma información en diferentes tablas y sea este campo el que nos sirva de unión entre las diversas tablas.

Estas columnas que nos sirven para recabar información entre tablas se denominan «claves foráneas» y la operación que nos permite hacer esto corresponde a la operación de «producto natural» o *join*[1].

### 5.1.2. Producto cartesiano

La operación de «producto cartesiano» une cada fila de una tabla con cada una de las filas de la otra tabla.

[1] En álgebra relacional esta operación se representa con el símbolo $\bowtie$.

**Formato:**

```
SELECT tab1.nom_colum, tab2.nom_colum,...
FROM tab1, tab2, ...;
```

El proceso que se sigue es el siguiente: si la cláusula `from` incluye varias tablas, se comienza realizando el producto cartesiano de las mismas. Dicho producto es, a su vez, una tabla derivada de la concatenación de las filas de una tabla original con las filas de otra (o de otras) también original; es decir, cada fila de una tabla es emparejada sucesivamente con todas las filas de la otra. Si la primera tabla contiene $M$ filas, y la segunda $N$, el producto cartesiano de estas dos tablas estará formado por una tabla con $M$ $x$ $N$ filas. Posteriormente, se proyectarán las columnas seleccionadas en la cláusula `select` y éstas aparecerán en el mismo orden especificado. La tabla resultante de esta operación contiene información que no es cierta.

La consulta del ejemplo 5.1 nos da una tabla donde aparecen los datos principales de los artículos cuyos números sean menores que 4 y de sus proveedores. Podemos comprobar, observando este ejemplo, que los proveedores están asignados a todos los artículos, cuando sabemos que realmente un artículo sólo lo suministra un proveedor. Si obsevamos más detenidamente dicho ejemplo, vemos que en primer lugar hemos hecho un producto cartesiano entre las tablas *Proveedores* y *Articulos* y, posteriormente, hemos aplicado un criterios de selección «dodne el nº del artículos sea menor que 4». Esta dos operaciones se denominan *producto theta*[2]

### 5.1.3. Producto natural

Esta operación se realizará sobre las columnas que contienen la misma información en tablas distintas ya que si esto no lo tenemos en cuenta, lo que realizaremos es un «producto cartesiano» con las consecuencias de pérdida de información, información redundante e inconsistencia de los datos. Si lo que queremos es reunir información entre tablas pero que el resultado sea correcto, debemos realizar un «producto natural» o *join* entre tabla y así no ocurrirá lo mismo que lo visto en el ejemplo 5.1.

Para ello, debemos tener al menos una columna común en las tablas sobre las que vamos a realizar esta operación, o sea, la unión de tablas se hace mediante la igualdad de valores entre esas columnas. Las columnas comunes se corresponden con las claves foráneas.

[2]En álgebra relacional se expresa con el símbolo $\bowtie_\theta$.

**Ejemplo** 5.1 Producto cartesiano entre proveedores y pesos

```
SQL> select art_num numero, art_nom nombre, proveedores.*
  2  from articulos, proveedores
  3  where art_num  < 4;

    NUMERO NOMBRE                   PRV_NUM PRV_NOM
---------- -------------------- ---------- -------------------------
         1 impresora                     1 catio electronic
         2 calculadora                   1 catio electronic
         3 calendario                    1 catio electronic
         1 impresora                     2 estilograficas reunidas
         2 calculadora                   2 estilograficas reunidas
         3 calendario                    2 estilograficas reunidas
         1 impresora                     3 mecanica de precision
         2 calculadora                   3 mecanica de precision
         3 calendario                    3 mecanica de precision
         1 impresora                     4 sanjita
         2 calculadora                   4 sanjita
         3 calendario                    4 sanjita
         1 impresora                     5 electrolamp
         2 calculadora                   5 electrolamp
         3 calendario                    5 electrolamp

15 filas seleccionadas.
```

Para realizar un producto natural basta con especificar en la cláusula `select` las columnas buscadas y escribir una condición de igualdad en la cláusula `where` entre las columnas que permiten la unión de las tablas. La cláusula `from` debe contener el nombre de todas las tablas.

**Formato:**

```
SELECT tab1.nom_colum, tab2.nom_colum, ...
FROM tab1, tab2, ...
WHERE tab1.colum = tab2.colum AND ... ;
```

Vemos que la cláusula `from` contiene más de una tabla, luego se comenzará realizando el producto cartesiano. Posteriormente, se descartarán todas las filas resultantes del producto que no cumplan el criterio de unión contenido en la cláusula `where`. La tabla así formada será posteriormente ordenada según las columnas indicadas en la cláusula `order by`. La búsqueda concluye con la proyección de las columnas que se especifican en la cláusula `select`.

Esta descripción del proceso se corresponde con una concepción típica del razonamiento humano. En la realidad, el optimizador desempeña un importante papel, efectuando, de forma inmediata, las selecciones y proyecciones sobre las tablas iniciales, de modo que la tabla resultante del producto cartesiano jamás se genera en el sistema.

Por ejemplo, si realmente lo que queremos es saber quien suministra realmente cada artículos, la consulta sería la mostrada en el ejemplo 5.2. Sólo se van a seleccionar las tuplas en las que el código del proveedor sea el mismo en las dos tablas.

**Ejemplo** 5.2 Datos de los artículos y del proveedor que lo suministra

```
SQL> select art_num numero, art_nom nombre, proveedores.*
  2  from articulos, proveedores
  3  where art_prv = prv_num;

    NUMERO NOMBRE                  PRV_NUM PRV_NOM
---------- -------------------- ---------- -------------------------
         2 calculadora                   1 catio electronic
        10 boligrafo                     2 estilograficas reunidas
        11 boligrafo                     2 estilograficas reunidas
        12 boligrafo lujo                2 estilograficas reunidas
        15 boligrafo lujo                2 estilograficas reunidas
        14 boligrafo lujo                2 estilograficas reunidas
        13 boligrafo lujo                2 estilograficas reunidas
         8 pesacartas 1-500              3 mecanica de precision
         9 pesacartas 1-1000             3 mecanica de precision
         1 impresora                     4 sanjita
         3 calendario                    4 sanjita
         4 lampara                       5 electrolamp
         5 lampara                       5 electrolamp
         6 lampara                       5 electrolamp
         7 lampara                       5 electrolamp

15 filas seleccionadas.
```

La cláusula `where` puede soportar condiciones de selección de tuplas, utilizando los operadores ya vistos, junto con los criterios de unión entre tablas (producto natural) siempre que estén combinados con operadores lógicos. Por ejemplo, en el ejemplo 5.1 aparecen artículos con proveedores que no son realmente quienes los suministra. Esto lo podemos subsanar realizando un producto natural más el mismo criterio de selección que hemos aplicado en dicho ejemplo (ver ejemplo 5.3).

### 5.1.4. Resumen

A modo de resumen, podemos resaltar los siguientes puntos:

- Las tablas que van a formar parte de un producto natural se especifican en la cláusula `from`.
- La cláusula `where` especifica los criterios para unir las tablas, así como el de selección de los registros.
- Las columnas que se quieren proyectar de una u otra tabla irán en la cláusula `select`.

- Puede utilizarse la expresión *nom_tabla.** para obtener todas las columnas de una tabla en los productos cartesiano y natural.
- Las columnas con igual nombre en ambas tablas deberán llevar un prefijo con el nombre de la tabla a la que pertenece, o bien un «alias», separado por un punto.
- Si los nombres de las columnas son únicos en todas las tablas del producto, no será necesario usar como prefijo el nombre de su tabla.
- Pueden realizarse tantos productos sobre tablas como se desee.
- El «criterio de correspondencia» para las tablas se denomina predicado de unión o criterio de unión.
- Pueden utilizarse más de un par de columnas para especificar la condición de unión entre dos tablas cualquiera.
- Cuando se quiere realizar el producto natural entre $n$ tablas, es necesario tener al menos *n-1* condiciones de unión para evitar productos cartesianos.

**Ejemplo** 5.3 Datos de los artículos y sus proveedores cuyos nº son menores que 4

```
SQL> select art_num numero, art_nom nombre, proveedores.*
  2  from articulos, proveedores
  3 where art_prv = prv_num and art_num < 4;

    NUMERO NOMBRE                PRV_NUM PRV_NOM
---------- -------------------- ---------- -------------------------
         2 calculadora                   1 catio electronic
         1 impresora                     4 sanjita
         3 calendario                    4 sanjita

3 filas seleccionadas.
```

### 5.1.5. Unión externa

Esta unión, también llamada *outer-joins*, permite resolver problemas del siguiente tipo: se desea obtener una lista de filas en función de un criterio que es función de la unión con otra tabla, si la condición de unión no se cumple, la fila normalmente no aparece en el resultado. Pues bien, para visualizar todas las filas, incluidas las que no cumplen este criterio, es preciso realizar una unión «externa».

En Oracle una unión «externa» se identifica mediante la presencia del símbolo (+) en la cláusula `where`, después del nombre de la columna donde pueden no darse valores determinados.

La unión externa devuelve todas las filas recuperadas por medio de un producto natural y, además, añade las tuplas de una tabla que no están emparejadas con las tuplas de la otra tabla.

Una unión externa hace que el *SQL* suministre columnas nulas o vacías en una tabla para las filas que no cumplen la condición de unión.

