# Cuaderno de Prácticas de Bases de Datos. Curso 2004-2005

Autores: **Medina J.M., Pons O., Vila M.A., Bailón A., Marín N., Acid S.**

Profesores de prácticas:
**Pons O.**, email:opc@decsai.ugr.es,
**Marín N.**, email: nicm@decsai.ugr.es
**Acid S.**, email: acid@decsai.ugr.es
**Martín M.J.**, email: mbautis@goliat.ugr.es
**Mantas C.** email: cmantas@decsai.ugr.es

*Dpto. Ciencias de la Computación e Inteligencia Artificial*
*E.T.S. de Ingeniería Informática. Universidad de Granada*

# Índice general

# Capítulo 1

# Introducción General

Los objetivos que se persiguen en la elaboración del cuaderno de prácticas son:

- Familiarizar al alumno con los principales elementos de un sistema de gestión de bases de datos (SGDB) comercial, ORACLE®.
- Conocer la configuración del sistema y del entorno de trabajo en el que se realizarán las prácticas de la asignatura.
- Utilizar un lenguaje estándar para sistemas gestores de bases de datos relacionales, SQL. Algunos sistemas que usan SQL son: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Aunque la mayoría de estos sistemas tienen ciertas adaptaciones propias del lenguaje, en las prácticas se verán comandos estándares para la creación y manipulación de un esquema de base de datos.

Algunas consideraciones sobre este cuaderno de prácticas:

- El cuaderno está organizado en varias sesiones de trabajo, una de las cuales –la segunda– se corresponde en realidad con tres sesiones de dos horas. Para que el tiempo de trabajo en prácticas sea óptimo, conviene leer el guión y resolver los ejercicios previamente.

- No existe ninguna recogida de prácticas: la realización de los ejercicios del cuaderno es voluntaria y muy recomendable. El examen práctico (puntuable en la calificación final de la asignatura) se realizará en las mismas condiciones de las clases de prácticas.

## 1.1. Introducción al SGBD ORACLE

### 1.1.1. Nota histórica

ORACLE®es uno de los primeros sistemas de gestión de bases de datos que soportan completamente el modelo de datos relacional. Los comandos de creación de sus últimas versiones recogen sus principales elementos de representación incluyendo las reglas de integridad de entidad y referencial. Así mismo, su lenguaje de consulta es el SQL que tiene todas las características de un lenguaje de consulta relacionalmente completo.

ORACLE fue desarrollado inicialmente por IBM para el SYSTEM/R, pero en 1977 esta compañía vendió ORACLE a Relational Software Inc. quién, en 1979, produjo la versión 2 del sistema. Esta versión fue el primer SGBD relacional disponible comercialmente, y el primero que utilizó SQL como lenguaje de consulta y manipulación.

Desde esta fecha ORACLE ha ido desarrollando su sistema de gestión, creando al mismo tiempo productos adicionales y lanzando al mercado distintas versiones tanto del sistema básico como de los productos asociados. La última versión de Oracle disponible, la 9.2, será la empleada en estas prácticas.

### 1.1.2. Una visión de conjunto del sistema

Como hemos comentado antes, ORACLE®es un sistema modular que se compone de distintos productos. Su elemento básico es el sistema de gestión ORACLE9. Dispone de un catálogo extenso de productos para desarrollo y explotación, todos ellos basados en su servidor de BD y perfectamente acoplados con la filosodía de desarrollo Cliente/Servidor, Internet e Intranet.

Los productos más comunes que se encuentran en casi todas las instalaciones son:

- ORACLE RDBMS que, como ya hemos dicho antes, es el verdadero sistema de gestión.
- Utilidades de ORACLE que son las utilidades de administración de la base de datos.
- SQL*Plus es la herramienta básica de programación que permite a los usuarios la manipulación directa de la base de datos mediante SQL.

En esta serie de prácticas nos vamos a centrar en el uso de SQL*Plus, que es una herramienta que nos va a permitir crear, consultar, manipular y generar informes sobre una base de datos. Intentaremos proponer sesiones de trabajo que, supuesto un buen conocimiento de SQL, proporcionen un conocimiento medio de esta herramienta.

# Capítulo 2

# Sesión de Trabajo 1ª

*En esta primera sesión se pretende que el alumno se familiarice con el entorno de trabajo en el que se van a desarrollar las prácticas. Para ello, al final de la misma, deberá ser capaz de entrar a trabajar en SQL*Plus, editar y ejecutar un fichero de comandos para la creación de tablas e inserción de tuplas. Además, se pretende preparar las tablas con la inserción en ellas de algunos datos figurados, que nos van a servir para desarrollar ejemplos de consultas en sesiones posteriores.*

## 2.1. Conexión y acceso al SGBD

El entorno de trabajo que se va a utilizar será Windows95. El procedimiento para trabajar con ORACLE consta de los siguientes pasos:

1. Iniciar el ordenador
2. Aparece una ventana de diálogo con tres campos, con las que nos identificaremos con nuestros datos de usuario de la ETSII, **login, clave**, el **código** a introducir es *orag*. (En ocasiones, dependiendo de la configuración del ordenador, tamaño pequeño de disco, es necesario reiniciar el ordenador con `Control+ALt+Del` y proceder como en el paso 2).
3. Se produce la descarga e inicializacón de una instalación de windows95 habilitada para acceder al servidor Oracle desde SQL*Plus.
4. Seleccionamos el programa SQL Plus a partir del menú de programas de inicio.
5. A través de una nueva ventana de diálogo introducimos **login** de usuario: **x**+los dígitos del dni - el primero, esto es, la **x** sustituye el primer dígito del dni o pasaporte; como **clave** la misma cadena del login [1], por último la **cadena de conexión** es *oracle0*.
6. Si todo va bien estaremos conectados al servidor Oracle y aparece el prompt del sistema

   ```
   SQL>
   ```

   Ya está todo dispuesto para introducir cualquier comando SQL. Indicar aquí, que éste no es sensible a mayúsculas o minúsculas.

[1] Esta es la clave asignada por defecto, se podrá cambiar

## 2.2. Cambiar la clave

La primera vez que nos conectamos se recomienda cambiar nuestra clave en Oracle, que como hemos indicado inicialmente coincide con la cadena del login. El cambio se realiza mediante la sentencia SQL, ALTER USER, en la forma que se indica:

`SQL> ALTER USER x-login IDENTIFIED BY password ;`

donde, `x-login` es el identificador de vuestra cuenta en el sistema, y está compuesto de `x` seguido de los restantes dígitos del dni o pasaporte p.e. `x2345678` y `password` es la 'clave'que querais establecer para acceder a Oracle, no tiene por qué ser igual al que tengais en la escuela, aunque puede serlo. Como hemos indicado las palabras reservadas `ALTER` ... se pueden introducir en mayúsculas o no.

Una vez ejecutada esta sentencia, necesitareis introducir ese 'login' y ese 'password' cada vez que entreis a Oracle desde el cliente SQL*Plus de Windows95.

## 2.3. Salir de SQL*Plus

Para finalizar cualquier sesión de trabajo, se recomienda ejecutar desde la línea de comandos la instrucción

```
COMMIT;
```

que guarda los cambios realizados en la sesión de trabajo. Y finalmante, para abandonar el entorno con el comando

```
QUIT;
```

## 2.4. Ejecutar comandos de SQL

### Desde la línea de comandos

Por ejemplo, para la creación de una tabla prueba1

```
CREATE TABLE prueba1 (
   cad char(3),
   n int,
   x float);
```

Si todo ha ido bien, devolverá `TABLE CREATED`, en caso contrario dará un mensaje de error adecuado (leedlo atentamente), en ocasiones se indica con un subrayado la palabra 'responsable' del error.

### Desde el buffer

En realidad SQL*Plus siempre trabaja con un fichero de comandos que, por defecto, se llama **afidiet.buf**. Este fichero contiene siempre el último comando que se ha tecleado en "línea". De hecho el ";" que se pone al final de un comando es la señal de fin de fichero y la orden de ejecutar el contenido del mismo. El contenido de este fichero se puede editar tecleando simplemente `edit` y volverlo a lanzar mediante el comando `run`.

