

---

<b>-PARTE III-</b> .....	<b>3</b>
<b>TEMA 8. CREACIÓN DE VISTAS:</b> .....	<b>3</b>
Vistas: ¿qué son y para qué sirven?.....	3
Creación de vistas.....	4
<b>Tipos de vistas</b> .....	<b>7</b>
Vistas con filtro de filas y de columnas.....	7
Agrupaciones.....	7
Composiciones.....	8
Vistas sobre vistas.....	10
<b>Restricciones para la creación y utilización de vistas</b> .....	<b>11</b>
<b>Actualizaciones en vistas</b> .....	<b>11</b>
<b>Vistas con validación</b> .....	<b>12</b>
<b>Eliminación de vistas</b> .....	<b>14</b>
<b>TEMA 9. CREACIÓN DE TABLAS</b> .....	<b>16</b>
Formato genérico para la creación de tablas.....	16
Restricciones de tabla y de columna.....	19
Modificación de la definición de tabla.....	35
<b>UNIDAD 10. SEGURIDAD EN SQL</b> .....	<b>44</b>
<b>Introducción a la seguridad en los SGBDR</b> .....	<b>44</b>
<b>Usuarios: creación</b> .....	<b>44</b>
<b>Privilegios</b> .....	<b>45</b>
Privilegios del sistema: .....	45
Privilegios sobre objetos: .....	46
<b>Retirada de privilegios</b> .....	<b>47</b>
Retirada de privilegios del sistema.....	47
Retirada de privilegios de objeto.....	48
<b>Roles</b> .....	<b>49</b>
Roles predefinidos.....	50

---

<b>Privilegios con opción de administración.....</b>	<b>52</b>
Privilegios del sistema con opción de administración.....	52
Privilegios de objeto con opción de administración.....	53
Retirada de la opción de administración de privilegios.....	54
<b>Utilización de sinónimos públicos y privados.....</b>	<b>55</b>
Sinónimos públicos.....	57
Eliminación de sinónimos.....	57
<b>TEMA 11. SEGURIDAD EN ACCESS.....</b>	<b>59</b>
<b>Habilitar una contraseña para abrir la base de datos.....</b>	<b>59</b>
<b>Protección de objetos mediante la seguridad por usuarios.....</b>	<b>60</b>
<b>Gestión de usuarios y grupos.....</b>	<b>67</b>
Dar permisos a los usuarios y grupos sobre los objetos de la base de datos.....	67
Asignar un grupo a un usuario.....	70
<b>ANEXO .....</b>	<b>72</b>
<b>FUNCIONES AVANZADAS Y CARACTERÍSTICAS ESPECIALES.....</b>	<b>72</b>
<b>Introducción.....</b>	<b>72</b>
<b>Recuperación jerárquica.....</b>	<b>72</b>
La pseudocolumna LEVEL.....	73
<b>La función DECODE.....</b>	<b>74</b>
<b>Disparadores.....</b>	<b>76</b>
<b>Utilización del diccionario de datos.....</b>	<b>77</b>
<b>BIBLIOGRAFÍA E INFORMACIÓN COMPLEMENTARIA.....</b>	<b>81</b>

---

## -PARTE III-

### Tema 8. CREACIÓN DE VISTAS:

---

*Autor: Fernando Montero*

#### ***Vistas: ¿qué son y para qué sirven?.***

---

**Podemos definir una vista como una consulta almacenada en la base de datos que se utiliza como una tabla virtual.**

Se trata de una perspectiva de la base de datos o ventana que permite a uno o varios usuarios ver solamente las filas y columnas necesarias para su trabajo.

Entre las ventajas que ofrece la utilización de vistas cabe destacar:

- **Seguridad y confidencialidad:** ya que la vista ocultará los datos confidenciales o aquellos para los que el usuario no tenga permiso.
- **Comodidad :** ya que solamente muestra los datos relevantes, permitiendo, incluso trabajar con agrupaciones de filas como si se tratase de una única fila o con composiciones de varias tablas como si se tratase de una única tabla.
- **Independencia respecto a posibles cambios** en los nombres de las columnas, de las tablas, etcétera.

Por ejemplo, la siguiente consulta permite al departamento de VENTAS realizar la gestión de sus empleados ocultando la información relativa a los empleados de otros departamentos.

```
SQL> SELECT * FROM EMPLEADOS WHERE DEP_NO = 30;
```

EMP_NO	APELLIDO	OFICIO	DIRECTOR	FECHA_AL	SALARIO	COMISIÓN	DEP_NO
7499	ALONSO	VENDEDOR	7698	20/02/81	140000	40000	30
7654	MARTIN	VENDEDOR	7698	28/09/81	150000	160000	30
7698	GARRIDO	DIRECTOR	7839	01/05/81	385000		30
7844	CALVO	VENDEDOR	7698	08/09/81	180000	0	30

4 filas seleccionadas.

La siguiente consulta permite a cualquier empleado de la empresa obtener información no confidencial de cualquier otro empleado ocultando las columnas SALARIO y COMISION:

```
SQL> SELECT emp_no, apellido, oficio, director, fecha_alta, dep_no FROM empleados;
```

EMP_NO	APELLIDO	OFICIO	DIRECTOR	FECHA_AL	DEP_NO
7499	ALONSO	VENDEDOR	7698	20/02/81	30

7521	LOPEZ	EMPLEADO	7782	08/05/81	10
7654	MARTIN	VENDEDOR	7698	28/09/81	30
7698	GARRIDO	DIRECTOR	7839	01/05/81	30
7782	MARTINEZ	DIRECTOR	7839	09/06/81	10
7839	REY	PRESIDENTE		17/11/81	10
7844	CALVO	VENDEDOR	7698	08/09/81	30
7876	GIL	ANALISTA	7782	06/05/82	20
7900	JIMENEZ	EMPLEADO	7782	24/03/83	20

9 filas seleccionadas.

## Creación de vistas.

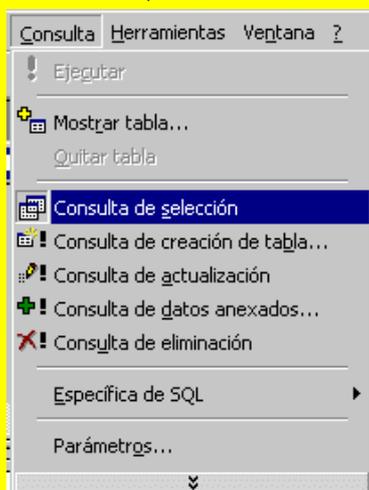
Para crear una vista se utiliza el comando CREATE VIEW según el siguiente formato genérico:

```
CREATE VIEW nombredevista [(listadecolumnas)] AS consulta;
```

Donde:

- *nombredevista* es el nombre que tendrá la vista que se va a crear.
- *listadecolumnas* es opcional. Permite especificar un nombre para cada columna de la vista. Si no se especifica, cada columna quedará con el nombre asignado por la consulta.
- *consulta* es la SELECT que define la vista.

En Access para crear vistas lo que haremos será guardar la consulta SELECT. Es la opción por defecto que se inicia en *Crear consulta en vista de diseño*. En la vista del diseño de la consulta podremos crear la **Vista** o bien pulsando al botón  *Vista SQL*, o desde el menú *Consulta / Consulta de selección*  ver Figura 19.