Por ejemplo, queremos saber los datos de los articulos y la fecha en que se han vendido cuyos números sean mayores que 4. Si un artículo aún no se han vendio también queremos que aparezcan sus datos. Esto lo podemos ver resuelto en el ejemplo 5.4. Obsérvese que le hemos dado formato al atributo fecha para que no aparezca tal como se han introducido los valores. Ver capítulo 6.

**Ejemplo** 5.4 Datos de artículos cuyos nº es mayor que 4 que se han vendido o no

```
SQL> select art_num numero, art_nom nombre,
  2         to_date(vnt_fch,'yymmdd') fecha
  3  from articulos,ventas
  4  where art_num = vnt_art(+) and art_num > 3;

    NUMERO NOMBRE               FECHA
---------- -------------------- --------
         4 lampara              06/01/91
         4 lampara              09/01/91
         5 lampara
         6 lampara              11/01/91
         7 lampara
         8 pesacartas 1-500
         9 pesacartas 1-1000    11/01/91
        10 boligrafo            06/01/91
        10 boligrafo            11/01/91
        11 boligrafo            06/01/91
        12 boligrafo lujo       10/01/91
        13 boligrafo lujo       10/01/91
        13 boligrafo lujo       22/02/92
        14 boligrafo lujo       06/01/91
        15 boligrafo lujo       09/01/91
        15 boligrafo lujo       09/05/01

16 filas seleccionadas.
```

### 5.1.6. Autouniones

A veces queremos realizar una unión entre dos tablas que no son distintas, es decir, las dos tablas que queremos unir son, en realidad, la misma tabla. Esto se produce cuando el criterio de unión afecta al valor de una columna en relación al valor de esta misma columna en otra fila de la misma tabla. Una tabla puede estar unida dentro de sí misma en columnas que contengan el mismo tipo de información. Una autounión une filas de una tabla con ella misma o con otras filas en la misma tabla.

**Formato:**

```
SELECT alias1.columna, alias2.columna, ...
FROM tabla1 alias1, tabla1 alias2
WHERE alias1.columna = alias2.columna;
```

Cuando realizamos este tipo de unión es necesario utilizar «alias» pues tenemos que diferenciar cuando estamos haciendo referencia a una tabla y cuando hacemos referencia a la otra tabla, sabiendo que las dos tablas son la misma.

**Ejemplo** 5.5 Autouniones

```
SQL> select t1.clt_num numero, t1.clt_nom nombre, t1.clt_apell
  2  from clientes t1, clientes t2
  3  where t1.clt_pais = t2.clt_pais and t2.clt_nom like 'm%'
  4  and t2.clt_num > 10;

    NUMERO NOMBRE               CLT_APELL
---------- -------------------- -------------------------
         1 margarita            borras
         2 miguel               perez
         5 antoni               llopis
         7 pablo                goqi
         9 consuelo             roman
        10 pau                  roca
        11 jorge                mancha
        12 pablo                curro
        13 diego                cortes
        14 joaquin              fernandez
        15 jacinto              duran

11 filas seleccionadas.
```

El ejemplo 5.5 nos muestra los datos de los clientes que viven en el mismo país de los que su número es mayor que 10 y su nombre comienza por *m*.

### 5.1.7. Resumen

- Una tabla puede unirse consigo misma como si fueran dos tablas separadas.
- La autounión es útil para unir una fila de una tabla con otra de la misma tabla.
- Al igual que en cualquier otra unión, la unión se realiza entre columnas que contienen el mismo tipo de información.
- A la tabla se deberá asignar un «alias» para sincronizar qué columnas van a ser obtenidas de la tabla.

### 5.1.8. Uniones no comunes

Cabe observar que la unión entre tablas puede quedar especificada mediante cualquiera de los operadores de comparación (=, <>, !=, <, >, <=, >=, between, in, like). También podemos llamar a esta unión «unión mediante la no igualdad».

**Ejemplo** 5.6 Clasificar los artículos cuyos nº estén comprendidos entre 2 y 10, según su peso

```
SQL> select art_num numero, art_nom nombre, peso_nom tipo
  2  from articulos, pesos
  3  where art_peso between peso_min and peso_max
  4    and art_num between 2 and 10;

    NUMERO NOMBRE               TIPO
---------- -------------------- ---------
         3 calendario           leve
        10 boligrafo            leve
         2 calculadora          ligero
         4 lampara              medio
         5 lampara              medio
         6 lampara              medio
         7 lampara              medio

7 filas seleccionadas.
```

El ejemplo 5.6 nos muestra cómo podemos hacer estos tipos de uniones. Según el peso de cada artículo los hemos clasificados utilizando la clasificación expresada en la tabla *Pesos*.

## 5.2. Operadores conjuntistas

Las operaciones conjuntistas son cuatro: unión, intersección, diferencia y producto cartesiano (ya visto anteriormente) de conjuntos. Sólo la unión y el producto cartesiano es común a todos los *SGBD*. Las otras dos operaciones son muy potentes pero sólo se dan de forma excepcional en los *SGBD*. En cuanto al producto cartesiano, su realización es automática cuando se hacen consultas que afectan a varias tablas y no indicamos un criterio de unión entre tablas.

Hay ocasiones en las que se necesita combinar informaciones de tipo similar. Un ejemplo clásico de esto puede ser dos o más listas de direcciones que se unen antes de empezar una campaña de publicidad por correos. Dependiendo del propósito particular de la campaña se puede enviar cartas de una de las siguientes maneras:

- A todo el mundo de ambas listas (mientras se evite enviar dos cartas a alguien que se encuentra en las dos listas)

- Sólamente a aquellas personas que están en ambas listas.

- Sólamente a aquellas personas que están en una lista y no en la otra.

Para realizar estas tres combinaciones de listas, Oracle utiliza los operadores `union`, `intersect` y `minus`.

### 5.2.1. El operador `union`

Este operador efectúa la unión de los resultados de dos consultas `select`. Esto significa que, partiendo de dos tablas temporales, se puede crear una tercera que contenga el conjunto de filas de las dos tablas iniciales, eliminando o no las filas repetidas. Si al operador `union` le sigue la palabra clave `all`, entonces se seleccionan también las filas idénticas.

En las dos tablas el tipo de columnas debe ser compatible. Si las tablas unidas pertenecen a subconjuntos de la misma tabla, el uso de este operador es completamente equivalente al de un operador lógico `or` colocado entre dos condiciones.

Este operador retorna todas las filas de múltiples consultas.

**Formato:**

```
SELECT nom_columna, ...
FROM nom_tabla, ...
[WHERE condiciones]
     {UNION | UNION ALL}
SELECT nom_columna, ...
FROM nom_tabla, ...
[WHERE condiciones]
[ORDER BY {nom_columna | entero | alias];
```

La cláusula `order by` no afecta a la segunda consulta, sino al resultado global.

En el ejemplo 5.7 se ha obtenido una lista con todos los números de los artículos independientemente de sí se han vendido o no.

**Ejemplo** 5.7 Los números de todos los artículos que se han vendido o no

```
SQL> select art_num numero
  2  from articulos
  3  union
  4  select vnt_art
  5  from ventas;

    NUMERO
----------
         1
         2
         3
         4
         5
         6
         7
         8
         9
        10
        11
        12
        13
        14
        15

15 filas seleccionadas.
```

### 5.2.2. El operador `intersect`

Este operador encuentra las tuplas comunes en las dos consultas.

**Formato:**

```
SELECT nom_columna, ...
FROM nom_tabla, ...
[WHERE condiciones]
     INTERSECT
SELECT nom_columna, ...
FROM nom_tabla, ...
[WHERE condiciones]
[ORDER BY {nom_columna | entero | alias];
```

En el ejemplo 5.8 solamente aparecen los números de los artículos que se han vendido.

**Ejemplo** 5.8 Los números de todos los artículos que se han vendido

```
SQL> select art_num numero
  2  from articulos
  3  intersect
  4  select vnt_art
  5  from ventas;

    NUMERO
----------
         1
         2
         3
         4
         6
         9
        10
        11
        12
        13
        14
        15

12 filas seleccionadas.
```

### 5.2.3. El operador `minus`

El operador `minus` obtiene todas las filas seleccionadas por la primera consulta pero no por la siguiente.

**Formato:**

```
SELECT nom_columna, ...
FROM nom_tabla, ...
[WHERE condiciones]
     MINUS
SELECT nom_columna, ...
FROM nom_tabla, ...
[WHERE condiciones]
[ORDER BY {nom_columna | entero | alias];
```

El ejemplo 5.9 nos muestra solamente los artículos que aún no se han vendido y podemos hacer una comparación con los ejemplos 5.7 y 5.8 ya mostrados.

### 5.2.4. Resumen

Se han detallado los tres operadores conjuntistas y vamos a detallar algunos detalles a tener en cuenta. Al combinar dos tablas, Oracle no se preocupa de los nombres de las columnas que hay a cada lado del operador de combinación. Es decir, Oracle requiere que cada instrucción `select` sea válida y tenga columnas válidas para su propia tabla o tablas, pero los nombres de las columnas de la primera instrucción `select` no tienen por qué ser los mismos que los de la segunda. Lo que sí exige Oracle es que se cumplan las siguientes normas:

- Las instrucciones `select` deben tener el mismo número de columnas. Si las dos tablas que se consultan tienen un número diferente de columnas seleccionadas, puede seleccionar cadenas en lugar de columnas para conseguir que las listas de columnas de las dos consultas coincidan.
- Las columnas correspondientes en las instrucciones `select` deben tener el mismo tipo de datos (no es necesarios que tengan la misma longitud).