### Desde un fichero

Otra alternativa para trabajar es editar y lanzar ficheros de comandos que se pueden guardar y volver a relanzar desde SQL*Plus. Estos ficheros de comandos SQL se pueden crear con cualquier editor (conviene dejarlos en un directorio de Bases de Datos preparado a tal efecto) deberán tener extensión `.sql`. Para ello hay varias posibilidades:

1. Tecleando `edit nombre-de-fichero`, siendo éste un fichero no creado previamente y escribiendo en él antes de abandonar el editor.
2. Tecleando `save nombre-de-fichero`. De esta forma se copia el contenido del buffer al fichero en cuestión.
3. Otra herramienta frecuentemente utilizada es el bloc de notas de Windows

En las situaciones 1 y 2 el fichero se crea con la extensión adecuada y se almacena en el directorio de usuario correspondiente. En el último caso es necesario renombrar el fichero para fijarle la extensión (sql), ya que el editor le asigna otra por defecto.

Para ejecutar un fichero desde SQl*Plus, basta teclear `startcamino/nombre-de-fichero` o bien `@camino/nombre-de-fichero`. Donde `@camino` es el camino absoluto o relativo desde el que se ejecuta SQL*Plus (por defecto está en /**bin**. Por simplicidad se recomienda establecer el directorio de trabajo, en el entorno, al directorio donde están nuestros archivos sql. Para ello basta con seleccionar en la ventana de SQL*Plus la opción OPEN del menu FILE y estableciendo cuál es la carpeta donde están nuestros archivos para después pulsar el botón CANCEL.[2]
*Ejercicio:*

Editar el fichero **pp.sql** en el directorio de trabajo, debe contener las instrucciones:

```
CREATE TABLE prueba2(
    cad1 char(2),
    num int);
INSERT INTO prueba2 VALUES ('aa',1);
INSERT INTO prueba2 VALUES ('Aa',2);
```

ejecutar el fichero.

## 2.5. Información acerca de la base de datos

Para saber cuáles son las tablas que tenemos creadas en nuestro espacio de usuario se puede hacer la consulta al sistema de la forma

```
SELECT TABLE_NAME
FROM USER_TABLES;
```

En este momento deberíamos tener las tablas PRUEBA1 y PRUEBA2. **Consultar** las tuplas de cada una de las tablas.

## 2.6. Descripción de una tabla

Con el comando CREATE TABLE se crea una tabla con los atributos especificados en el momento de creación que serán referenciados en consultas específicas como

```
SELECT CAD1
FROM PRUEBA2;
```

[2]Es un amago de apertura de un fichero, abrir un fichero no lo ejecuta, tan sólo son válidas las dos formas de start y @ para ejecutar un fichero.

En cualquier momento, podemos obtener información acerca de los atributos de una tabla como los nombres si puede tomar valores nulos etc. con el comando

```
DESCRIBE nombre-tabla;
```

## 2.7. Eliminar tablas

Se puede eliminar una tabla con todas las tuplas que contiene, liberando el espacio con el comando

```
DROP TABLE nombre-tabla;
```

## 2.8. Petición de ayuda sobre SQL

El cliente SQL*Plus para Windows dispone de un menú de ayuda que nos puede proporcionar información acerca de la sintaxis de las sentencias SQL y SQL*Plus. Desde el "prompt" de `SQL>` para obtener ayuda.

Se tienen instaladas las páginas de manual de Oracle en formato (.pdf) en la unidad `H`, directorio **orawin95/pdfdoc**. Por tanto, para consultar basta con navegar através del icono **MiPC** de windows por los directorios indicados y pinchar en un fichero (.pdf) y se invoca automáticamente al Acrobat Reader, y muestra el índice de los manuales. Las páginas de manual contienen hipertexto por lo que es muy sencilla su utilización.

## 2.9. Algunas facilidades

### Paginar la información

En ocasiones los resultados de consultas pueden ser muy grandes para ser mostrados en pantalla de forma aceptable, por ello es conveniente activar los comandos de paginado con

```
 set pause on;
```

y luego

```
SQL>set pause '.....Pulsa una Tecla para continuar.....'
```

lo que hace que el barrido de pantalla se detenga cada vez que se llene ésta y nos permita observar los resultados de nuestras operaciones con mayor comodidad.

### Guardar una copia de lo realizado

Este comando sirve para activar la salida de la pantalla de SQL*Plus a un cierto fichero. De esta forma se guarda tanto los comandos como las salidas de éstos. Su forma general es:

```
SPO[OL] [file_name[.ext ] | OFF | OUT ]
```

- `file_name` representa el fichero de texto donde se quiere enviar la salida. Como puede verse la extensión es opcional. En el caso de que no la lleve se pone por defecto la extensión `lst`.
- `OFF` termina la salida al fichero.
- `OUT` termina la salida al fichero y envía este a la impresora.

## 2.10. Creación de la Base de Datos de las prácticas

En esta ocasión, vamos a crear las tablas que vamos a usar en futuras sesiones de prácticas. En la definición de estas tablas vamos a especificar claves primarias, claves externas, restricciones a campos... Se recomienda utilizar un editor, aunque puede ser empleada cualquiera de las anteriores alternativas. Es decir, vamos a editar diferentes ficheros de creación de las tablas: *proveedores, piezas, proyectos* y *ventas*. Se recomienda insertar las tuplas que aparecen en los ejemplos para poder hacer comparaciones de resultados en las consultas.

### Creación de la tabla de Proveedores

Se pide crear el fichero proveedor.sql con la información siguiente:

```
create table proveedor(
codpro varchar2(3) constraint codpro_no_nulo not null constraint codpro_clave_primaria primary key,
nompro varchar2(30) constraint nompro_no_nulo not null,
status number constraint status_entre_1_y_10 check(status>=1 and status<=10),
ciudad varchar2(15));
```

Como resultado de la ejecución de este fichero se crea la tabla de proveedores definiendo 'codpro' como llave primaria y una restricción de integridad sobre el valor de 'status'.

Para ejecutar este fichero teclear:

```
SQL> @proveedor
```

Para ver la estructura de la tabla creada ... ver sección 2.6. El resultado debe ser:

```
 Name                           Null?    Type
 ------------------------------ -------- ----
 CODPRO                         NOT NULL VARCHAR2(3)
 NOMPRO                         NOT NULL VARCHAR2(30)
 STATUS                                  NUMBER
 CIUDAD                                  VARCHAR2(15)
```

### Creación de la tabla de piezas

Puesto que el proceso es el mismo nos limitamos a mostrar el contenido del fichero `pieza.sql`:

```
create table pieza(
codpie varchar2(3)  constraint codpie_clave_primaria primary key,
nompie varchar2(10) constraint nompie_no_nulo not null,
color varchar2(10),
peso number(5,2) constraint peso_entre_0_y_100 check(peso>0 and peso<=100),
ciudad varchar2(15));
```

grabar y salir del editor

```
SQL>start pieza;
```

La información acerca de la tabla es:

```
 Name                           Null?    Type
 ------------------------------ -------- ----
 CODPIE                         NOT NULL VARCHAR2(3)
 NOMPIE                         NOT NULL VARCHAR2(10)
 COLOR                                   VARCHAR2(10)
 PESO                                    NUMBER(5,2)
 CIUDAD                                  VARCHAR2(15)
```

## Creación de la tabla de proyectos

Editamos el fichero `proyecto.sql`, con la siguiente información:

```
create table proyecto(
codpj varchar2(3)  constraint codpj_clave_primaria primary key,
nompj varchar2(20) constraint nompj_no_nulo not null,
ciudad varchar2(15));
```

guardamos y salimos del editor

```
SQL>start proyecto;
```

La información acerca de la tabla es:

```
 Name                           Null?    Type
 ------------------------------ -------- ----
 CODPJ                          NOT NULL VARCHAR2(3)
 NOMPJ                          NOT NULL VARCHAR2(20)
 CIUDAD                                  VARCHAR2(15)
```

## Creación de la tabla de ventas

Editamos `ventas.sql` con el contenido:

```
create table ventas(
codpro constraint codpro_clave_externa_proveedor references  proveedor(codpro),
codp constraint codpie_clave_externa_pieza references  pieza(codp),
codpj constraint codpj_clave_externa_proyecto references proyecto(codpj),
cantidad number(4),
constraint  clave_primaria primary key (codpro,codp,codpj));
```

grabamos y salimos del editor. *Ojo: hay una errata.* Corregimos y ejecutamos.