<- Figura 19. Consulta de selección.

Para que sea una vista de la Base de datos, guardamos la consulta y la damos un nombre.

El siguiente ejemplo crea la vista `emple_dep30` para la gestión de los empleados del departamento 30 mencionada en el apartado anterior.

```
SQL> CREATE VIEW emple_dep30 AS
      SELECT * FROM EMPLEADOS WHERE DEP_NO = 30;
```

En Access:

1° Escribimos la consulta:

```
SELECT * FROM EMPLEADOS WHERE DEP_NO = 30;
```

2° La ejecutamos pulsando  para comprobar datos.

3° Pulsamos al botón  y la llamamos `Emple_dep30`.

4° Cerramos la vista SQL y vemos la consulta que aparece creada en la ventana de la base de datos.

A continuación se muestra el comando que crea la vista `datos_emple` que contiene información de todos los empleados ocultando la información confidencial.

```
SQL> CREATE VIEW datos_emple AS
      SELECT emp_no, apellido, oficio,
             director, fecha_alta, dep_no
      FROM empleados;
```

En Access:

1° Escribimos la consulta:

```
SELECT emp_no, apellido, oficio, director, fecha_alta,
       dep_no FROM empleados;
```

2° La ejecutamos pulsando  para comprobar datos.

3° Pulsamos al botón  y la llamamos `Datos_emple`.

4° Cerramos la vista SQL y vemos la consulta que aparece creada en la ventana de la base de datos.

Una vez creada la vista se puede utilizar como si se tratase de una tabla (observando algunas restricciones que se verán más adelante).

Por ejemplo si escribimos:

```
SQL> SELECT * FROM emple_dep30;
```

O bien:

```
SQL> SELECT * FROM datos_emple;
```

En ambos casos obtendremos el mismo resultado que si introducimos la cláusula de selección en la que se basa la vista, mostrado en el apartado anterior.

También aplicar condiciones de selección, de filas y de columnas al recuperar datos de la vista:

```
SQL> SELECT apellido FROM emple_dep30:
```

```
APELLIDO
-----
ALONSO
MARTIN
GARRIDO
CALVO
```

```
SQL> SELECT apellido, director FROM datos_emple
      WHERE oficio = 'VENDEDOR';
```

```
APELLIDO  DIRECTOR
-----  -
ALONSO      7698
MARTIN      7698
CALVO       7698
```

Las vistas no ocupan espacio en la base de datos ya que lo único que se almacena es la definición de la vista. El gestor de la base de datos se encargará de comprobar los comandos SQL que hagan referencia a la vista, transformándolos en los comandos correspondientes referidos a las tablas originales, todo ello de forma transparente para el usuario.

En Access las consultas de selección las podremos utilizar igual que una tabla. En la Figura 20 vemos distintas consultas creadas en la ventana de la base de datos, cada consulta se identifica por el icono asociado:

Figura 20. Ventana de la base de datos y consultas.



## ***Tipos de vistas***

Atendiendo al tipo de consulta en la que se basa la vista podemos distinguir los siguientes tipos:

### **Vistas con filtro de filas y de columnas.**

Se crean basándose en **consultas que filtran determinadas filas o columnas**. Las vistas *emple\_dep30* y *datos\_emple* son ejemplos de este tipo de vistas.

También se pueden crear vistas que establezcan filtros de selección tanto a nivel de fila como a nivel de columna. El siguiente ejemplo crea la vista *datos\_vendedores* que muestra solamente las columnas *emp\_no*, *apellido*, *director*, *fecha\_alta*, *dep\_no*, de aquellos empleados cuyo oficio es *VENDEDOR*.

```
SQL> CREATE VIEW datos_vendedores
      (numvendedor, apellido, director, fecha_alta, dep_no) AS
      SELECT emp_no, apellido, director, fecha_alta, dep_no
      FROM empleados
      WHERE oficio = 'VENDEDOR';
```

Vista creada.

En Access:

1° Escribimos la consulta:

```
SELECT emp_no, apellido, director, fecha_alta, dep_no
FROM empleados WHERE oficio = 'VENDEDOR';
```

2° La ejecutamos pulsando  para comprobar datos.

3° Pulsamos al botón  y la llamamos Datos\_vendedores.

4° Cerramos la vista SQL.

Los datos accesibles mediante la vista creada serán:

```
SQL> select * from datos_vendedores;
```

NUMVENDEDOR	APELLIDO	DIRECTOR	FECHA_AL	DEP_NO
7499	ALONSO	7698	20/02/81	30
7654	MARTIN	7698	28/09/81	30
7844	CALVO	7698	08/09/81	30

## **Agrupaciones**

También se pueden crear vistas a partir de consultas que incluyen agrupaciones, como en el siguiente ejemplo:

```
SQL> CREATE VIEW resumen_dep
      (dep_no, num_empleados, suma_salario, suma_comision) AS
      SELECT dep_no, COUNT(emp_no), SUM(salario), SUM(comision)
      FROM empleados
      GROUP BY dep_no;
```

Vista creada.

En Access:

- 1° Escribimos la consulta:  

```
SELECT    dep_no,    COUNT(emp_no)    AS    num_empleados,
SUM(salario)
AS Suma_salario, SUM(comision) AS Suma_comision
FROM empleados GROUP BY dep_no;
```
- 2° La ejecutamos pulsando  para comprobar datos.
- 3° Pulsamos al botón  y la llamamos Resumen\_dep.
- 4° Cerramos la vista SQL.

En estos casos, **cada fila de la vista corresponderá a varias filas en la tabla original** tal como se puede comprobar en la siguiente consulta:

```
SQL> select * from resumen_dep;
```

DEP_NO	NUM_EMPLEADOS	SUMA_SALARIO	SUMA_COMISION
10	3	980000	
20	2	475000	
30	4	855000	200000

Normalmente la mayoría de las columnas de este tipo de vistas corresponden a funciones de columna tales como *SUM*, *AVERAGE*, *MAX*, *MIN*, etcétera. Por ello el estándar SQL establece en estos casos la obligatoriedad de especificar la lista de columnas.

Aunque algunos gestores de bases de datos permiten saltar esta restricción. No es aconsejable ya que las columnas correspondientes de la vista quedarán con nombres como *COUNT(EMP\_NO)*, *SUM(SALARIO)*, *SUM(COMISION)* lo cual no resulta operativo para su posterior utilización.

## Composiciones.

Una vista se puede crear a partir de una consulta que recupera información de varias tablas relacionadas.

La siguiente vista incluye información de las tablas *empleados* (*emp\_no*, *apellido* y *oficio*) y *departamentos* (*dnombre*, *localidad*) relacionadas a partir de la columna común *dep\_no*.

```
SQL> CREATE VIEW datos_emp_dep AS
      SELECT emp_no, apellido, oficio, dnombre, localidad
      FROM empleados, departamentos
      WHERE empleados.dep_no = departamentos.dep_no;
```

En Access:

1° Escribimos la consulta:

```
SELECT emp_no, apellido, oficio, dnombre, localidad
FROM empleados, departamentos
WHERE empleados.dep_no = departamentos.dep_no;
```

2° La ejecutamos pulsando  para comprobar datos.