Para ordenar la salida, hay que tener en cuenta que Oracle usa los nombres de columna de la primera instrucción `select` para proporcionar los resultados de la consulta. Por tanto, en `order by` sólo se pueden emplear los nombres de columna de la primera instrucción `select`.

## 5.3. Consejos prácticos para escribir una consulta

Vamos a resumir los requisitos necesarios para realizar una consulta de la mejor manera posible.

- Determinar las tablas que han de participar, e incluirlas en la cláusula `from`.
- Decidir qué columnas se desean visualizar o proyectar y su orden de aparición para incluirlas en la cláusula `select`.
- Si la cláusula `select` conlleva funciones sobre grupos, será necesario una cláusula `group by` que agrupe todas las columnas citadas en la cláusula `select`, salvo las funciones en cuestión.
- Determinar las condiciones de selección que limitan la consulta. Las condiciones con proyección sobre grupos deben figurar en una cláusula `having`; las que se refieren a valores individuales, aparecerán en una cláusula `where`.
- Cuando se tenga que emplear una función sobre grupos en una cláusula `where`, o cuando haya necesidad de un valor de una columna de otra tabla distinta, hay que realizar una consulta anidada.

- Para fusionar los resultados procedentes de dos cláusulas `select`, basta con emplear el operador `union`.
- Hay que precisar el orden de aparición de las filas del resultado de una consulta con una cláusula `order by`.
- Los posibles «alias» que demos a las columnas, deberán estar en la primera consulta cuando usamos los operadores conjuntistas.

**Ejemplo** 5.9 Los números de todos los artículos que no se han vendido

```
SQL> select art_num numero
  2  from articulos
  3  minus
  4  select vnt_art
  5  from ventas;

    NUMERO
----------
         5
         7
         8

3 filas seleccionadas.
```

## 5.4. Optimización de las consultas

### 5.4.1. Introducción

Con el lenguaje *SQL* es posible expresar una misma consulta de distintos modos, aunque no siempre es fácil decidir la mejor forma de expresar una consulta compleja. ¿Es mejor usar una unión de tablas o una consulta anidada? ¿Interesa el empleo del operador `exists`? A todas estas preguntas vamos a intentar dar respuesta en este apartado. Las reglas que siguen facilitarán a buen seguro la tarea del programador novel.

Para ilustrar nuestro punto de vista examinemos una consulta sencilla que afecta a dos tablas: visualizar los datos de los clientes que han comprado el artículo nº 3.

Existen, al menos, cuatro formulaciones diferentes para esta consulta: una unión entre tablas, una consulta anidada, una consulta correlacionada y una consulta basada en el operador `exists`[3].

1) *Unión entre tablas*: establecer un producto cartesiano entre las tablas *Clientes* y *Ventas* mediante una igualdad en el número del cliente en ambas tablas, teniendo en cuenta

[3] Ver preguntas 5.4.4 y 5.4.5.

que sólo necesitamos los datos de aquellos clientes que han comprado el artículo nº 3. Ver ejemplo 5.10.

2) *Consulta anidada*: listar a todos los clientes cuyo número se incluye en el conjunto de los que han comprado el artículo nº3. Ver ejemplo 5.11.

3) *Consulta correlacionada*: listar a los clientes cuya número pertenezca al conjunto de números de clientes que han comprado el artículo nº 3. Ver ejemplo 5.12.

4) *Consulta de existencia*: listar a los clientes si existen que han comprado el artículo nº 3 en la tabla *Ventas*. Ver ejemplo 5.13.

### 5.4.2. Eficacia

Autorizar varias formulaciones para una misma consulta puede considerarse una ventaja del lenguaje, si cada usuario es libre de elegir su propia manera de plantear el problema. Por desgracia, en la situación actual de las versiones del *SQL*, estas distintas formulaciones arrojan tiempos de ejecución con frecuencia muy distintos y a veces hasta inesperados. Ello obliga al usuario a comprobar el comportamiento de su versión de *SQL* para cada tipo de consulta y para cada formulación, si pretenden optimizar sus aplicaciones.

La razón de ello estriba en que las técnicas utilizadas por el optimizador para organizar la consulta depende de la formulación de la consulta y del *SGBDR*[4].

### 5.4.3. Unión entre tablas o consulta anidada

Como se ve en el ejemplo propuesto en el apartado 5.4.1, la unión ofrece con frecuencia una formulación más concisa, pero quizás menos clara. Las subconsultas tienen la ventaja de descomponer el problema, lo que hace que las consultas sean más fáciles de escribir y de comprender.

> *La unión es indispensable para transcribir una consulta que pretende recuperar información derivada de distintas tablas.*

En efecto, una consulta mediante subconsulta permite sólo la visualización de informaciones situadas en la tabla exterior.

Por el contrario, las uniones resultan inoperantes en determinadas consultas multitablas.

> *Cuando la consulta consiste en recuperar información de una sola tabla, pero siendo necesaria otra para condicionar la selección de las filas de la primera, siempre es mejor el uso de una consulta anidada.*

[4]Sistema de Gestión de Bases de Datos Relacionales.

Así, se puede comprobar fácilmente que no es posible ninguna condición de unión para solucionar una consulta que efectúa una operación conjuntista de diferencia. Esto es lógico, dado que la consulta niega el enlace entre las tablas. Por el contrario, la solución es inmediata con el operador `in` o con `exists`, dado que basta con invertir las condiciones en el ejemplo anterior (`not` delante de `in` o `exists`).

### 5.4.4. Empleo del cuantificador existencial

Recordemos que la expresión «`exists` (`select` * `from` ...)» es un predicado que vale verdadero (`true`) si la consulta tiene resultado no vacío. Por lo tanto, una fila de la consulta exterior forma parte del resultado «si existe» en la tabla interior «al menos una» fila que satisface las condiciones de la cláusula `where` de la subconsulta. Cuando la expresión está precedida de `not`, el predicado se evalua como verdadero si no existe ninguna fila de la tabla interior que satisface las condiciones exigidas.

Examinando las consultas que se pueden hacer a una base de datos, se llega fácilmente a la conclusión de que toda subconsulta precedida por el operador `in` puede ser expresada con el operador `exists`. Pero lo inverso no es verdad; algunas consultas no pueden expresarse sino con la ayuda del operador `exists`.

Como conclusión, el operador `in` ante una subconsulta puede ser sustituido por `exists`. El operador `in` es, pues, redundante en *SQL*, si se exceptúa el hecho de que permite expresar una consulta de manera conjuntista, algo que algunos usuarios prefieren en lugar del enfoque predicativo de `exists`. El operador `in` a veces es necesario, porque los optimizadores no son perfectos y el tiempo de ejecución de la consulta con `exists` puede ser mucho mayor en algunos casos.

Por otra parte, `exists` es necesario para expresar el cuantificador universal *for all*, muy conocido en lógica, pero que no está representado directamente en *SQL*. En general, la expresión «para todo» puede formularse con la ayuda de uno o dos `not exists`.

### 5.4.5. Solución al ejercicio propuesto

1) Producto natural entre las tablas: se realiza el producto natural entre las tablas *Clientes* y *Ventas*, y se seleccionan las tuplas de los clientes que han comprado el artículo 3.

**Ejemplo** 5.10 Producto natural

```
SQL> select clt_num numero, clt_nom nombre, clt_apell apellido
  2  from clientes, ventas
  3  where clt_num = vnt_clt and vnt_art = 3;

    NUMERO NOMBRE               APELLIDO
---------- -------------------- -------------------------
         6 marcel               souris
        13 diego                cortes

2 filas seleccionadas.
```

2) Consulta anidada: se realiza una consulta anidada por obtener los nº de clientes que han comprado el artículo nº 3. Esta consulta sirve de criterio de selección para obtener los datos de los clientes que lo han comprado.

**Ejemplo** 5.11 Consulta anidada

```
SQL> select clt_num numero, clt_nom nombre, clt_apell apellido
  2  from clientes
  3  where clt_num in
  4     (select vnt_clt
  5     from ventas
  6     where vnt_art = 3);

    NUMERO NOMBRE               APELLIDO
---------- -------------------- -------------------------
         6 marcel               souris
        13 diego                cortes

2 filas seleccionadas.
```

3) Consulta correlacionada: para cada tupla de la tabla *Clientes* se comprueba mediante una consulta correlacionada si ha comprado el artículo nº 3.

**Ejemplo** 5.12 Consulta correlacionada

```
SQL> select clt_num numero, clt_nom nombre, clt_apell apellido
  2  from clientes
  3  where 3 in
  4     (select vnt_art
  5     from ventas
  6     where vnt_clt = clt_num);

    NUMERO NOMBRE               APELLIDO
---------- -------------------- -------------------------
         6 marcel               souris
        13 diego                cortes

2 filas seleccionadas.
```

4) Consulta de existencia: para cada cliente de la tabla *Clientes* se comprueba mediante una consulta si es verdad (existe) que el cliente ha comprado el artículo nº 3.