Como puede verse hemos definido ventas con tres llaves externas a proveedor, pieza y proyecto y con llave primaria incluyendo tres de sus atributos.

```
SQL> start ventas;
SQL> describe ventas
 Name                           Null?    Type
 ------------------------------ -------- ----
 CODPRO                                  VARCHAR2(3)
 CODP                                    VARCHAR2(3)
 CODPJ                                   VARCHAR2(3)
 CANTIDAD                                NUMBER(4)
```

Comprobad que las tablas que tenemos creadas hasta el momento En nuestro caso aparecerá

```
TABLE_NAME
------------------------------
PRUEBA1
PRUEBA2
PIEZA
PROYECTO
PROVEEDOR
VENTAS
```

### 2.10.1. Inserción de tuplas en las tablas

Una vez creadas las tablas es preciso introducir valores en ellas. Para ello se hará un uso intensivo del comando `insert` de SQL, ya que es la única forma que, por ahora, tenemos para introducir datos en una tabla. Vamos a mostrar un ejemplo de introducción y visualización sencilla de datos en la tabla de proveedores. Las tuplas de las tablas de proveedor, pieza y proyecto se introducirán de la misma manera, para ello se recomienda usar las tuplas que aparecen listadas a continuación para que las consultas posteriores tengan respuestas no vacías.

La forma general del comando `insert` es la siguiente:

```
INSERT INTO table_name [(column1, column2,...)]
   VALUES(value1, value2,...);
```

Para utilizarlo vamos a editar el fichero `Intro_proveedor.sql`, cuyo proceso es muy parecido al anterior:

```
insert into proveedor
values ('S1','Jose Fernandez',2, 'Madrid');
```

grabar y salir del editor. Ejecutar `SQL> start Intro_proveedor;`
si la tupla es correcta (tipos de datos acordes, etc.) devolverá: `1 record created`. En caso contrario dará el correspondiente mensaje de error.

Se continúa el proceso de inserción de tuplas bien modificando adecuadamente el fichero `Intro_proveedor.sql` y lanzándolo a ejecutar sucesivas veces o bien incluyendo en el fichero varios comandos `insert`. A continuación se muestran los contenidos de las tablas proveedor, pieza y proyecto tal y como deberían quedar para la resolución de las consultas:

```
SQL> select * from proveedor;

COD NOMPRO                          STATUS CIUDAD
--- ------------------------------ --------- --------
S1  Jose Fernandez                         2 Madrid
S2  Manuel Vidal                           1 Londres
S3  Luisa Gomez                            3 Lisboa
S4  Pedro Sanchez                          4 Paris
S5  Maria Reyes                            5 Roma
```

```
SQL> select * from pieza;

COD NOMPIE     COLOR           PESO CIUDAD
--- ---------- ---------- --------- ---------------
P1  Tuerca     Gris             2,5 Madrid
P2  Tornillo   Rojo            1,25 Paris
P3  Arandela   Blanco              3 Londres
P4  Clavo      Gris             5,5 Lisboa
P5  Alcayata   Blanco             10 Roma

SQL> select * from proyecto;

COD NOMPJ                CIUDAD
--- -------------------- ---------------
J1  Proyecto 1           Londres
J2  Proyecto 2           Londres
J3  Proyecto 3           Paris
J4  Proyecto 4           Roma
```

Para realizar la introducción de tuplas en la tabla ventas del usuario se dispone de una tabla ya rellena en el servidor, la tabla **opc.ventas**. A continuación se muestra su contenido.

```
SQL> select * from opc.ventas;

COD COD COD  CANTIDAD FECHA
--- --- --- --------- --------
S1  P1  J1        150 18/09/97
S1  P1  J2        100 06/05/96
S1  P1  J3        500 06/05/96
S1  P2  J1        200 22/07/95
S2  P2  J2         15 23/11/04
S4  P2  J3       1700 28/11/00
S1  P3  J1        800 22/07/95
S5  P3  J2         30 21/01/04
S1  P4  J1         10 22/07/95
S1  P4  J3        250 09/03/94
S2  P5  J2        300 23/11/04
S2  P2  J1       4500 15/08/04
S3  P1  J1         90 09/06/04
S3  P2  J1        190 12/04/02
S3  P5  J3         20 28/11/00
S4  P5  J1         15 12/04/02
S4  P3  J1        100 12/04/02
S4  P1  J3       1500 26/01/03
S1  P4  J4        290 09/03/94
S1  P2  J4        175 09/03/94
S5  P1  J4        400 21/01/04
S5  P3  J3        400 21/01/04
```

```
22 filas seleccionadas.
```

Luego hablamos de dos tablas con nombres iguales. Por un lado, nuestra tabla ventas y por otro, una tabla **ventas** diferente perteneciente al usuario **opc**, que tiene activado el acceso de lectura.
Pues bien, podemos rellenar fácilmente nuestra tabla *ventas* a partir de la anterior con el comando **insert into** ventas **select** * from opc.ventas .... Pero comprueba si ambas ventas son compatibles...
Por último, antes de terminar la sesión, se pide dejar únicamente las tablas necesarias para las próximas sesiones. Para ello:

1. Comprobar las tablas de usuario
2. Borrar el resto de tablas excepto las indicadas
3. Ejecutar COMMIT antes de abandonar la sesión.

# Capítulo 3

# Sesión de Trabajo 2ª

*Dedicaremos esta sesión de trabajo al proceso de consulta interactiva mediante el comando SELECT de SQL. Al final de esta sesión los alumnos deben ser capaces de expresar una consulta a la base de datos. Con objeto de aprovechar convenientemente el tiempo se recomienda que, antes de comenzar la sesión, los alumnos tengan previamente codificadas en SQL las consultas que se proponen.*

## 3.1. Instrucción para la consulta

La instrucción `SELECT` permite consultar las tablas seleccionando datos en filas y columnas de una o varias tablas.

```
SELECT [ DISTINCT | ALL]
        expresion [alias_columna_expresion]
        {,expresion [alias_columna_expresion]}
FROM [esquema.]tabla|vista [alias_tabla_vista]
WHERE condicion
GROUP BY expresion,[expresion]
[{UNION | UNION ALL | INTERSECT | MINUS} instruccion SELECT]
HAVING condicion
ORDER BY {expresion} [ASC | DESC]
```

**Problema:** *Encontrar todos los nombres de todos los proveedores con status igual a 3*
**Consulta:** `select nompro from proveedor where status=3;`
**Salida:**

```
NOMPRO
----------------------------
Juan Martinez
Jose Lopez
Carlos Ruiz
```

Para realizar las siguientes consultas los términos **ventas, suministros y pedidos** son sinónimos e implican una terna formada por (código de proveedor, código de pieza y código de proyecto).

## 3.2. Consultas a realizar

Se proponen las siguientes consultas, que recogen gran parte de las posibilidades de SQL. Es interesante que se resuelvan todas ellas, así como que se propongan por parte del alumno consultas adicionales.

1. Encontrar los códigos de los proveedores que suministran al proyecto J1.
2. Encontrar todos los suministros cuya cantidad esté entre 200 y 300 inclusive.
3. Listar los nombres, colores y pesos de las piezas por orden ascendente de peso.
4. Encontrar todos los triples de códigos de proveedor, proyecto y piezas que estén en la misma ciudad.
5. Encontrar todos los triples de códigos de proveedor, proyecto y piezas que estén en la misma ciudad y que estén relacionados mediante alguna venta.
6. Encontrar los nombres de piezas suministradas por los proveedores de Londres por orden ascendente de nombres.
7. Encontrar la ciudad y los códigos de piezas suministradas a cualquier proyecto por un proveedor que esté en la misma ciudad que este proyecto por orden ascendente de ciudad.
8. Encontrar aquellos proyectos que usan una pieza de las que suministra S1.
9. Encontrar los códigos de los proveedores que tienen un status mayor que el de S3.
10. Encontrar los códigos de aquellos proyectos que no utilizan ninguna pieza roja que esté suministrada por un proveedor de Londres.
11. Encontrar los códigos de aquellos proyectos a los que sólo abastece S1.
12. Encontrar los códigos de las piezas suministradas a todos los proyectos localizados en Londres.
13. Encontrar aquellos proveedores que envían piezas procedentes de todas las ciudades donde hay un proyecto.
14. Encontrar el número de envíos hechos por cada proveedor.
15. Encontrar la cantidad total de cada pieza enviada a cada proyecto.
16. Encontrar para cada proyecto su nombre y la cantidad media de piezas que recibe por proveedor.
17. Encontrar los nombres de proveedores tales que todas sus ventas superen la cantidad de 100 unidades.
18. Encontrar los proyectos que no tengan ningún pedido.