3° Pulsamos al botón  y la llamamos Datos\_emp\_dep.

4° Cerramos la vista SQL.

El contenido de la vista será:

```
SQL> select * from datos_emp_dep;
```

EMP_NO	APELLIDO	OFICIO	DNOMBRE	LOCALIDAD
7499	ALONSO	VENDEDOR	VENTAS	MADRID
7521	LOPEZ	EMPLEADO	CONTABILIDAD	BARCELONA
7654	MARTIN	VENDEDOR	VENTAS	MADRID
7698	GARRIDO	DIRECTOR	VENTAS	MADRID
7782	MARTINEZ	DIRECTOR	CONTABILIDAD	BARCELONA
7839	REY	PRESIDENTE	CONTABILIDAD	BARCELONA
7844	CALVO	VENDEDOR	VENTAS	MADRID
7876	GIL	ANALISTA	INVESTIGACION	VALENCIA
7900	JIMENEZ	EMPLEADO	INVESTIGACION	VALENCIA

Así mismo, se pueden crear vistas que incluyan todas o varias de las posibilidades estudiadas. Por ejemplo la siguiente vista permite trabajar con datos de dos tablas, agrupados y seleccionando las filas que interesan (en este caso todos los departamentos que tengan mas de un empleado):

```
SQL> CREATE VIEW resumen_emp_dep
      (departamento, localidad, num_empleados, suma_salario)
      AS
      SELECT dnombre, localidad, COUNT(emp_no), SUM(salario)
      FROM empleados, departamentos
```

```
WHERE empleados.dep_no = departamentos.dep_no
GROUP BY dnombre, localidad
HAVING COUNT(emp_no) > 1;
```

En Access:

1° Escribimos la consulta:  
 SELECT dnombre, localidad, COUNT(emp\_no) AS Num\_empleados,  
 SUM(salario) as Suma\_salario  
 FROM empleados, departamentos  
 WHERE empleados.dep\_no = departamentos.dep\_no  
 GROUP BY dnombre, localidad HAVING COUNT(emp\_no) > 1;

2° La ejecutamos pulsando  para comprobar datos.

3° Pulsamos al botón  y la llamamos Resumen\_emp\_dep.

4° Cerramos la vista SQL.

```
SQL> SELECT * FROM RESUMEN_EMP_DEP;
```

DEPARTAMENTO	LOCALIDAD	NUM_EMPLEADOS	SUMA_SALARIO
CONTABILIDAD	BARCELONA	3	980000
INVESTIGACION	VALENCIA	2	475000
VENTAS	MADRID	4	855000

## Vistas sobre vistas.

**La consulta que define a una vista puede incluir a su vez una vista.**

Por ejemplo, la siguiente vista *dat\_emp\_dep30* se crea a partir de la vista *emple\_dep30* filtrando algunas de las columnas:

```
SQL> CREATE VIEW dat_emp_dep30 AS
SELECT emp_no, apellido, oficio, director,
fecha_alta, dep_no
FROM emple_dep30;
```

En Access:

1° Escribimos la consulta:  
 SELECT emp\_no, apellido, oficio, director,  
 fecha\_alta, dep\_no FROM **emple\_dep30**;

2° La ejecutamos pulsando  para comprobar datos.

3° Pulsamos al botón  y la llamamos Dat\_emp\_dep30.

4° Cerramos la vista SQL.

```
SQL> select * from dat_emp_dep30;
```

EMP_NO	APELLIDO	OFICIO	DIRECTOR	FECHA_AL	DEP_NO
7698	GARRIDO	DIRECTOR	7839	01/05/81	30
7499	ALONSO	VENDEDOR	7698	20/02/81	30
7654	MARTIN	VENDEDOR	7698	28/09/81	30
7844	CALVO	VENDEDOR	7698	08/09/81	30

### ***Restricciones para la creación y utilización de vistas.***

- No se puede usar la cláusula ORDER BY en la creación de una vista ya que las filas de una tabla no están ordenadas (la vista es una tabla virtual). No obstante, si se puede utilizar dicha cláusula a la hora de recuperar datos de la vista.
- Es obligatorio especificar la lista de nombres de columnas cuando la consulta devuelve funciones de agrupamiento como SUM, COUNT, etcétera.
- No se pueden utilizar funciones de agrupación sobre columnas de vistas que se basan a su vez en funciones de agrupación ya que en la práctica supondría un doble agrupamiento que no está permitido por el estándar.

```
SELECT MIN(num_empleados), MIN( suma_salario)
FROM resumen_emp_dep
GROUP BY LOCALIDAD;
```

En Access no da error si creamos una vista que visualice esta SELECT.

### ***Actualizaciones en vistas.***

Como ya hemos explicado, las vistas son tablas virtuales, y los datos que manejan, en realidad, pertenecen a otras tablas. Por tanto, **cualquier comando de actualización (INSERT, UPDATE o DELETE) sobre una vista será traducido por el SGBD a una actualización de la tabla sobre la que se basa.**

Por ejemplo, el siguiente comando introducirá, a través de la vista *emple\_dep30*, una nueva fila en la tabla *departamentos*:

```
SQL> INSERT INTO emple_dep30
values (8998, 'CORTES', 'VENDEDOR', 7698,
'20/02/99', 180000, NULL, 30);
```

```

En Access:
INSERT INTO emple_dep30 ( EMP_NO, APELLIDO, OFICIO, DIRECTOR,
                          FECHA_ALTA, SALARIO, COMISION, DEP_NO )
Values (8998, 'CORTES', 'VENDEDOR', 7698, #2/20/1999#, 180000, Null, 30);

```

Podemos comprobar la inserción consultando la tabla *empleados*:

```
SQL> SELECT * FROM empleados WHERE emp_no = 8998;
```

EMP_NO	APELLIDO	OFICIO	DIRECTOR	FECHA_AL	SALARIO	COMISIÓN	DEP_NO
8998	CORTES	VENDEDOR	7698	20/02/99	180000		30

**Para que se puedan utilizar comandos de actualización sobre una vista, deberá existir una correspondencia inequívoca en filas y columnas entre la vista y la tabla sobre la que basa.**

La regla anterior se puede desglosar en:

- **Las vistas basadas en agrupaciones no pueden ser actualizadas** ya que no existe una correspondencia directa entre las filas de la vista y las filas de las tablas originales.
- Por la misma razón **no se pueden actualizar las vistas creadas a partir de composiciones con varias tablas.**
- **Tampoco se pueden actualizar vistas cuyas columnas se han creado a partir de expresiones o funciones** que pueden enmascarar el valor de la columna en la tabla original.

Nota: Algunos gestores de base de datos permiten saltar estas restricciones en ciertas circunstancias. Estas características suelen estar documentadas en los manuales del producto.

## ***Vistas con validación***

Cuando una vista se crea utilizando una consulta que incluye una condición de selección (cláusula WHERE) e intentamos insertar una fila que no cumple la condición, ¿permitirá el SGBD la inserción?