**Ejemplo** 5.13 Consulta de existencia

```
SQL> select clt_num numero, clt_nom nombre, clt_apell apellido
  2  from clientes
  3  where exists
  4     (select *
  5     from ventas
  6     where vnt_clt = clt_num and vnt_art = 3);

    NUMERO NOMBRE               APELLIDO
---------- -------------------- -------------------------
         6 marcel               souris
        13 diego                cortes

2 filas seleccionadas.
```

# Capítulo 6

# Tratamiento de fechas

Oracle proporciona un buen método para el tratamiento de las fechas. Este capítulo nos muestra las diferentes maneras que tenemos de expresarlas.

## 6.1. Aritmética de fechas

Oracle posee la capacidad de almacenar y calcular fechas, así como el número de segundos, minutos, horas, días, meses y años entre fechas dadas. También tiene la facilidad de formatear las fechas de cualquier manera que se pueda concebir, desde un simple «15-MAR-97» a «Decimoquinto día del mes de marzo en el año 764 del reinado de Luis IX».

Para almacenar y operar con fechas, Oracle proporciona el tipo de datos `date` que posee unas propiedades únicas y especiales y que se almacena en un formato interno especial que incluye, no sólo el mes, día y año, sino también horas, minutos y segundos.

Los beneficios que proporcionan todos estos detalles deberían ser obvios. Así podemos siempre almacenar automáticamente la fecha, la fecha y hora, la fecha, hora y minuto, o la fecha, hora, minuto y el segundo de cualquier suceso.

Los productos *SQL*Plus* y *SQL* reconocen las columnas que tienen como tipo de dato el `date` y comprenden las instrucciones aritméticas que se hacen con ellas, denominadas aritmética de fechas.

Tenemos que tener en cuenta que si sumamos un 1 a una fecha se obtiene otra fecha: el siguiente día. Mientras que si restamos una fecha de otra se obtiene un número: la cantidad de días entre las dos fechas.

Sin embargo, puesto que las fechas de Oracle contienen horas, minutos y segundos, realizar una aritmética de fechas puede provocar un poco de confusión, por ejemplo Oracle puede decir que la diferencia entre hoy y mañana es de «.5339 días». La razón de que se obtenga este número fraccional de días entre hoy y mañana es que Oracle guarda también las horas, los minutos y los segundos junto a las fechas por lo que la diferencia entre fechas no es un día exacto.

El formato por defecto de la fecha se incluye en el fichero de arranque del sistema como un parámetro más. Cualquier usuario puede cambiar el formato para una sesión de trabajo, o bien de forma permante incluyendo las órdenes en su fichero de arraque (`login.sql`).

En la tabla 6.1 vemos la expresión, la definición y el tipo de dato que resulta de hacer algunas operaciones con las fechas. Suponemos que «x» representa a un número entero.

| **Expresión** | **Definición** | **Tipo de dato devuelto** |
|---|---|---|
| fecha + x | Añadiría un número de días. | `date` |
| fecha + x/24 | Añadiría un número de horas. | `date` |
| fecha - x | Restaría un número de días. | `date` |
| fecha - x/24 | Restaría un número de horas. | `date` |
| fecha - fecha | Determina el número de días entre ambas fechas. | numérico |

Tabla 6.1: Operaciones con fechas

La tabla *dual*[1] proporcionada por Oracle, es la tabla que se emplea para probar las funciones o realizar cálculos rápidos.

## 6.2. La función `sysdate`

Esta función es bastante importante ya que Oracle la utiliza para extraer del sistema operativo la fecha y la hora actual[2]. No necesita argumentos.

Por ejemplo para saber la fecha de hoy, debemos usar esta función junto con la tabla `dual`, ya que el sistema debe devolvernos la consulta en formato tabla.

[1] Ver capítulo 3.

[2] Esta función se puede usar igual que cualquier otra función, ver capítulo 3.

**Ejemplo** 6.1 Fecha actual

```
SQL> select sysdate
  2  from dual;

SYSDATE
--------
18/11/04

1 fila seleccionada.
```

El ejemplo 6.1 nos muestra el resultado de la utilización de dicha función. El formato en que nos aparece es el formato por defecto que esté definido en el fichero de arranque de Oracle.

## 6.3. Funciones de fecha

Las funciones de fecha operan con valores de tipo de datos `date` y devuelven valores de este mismo tipo de datos, excepto la función `months_between` que devuelve un dato de tipo numérico.

En la tabla 6.2 se muestran las funciones de fechas más usuales.

| **Función** | **Uso** |
|---|---|
| `add_months(d,n)` | Añade *n* meses a la fecha *d*. |
| `last_day(d)` | Devuelve la fecha del último día del mes que contiene la fecha *d*. |
| `months_between(d1,d2)` | Devuelve el número de meses entre las fechas *d1* y *d2*. |
| `next_day(d,char)` | Devuelve la fecha del siguiente día después de la fecha dada donde `char` es el nombre de los días de la semana. |
| `round(d[,'formato'])` | Devuelve la fecha (en formato por defecto) más cercana (anterior o posterior) según la precisión que le indiquemos con *formato*. |
| `sysdate` | Devuelve la hora y el día actual. No requiere argumentos. |
| `trunc(d[,'formato'])` | Devuelve la fecha (en formato por defecto) anterior que primero cumple con la precisión indicada con *formato*. |

Tabla 6.2: Funciones de fecha

La función `sysdate` no se puede usar en la cláusula de restricción de integridad `check`[3].

## 6.4. Formatos y conversión de fechas

Anteriormente hemos dicho que las fechas se pueden expresar en distintos formatos. También podemos convertir cadenas de caracteres en fechas y viceversa[4].

Las funciones `to_char` y `to_date` se parecen en que ambas tienen poderosas capacidades para formatear, a parte de realizar una conversión de datos.

El formato es el siguiente:

**Formato:**

```
TO_CHAR(fecha[,'formato'])
TO_DATE(cadena[,'formato'])
```

**fecha** La fecha debe ser una columna definida como tipo de datos `date` en Oracle y no puede ser una cadena ni en el caso de que se encuentre en el formato `dd-mm-yy`. La única manera de usar una cadena donde aparece *fecha* en la función `to_char` es encerrándola en el interior de una función `to_date`.

**cadena** Es una cadena literal, un número literal o una columna de la base de datos que contiene cadenas o números. En todos los casos, menos en uno, el formato se corresponderá con el descrito con *formato*. Sólo en el caso en que la cadena esté en el formato definido en el fichero de arranque, éste se puede omitir.

**formato** Es una colección de más de cuarenta opciones, que se pueden combinar, virtualmente, de un número infinito de formas. En la tabla 6.3 se muestran las distintas opciones con sus respectivos efectos.

Los formatos de fechas vistos se usan tanto con la función `to_char` como con la función `to_date`.

El ejemplo 6.2 nos muestra los datos de los clientes que han relizado alguna compra y la fecha en que la realizaron. La fecha aparece con formato por defecto.

El ejemplo 6.3 nos muestra la fecha y la hora actual según el formato especificado. Para ello, la función `sysdate` debemos convertirla al tipo de datos `char`.

[3] Creación de una tabla con el lenguaje de definición de datos.
[4] Ver capítulo 3.

**Ejemplo** 6.2 Los clientes y las fechas de compras

```
SQL> select vnt_clt numero, clt_nom nombre, clt_apell apellido,
  2         to_date(vnt_fch,'yymmdd') fecha
  3  from clientes, ventas
  4  where clt_num=vnt_clt and clt_pais = 'e';

    NUMERO NOMBRE               APELLIDO                  FECHA
---------- -------------------- ------------------------- --------
         1 margarita            borras                    10/01/91
         1 margarita            borras                    10/01/91
         1 margarita            borras                    10/01/91
         5 antoni               llopis                    06/01/91
         7 pablo                goqi                      06/01/91
         7 pablo                goqi                      06/01/91
         7 pablo                goqi                      06/01/91
        13 diego                cortes                    10/01/91
        13 diego                cortes                    09/01/91

9 filas seleccionadas.
```

**Ejemplo** 6.3 Fecha y hora

```
SQL> select to_char(sysdate,'dd-mm-yy hh:mi:ss') "Fecha y hora"
  2  from dual;

Fecha y hora
------------------
15-12-04 02:33:07

1 fila seleccionada.
```

| Elemento | Uso |
|---|---|
| MM | Número del mes. |
| MON | Abreviatura de tres letras del mes. |
| Mon | Como MON pero sólo con la primera letra en mayúscula. |
| mon | Como MON pero en minúscula. |
| MONTH | El nombre del mes. |
| Month | Como MONTH pero con la primera letra en mayúscula. |
| month | El nombre del mes en minúscula. |
| DDD | Número del día del año. |
| DD | Número del día del mes. |