**Preguntas adicionales**

1. Encontrar aquellos proveedores que hayan hecho al menos dos pedidos. (Realizar el ejercicio con y sin operadores de agregación).
2. Encontrar aquellos proveedores que hayan hecho al menos veinte pedidos.
3. *a*) Encontrar el nombre de las piezas suministradas por el proveedor S1.
   *b*) Encontrar aquellos proveedores que venden todas las piezas suministradas por S1.
   *c*) Encontrar la cantidad total de piezas que ha vendido cada proveedor que cumple la condición de vender todas las piezas suministradas por S1.

4. *a*) Encontrar el nombre de los proveedores que suministran la pieza P3.
   *b*) Encontrar qué proyectos están suministrados por todos los proveedores que suministran la pieza P3.
   *c*) Encontrar la cantidad media de piezas suministrada a aquellos proveedores que venden la pieza P3.
5. Piensa con detenimiento el significado de la palabra todos/as en las siguientes consultas y resuélvelas convenientemente:
   *a*) Encontrar todos los proveedores que venden una pieza roja.
   *b*) Encontrar todos los proveedores que venden todas las piezas rojas.
   *c*) Encontrar todos los proveedores tales que todas las piezas que venden son rojas.
   *d*) Encontrar el nombre de aquellos proveedores que venden más de una pieza roja.
   *e*) Encontrar todos los proveedores que vendiendo todas las piezas rojas cumplen la condición de que todas sus ventas son de más de 10 unidades.

## 3.3. Un nuevo problema

[1] Se dispone del siguiente esquema de una base de datos, correspondiente a la especialización de una serie de productos en impresoras, ordenadores personales PC, y portátiles.

```
PRODUCTOS(ref#,fabricante,precio)
IMPRESORAS(ref#,color,resolV,tipo)
PC(ref#,procesador,velocidad,ram,disco)
PORTATILES(ref#,procesador,velocidad,ram,disco,pantalla)
```

- Crear las tablas correspondientes definiendo tipos de los atributos, claves primarias, externas y las siguientes restricciones:
  - La resolución vertical de las impresoras tiene valor mínimo=1200
  - Para la tabla Portátiles el atributo disco tiene valor mínimo=20
  - Para la tabla PC el atributo disco tiene valor mínimo=30
  - El atributo velocidad de las tablas PC y Portatil tiene valor mínimo=1.4

La tabla `Productos` contiene la siguiente información:

```
1300X  ACERo   1339,00
1400XC ACERo  1057,00
1403XC ACERo  1399,00
1400LC ACERo  1535,25
1403LC ACERo  1879,00
1407XE  HPp   1499,00
1500XE  HPp   2049,00
8004Ev  IBMm   958,00
8100Ev  HPp    1228.00
```

[1] Este problema ha sido elaborado inspirándose en un ejercicio (del capítulo 4) del libro Introducción a los Sistemas de Bases de Datos de los autores J.D. Ullman y J. Widom.

```
8077Ev  COMPACq 3359,00
4300LSC BROTHERr 199.00
4650BJC CANONn   240.20
4400BJC EPSONn   689.00
5000BJC HPp      263.63
```

La tabla PC contiene:

```
8004Ev Pentium 4  2.53-GHz  256MB 30GB
8100Ev Pentium 4  1.6-GHz 256MB 41GB
8077Ev Pentium 4  2.53-GHz 1024MB 40GB
```

La tabla Portátiles contiene la siguiente información:

```
1300XC ATH  1.4,  256 MB, 20GB, 14.1
1400XC ATH  1.4,  128 MB, 20GB, 15.1
1403XC Pentium IV 1.7 GHZ, 256 MB, 20GB, 14.1
1403LC Pentium IV 1.7 GHZ, 256 MB, 20GB, 14.1
1403LC Pentium IV 2,0 GHZ, 512 MB, 30GB, 14.1
1407XE Pentium Cel 1.13 Ghz, 128 MB, 20GB, 15.1
1500XE Pentium IV-M  1.6 Ghz, 256 MB,30GB, 15.1
```

La tabla Impresoras contiene

```
4300BJC 1200DPI, B&W, 2MB, Laser
4650BJC 2400DPI, Color, 80KB, Inkjet
4400BJC 2880DPI, Color, 256KB, Inkjet
5000BJC 4800DPI, Color, 16MB, Thermal
```

Las tablas anteriores se ilustran acompañadas de información adicional para facilitar su lectura, quizá alguna sea prescindible para definir los tipos de los atributos en las tablas.

1. Muestra las referencias de PC que tienen una velocidad de al menos 1.7 GHz
2. Muestra los fabricantes que producen portátiles de discos mayores de 20 GB
3. Encuentra las referencias y precios de todos los productos realizados por el fabricante HPp.
4. Muestra las referencias de las impresoras de injección de tinta de color.
5. Encuentra los fabricantes que venden portátiles, pero no PC.
6. Muestra los tamaños de disco duro que se repiten en dos o más portátiles.
7. Para realizar una comparación muestra los parejas de referencias de portátiles que tienen la misma velocidad y la misma RAM. Cada pareja aparece una sola vez. Esto es, se muestra (i,j) pero no (j,i).
8. Muestra las referencias de PC más caros.
9. Muestra las referencias de PC más baratos.
10. Muestra los fabricantes de, al menos, dos ordenadores diferentes (PC o portatil) con una velocidad mínima de 1.7 GHz.
11. Muestra los fabricantes de ordenadores (PC o portatil) de máxima velocidad.
12. Muestra los fabricantes de Portátiles de al menos tres velocidades diferentes.

13. Muestra los fabricantes de Portátiles que utilizan tan sólo Pentium.
14. Muestra los fabricantes que venden todos los modelos de procesador Pentium.
15. Muestra los fabricantes que venden exactamente dos referencias diferentes de Portátiles.
16. Muestra las referencias fabricadas por cada fabricante.
17. Muestra el precio medio de portátiles por fabricante.
18. Para cada fabricante muestra el número de artículos diferentes.
19. Muestra las referencias de impresoras con mayor resolucion vertical que la impresora 4650BJC.

# Capítulo 4

# Sesión de Trabajo 3ª

*El objetivo de esta sesión es conocer el tipo de dato fecha y su manejo en consultas, inserciones y modificaciones. Para realizar las prácticas correspondientes a esta sesión vamos a alterar el esquema de alguna de nuestras tablas. Por ello, comenzamos con:*

## 4.1. Modificación de una tabla

Vamos a modificar nuestra tabla `ventas` añadiéndole una nueva columna llamada `FECHA` de tipo date que será rellenada con la fecha de hoy:

```
alter table ventas add(fecha date default SYSDATE);
```

La función SYSDATE devuelve la fecha y hora del sistema.

- Comprobar que se ha variado el esquema de dicha tabla y su contenido.

## 4.2. El tipo DATE

El tipo DATE sirve para almacenar información de fechas y su rango va de 1 de Enero de 4712 BC a 31 de Diciembre de 9999.

Aunque los datos de fecha podrían representarse mediante los tipos VARCHAR y NUMBER, el tipo DATE ofrece, además, funciones específicas para su manejo que tienen en cuenta su semántica.

Para especificar un valor de tipo DATE, debe convertirse un valor de cadena de caracteres o un número a un valor válido de fecha, usando la función **TO_DATE**. Con esta función, el valor de fecha suministrado como cadena se traducirá a un valor de fecha con un formato por defecto, que es el que se suministró en la instalación (actualmente se corresponde con DD/MM/AA, esto es, dos dígitos para el día, dos dígitos para el mes y dos dígitos para el año).

Un ejemplo de uso de la función `TO_DATE` es el siguiente:

```
SQL> insert into ventas values ('S5','P2','J4',1200,TO_DATE('17/02/67'));
```

Para la recuperación de datos de tipo fecha en *un formato concreto* [1], la función que debe utilizarse es **TO_CHAR**, que transforma un valor de fecha en su formato interno a una cadena de caracteres imprimible según el formato fecha especificado.

[1] En el caso de que se desee tener un formato distinto del que hay establecido por defecto.