Supongamos que queremos insertar en la vista *emple\_dep30* un empleado cuyo departamento será el 20:

```
SQL> INSERT INTO emple_dep30
      values (8999, 'LUCAS', 'ANALISTA', 7566,
```

```
'20/02/99', 380000, NULL, 20);
```

En Access:

```
INSERT INTO emple_dep30 ( EMP_NO, APELLIDO, OFICIO, DIRECTOR,
                          FECHA_ALTA, SALARIO, COMISION, DEP_NO )
VALUES (8999, 'LUCAS', 'ANALISTA', 7566, #20/02/99#, 380000, NULL, 20);
```

Después de esta orden el sistema devolverá el mensaje:

Una fila creada.

Sin embargo, al intentar recuperar la información insertada nos encontraremos lo siguiente:

```
SQL> SELECT * FROM emple_dep30 WHERE emp_no = 8999;
```

**Ninguna fila seleccionada.**

En realidad la fila se ha insertado en la tabla *empleados* a través de la vista *emple\_dep30* (podemos comprobarlo mediante una consulta a la tabla *empleados*). Pero la vista no puede acceder a la información que se acaba de introducir ya que no cumple la condición especificada al crear la vista.

Este comportamiento puede parecer incoherente y resulta desconcertante para el usuario. **Se puede evitar indicando al crear la tabla que se compruebe que cualquier inserción o modificación satisface la condición establecida añadiendo la cláusula *WITH CHECK OPTION* al final de la instrucción de creación.**

En Access se pone **WITH OWNERACCESS OPTION;**

El formato de creación de vistas ampliado con esta opción será:

```
CREATE VIEW nombredevista [(listadecolumnas)]
AS consulta [WITH CHECK OPTION];
```

En nuestro ejemplo:

```
SQL> CREATE VIEW emple_dep30 AS
      SELECT * FROM EMPLEADOS
      WHERE DEP_NO = 30
      WITH CHECK OPTION;
```

En Access:

```
SELECT * FROM EMPLEADOS
WHERE DEP_NO = 30 WITH OWNERACCESS OPTION;
```

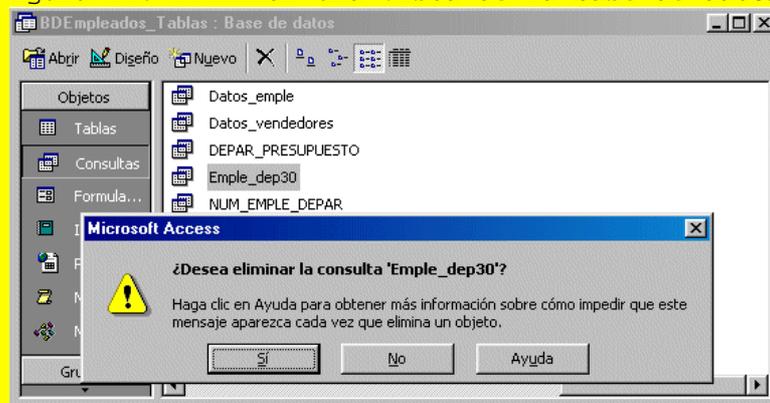
## ***Eliminación de vistas.***

La sentencia DROP VIEW permite eliminar la definición de una vista.

```
DROP VIEW nombredevista ;
```

En Access esta orden no se puede utilizar para borrar vistas. Para borrar una vista se hace desde la ventana de la base de datos, simplemente basta con seleccionar la vista , pulsar la tecla suprimir y confirmar la eliminación. Ver Figura 21:

Figura 21. Eliminar una vista de la base de datos.



El siguiente ejemplo borrará la vista emple\_dep30:

```
SQL> drop view emple_dep30 cascade constraints;
```

Vista borrada.

Recordemos que la vista *emple\_dep30* se utilizaba para crear la vista *dat\_emp\_dep30*, que ahora al intentar consultarla dará el siguiente error:

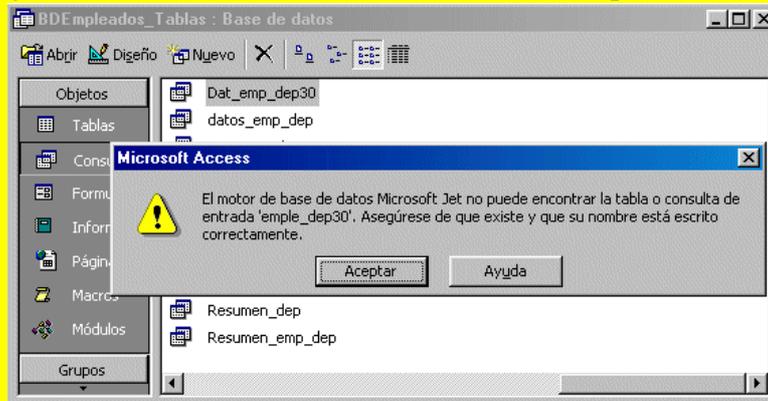
```
SQL> select * from dat_emp_dep30;
select * from dat_emp_dep30
      *
```

ERROR en línea 1:

```
ORA-04063: view "CURSOSQL.DAT_EMP_DEP30" tiene errores
```

En Access aparece la ventana de error que se muestra en la Figura22.

Figura 22. Error al utilizar una vista que no existe



Algunos gestores de base de datos incorporan opciones como *CASCADE* que añadidos al final del comando de eliminación provocan que se borren también todas las vistas que utilizan la que se pretende borrar.

---

## Tema 9. CREACIÓN DE TABLAS

---

*Autora: María Teresa Miñana*

### **Formato genérico para la creación de tablas.**

---

#### Consideraciones previas a la creación de una tabla

- El nombre de la tabla.
- El nombre de cada columna.
- El tipo de dato almacenado en cada columna.
- El tamaño de cada columna.
- Otra información.

La sentencia SQL que permite crear tablas es **CREATE TABLE**.

#### Formato básico para la creación de tablas

```
CREATE TABLE nombre_de_tabla ??????>
>???(???definición_de_columna ??????>);
?????????? , ??????????????
```

- **nombre\_de\_tabla** permite un conjunto de hasta 30 caracteres, comenzando por uno alfabético y con posibilidad de contener alfanuméricos y subrayados, así como mayúsculas y minúsculas indistintamente.
- **definición\_de\_columna** consiste en indicar para cada columna de la tabla:

- . **Nombre de columna.** Por ejemplo Apellido
- . **Tipo de dato** que se va a almacenar en esa columna. En el ejemplo:  
Apellido VARCHAR
- . **Tamaño previsto** para esa columna. En el ejemplo:  
Apellido VARCHAR(8)

**Sugerencia:** Ver TIPOS DE DATOS (En TEMA 2).

Existirán tantas definiciones de columna como datos diferentes se vayan a almacenar en la tabla que estamos creando.