*continúa en la siguiente página*

*continuación de la página anterior*

| Elemento | Uso |
|---|---|
| D | Número del día de la semana. |
| DY | Abreviatura de tres letras del día. |
| Dy | Como DY, pero teniendo la primera letra en mayúscula. |
| dy | Como DY, pero todo en minúsculas. |
| DAY | El nombre del día de la semana. |
| Day | El día de la semana con la primera en mayúscula. |
| day | El día de la semana en minúscula. |
| YYYY | El año con los cuatro dígitos. |
| SYYYY | Año con signo, 1000 a.C. = -1000. |
| YYY | Los tres últimos dígitos del año. |
| YY | Los dos últimos dígitos del año. |
| Y | El último dígito del año. |
| YEAR | El año escrito con letras. |
| Year | Como YEAR con las iniciales en mayúsculas. |
| year | Como YEAR con todas las letras en minúsculas. |
| Q | Número de trimestre. |
| WW | Número de semana en el año. |
| W | Número de semana en el mes. |
| J | Días Julianos desde el 31 de diciembre de 4713 a.C.. |
| HH | Hora del día, siempre de 1-12. |
| HH12 | Lo mismo que HH. |
| HH24 | Hora del día de 1-24. |
| MI | Minuto de la hora. |
| SS | Segundo del minuto. |
| SSSSS | Segundos desde la medianoche, siempre desde 0-86399. |
| /,-:.;"texto" | Puntuación para ser incorporada. |
| A.M. | Muestra A.M. o P.M., dependiendo del momento del día. |
| a.m. | Lo mismo pero en minúscula. |
| P.M. | El mismo efecto que A.M. |
| p.m. | El mismo efecto que a.m. |
| AM | Lo mismo que A.M. pero sin puntos. |
| am | Lo mismo que a.m. pero sin puntos. |
| PM | Lo mismo que P.M. pero sin puntos. |
| pm | Lo mismo que p.m. pero sin puntos. |
| B.C. | Muestra B.C. o A.D. dependiendo de la fecha. |
| A.D. | Lo mismo que B.C. |
| b.c. | Lo mismo que B.C. pero en minúsculas. |
| a.d. | Lo mismo que b.c. |
| BC o AD | Lo mismo que B.C. pero sin los puntos. |

*continúa en la siguiente página*

*continuación de la página anterior*

| Elemento | Uso |
|---|---|
| bc o ad | Lo mismo que b.c. pero sin los puntos. |

Tabla 6.3: Formatos de fechas

# APÉNDICES

# Apéndice A

# El lenguaje *SQL*

Se muestran las características fundamentales de este lenguaje, como son: los principios básicos, la estructura general y las ventajas que presenta.

## A.1. Introducción

El lenguaje *SQL* es un lenguaje de cuarta generación, no procedimental ya que le decimos lo que queremos y el sistema se encarga de plantear el mejor modo de obtenerlo.

Como lenguaje de datos que es, nos permite:

- Definir datos.
- Consultar datos.
- Manipular datos.
- Controlar el acceso a los datos.

El *SQL* se puede usar:

- Por los usuarios no habituales (no-programadores),
- Por los programadores, analistas, etc., o
- De forma interactiva, o en segundo plano.

## A.2. Evolución histórica del lenguaje *SQL*

La historia del lenguajes *SQL* está íntimamente relacionada con el desarrollo de las bases de datos relacionales. El artículo del Dr. *E. Codd* desencadenó una racha de investigaciones en base de datos relacionales, incluyendo un importante proyecto de investigación dentro de *IBM*. El objetivo del proyecto llamado *System/R*, fue demostrar la operabilidad del concepto relacional y proporcionar alguna experiencia en la implementación efectiva de un *SGBD* relacional.

**1974–1975:** La primera fase del proyecto *System/R* produjo un mínimo prototipo de un *SGBD* relacional. Además del propio *SGBD*, el proyecto *System/R* incluía trabajos sobre lenguajes de consulta de bases de datos. Uno de estos lenguajes fue denominado *SEQUEL*[1].

**1976–1977:** El prototipo de investigación *System/R* fue reescrito desde el principio. La nueva implementación soportaba consultas multitabla y permitía que varios usuarios compartieran el acceso a los datos. En la tabla A.1 podemos ver un resumen de la evolución de las bases de datos relacionales.

El lenguaje *SQL*, construido en un principio como un lenguaje algebraico, fue enriqueciendose con funciones predicativas (cláusula existencial, bloques de cualificación correlativos, ...).

**1977–1979:** La implementación de *System/R* fue distribuida a una serie de instalaciones de clientes de *IBM* para evaluación en 1978 y 1979. Estas primeras instalaciones de usuario proporcionaron cierta experiencia efectiva en el uso de *System/R* y de su lenguaje de base de datos, que había sido renombrado como *SQL*[2], por razones legales. A pesar del cambio de nombre, la pronunciación *SEQUEL* permaneció y continua hoy día. En 1979 el proyecto de investigación *System/R* llegó al final, e *IBM* concluyó que las bases de datos relacionales no solamente eran factibles, sino que podrían ser la base de un producto comercial útil.

Este período fue aprovechado por varias firmas para proponer, antes que *IBM*, productos relacionales. Un ejemplo de éstos fue *Ingre*, lanzado por *RTI* en 1976 con el lenguaje predicativo *QUEL*, un derivado del *ALPHA*; otro ejemplo fue el *Oracle*, producto que, inspirándose en el *System/R*, ya se aprovechaba del *SQL* en 1979.

**1980–1989:** Esta tecnología desarrollada por *IBM* se implantó por fin en productos comerciales en el año 1981 con el *SQL/DS* y, especialmente, en el año 1982 con el *DB2*. Este mismo año, el grupo *X3H2* del *ANSI* propuso el *SQL* como un estándar oficial, ratificando en 1986, cuando ya existían por todo el mundo varias decenas de versiones *SQL*, de momento incompatibles entre sí.

La norma *ANSI* de 1986 (*SQL-86*), que cubre dos primeros niveles de normalización, es el resultado de presiones por parte de los fabricantes y, por ello, sólo puede ser

[1] Acrónimo de *Structured English QUEry Language*, lenguaje estructurado de consulta inglés.
[2] *Structured Query Language*, lenguaje estructurado de consultas.

considerado como un pobre conjunto de principios que sólo imperfectamente representa el modelo relacional. Los analistas esperan con impaciencia las ampliaciones anunciadas de la norma y, especialmente, su generalización para los productos comercializados. El nivel 2 de la norma *ANSI* se toma como punto de referencia por los fabricantes de *SGBDR*, aunque el mismo *DB2* en el que se inspira la norma se aleja bastante de ésta. De igual modo, los productos comerciales por lo general se alejan más o menos de la norma y, a menudo, son mucho más complejos. El *SQL-86* se convirtió en un estándar legal establecido por *ANSI*.

En 1989 se publicó una norma extendida para *SQL* denominada *SQL-89* y actualmente los sistemas de bases de datos son normalmente compatibles al menos con las características de *SQL-89*.

**1990–1999:** En 1992 se ratifica el estándar *ANSI* denominado *SQL-92* que ha sido ampliamente implementado y que fue una revisión importante del estándar *SQL-89*. A partir de este estándar se sabía que existían muchos aspectos por resolver para que realmente el *SQL* pudiera ofrecer las capacidades de bases de datos requeridas y para satisfacer las necesidades de los usuarios. Durante siete años se lanzaron varios estándares que atacaban aspectos específicos del *SQL*, hasta que en 1999, se produjo el siguiente estándar importante del *SQL*, el *SQL-99* o *SQL3*.

**2003– :** En el año 2003 salió un nuevo estándar del *SQL* llamado *SQL:2003*, el cual introdujo la noción de *XML* y estandarizó los generadores de secuneicas o valores autogenerados, esto incluye comumnas que se utilizan como identificadores.

## A.3. *SQL* en los *SGBDR*

Todos los informáticos están de acuerdo en afirmar que la generación de las bases de datos relacionales ha modificado considerablemente los hábitos de los programadores.

Los *SGBDR*[3] disponen no sólo de un lenguaje de consultas poderoso y conciso, sino también de un conjunto integrado de herramientas eficaces que explotan en toda su extensión dicho lenguaje, permitiendo la escritura rápida de complejas aplicaciones. El incremento de la productividad es espectacular: los usuarios apuntan de manera uniforme muchas ventajas en las aplicaciones de cierta entidad con respecto al entorno clásico de *COBOL* o *C* con *SGF*[4].

Este *software*, llamado de cuarta generación, está formado generalmente por un conjunto de herramientas que configuran una estructura de anillo:

- *Un generador de pantallas anidadas* para consultas o tomas de datos multi-tablas. En cualquier momento de la gestión de una pantalla, el programador puede soli-

[3]Sistemas de Gestión de Bases de Datos Relacionales.
[4]Sistema de Gestión de Ficheros.

| *Fecha* | *Acontecimiento* |
|---|---|
| 1970 | *Codd* define el modelo de *Base Datos relacional* |
| 1974 | Comienza el proyecto *System/R* de *IBM* |
| 1974 | Primer artículo que describe el lenguaje *SEQUEL* |
| 1978 | Test de clientes del *System/R* |
| 1979 | ORACLE introduce el primer *SGBDR* comercial |
| 1981 | *Relacional Technology* introduce *Ingres* |
| 1981 | *IBM* anuncia *SQL/DS* |
| 1982 | *ANSI* forma el comité de estándares *SQL* |
| 1983 | *IBM* anuncia *DB2* |
| 1986 | Se ratifica el estándar *ANSI SQL* |
| 1986 | *Sybase* introduce *SGBDR* para procesamiento de transacciones |
| 1987 | Se ratifica el estándar *ISO SQL* |
| 1988 | *Ashton-Tate* y *Microsoft* anuncian *SQL Server* para *OS/2* |
| 1988 | *IBM* anuncia la versión 2 de *DB2* |
| 1989 | Primera entrega de servidores de bases de datos *SQL* para *OS/2* |
| 1992 | Se ratifica el estándar *ANSI SQL2* |
| 1999 | Se ratifica el estándar *SQL3* |
| 2003 | Nuevo estándar llamado *SQL:2003* |

Tabla A.1: Resumen cronológico

citar la ejecución automática de una consulta *SQL* que tanto puede realizar una actualización, como visualizar ciertos datos de otra tabla.