Algunos ejemplos de uso de la función `TO_CHAR` son los siguientes:

```
SQL> select TO_CHAR(fecha,'dd-mon-yy') from ventas;

Fecha
-----
17-Feb-67

SQL> select TO_CHAR(TO_DATE('27-OCT-98', 'DD-MON-YY') ,'YYYY') "Year" from DUAL;

Year
----
1998
```

### 4.2.1. Formato de fecha

La especificación de un formato de fecha puede contener cualquiera de los símbolos que aparecen en la tabla 4.1 combinados de diferentes formas:

| | |
|---|---|
| -/,.;:"texto" | Separadores permitidos. |
| D | Día de la semana entre 1 y 7. |
| DAY/day | Nombre del día (lunes, martes,...) |
| DD | Día del mes entre 1 y 31. |
| DDD | Dìa del año entre 1 y 365. |
| J | Número de día según calendario Juliano. |
| MM | Dos dígitos para el mes. |
| MON | Tres primeros caracteres del mes. |
| MONTH´/month | Nombre completo del mes. |
| YYYY | Cuatro dígitos para el año. |
| Y,YYY | Cuatro dígitos con separador. |
| YEAR/year | Nombre del año. |
| YY | Ultimos dos dígitos del año. |

Cuadro 4.1: Elementos de un formato de fecha

Por ejemplo, formatos de fecha válidos serían:

```
'"La fecha es":DAY, DD "de" MONTH "de" YYYY'
'dd:month:year'
'mm/yy'
```

Los caracteres de la fecha recuperada mediante una sentencia `select` aparecerán en la misma posición y con las mismas características que se especificaron en el formato de la función TO_CHAR..

### 4.2.2. Aritmética de fechas

Oracle permite sumar y restar valores constantes y otras fechas a los datos de tipo fecha. Para ello, la fecha se representa internamente como un único número (número de días); así, por ejemplo, SYSDATE + 1 es mañana , SYSDATE - 7 es *hace una semana* y SYSDATE + (10/1440) es *dentro de diez minuto.*

### 4.2.3. Uso del calendario Juliano

Una fecha según el calendario Juliano, es un número de día desde el 1 de Enero de 4712 BC. Este formato de fecha permite tener un referencial contínuo para el almacenamiento y la manipulación de fechas. El formato de fecha *J*, utilizado en las funciones TO_DATE y TO_CHAR establece esta representación.

*Ejemplo.-*
La siguiente sentencia devuelve el valor juliano equivalente a 1 de Enero de 1997.

```
select TO_CHAR(TO_DATE('01-01-1997', 'MM-DD-YYYY'),'J') from  DUAL;

TO_CHAR
--------
2450450
```

### 4.2.4. Ejemplo

A continuación vamos a insertar algunas tuplas nuevas en `ventas` mediante la edición de un nuevo archivo en el que se considere la existencia de la nueva columna `FECHA`. Un ejemplo de tupla a insertar sería:

```
insert into ventas values('S5','P1','J4',400,TO_DATE('25/12/00'));
```

Para seleccionar la columna FECHA con un formato específico e imprimirla:

```
select codpro,codpie,to_char(fecha,'"Dia" day,dd/mm/yy') from ventas;
```

donde el texto que se quiere incluir como parte de la fecha debe ir entre comillas dobles.
Si se omite la función `TO_CHAR` en la sentencia `select`, el formato aplicado será el que haya por defecto.

### 4.2.5. Ejercicios

1. Recuperar la información de la columna FECHA con diferentes formatos.
2. Encontrar los nombres de los proveedores que suministraron sus pedidos antes del 1 de enero de 2001.
3. Agrupar los suministros de la tabla de ventas por años y sumar las cantidades totales anuales.
4. Encontrar la media de productos suministrados cada mes.
5. Insertar una antigua venta realizada por S4 de 377 piezas P5 a J3 el 12 de Enero del 98.
6. Comprueba la fecha de la venta recién introducida con 4 dígitos para el año.

## 4.3. Borrado de una tupla

La instruccion DELETE se utiliza para eliminar tuplas de una tabla, el comando es:

```
DELETE [FROM] nombre_tabla
[WHERE (condicion)];
```

Donde `[]` indica opcionalidad, esto es la clausula WHERE con su sintaxis habitual es opcional. `condicion` es cualquier expresión lógica similar a las utilizadas en la sentencia SELECT.
*Ejemplo:*

```
DELETE FROM  pieza;
```

La instrucción de borrado sin clausula WHERE, borra todas las tuplas de la tabla. En este caso da un mensaje de error (¿porqué?). Pero sí podríamos borrar la tabla `ventas`.

*Ejercicios:*

- Elimina todas las ventas realizadas antes del 2001.
- Elimina las ventas con cantidades comprendidas entre 291 y 301.
- Elimina los proyectos que no tienen ningún pedido.

## 4.4. Modificación de una tupla

Hasta el momento, si queremos modificar una tupla, la borramos previamente y la introducimos nuevamente con los nuevos valores. Esto, se evitará con el comando de modificación UPDATE. Este comando modifica el o los registros que se ajustan al criterio especificado en la clausula WHERE.

```
UPDATE nombre_tabla
SET nombre_atributo =
    'nuevovalor'
[, nombre_atributo2 =
   'nuevovalor2'...]
[WHERE  'condicion' ];
```

Donde `[]` indica opcionalidad. Así se puede modificar un atributo o más de un atributo simultáneamente. Por otro lado, igual que para la instrucción DELETE, la clausula WHERE con su sintaxis habitual es opcional. Ejemplo:

```
update ventas
  set cantidad = cantidad*3
  where codpie='P1' and codpro='S1' and codpj='J1';
```

*Ejercicios:*

- Modifica las fechas de todas las ventas para pasarlas al día después.
- Todos los proveedores de Madrid se han mudado a Chinchon; modifica las tuplas pertinentes.
- Se ha cambiado el convenio para los pesos de las piezas y hay que reducirlos todos en un tercio.
- El proyecto 4 se renombra a Proyecto último y su ciudad pasa a ser Paris
- Las ventas comprendidas entre 1500 y 2000 se establecen a 1800.

# Capítulo 5

# Sesión de Trabajo 4ª

## 5.1. Gestión de índices y clusters

### 5.1.1. Índices

El índice de un libro nos permite localizar información de una manera rápida sin la necesidad de hacer una búsqueda intensiva por todas las páginas del libro. El índice es una estructura adicional al contenido del libro en que se encuentran ordenadas las palabras importantes del mismo junto con un número que indica la página exacta en que se encuentra cada una de ellas. Por supuesto, la información contenida en el libro es la misma con índice o sin él. El índice sólo influye en la velocidad con que se encuentra la información buscada.

En el mundo de las bases de datos relacionales, los índices son estructuras que se pueden crear asociadas a tablas con el objetivo de acelerar algunas sentencias SQL ejecutadas sobre ellas.

La ausencia o presencia de un índice asociado a una tabla no influye en la sintaxis de las sentencias SQL ejecutadas sobre esa tabla. El índice es un medio usado de forma automática por Oracle para acceder de forma rápida a la información de la tabla.

Oracle hace uso de los índices para incrementar el rendimiento cuando:

- Busca registros con valores específicos en columnas indexadas
- Accede a una tabla en el orden de las columnas del índice

#### 5.1.1.1. Selección de índices

Los índices son creados para aumentar la eficiencia en los accesos a las tablas de la base de datos. A cambio se paga con el espacio extra ocupado por los índices y el tiempo necesario para mantenerlos.

Aunque los índices agilizan la recuperación de los datos de las tablas, sin embargo ralentizan las actualizaciones y ocupan espacio en disco, ya que, cuando se realizan actualizaciones de una tabla de la base de datos, cada uno de los índices basados en esa tabla necesita también una actualización.

Antes de crear un índice se debe determinar si es necesario, es decir, se debe estimar si el beneficio en el rendimiento que se obtendrá supera al costo derivado de su mantenimiento.

El usuario sólo debe crear aquellos índices que considere necesarios y Oracle se encargará de usarlos y mantenerlos automáticamente. El mantenimiento implica la modificación necesaria del índice cuando se añaden, modifican o eliminan registros de la tabla.

Cuando creamos una tabla, Oracle crea automáticamente un índice asociado a la llave primaria de la misma. Por tanto, no es conveniente crear índices sobre la llave primaria porque no sólo no se consigue mejorar el rendimiento sino que se empeora al tener que mantener un índice más.

Por tanto, es importante utilizar los índices que se necesiten realmente y se deben borrar si no van a necesitarse de nuevo.

#### 5.1.1.2. Creación de índices

Para la creación de índices usaremos la sentencia CREATE INDEX cuya sintaxis básica es:

```
CREATE INDEX nombre_del_indice ON tabla(campo [ASC | DESC],...)
```

Por ejemplo, si queremos acelerar las consultas cuando busquemos a un proveedor por su nombre, podemos crear un índice asociado al campo nompro de la tabla proveedor.

```
CREATE INDEX indice_proveedores ON proveedor(nompro);
```

Cuando se crea una tabla es mejor insertar los registros antes de crear el índice. Si se crea antes el índice, Oracle deberá actualizarlo cada vez que se inserte un registro.