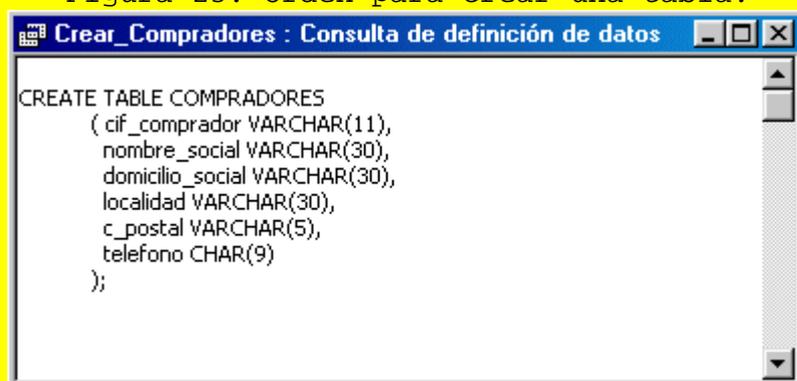
En Access podemos crear tablas siguiendo dos caminos.

**A) Desde la vista SQL.** El icono identificativo de una una sentencia de creación de tablas es .

Pasos:

1º- Desde la *Ventana de la base de datos* elegimos el objeto *Consultas* y doble clic en *Crear consulta en vista de diseño*. Cerrar la ventana *Mostrar tabla*, a continuación abrir la vista SQL. En esta ventana escribimos la orden CREATE. Ver Figura 23:

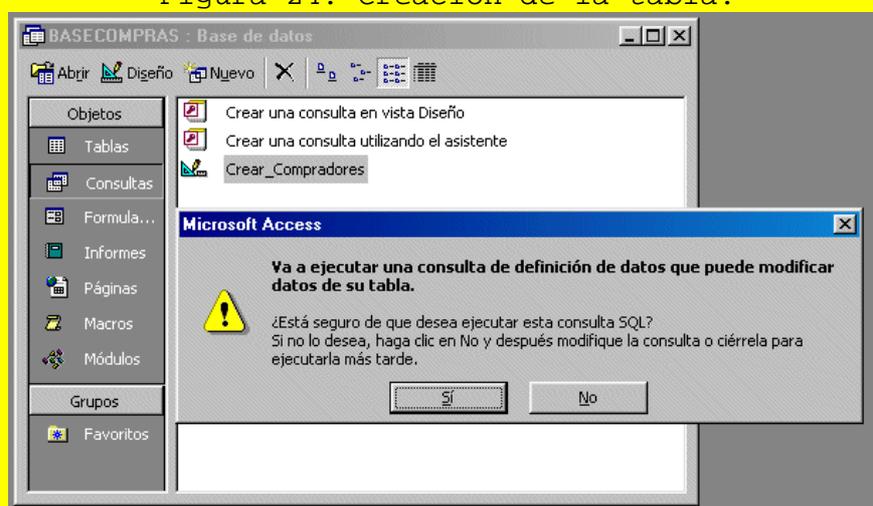
Figura 23. Orden para crear una tabla.

Una ventana de Microsoft Access con el título "Crear Compradores : Consulta de definición de datos". El contenido de la ventana es un código SQL para crear una tabla.

```
CREATE TABLE COMPRADORES
( cif_comprador VARCHAR(11),
  nombre_social VARCHAR(30),
  domicilio_social VARCHAR(30),
  localidad VARCHAR(30),
  c_postal VARCHAR(5),
  telefono CHAR(9)
);
```

2º- Por último para ejecutar la orden CREATE es necesario guardar la consulta de creación, darla un nombre, cerrar la vista SQL, ( por ejemplo la orden se ha guardado con el nombre *CrearCompradores*) y a continuación desde la ventana de la base de datos la seleccionamos y la ejecutamos haciendo doble clic. Ver Figura 24. Aparece un mensaje que pide la confirmación de la orden, pulsamos a *Aceptar*.

Figura 24. Creación de la tabla.



**B) Desde la vista de Diseño de la tabla.**

En la ventana de la base de datos elegimos el objeto Tablas, Elegimos la opción Crear una tabla en Vista de diseño. Aparece una pantalla dividida en dos partes: en la superior podemos distinguir un pequeño casillero, donde describiremos los campos de la tabla, mientras que en la inferior encontramos una serie de fichas vacías en el primer momento, pero que luego se utilizarán para asignar propiedades a los campos creados. La inserción de campos se realiza de forma manual, en la columna *Nombre del campo* iremos añadiendo los atributos o campos de la tabla. En la columna *Tipo de datos* elegiremos de la lista el tipo de dato que se almacenará en ese campo. Y en la columna *Descripción* se puede escribir un comentario sobre el campo. Ver Figura25

Figura25.Creación de tablas utilizando la vista de diseño.



A la izquierda de la fila en la que estamos posicionados aparece un marcador de fila , podremos marcar toda la fila y mover el campo a otro lugar arrastrándolo con el ratón o suprimirlo si ya no se necesita.

Los siguientes ejemplos van a utilizar nuevas tablas para no alterar las tablas anteriormente utilizadas.

Recuerda en Access la definición de datos tipo NUMBER no va acompañada de la longitud, por defecto se definen como numéricos doble.

Para crear las tablas en Access vamos a crear una nueva Base de datos en Blanco y llamarla BDCOMPRADORES.

## Ejemplos

1. Crear una *tabla de compradores* con las siguientes definiciones de columna:

- CIF\_comprador → alfabético de 11 caracteres.
- Nombre\_social → alfabético de 30 caracteres.
- Domicilio\_social → alfabético de 30 caracteres.
- Localidad → alfabético de 30 caracteres.
- Teléfono → alfabético de 9 caracteres.

```
SQL> CREATE TABLE compradores
      ( cif_comprador VARCHAR(11),
        nombre_social VARCHAR(30),
        domicilio_social VARCHAR(30),
        localidad VARCHAR(30),
        c_postal VARCHAR(5),
        telefono CHAR(9)
      );
```

En Access y desde la vista SQL:

1º Escribimos la orden:

```
CREATE TABLE COMPRADORES
      ( CIF_COMPRADOR VARCHAR(11),
        NOMBRE_SOCIAL VARCHAR(30),
        DOMICILIO_SOCIAL VARCHAR(30),
        LOCALIDAD VARCHAR(30),
        C_POSTAL VARCHAR(5),
        TELEFONO CHAR(9)
      );
```

- 3º Pulsamos al botón  y la llamamos CREAR\_COMPRADORES.
- 4º Cerramos la vista SQL. Y desde la ventana de la base de datos hacemos doble clic sobre la consulta para crear la tabla.

2. Crear una tabla de artículos con las siguientes descripciones de columna:

- Referencia\_artículo → alfabético de 6 caracteres.
- Descripción\_artículo → alfabético de 30 caracteres.
- Precio\_unidad → numérico de 6 posiciones.
- IVA → numérico de 2 posiciones.
- Existencias\_actuales → numérico de 5 posiciones.
- Stock\_mínimo → numérico de 4 posiciones.

```
SQL> CREATE TABLE articulos
      ( referencia_articulo VARCHAR(6),
        descripcion_articulo VARCHAR(30),
        precio_unidad NUMBER(6),
        iva NUMBER(2),
        existencias_actuales NUMBER(5),
        stock_minimo NUMBER(4)
      );
```

En Access y desde la vista SQL:

1º Escribimos la orden:

```
CREATE TABLE ARTICULOS
      ( REFERENCIA_ARTICULO VARCHAR(6),
        DESCRIPCION_ARTICULO VARCHAR(30),
        PRECIO_UNIDAD NUMBER,
        IVA NUMBER,
        EXISTENCIAS_ACTUALES NUMBER,
        STOCK_MINIMO NUMBER
      );
```

3º Pulsamos al botón  y la llamamos CREAR\_ARTICULOS.