- *Un generador de estados de la base de datos* que se imprimen en papel o se visualizan en el monitor. Un estado de la base se obtiene ejecutando una consulta *SQL* asociada a una descripción precisa del formato de salida.

- *Un generador de aplicaciones* que utiliza las pantallas y los estados precedentes y que permite la definición de menús jerarquizados así como la ejecución de tratamientos diversos de los datos. Estos tratamientos pueden ser simples tareas de manipulación de datos escritos en *SQL*, o tratamientos más complejos que impliquen el empleo de los procedimientos de un lenguaje de tercera generación (*C* o *COBOL*, por ejemplo). De este modo, se aprovecha plenamente la potencia de las herramientas de cuarta generación, conservando la riqueza de los lenguajes de la tercera.

- *Un módulo de programas de utilidades* que facilitan principalmente el acceso a los datos o a aplicaciones distantes en un entorno de bases de datos distribuidas.

- *Un generador de esquemas conceptuales*, propuestos por algunos programas, que permiten la descripción interactiva completa de la estructura de los datos (entidades, atributos, enlaces, condiciones) a utilizar por los restantes módulos.

En todos los niveles, el lenguaje *SQL* se utiliza siempre que deba expresarse una

definición, una manipulación o un control de datos. El lenguaje *SQL* es, pues, un elemento central, el nexo imprescindible entre los diversos componentes del *SGDBR*.

El núcleo comprende los programas que generan el conjunto del sistema:

- Compilador *SQL*.
- Traductor externo/conceptual.
- Optimizador de las consultas.
- Traductor conceptual/interno.
- Gestión de los ficheros e índices.
- Conexión con el sistema operativo y el equipo físico.

## A.4. *SQL* en los sistemas no relacionales

El *SQL* se ha hecho tan indispensable que la mayoría de las casas de *software* han optado por adoptarlo, tanto en sus *SGF* como en el resto de los *SGBD* no relacionales. La razón de este hecho es doble.

Por una parte, cada vez está resultando comercialmente más difícil vender un *SGBD* que no presente las facilidades de una lenguaje de acceso de cuarta generación. Y es que el usuario llega a «pensar en *SQL*» después de un aprendizaje muy rápido. Con un poco de lógica, un programador de *C*, *DBASE* o *COBOL* puede utilizarlo al cabo de pocas horas.

Por otra parte, la conectividad de los sistemas es, hoy por hoy, un elemento fundamental de la organización de la información. Los usuarios y los analistas han de disponer de tratamientos distribuidos, de particiones de aplicaciones y de medios de comunicación transparentes. Por ejemplo, las grandes bases de datos de tipo red o jerárquica como *IDMS/R* de *CULLINET* o *DATACOM/DB* de *CCA* proponen interfases *SQL* que permiten la compatibilidad con el *DB2* de *IBM*. Estamos ante una proliferación de servidores de bases de datos, que permiten la elaboración de una informática realmente compartida en un entorno heterogéneo. Citemos, por ejemplo, el *SQL Server*, el *SQLBASE*, *SQL*STAR* o *INGRES*STAR*, que pretenden imponerse en un mercado vertiginoso, descabalgando la competencia ajena. Hoy en día es obligado para todos los *SGF* y *SGBD*, incompatibles de partida, que puedan comunicarse entre sí mediante el lenguaje normalizado del servidor. De aquí que el *SQL* se está imponiendo como un lenguaje universal.

Ya hemos señalado que estos sistemas no relacionales carecen de un núcleo *SQL*, por lo que estas emulaciones sólo funcionan a medias. No obstante, tal extensión ofrece apreciables servicios para la mayoría de las aplicaciones que emplean las funciones fundamentales del lenguaje *SQL*.

# A.5. Principios básicos del lenguaje *SQL*

## A.5.1. El enfoque conjuntista y el lenguaje algebraico

Para comprender bien la filosofía del lenguaje *SQL* expondremos brevemente los principios en que se basa. Veremos cómo el *SQL* aprovecha la sencillez del lenguaje algebraico, cobrando por una parte concisión y por otra claridad, mediante una formulación muy cercana al lenguaje natural.

### A.5.1.1. Principios

Todos los operadores algebraicos se aplican a la totalidad de las filas de las relaciones.

El resultado de una operación (consulta) es una nueva relación, susceptible de ser empleada a su vez en una nueva operación. Esto es lo que se conoce como *propiedad de cierre*. A este fin se dispone de la asignación relacional, que permite afectar a una relación el resultado de una operación.

*Codd* ha enumerado una veintena de operadores que se derivan de cinco primitivos, a saber: *proyección*, *selección*, *unión*, *diferencia* y *producto*. A estos cinco operadores básicos deben agregarse, por su importancia práctica, otros tres: *producto natural* o *unión*[5], *intersección* y *división* (ver figura A.1).

### A.5.1.2. Operadores primitivos propios

**Proyección** Este operador permite elegir sólo las columnas que interesan. Equivale a recortar verticalmente la tabla. Además de ello, elimina las repeticiones de filas que resulten de este recorte.

**Selección** Esta operación realiza un corte horizontal de la relación para retener sólo las filas que cumplen una determinada condición respecto a los valores de una columna o conjunto de columnas.

**Unión** La operación *unión* de conjuntos permite agrupar las filas, obtenidas mediante selección sobre varias tablas efectuando un «o lógico».

**Diferencia** La *diferencia* de conjuntos es una operación que partiendo de dos tablas se crea una tercera a partir de las filas de la primera tabla en la que se eliminan aquellas filas que se repiten en la segunda tabla.

**Producto cartesiano** El *producto* de dos tablas consiste en concatenar (emparejar) cada fila de la primera tabla con todas las filas de la segunda. Esto significa que, si la primera tabla tiene $M$ filas y la segunda $N$, la tabla resultante tendrá $M*N$ filas.

[5] En inglés llamado *join*.

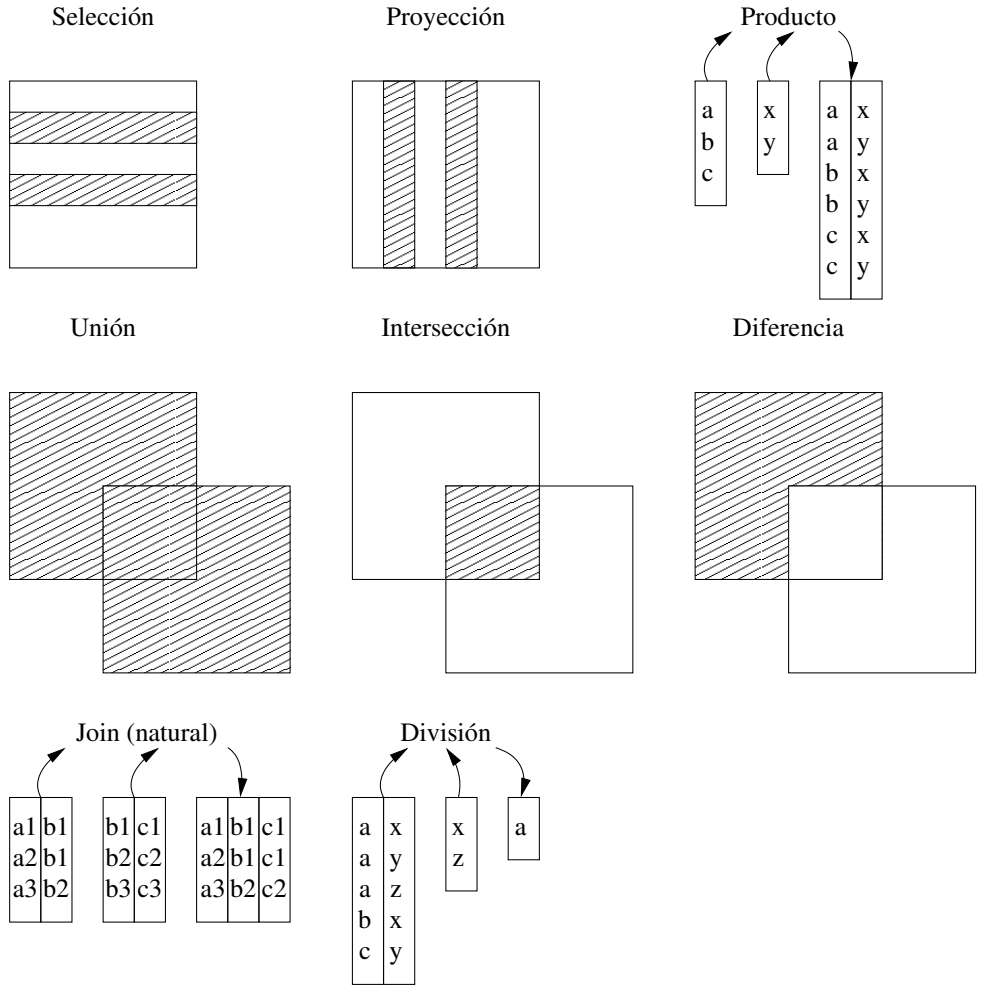


Figura A.1: Operadores

El *producto* presenta de por sí poco interés, pero combinado con una *selección* se obtiene un operador particularmente útil en las consultas multi-tablas: el *producto natural*.