#### 5.1.1.3. Índices compuestos

Un índice compuesto es aquél creado sobre más de un campo de la tabla.

Los índices compuestos pueden acelerar la recuperación de información cuando la condición de la consulta correspondiente referencie a todos los campos indexados o sólo a los primeros. Por tanto, el orden de las columnas usado en la definición del índice es importante; generalmente, los campos por los que se accede con mayor frecuencia se colocan antes en la creación del índice.

Supongamos que tenemos una tabla *libros* con los campos *código*, *título*, *autor*, *editorial* y *género* y creamos un índice compuesto sobre los campos *género*, *título* y *editorial* :

```
CREATE INDEX indicelibros ON libros(genero,titulo,editorial);
```

Dicho índice acelerará la recuperación de información de las consultas cuya condición incluya referencias al campo *género*, a los campos *género* y *título* o a los campos *género*, *título* y *editorial*. Por ejemplo:

```
SELECT codigo FROM libros WHERE genero='novela' and
                                titulo='Sin noticias de Gurb';

SELECT codigo FROM libros WHERE genero='novela' and
                                titulo='Sin noticias de Gurb' and
                                editorial='Planeta';
```

Sin embargo no mejorará el acceso cuando las consultas no incluyan referencias a los primeros campos del índice. Por ejemplo:

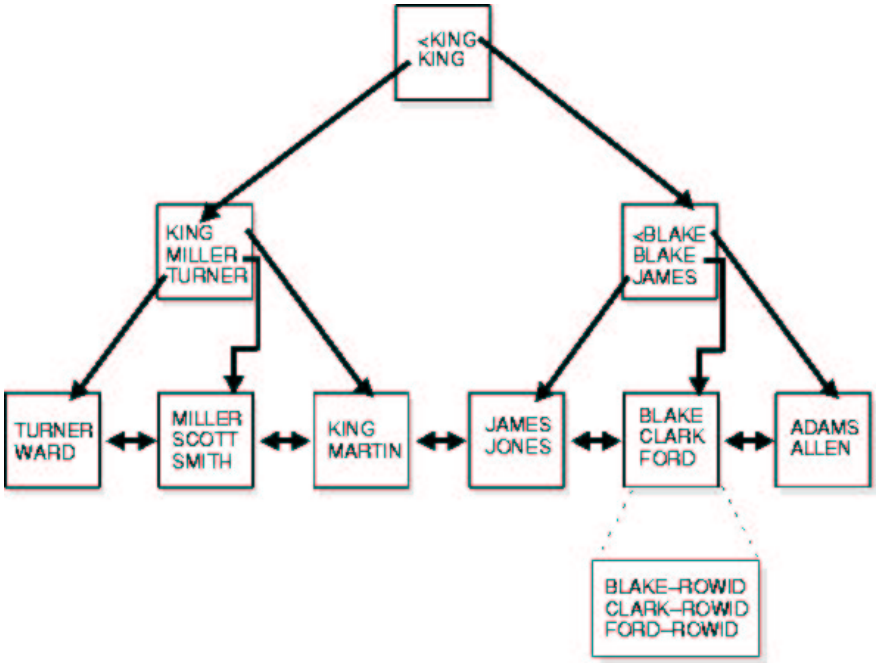
```
SELECT codigo FROM libros WHERE titulo='Sin noticias de Gurb';

SELECT codigo FROM libros WHERE genero='novela' and
                                editorial='Planeta';
```

#### 5.1.1.4. Estructura de los índices

Cuando se crea un índice, Oracle recupera los campos indexados de la tabla y los ordena. A continuación almacena en una estructura especial los valores de los campos indexados junto con un identificador (ROWID) del registro correspondiente.

Oracle usa árboles B* balanceados para igualar el tiempo necesario para acceder a cualquier fila.




#### 5.1.1.5. Eliminación de índices

Son varias las razones por las que puede interesar eliminar un índice:

- El índice no se necesitará más
- Por las características de la tabla el índice no mejora la eficiencia
- Necesitamos cambiar los campos que se indexan
- Necesitamos rehacer un índice muy fragmentado

Para eliminar un índice usaremos la sentencia:

```
DROP INDEX nombre_del_indice;
```

Se debe tener en cuenta que cuando se borra una tabla también se borran todos los índices asociados.

Los índices que Oracle crea asociados a las llaves primarias sólo se pueden borrar eliminando dichas restricciones.

### 5.1.2. Clusters

Un cluster proporciona un método alternativo de almacenar información en las tablas. Un cluster lo forman un conjunto de tablas que se almacenan en los mismos bloques de datos porque comparten campos comunes (llave del cluster) y son frecuentemente accedidas de forma conjunta.

Por ejemplo, las tablas proveedor y ventas comparten el campo codpro. Si se incluyen las dos tablas en el mismo cluster, Oracle físicamente almacena todas las filas de cada codpro de ambas tablas en los mismos bloques de datos. No se deben usar clusters con tablas que son accedidas frecuentemente de forma individual.

Supongamos que la tabla proveedor contiene la siguiente información:

| CODPRO | NOMPRO | STATUS | CIUDAD |
|---|---|---|---|
| S1 | Antonio Martín | 5 | Madrid |
| S2 | David Bailón | 8 | Granada |
| S3 | Lara Croft | 9 | Londres |

Supongamos que la tabla ventas contiene la siguiente información:

| CODPRO | CODPIE | CODPJ | CANTIDAD |
|---|---|---|---|
| S1 | P2 | J1 | 50 |
| S2 | P1 | J2 | 20 |
| S2 | P4 | J3 | 30 |
| S3 | P3 | J3 | 50 |
| S1 | P2 | J2 | 20 |
| S3 | P3 | J1 | 40 |

Si incluimos ambas tablas en el mismo cluster, compartiendo el campo codpro, el resultado será:

| CODPRO<br>S1 | NOMPRO | STATUS | CIUDAD |
|---|---|---|---|
| | Antonio Martín | 5 | Madrid |
| | CODPIE | CODPJ | CANTIDAD |
| | P2 | J1 | 50 |
| | P2 | J2 | 20 |
| S2 | NOMPRO | STATUS | CIUDAD |
| | David Bailón | 8 | Granada |
| | CODPIE | CODPJ | CANTIDAD |
| | P1 | J2 | 20 |
| | P4 | J3 | 30 |
| S3 | NOMPRO | STATUS | CIUDAD |
| | Lara Croft | 9 | Londres |
| | CODPIE | CODPJ | CANTIDAD |
| | P3 | J3 | 50 |
| | P3 | J1 | 40 |

Como los clusters almacenan registros relacionados en los mismos bloques de datos, los beneficios obtenidos son:

- Se reduce el acceso a disco y se mejora el rendimiento en reuniones de las tablas del cluster
- La llave del cluster (columnas comunes entre las tablas del cluster) almacena cada valor una sola vez de modo independiente al número de registros de las tablas que contengan dicho valor

Después de crear un cluster se pueden crear las tablas que pertenecen al cluster. No obstante, antes de

que se puedan insertar registros en las tablas del cluster es necesario indexar el cluster.

#### 5.1.2.1. Selección de tablas y llave del cluster

El cluster se debe usar para almacenar tablas que son principalmente consultadas y escasamente modificadas. Además, dichas consultas deben reunir varias tablas del cluster.

Si no se cumplen estos requisitos, el cluster resultante tendrá un efecto negativo sobre el rendimiento porque no aprovechamos las ventajas que nos ofrece y sin embargo estamos pagando el costo de su mantenimiento.

Se debe elegir la llave del cluster con cuidado. Si se usan varios campos en consultas que reunen las tablas, se debe usar una llave compuesta.

Una buena llave del cluster debe tener un número adecuado de valores distintos para que la información asociada a cada valor de la llave ocupe aproximadamente un bloque de datos.

Si la cantidad de valores distintos de la llave del cluster es muy pequeña se desperdicia espacio y la mejora en el rendimiento es muy baja. Por ello es conveniente evitar llaves de cluster demasiado específicas.

Cuando la cantidad de valores distintos es demasiado alta, la búsqueda del registro a partir de la llave del cluster es lenta. Por tanto, se deben evitar llaves de cluster demasiado generales.