4º Cerramos la vista SQL. Y desde la ventana de la base de datos hacemos doble clic sobre la consulta para crear la tabla.

## ***Restricciones de tabla y de columna.***

**Restricciones** son condiciones que imponemos a la hora de crear una tabla para que los datos se ajusten a una serie de características predefinidas que mantengan su integridad.

Se refieren a los siguientes conceptos:

- **DEFAULT.** Proporciona un valor por defecto cuando la columna a la que acompaña no recibe ningún dato cuando se está insertando.

Los valores por defecto pueden ser: constantes, funciones SQL o las variables USER o SYSDATE.

**Ejemplo.** La fecha de alta de los empleados será la del sistema o fecha del día en que se está realizando la entrada de datos en la base de datos, si no se le indica otra.

```
SQL> CREATE TABLE empleados
      (.....,
        fecha_alta DATE DEFAULT SYSDATE,
        .....
      );
```

En Access esta restricción **NO** se hace en la orden **CREATE** se hace en la **vista del diseño** de la tabla. En la propiedad **valor predeterminado**. Se verá en los ejercicios.

Las siguientes restricciones son conocidas en SQL por su nombre en inglés, es decir **CONSTRAINT**:

- **NOT NULL.** Exige la existencia de dato en la columna que lleva la restricción.  
**Ejemplo.** El número de empleado nunca irá sin información.

```
SQL> CREATE TABLE empleados
      (emp_no NUMBER(4) NOT NULL
        .....
      );
```

- **CHECK.** Comprueba si se cumple una determinada condición. No puede incluir una subconsulta, ni las variables SYSDATE o USER.

En Access esta restricción **NO** se hace en la orden **CREATE** se hace en la **vista del diseño** de la tabla. En la propiedad **Regla de validación**. Se verá en los ejercicios.

### Ejemplos

1. La columna APELLIDO del empleado siempre deberá ir con dato. Podría escribirse una de las restricciones siguientes:

```
SQL> CREATE TABLE empleados
      (.....,
        apellido VARCHAR(8) NOT NULL,
        .....
      );
```

o

```
SQL> CREATE TABLE empleados
      (.....,
        apellido VARCHAR(8) CHECK(apellido IS NOT NULL),
        .....
      );
```

2. La columna APELLIDO del empleado siempre deberá ir en mayúsculas.

```
SQL> CREATE TABLE empleados
      (.....,
        apellido VARCHAR(8) CHECK(apellido=UPPER(APELLIDO),
        .....),
      );
```

- **UNIQUE.** Evita valores repetidos en la misma columna. Admite valores NULL.  
**Ejemplo.**

```
SQL> CREATE TABLE empleados
      (emp_no NUMBER(4) NOT NULL UNIQUE
      .....),
      .....),
      );
```

- **PRIMARY KEY.** Indica una o varias columnas como dato o datos que identifican unívocamente cada fila de la tabla. Sólo existe una por tabla y en ninguna fila puede tener valor NULL. En nuestras tablas de *empleados* y *departamentos* serían **emp\_no** y **dep\_no**, respectivamente.
- **FOREIGN KEY.** Indica que una determinada columna de una tabla va a servir para referenciar a otra tabla en la que la misma columna está definida como **PRIMARY KEY** o **UNIQUE**. El valor de la clave extranjera o ajena deberá coincidir con uno de los valores de esta clave referenciada o ser NULL. No existe límite en el número de claves extranjeras o ajenas que pueda tener una tabla. Como caso particular, una clave extranjera o ajena puede referenciar a la misma tabla en la que está. Para poder crear una tabla con clave extranjera o ajena deberá estar previamente creada la tabla maestra en la que la misma columna es clave primaria. En nuestras tablas de *empleados* y *departamentos*, **dep\_no** sería clave extranjera o ajena en la tabla de *empleados* porque con ella podemos acceder a la tabla de *departamentos* y ésta tabla deberá crearse antes que la de *empleados*.
- Algunos sistemas gestores de bases de datos relacionales añaden a las anteriores restricciones, la posibilidad de generar automáticamente índices para las columnas **PRIMARY KEY** o **UNIQUE**.
- Algunos, también, permiten la posibilidad de establecer condiciones para el almacenamiento físico de las tablas que se están creando.

Las **CONSTRAINTS** se van a almacenar con un nombre. Si no se lo damos nosotros, el sistema las nombrará con el formato **sys\_cn°** que es poco representativo.

### **Criterio para dar nombres significativos a las CONSTRAINTS**

- **nn\_nombre\_tabla\_nombre\_columna** → para NOT NULL
- **cki\_nombre\_tabla\_nombre\_columna** → para posibilitar más de un CHECK a la misma columna. Se puede utilizar nombres más significativos.  
Ejemplos:  
*ck\_upper\_nombre\_columna* para comprobar si el dato de esa columna va en mayúsculas.  
*ck\_lugar\_nombre\_columna* para comprobar que el dato de esa columna es una de las localidades permitidas.
- **uq\_nombre\_tabla\_nombre\_columna** → para UNIQUE
- **pk\_nombre\_tabla\_nombre\_columna** → para PRIMARY KEY
- **fk\_nombre\_tabla\_nombre\_columna** → para FOREIGN KEY

Las restricciones que acabamos de relacionar se pueden establecer a nivel de tabla o a nivel de columna, con alguna consideración lógica:

- NOT NULL tiene más sentido a nivel de columna.
- Cuando existan claves compuestas, se definirán a nivel de tabla.

#### Formato para la creación de tablas con CONSTRAINTS definidas a nivel de tabla

```
CREATE TABLE nombre_de_tabla ??????????>
>??? (???definición_de_columna ??????????)
?????????? , ??????????????
>????????????????????????????????????????????????????????>
?? CONSTRAINT nombre_constraint ??
>????????????????????????????????????????????????????????>);
? ?? CHECK(condición) ?? ?
? ? ?
????? UNIQUE (columna(s)) ??????????
? ?? PRIMARY KEY (columna(s)) ?? ?
? ? ?
????????????????????????????????????????????????????????
? ?? FOREIGN KEY (columna(s)) *?? ?
????????????????????????????????????????????????????????
```

(\*) Si una columna se define como FOREIGN KEY, el formato se amplía con las siguientes indicaciones:

```

??? FOREIGN KEY (columna(s)) REFERENCES tabla_referenciada ?????>
    [(columna(s))]
>????????????????????????????????????????????????????????????????????>
  ? ON DELETE???CASCADE?????      ? ON UPDATE???CASCADE?????
      ??SET NULL?????                ??SET NULL?????
      ??SET DEFAULT?                ??SET DEFAULT?
      ??NO ACTION???                ??NO ACTION???