### A.5.1.3. Operadores primitivos derivados

**Producto natural** El *producto natural* sólo es posible realizarlo cuando dos tablas tienen un dominio común. El *producto natural* junta las dos tablas «concatenando» las filas cuyo valor de columna es idéntico en ambas.

De aquí se deduce el importante papel que desempeña una clave foránea[6], que permite establecer implícitamente un enlace entre dos tablas. Para ello el operador

[6]Se denomina clave foránea a un atributo, tal vez compuesto, de una relación *R2* cuyos valores deben concordar con los de la clave primaria de alguna relación *R1* (donde *R1* y *R2* no necesariamente son distintos). Una clave foránea debe tener todos los valores de sus atributos nulos o ninguno de ellos.

*producto natural* es fundamental, dado que puede reunir datos dispersos pero ligados entre sí. La condición de concatenación contiene un operador comparativo que puede ser de cualquier clase, aunque el tipo de comparación que más se utiliza es el de igualdad.

**Intersección** La *intersección* de dos tablas no es un operador primitivo básico, pero puede obtenerse fácilmente mediante dos diferencias. Esta operación permite crear una tabla a partir de filas comunes de dos de tablas.

**División** La operación de *división* permite expresar de forma sencilla el cuantificador universal «para todo elemento que».

#### A.5.1.4. Combinación de operadores primitivos

Se pueden realizar operaciones combinando operadores primitivos propios con operadores primitivos derivados para realizar consultas más complejas.

### A.5.2. Forma general de una consulta *SQL*

La forma de una consulta *SQL* es, por esencia, poco procedimental. El ideal está en expresar globalmente la consulta precedente entrelazando los operadores primitivos y sin recurrir a tablas intermedias. El *SQL* permite obtener este tipo de formulaciones.

La forma general de una consulta *SQL* está compuesta de varias cláusulas:

**Formato:**

```
SELECT     selecciona las columnas deseadas
FROM       lista de tablas que intervienen en la consulta
[WHERE]    criterio de selección de filas
[GROUP BY] formación de grupos por columna(s) con valores
           indénticos
[HAVING]   criterio de selección de grupos
[ORDER BY] ordenación de tuplas en la consulata
```

La cláusula `select` realiza una proyección algebraica sobre las tablas especificadas en la cláusula `from`, y suele ir acompañada de la cláusula `distinct`[7] a fin de evitar las repeticiones de filas.

La cláusula `where` permite expresar todas las condiciones de selección (horizontales) relativas a las tablas y estas condiciones expresan el modo de enlace entre las tablas.

Tenemos que tener en cuenta, además, que en caso de existir ambigüedad en el nombre de una columna, se debe escribir delante el nombre de la tabla a la que pertenece seguido de un punto.

[7] Ver capítulo 2.

**Formato:**

```
nombre_tabla.nombre_columna
```

Construir una consulta multi-tabla en *SQL* resulta extemadamente fácil: basta con escribir el nombre de las tablas afectadas tras la cláusula `from` y expresar en la cláusula `where` todas las condiciones a cumplir, incluidas las condiciones de enlace.

Esta sencillez de escritura se consigue gracias a que el lenguaje exime al programador de explicar el orden en que deben ejecutarse las operaciones. El intérprete o el compilador traducirá automáticamente la consulta, en forma de secuencia óptima de consulta algebraica. En la práctica, el usuario no tiene necesidad de conocer la existencia de índices para efectuar las consultas o clasificaciones.

De este modo, el *SQL* permite expresar, con claridad y precisión, un gran número de operadores del álgebra relacional.

Pero, una vez sentado el principio general, debemos advertir que la sintaxis del lenguaje no es siempre tan transparente. Las consultas que necesitan una lógica más compleja se construyen a su vez con una formulación menos inmediata. Además de que las versiones actuales del lenguaje presentan innumerables trampas capaces de desanimar a más de un principiante.

Por éstas, y otras razones que veremos, el *SQL* es más un lenguaje de programación que un lenguaje de usuario.

## A.5.3. Estructura y características generales

### A.5.3.1. Estructura

El lenguaje *SQL* sólo contiene, por convenio explícito, un limitado número de órdenes o palabras claves, distribuidos en seis grandes grupos funcionales: órdenes de *DDL*[8] y de *DCL*[9], órdenes de *DML*[10], órdenes de control de transacciones, órdenes de control de sesiones y órdenes de *SQL* embebido. La tabla A.2 muestra un resumen de estas órdenes[11].

El *DDL* permite crear todas las estructuras de la base de datos (tablas, vistas, columnas índices...), o sea, la creación del esquema conceptual. Con él definimos todas las estructuras necesarias para manipulación de dicha base de datos.

El *DCL* contiene los operadores que permiten o eliminan el acceso a los datos de un usuario por parte de otros usuario (`grant` y `revoke`).

El *DML* permite el manejo de las tablas y de las vistas mediante las correspondientes

[8] *Data Definition Language*, lenguaje de definición de datos.
[9] *Data Control Language*, lenguaje de control de datos.
[10] *Data Manipulation Language*, lenguaje de manipulación de datos.
[11] En esta tabla se muestran solamente algunas de las órdenes del *SQL* embebido.

| ddl | dml | dcl | Transacciones | Sesiones | Embebido |
|---|---|---|---|---|---|
| create | select | grant | commit | alter session | open |
| drop | insert | revoke | rollback | set role | close |
| alter | delete | | savepoint | | fetch |
| truncate | update | | | | execute |

Tabla A.2: Órdenes del lenguaje *SQL*

cuatro operaciones fundamentales sobre los datos.

Las órdenes de control de transacciones manejan los cambios realizados por las órdenes de *DML*, reflejándolos en la base de datos.

Las órdenes del control de sesiones permite a los usuarios controlar las propiedades de su sesión en curso, incluyendo la habilitación y deshabilitación de funciones o `role`[12].

Las órdenes del *SQL* embebido incorpora órdenes de los lenguajes *DML*, del *DDL* y órdenes de control en un programa de lenguaje procedural, usando necesariamente los precompiladores de *ORACLE*.

#### A.5.3.2. Características del lenguaje

El lenguaje *SQL* es manejable bajo dos modalidades distintas:

- En interactivo, consulta, crea y controla directamente a la base de datos.
- Como un módulo que proporciona un potente lenguaje de consultas interpretadas en un lenguaje huésped como el *C* o el *COBOL*. Corresponde al *SQL* embebido.

Además podemos destacar de este lenguaje lo siguiente:

- Es ensamblista pues, al igual que el lenguaje algebraico del que procede, enfoca las relaciones de forma global (cuantos más barridos de tablas, más test).
- Es un lenguaje cerrado: el resultado de una consulta es una nueva relación, lo que implica, entre otras cosas, que las consultas pueden ser anidadas[13].
- Respeta la independencia entre el nivel conceptual y las aplicaciones, ya que permite la creación y manejo de esquemas externos (o visualizaciones) personalizadas.
- Garantiza la independencia entre el nivel conceptual y el nivel interno. El usuario no nota la presencia de un índice o de un agrupamiento[14]. Es asunto del gestor de la base de datos conseguir la optimización de las ejecuciones.

[12] Es un objeto de Oracle que soporta privilegios tanto de objetos como del sistema.
[13] Ver capítulo 4.
[14] Estructuras adicionales para la mejora de la búsqueda y recuperación de información.

- Garantiza una seguridad total de los datos, gracias a una distribución selectiva de las prioridades de acceso, y a una gestión eficaz de las visualizaciones.
- Permite la gestión multi-usuario de los datos. Cada fila a la que se accede para su modificación queda automáticamente bloqueada por el sistema. En particular, el *SQL* contiene el concepto de transacción, que permite restaurar el estado anterior de la base en caso de anomalías.
- Utiliza constantemente un diccionario dinámico centralizado. El diccionario es accesible, mediante las prioridades requeridas, por el mismo lenguaje *SQL*.
- El cumplimiento de las condiciones de integridad está plenamente garantizado en las versiones actuales del *SQL*. Esto se resuelve con la cláusula `with check option`.

# Apéndice B

# Definición de las tablas

**Tiendas:** un número de identificación único no vacío ($tda\_num$), el área geográfica en que se encuentra la tienda ($tda\_pob$) y el nombre del gerente ($tda\_ger$).

**Clientes:** un número de identificación único no vacío ($clt\_num$), el apellido del cliente ($clt\_apell$), su nombre ($clt\_nom$), nacionalidad ($clt\_pais$) y su ciudad ($clt\_pob$).

**Artículos:** un número de identificación único ($art\_num$), el nombre del artículo ($art\_nom$), su peso ($art\_peso$), su color ($art\_col$), el precio de compra ($art\_pc$), el precio de venta ($art\_pv$) y el número del proveedor ($art\_prv$).

**Proveedores:** un número de identificación único ($prv\_num$), y el nombre del proveedor ($prv\_nom$).

**Ventas:** el número del cliente ($vnt\_clt$), el número de la tienda de adquisición ($vnt\_tda$), el número del artículo adquirido ($vnt\_art$), la cantidad vendida ($vnt\_cant$), el precio de venta total ($vnt\_precio$) y la fecha de la venta ($vnt\_fch$). Se supone que un cliente no va a comprar dos veces el mismo artículo en un mismo día y en la misma tienda.