#### 5.1.2.2. Creación de un cluster

Para la creación de un cluster se emplea la sentencia CREATE CLUSTER cuya sintaxis básica es:

```
CREATE CLUSTER nombre_del_cluster(campo tipo_de_dato);
```

Por ejemplo, para crer el cluster del ejemplo mostrado anteriormente, usamos la sentencia:

```
CREATE CLUSTER cluster_codpro(codpro varchar2);
```

#### 5.1.2.3. Creación de las tablas del cluster

Una vez que se ha creado un cluster se pueden crear las tablas que contendrá. Para ello se emplea la sentencia CREATE TABLE usando la opción CLUSTER en la que se indica el nombre del cluster al que pertenecerá la tabla y, entre paréntesis, el nombre del campo o campos que son la llave del cluster.

```
CREATE TABLE proveedor(
   codpro varchar2 primary key,
   nompro varchar2 not null,
   status number check(status>=1 and status<=10),
   ciudad varchar2)
   CLUSTER cluster_codpro(codpro);

CREATE TABLE ventas(
   codpro references proveedor(codpro),
   codpie references pieza(codpie),
   codpj references proyecto(codpj),
   cantidad numbre(4),
   primary key (codpro,codpie,codpj))
   CLUSTER cluster_codpro(codpro);
```

#### 5.1.2.4. Creación del índice del cluster

Antes de que se pueda insertar información en las tablas del cluster es necesario que se haya creado un índice. Esto se logra mediante la sentencia CREATE INDEX. Por ejemplo, para crear el índice del cluster del ejemplo usaremos:

```
CREATE INDEX indice_cluster ON cluster_codpro;
```

#### 5.1.2.5. Eliminación de clusters

Un cluster puede ser eliminado si las tablas del cluster no son necesarias. Cuando se borra el cluster también se borra el índice del cluster.

Para la eliminación de un cluster se emplea la sentencia DROP CLUSTER cuya sintaxis es:

```
DROP CLUSTER nombre_del_cluster
    [INCLUDING TABLES [CASCADE CONSTRAINTS]];
```

Si un cluster contiene tablas no podrá ser eliminado a menos que se eliminen antes las tablas o que se incluya en la sentencia la opción INCLUDING TABLES.

El cluster no podrá ser borrado si las tablas contienen campos que son referenciados por llaves exteriores localizadas en tablas externas al cluster. Para que se pueda borrar el cluster se deben eliminar las referencias externas mediante la opción CASCADE CONSTRAINTS.

#### 5.1.2.6. Eliminación de tablas del cluster

Se pueden eliminar tablas individuales de un cluster usando la sentencia DROP TABLE.

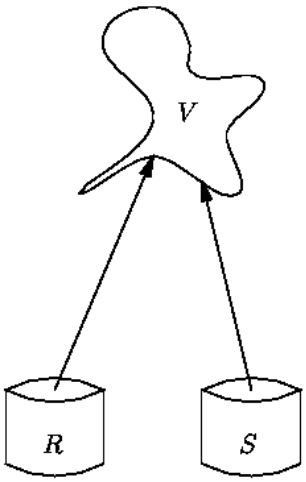
La eliminación de una tabla del cluster no afecta al cluster ni a las otras tablas ni al índice del cluster.

#### 5.1.2.7. Eliminación del índice del cluster

El índice de un cluster puede ser eliminado en cualquier momento sin afectar al cluster ni a las tablas del mismo. Sin embargo no será posible acceder a la información contenida en las tablas hasta que se reconstruya el índice. Para la eliminación del índice se emplea la sentencia DROP INDEX.

## 5.2. Gestión de vistas

Una vista es una tabla lógica cuya información se deriva de una tabla, de un conjunto de ellas o bien de otras vistas de la base de datos. Si el contenido de las tablas cambia, este cambio se ve reflejado en las vistas. Dicho de otra forma: es una expresión que describe una tabla sin crearla.




### Definición de vistas

```
CREATE VIEW <nombre> AS <consulta>;
```

Supongamos la vista `VentasLondres`, que representa el conjunto de suministros realizados sólo con integrantes procedentes de Londres.

```
CREATE VIEW VentasLondres AS
    SELECT codpro,codpie,codpj,cantidad
    FROM ventas
    WHERE (codpro,codpie,codpj) IN
        (SELECT codpro,codpie,codpj
         FROM proveedor,pieza,proyecto
         WHERE proveedor.ciudad='Londres' and
               pieza.ciudad='Londres' and
               proyecto.ciudad='Londres');
```

En la cláusula `AS` se especifica la consulta que determina qué filas y columnas de la tabla o tablas almacenadas forman parte de la vista.

### Consulta de vistas

Se trata la vista como una relación cualquiera. Así por ejemplo, podemos consultar la relación `VentasLondres` y de ella mostrar los códigos de proveedores que suministran al proyecto J4.

```
SELECT distinct codpro
FROM VentasLondres
WHERE codpj='J4';
```

### Modificación en las vistas

Se pueden utilizar los comandos `DELETE, INSERT, UPDATE` sólo en determinadas ocasiones. La razón es que puesto que la vista no existe como tabla base (relación almacenada), para el caso de insertar una nueva tupla, la cuestión es dónde encajaría esa tupla? En otros casos se plantea cómo habría que completar algunos atributos de las tuplas. Por ejemplo supongamos la vista siguiente:

```
CREATE VIEW PiezasLondres AS
       SELECT codpie,nompie, color,peso FROM Piezas
       WHERE  pieza.ciudad='Londres';
```

Si hacemos una inserción del tipo:

```
INSERT INTO  PiezasLondres VALUES('P9','Pieza nueve','rojo',90);
```

La vista PiezasLondres cumple las condiciones para actualizar la tabla Piezas, pero inserta NULL como valor para Ciudad. Bastaría con añadir un campo más en la inserción con el valor 'Londres'pero el atributo Ciudad no pertenece a la vista. Por tanto habría que redefinir la vista para que incluyera Ciudad.

El comando para **borrar una vista** es :

```
DROP VIEW <vista>
```

Para eliminar la vista de nuestro ejemplo la sentencia será:

```
DROP VIEW VentasLondres;
```

## 5.3. Generalidades sobre el catálogo de la base de datos

Vamos a dedicar esta sección a conocer qué es el catálogo o diccionario de datos de un sistema gestor de bases de datos. Además, vamos a estudiar la información que éste pone a disposición de todos los usuarios de nuestra base de datos.

### 5.3.1. Qué es el catálogo

El catálogo de la base de datos está formado por una serie de tablas y vistas que almacenan datos sobre todos los objetos que hay en nuestra base de datos (usuarios, roles, privilegios, tablas, restricciones, ...). Sobre estas tablas hay definidas una serie de vistas que nos permiten acceder a nuestra información. Para poder ver las tablas en su totalidad hacen falta privilegios especiales, y para modificarlas también.

#### 5.3.1.1. Algunas vistas relevantes del catálogo de la base de datos

La tabla 5.1 muestra algunas vistas relevantes que podemos consultar en el catálogo. Algunas de ellas ya las conocemos. Por ejemplo la definición de la vista USER_TABLES contiene información no sólo sobre el nombre de la tabla TABLE_NAME (que hasta hora hemos estado utilizando) sino también sobre el espacio de tablas donde se encuentra almacenada (TABLESPACE_NAME), información estadística sobre su tamaño (NUM_ROWS, AVG_SPACE), y muchas otras cosas más. La vista USER_CONSTRAINTS nos explica para cada restricción su nombre CONSTRAINT_NAME, su tipo CONSTRAINT_TYPE, la tabla sobre la que está definida TABLE_NAME, y otros datos específicos de cada tipo de restricción. Si queremos consultar qué columnas están asociadas a la restricción lo tendremos que hacer sobre la vista USER_CONS_COLUMNS. La definición de todas las columnas de nuestras tablas, la podemos consultar en la vista USER_TAB_COLUMS: allí encontraremos información como el nombre de la columna (COLUMN_NAME), la tabla en la que está definida (TABLE_NAME), su orden en la definicición (COLUMN_ID), su tipo (DATA_TYPE), y otra serie de campos para determinar las distintas peculiaridades de cada tipo de dato (longitud, precision, posibilidad de admitir nulos, ...).