```

- La *columna* o *columnas* que siguen a la cláusula `FOREIGN KEY` es aquella o aquellas que están formando la clave ajena o extranjera. Si hay más de una se separan por comas.
- La *tabla referenciada* es el nombre de la tabla a la que se va a acceder con la clave ajena o extranjera y donde la misma es clave primaria.
- Si la *columna* o *columnas* que forman la clave primaria en la tabla referenciada no tiene el mismo nombre que en la clave ajena, debe indicarse su nombre detrás del de la tabla referenciada y dentro de paréntesis. Si son más de una columna se separan por comas. Si los nombres de columnas coinciden en la clave ajena y en la primaria, no es necesario realizar esta indicación.
- Para mantener la integridad de los datos, al borrar (`DELETE`) o modificar (`UPDATE`) una fila de la tabla referenciada, existen las siguientes opciones:
  - **CASCADE**. El borrado o modificación de una fila de la tabla referenciada lleva consigo el borrado o modificación en cascada de las filas de la tabla que contiene la clave ajena. Es la más utilizada.
  - **SET NULL**. El borrado o modificación de una fila de la tabla referenciada lleva consigo poner a `NULL` los valores de las claves ajenas en las filas de la tabla que referencia.
  - **SET DEFAULT**. El borrado o modificación de una fila de la tabla referenciada lleva consigo poner un valor por defecto en las claves ajenas de la tabla que referencia.
  - **NO ACTION**. El borrado o modificación de una fila de la tabla referenciada sólo se lleva a cabo si no existe ninguna fila con el mismo valor en la clave ajena en la tabla que referencia. En algunos gestores se conoce como `RESTRICT`. Es la opción por defecto.

**En SQLPLUS de Oracle, sólo está permitida la opción `ON DELETE CASCADE`.**

**En ACCESS.**

Para mantener la integridad de los datos, al borrar (DELETE) o modificar (UPDATE) en **Access** se hace lo siguiente:

1º Se crean todas las consultas de creación de tablas, una a una indicando las claves primarias y las ajenas.

2º Se ejecutan las consultas una a una en el orden de creación. Al crear las tablas se crean las distintas relaciones entre ellas.

3º Desde la ventana de la base de datos pulsamos al botón

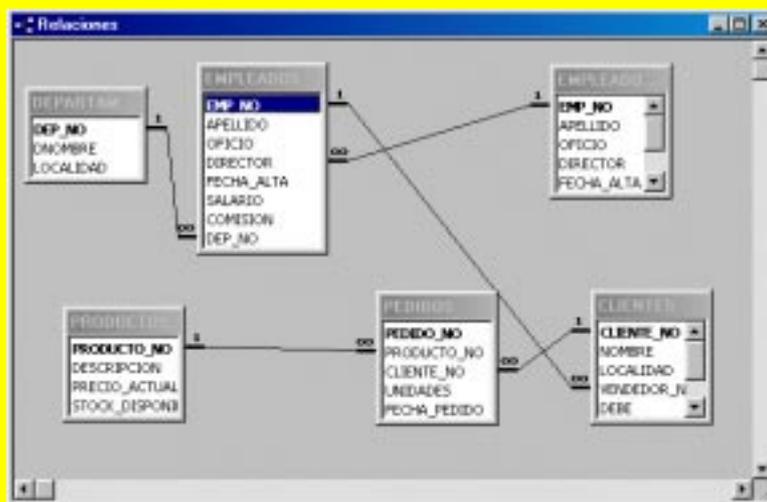
**Relaciones**  que aparece en la barra de herramientas.

4º Desde la ventana *Mostrar tabla* elegimos todas las tablas. Si no se ven las tablas, pulsamos al botón

*Mostrar tabla*  y seleccionamos todas.

5º Aparecen las tablas y sus relaciones. Ver Figura 26.

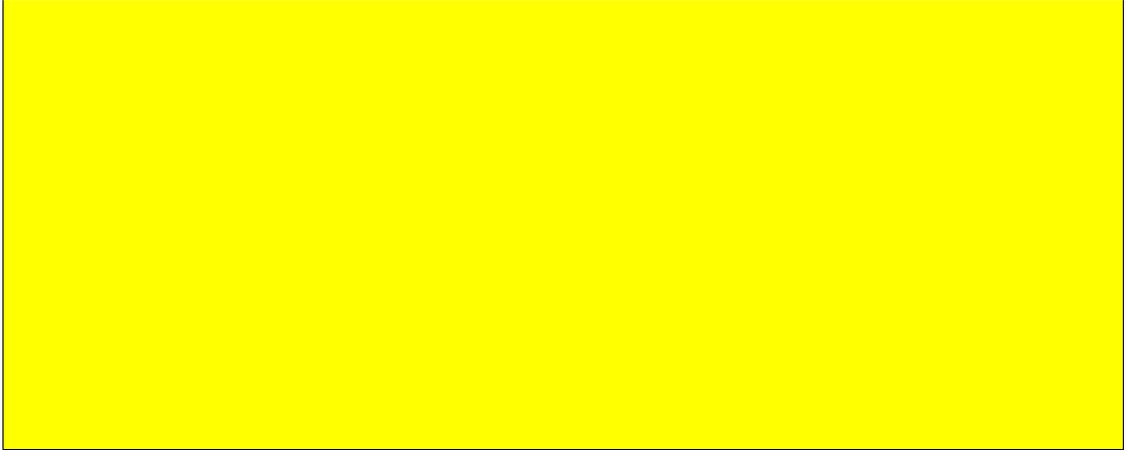
Figura 26. Ventana de relaciones entre tablas.



6º Para mantener la integridad de datos entre las relaciones, hacemos lo siguiente.

- Hacemos doble clic sobre la relación (sobre la línea).
- Aparece la ventana de la figura 27. Y elegimos la opción deseada: **Actualizar en cascada los campos relacionados** (ON UPDATE CASCADE), o **Eliminar en cascada los campos relacionados** (ON DELETE CASCADE).

Figura 27. Opciones en la relación de tablas.





```
descripcion_articulo VARCHAR(30),  
precio_unidad NUMBER(6),  
iva NUMBER(2),  
existencias_actuales NUMBER(5),  
stock_minimo NUMBER(4) DEFAULT 0  
);
```

**En Access:**

1º Escribimos la orden:

```
CREATE TABLE ARTICULOS
( REFERENCIA_ARTICULO VARCHAR(6),
  DESCRIPCION_ARTICULO VARCHAR(30),
  PRECIO_UNIDAD NUMBER,
  IVA NUMBER,
  EXISTENCIAS_ACTUALES NUMBER,
  STOCK_MINIMO NUMBER
);
```

En Access la cláusula **DEFAULT** no la podemos poner en la orden CREATE, se añadirá luego en el diseño de la tabla.

2º Guardamos esa consulta de creación con un nombre.

3º Desde la ventana de la base de datos ejecutamos la consulta para crear la tabla.