**Suministros:** el número del artículo ($smt\_art$) y el número del proveedor que los suministra ($smt\_prv$).

**Pesos:** el nombre de la clasificación del peso ($peso\_nom$), el peso mínimo ($peso\_min$) y máximo ($peso\_max$) para cada uno.

# Apéndice C

# Tablas

| ART_NUM | ART_NOM | ART_PESO | ART_COL | ART_PC | ART_PV | ART_PRV |
|---|---|---|---|---|---|---|
| 1 | impresora | 150 | rojo | 400 | 580 | 4 |
| 2 | calculadora | 150 | negro | 4000 | 4700 | 1 |
| 3 | calendario | 100 | blanco | 420 | 600 | 4 |
| 4 | lampara | 550 | rojo | 2100 | 2980 | 5 |
| 5 | lampara | 550 | blanco | 2000 | 2900 | 5 |
| 6 | lampara | 550 | azul | 2100 | 2980 | 5 |
| 7 | lampara | 550 | verde | 2100 | 2980 | 5 |
| 8 | pesacartas 1-500 | | | 2400 | 4000 | 3 |
| 9 | pesacartas 1-1000 | | | 3000 | 5000 | 3 |
| 10 | boligrafo | 20 | rojo | 20 | 40 | 2 |
| 11 | boligrafo | 20 | azul | 20 | 40 | 2 |
| 12 | boligrafo lujo | 20 | rojo | 60 | 100 | 2 |
| 13 | boligrafo lujo | 20 | verde | 60 | 100 | 2 |
| 14 | boligrafo lujo | 20 | azul | 60 | 100 | 2 |
| 15 | boligrafo lujo | 20 | negro | 60 | 100 | 2 |

**Artículos**

| TDA_NUM | TDA_POB | TDA_GER |
|---|---|---|
| 1 | madrid-batan | contesfosques, jordi |
| 2 | madrid-centro | martinez, juan |
| 3 | pamplona | dominguez, julian |
| 4 | barcelona | peqa, jose maria |
| 5 | trujillo | mendez, pedro |
| 6 | jaen | marin, raquel |
| 7 | valencia | petit, joan |
| 8 | requena | marcos, pilar |
| 9 | palencia | castroviejo, lorenzo |
| 10 | gerona | gomez, gabriel |
| 11 | lyon | madoux, jean |
| 12 | paris | fouet, paul |

### Tiendas

| CLT_NUM | CLT_APELL | CLT_NOM | CLT_PAIS | CLT_POB |
|---|---|---|---|---|
| 1 | borras | margarita | e | madrid |
| 2 | perez | miguel | e | madrid |
| 3 | dupont | jean | f | paris |
| 4 | dupret | michel | f | lyon |
| 5 | llopis | antoni | e | barcelona |
| 6 | souris | marcel | f | paris |
| 7 | goqi | pablo | e | pamplona |
| 8 | courbon | gerad | f | lyon |
| 9 | roman | consuelo | e | jaen |
| 10 | roca | pau | e | gerona |
| 11 | mancha | jorge | e | valencia |
| 12 | curro | pablo | e | barcelona |
| 13 | cortes | diego | e | madrid |
| 14 | fernandez | joaquin | e | madrid |
| 15 | duran | jacinto | e | pamplona |
| 16 | minguin | pedro | e | pamplona |

### Clientes

| PRV_NUM | PRV_NOM |
|---|---|
| 1 | catio electronic |
| 2 | estilograficas reunidas |
| 3 | mecanica de precision |
| 4 | sanjita |
| 5 | electrolamp |

**Proveedores**

| VNT_CLT | VNT_TDA | VNT_ART | VNT_CANT | VNT_PRECIO | VNT_FC |
|---|---|---|---|---|---|
| 5 | 4 | 4 | 1 | 2980 | 910106 |
| 7 | 3 | 10 | 1 | 40 | 910106 |
| 7 | 3 | 11 | 2 | 80 | 910106 |
| 7 | 3 | 14 | 3 | 300 | 910106 |
| 8 | 11 | 2 | 1 | 4700 | 910109 |
| 6 | 12 | 3 | 2 | 1200 | 910109 |
| 6 | 12 | 15 | 2 | 200 | 910109 |
| 13 | 1 | 4 | 1 | 2980 | 910109 |
| 13 | 1 | 3 | 1 | 600 | 910110 |
| 1 | 2 | 2 | 1 | 4700 | 910110 |
| 1 | 2 | 12 | 1 | 100 | 910110 |
| 1 | 2 | 13 | 10 | 1000 | 910110 |
| 4 | 11 | 1 | 8 | 4640 | 910111 |
| 4 | 11 | 10 | 7 | 280 | 910111 |
| 3 | 7 | 6 | 1 | 2980 | 910111 |
| 3 | 7 | 9 | 2 | 10000 | 910111 |

**Ventas**

| PESO_NOM | PESO_MIN | PESO_MAX |
|---|---|---|
| leve | 0 | 100 |
| ligero | 101 | 500 |
| medio | 501 | 2500 |
| pesado | 2501 | 9999 |

**Pesos**

# Apéndice D

# Descripción de las tablas

| Nombre | ?Nulo? | Tipo |
|---|---|---|
| ART_NUM | NOT NULL | NUMBER(38) |
| ART_NOM | NOT NULL | VARCHAR2(20) |
| ART_PESO | | NUMBER(38) |
| ART_COL | | VARCHAR2(7) |
| ART_PC | NOT NULL | NUMBER(38) |
| ART_PV | NOT NULL | NUMBER(38) |
| ART_PRV | | NUMBER(38) |

**Articulos**

| Nombre | ?Nulo? | Tipo |
|---|---|---|
| CLT_NUM | NOT NULL | NUMBER(38) |
| CLT_APELL | NOT NULL | VARCHAR2(25) |
| CLT_NOM | | VARCHAR2(20) |
| CLT_PAIS | | VARCHAR2(8) |
| CLT_POB | | VARCHAR2(20) |

**Clientes**

```
Nombre                                    ?Nulo?   Tipo
----------------------------------------- -------- ----------------------------
TDA_NUM                                   NOT NULL NUMBER(38)
TDA_POB                                   NOT NULL VARCHAR2(20)
TDA_GER                                            VARCHAR2(25)
```

### Tiendas

```
Nombre                                    ?Nulo?   Tipo
----------------------------------------- -------- ----------------------------
PRV_NUM                                   NOT NULL NUMBER(38)
PRV_NOM                                   NOT NULL VARCHAR2(25)
```

### Proveedores

```
Nombre                                    ?Nulo?   Tipo
----------------------------------------- -------- ----------------------------
VNT_CLT                                   NOT NULL NUMBER(38)
VNT_TDA                                   NOT NULL NUMBER(38)
VNT_ART                                   NOT NULL NUMBER(38)
VNT_CANT                                           NUMBER(38)
VNT_PRECIO                                         NUMBER(38)
VNT_FCH                                   NOT NULL VARCHAR2(6)
```

### Ventas

```
Nombre                                    ?Nulo?   Tipo
----------------------------------------- -------- ----------------------------
PESO_NOM                                  NOT NULL VARCHAR2(9)
PESO_MIN                                  NOT NULL NUMBER(38)
PESO_MAX                                  NOT NULL NUMBER(38)
```

### Pesos

# Bibliografía


[Abbe02] Abbey, M.; Corey, M. & Abramson, I.
*Oracle9i. Guía de aprendizaje*
Osborne McGraw-Hill, 2002.

[Abra06] Abramson, I.; Abbey, M. & Corey, M.
*Oracle Database 10g. Guía de aprendizaje*
Osborne McGraw-Hill, 2006.

[Conn05] Connolly, T. & Begg, C.
*Sistemas de Bases de Datos*
Pearson Addison-Wesley, 4ª edición, 2005.

[Elma07] Elmasri, R. & Navathe, S.B.
*Fundamentos de sistemas de Bases de Datos*
Addison-Wesley, 5ª edición, 2007.

[Garc96] Garví García, E.; Rodríguez Almendros, M. L.; Velasco Anguita, F.
*Uso de Oracle: SQL y PL/SQL*
Proyecto Sur de Ediciones, S.L., 1996.

[Koch94] Koch, G.
*Oracle. Manual de referencia*
McGraw-Hill/Interamericana de España, S.A., 1994.

[Lone02] Loney, K. & Theriault, M.
*Oracle9i. Manual del administrador*
Osborne McGraw-Hill, 2006.

[Lone06] Loney, K. & Bryla, B.
*Oracle Database 10g. Manual del administrador*
Osborne McGraw-Hill, 2006.

[Mare92] Marêe, C.; Ledant, G.
*SQL Iniciación, programación y prácticas avanzadas*
Masson, S.A., 1992.

[Orac92] *Manuales de Oracle*
Oracle Corporation, 1992.

[Pere02] Pérez, C.
*Oracle9i. Administración y Análisis de Bases de Datos*
Ra-Ma, 2002.

[Silb06] Silberschatz, A.; Korth, H. & Sudarshan, S.
*Fundamentos de Bases de Datos*
McGraw-Hill, 5ª edición, 2006.

[Smin93] Smine, H.
*ORACLE. Arquitectura, administración y optimización*
Díaz de Santos, 1993.

# Referencias electrónicas

http://www.oracle.com

http://juno.uca.es/index.htm

http://ora.u440.com