| Vista | Descripción |
|---|---|
| ALL_ALL_TABLES | Todas las tablas accesibles para el usuario. |
| ALL_CONS_COLUMNS | Columnas y definiciones de restricciones. |
| ALL_CONSTRAINTS | Definiciones de restricciones en las tablas accesibles. |
| ALL_ERRORS | Errores en todos los objetos accesibles para el usuario. |
| ALL_INDEXES | Indices en tablas accesibles para el usuario |
| ALL_OBJECTS | Objetos accesibles para el usuario. |
| ALL_SEQUENCES | Secuencias accesibles para el usuario. |
| ALL_TAB_COLUMNS | Columnas de todas las tablas, vistas y clusters accesibles para el usuario. |
| ALL_TAB_PRIVS | Privilegios sobre objetos para los que el usuario o PUBLIC están autorizados. |
| ALL_TAB_PRIVS_MADE | Privilegios autorizados por y para el usuario sobre sus objetos. |
| ALL_TABLES | Tablas relacionales accesibles para el usuario. |
| ALL_USERS | Todos los usuarios de la base de datos. |
| ALL_VIEWS | Vistas accesibles para el usuario. |
| DICTIONARY | Tablas y vistas del catálogo. |
| USER_ALL_TABLES | Todas las tablas accesibles para el usuario. |
| USER_CATALOG | Tablas, vistas, sinónimos y secuencias propiedad del usuario. |
| USER_CONSTRAINTS | Definiciones de restricciones en las tablas del usuario. |
| USER_CONS_COLUMNS | Columnas y definiciones de restricciones propiedad del usuario. |
| USER_ERRORS | Errores en todos los objetos almacenados del usuario. |
| USER_FREE_SPACE | Espacio libre en los espacios de tablas asignados al usuario. |
| USER_INDEXES | índices propios del usuario. |
| USER_OBJECTS | Objetos propiedad del usuario. |
| USER_ROLE_PRIVS | Roles autorizados al usuario. |
| USER_SEQUENCES | Secuencias del usuario. |
| USER_SYS_PRIVS | Privilegios del sistema concedidos al usuario. |
| USER_TAB_COLUMNS | Columnas en las tablas, vistas y clusters del usuario. |
| USER_TAB_PRIVS | Privilegios sobre objetos para los que el usuario es el propietario, autorizador o autorizado. |
| USER_TABLES | Tablas relacionales del usuario. |
| USER_TABLESPACES | Espacios de tablas accesibles. |
| USER_TRIGGERS | Disparadores del usuario. |
| USER_USERS | Información sobre el usuario actual. |
| USER_VIEWS | Vistas propiedad del usuario. |

Cuadro 5.1: Vistas relevantes del catálogo

#### 5.3.1.2. Ejercicios

- Consulta la definición de alguna de las vistas anteriores utilizando el comando DESCRIBE. Para entender el significado de una buena parte de los campos que las caracterizan, hay que tener un conocimiento muy profundo del funcionamiento de la base de datos, que escapa al objetivo de estas prácticas. Sin embargo, con lo que sabes hasta ahora, si debes intentar comprobar el sentido general de estas vistas e identificar los campos relevantes que te permiten extraer información de las mismas.
- Crea en tu cuenta de la base de datos una tabla nueva

  *DD_Prueba*. Procura que su definición sea completa (varias columnas, tipos de datos, restricciones, privilegios, etc.). Haz un seguimiento de los cambios que se producen en las vistas enumeradas en la tabla anterior.

### 5.3.2. Gestión de privilegios

El objetivo de este apartado es identificar los distintos tipos de privilegios que se pueden manejar en Oracle, distinguiendo entre privilegios sobre el sistema y sobre los objetos. También haremos algunas prácticas sobre cómo autorizar y derogar privilegios.

#### 5.3.2.1. Privilegios del sistema

Este tipo de privilegios permiten al usuario llevar a cabo acciones particulares en la base de datos. Hay aproximadamente 80 privilegios, y su número continúa creciendo versión a versión.
Estos privilegios pueden clasificarse como sigue:

- Privilegios que autorizan operaciones de alto nivel en el sistema (como por ejemplo CREATE SESSION y CREATE TABLESPACE).
- Privilegios que autorizan la gestión de objetos en el propio esquema de usuario (como por ejemplo CREATE TABLE).
- Privilegios que autorizan la gestión de objetos en cualquier esquema (como por ejemplo CREATE ANY TABLE).

Hay dos comandos del DDL que se encargan de controlar los privilegios, GRANT y REVOKE. Estos permiten conceder y derogar privilegios a un usario o a un role.

**◇ Concesión de un privilegio de sistema**

GRANT {system_priv | role} [,{system_priv | role}] ... TO {user | role | PUBLIC} [,{user | role | PUBLIC}]... WITH ADMIN OPTION

- PUBLIC se refiere a todos los usuarios.
- WITH ADMIN OPTION permite que el usuario autorizado pueda conceder a su vez el privilegio a otros.

**◇ Derogación de privilegios de sistema**

REVOKE {system_priv | role} [,{system_priv | role}] ... FROM {user | role | PUBLIC} [,{user | role | PUBLIC}]...

- Sólo permite derogar privilegios que hayan sido explícitamente concedidos mediante el uso de GRANT.
- Hay que vigilar los efectos sobre otros privilegios al derogar uno dado.
- No hay efecto de cascada aunque se haya usado en el GRANT la opción WITH ADMIN OPTION.

#### 5.3.2.2. Privilegios sobre los objetos

La concesión de este tipo de privilegios autoriza la realización de ciertas operaciones sobre objetos concretos. La siguiente tabla muestra los distintos privilegios que se pueden utilizar en relación con los diferentes objetos de la base de datos.

| Privilegio | Tabla | Vista | Secuencia | Procedimiento |
|---|---|---|---|---|
| ALTER | X | | X | |
| DELETE | X | X | | |
| EXECUTE | | | | X |
| INDEX | X | | | |
| INSERT | X | X | | |
| REFERENCES | X | | | |
| SELECT | X | X | X | |
| UPDATE | X | X | | |

◇ **Concesión de privilegios de objetos**

GRANT {object_priv [(column_list)] [,object_priv [(column_list)]] ... | ALL [PRIVILEGES]
ON [schema.]object
TO {user | role | PUBLIC} [,{user | role | PUBLIC}]... [WITH GRANT OPTION]

- La lista de columnas sólo tiene sentido cuando se refieren a privilegios INSERT, REFERENCES o UPDATE.
- ALL se refiere a todos los privilegios que han sido concedidos sobre el objeto con WITH GRANT OPTION.
- WITH GRANT OPTION autoriza al usuario para conceder a su vez el privilegio. No se puede usar al conceder privilegios a roles.

◇ **Derogación de privilegios de objeto**

REVOKE {object_priv [(column_list)] [,object_priv [(column_list)]]... | ALL [PRIVILEGES]} ON [schema.]object
FROM {user | role | PUBLIC} [,{user | role | PUBLIC}]... [CASCADE CONSTRAINTS]

- CASCADE CONSTRAINTS propaga la eliminación hacia restricciones de integridad referencial relacionadas con el privilegio REFERENCES o ALL.
- Un usario sólo puede derogar aquellos privilegios que él mismo haya concedido mediante GRANT a otro.

- Siempre se hace cascada con relación a la opción WITH GRANT OPTION.

#### 5.3.2.3. Ejercicios de gestión de privilegios

- Crear en la cuenta una tabla cualquiera.

```
CREATE TABLE acceso (testigo number);
```

- Insertar algunas tuplas de prueba.

```
INSERT INTO acceso VALUES(1);
INSERT INTO acceso VALUES(2);
```

- Autorizar al usuario de tu derecha para que pueda hacer consultas sobre esa tabla.

```
GRANT SELECT ON acceso TO usuario_derecha;
```

- Comprobar que se puede acceder a la tabla del usuario de la izquierda.

```
SELECT * FROM usuario_izquierda.acceso;
```

- Retirar el privilegio de consulta antes concedido.

```
REVOKE SELECT ON acceso FROM usuario_derecha;
```

- Autorizar ahora al usuario de la derecha para que pueda hacer consultas sobre la tabla, pero ahora con posibilidad de que este propague ese privilegio.

```
GRANT SELECT ON acceso TO usuario_derecha WITH GRANT OPTION;
```

- Propagar el privilegio concedido por el usuario de la izquierda hacia el usuario de la derecha.

```
GRANT SELECT ON usuario_izquierda.acceso TO usuario_derecha;
```

- Comprobar que se pueden acceder a las tablas del usuario de la derecha y del anterior.

```
SELECT * FROM usuario_izquierda.acceso;
SELECT * FROM usuario_izquierda_del_usuario_izquierda.acceso;
```

- Retira el privilegio antes concedido. ¿Qué ocurre con los accesos?

```
REVOKE SELECT ON acceso TO usuario_derecha;
SELECT * FROM usuario_izquierda.acceso;
SELECT * FROM usuario_izquierda_del_usuario_izquierda.acceso;
```