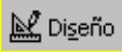
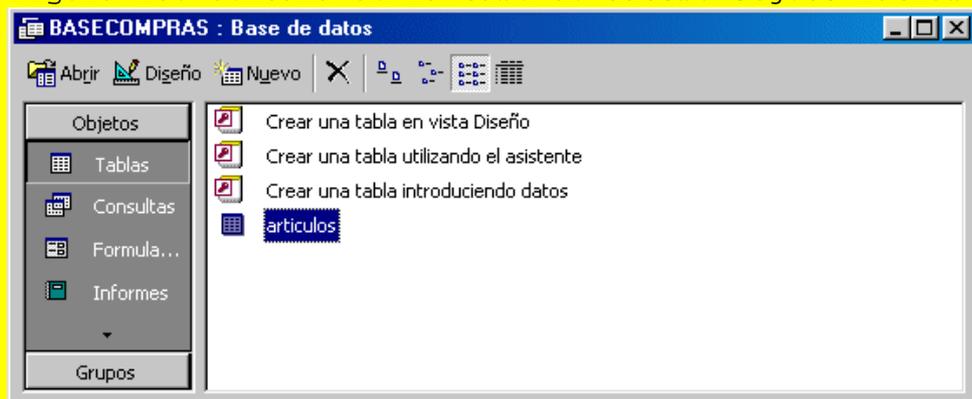
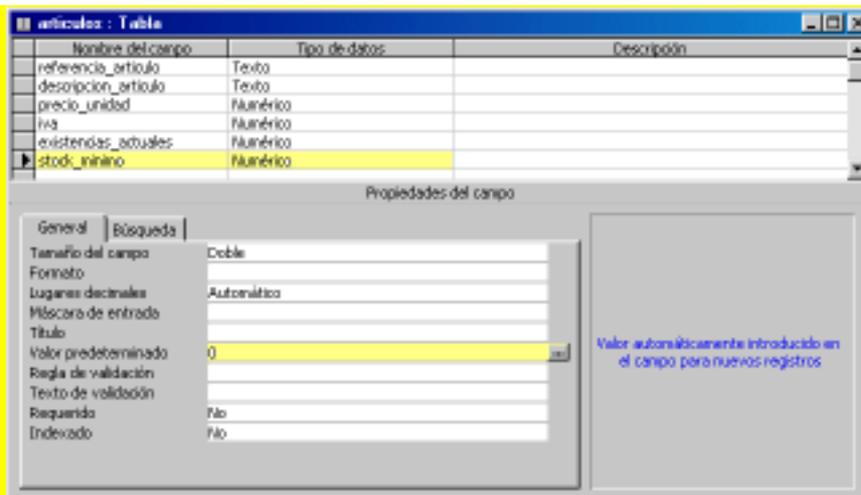
4º Una vez creada y desde la ventana de la base de datos, seleccionamos la tabla y pulsamos al botón diseño  de la ventana de la base de datos. Ver Figura 28.

Figura 28. Ventana de la base de datos. Objeto Tablas.



5º Y en el diseño de la tabla seleccionamos el campo STOCK\_MINIMO y nos posicionamos en la propiedad **Valor predeterminado**. Ver Figura 29. En esa casilla ponemos 0, es el valor por defecto.

Figura 29. Ventana del diseño de la tabla.



6° Guardamos los cambios y cerramos la ventana del diseño de la tabla.

---

2. Crear la *tabla de compradores* con la columna *c\_postal* comenzando por uno de los códigos de provincia existentes, usando una CONSTRAINT de columna.

- Sin nombre de CONSTRAINT:

```
SQL> CREATE TABLE COMPRADORES
      ( CIF_COMPRADOR VARCHAR(11),
        NOMBRE_SOCIAL VARCHAR(30),
        DOMICILIO_SOCIAL VARCHAR(30),
        LOCALIDAD VARCHAR(30),
        C_POSTAL VARCHAR(5)
        CHECK(SUBSTR(C_POSTAL,1,2) BETWEEN '01' AND '52'),
        TELEFONO CHAR(9)
      );
```

- Con nombre de CONSTRAINT:

```
SQL> CREATE TABLE compradores
      ( cif_comprador VARCHAR(11),
        nombre_social VARCHAR(30),
        domicilio_social VARCHAR(30),
        localidad VARCHAR(30),
        c_postal VARCHAR(5) CONSTRAINT ck_compradores_postal
        CHECK(SUBSTR(c_postal,1,2) BETWEEN '01' AND '52'),
        telefono CHAR(9)
      );
```

**En Access:**

1º Escribimos la orden:

```
CREATE TABLE COMPRADORES
( CIF_COMPRAADOR VARCHAR(11),
  NOMBRE_SOCIAL VARCHAR(30),
  DOMICILIO_SOCIAL VARCHAR(30),
  LOCALIDAD VARCHAR(30),
  C_POSTAL VARCHAR(5),
  TELEFONO CHAR(9)
);
```

En Access la cláusula **CHECK** no la podemos poner en la orden CREATE, se añadirá luego en el diseño de la tabla.

2º Guardamos esa consulta de creación con un nombre.

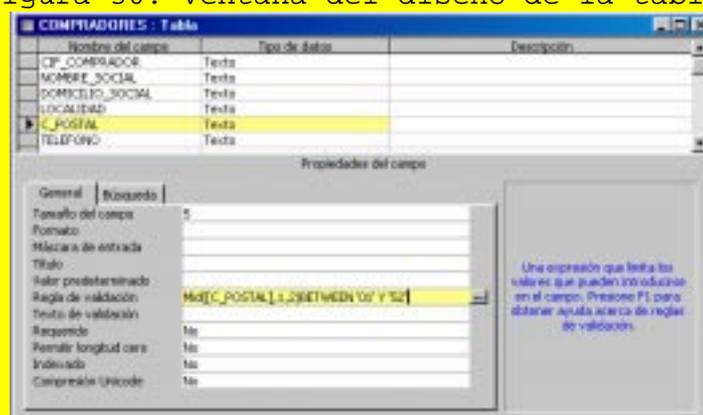
3º Desde la ventana de la base de datos ejecutamos la consulta para crear la tabla haciendo doble clic.

4º Una vez creada y desde la ventana de la base de datos, seleccionamos la tabla y pulsamos al botón diseño  de la ventana de la base de datos. Ver

5º Y en el diseño de la tabla seleccionamos el campo C\_POSTAL y nos posicionamos en la propiedad **Regla de validación**. Ver Figura 30. En esa casilla escribimos la fórmula:

**Mid([C\_POSTAL],1,2)BETWEEN '01' Y '52'**

Figura 30. Ventana del diseño de la tabla.



6º Guardamos los cambios y cerramos la ventana del diseño de la tabla.

Si volvemos a entrar en el diseño de la tabla vemos que access traduce la fórmula a español y la escribe así:

**Medio([C\_POSTAL],1,2) Entre '01' Y '52'**

3. Crear la *tabla de compradores* con la columna *cif\_comprador* como NOT NULL.

- Sin nombre de CONSTRAINT:

```
SQL> CREATE TABLE COMPRADORES
      ( CIF_COMPRADOR VARCHAR(11) NOT NULL,
        NOMBRE_SOCIAL VARCHAR(30),
        DOMICILIO_SOCIAL VARCHAR(30),
        LOCALIDAD VARCHAR(30),
        C_POSTAL VARCHAR(5),
        TELEFONO CHAR(9)
      );
```

**En Access:**

1º Escribimos la orden:

```
CREATE TABLE COMPRADORES
      ( CIF_COMPRADOR VARCHAR(11) NOT NULL,
        NOMBRE_SOCIAL VARCHAR(30),
        DOMICILIO_SOCIAL VARCHAR(30),
        LOCALIDAD VARCHAR(30),
        C_POSTAL VARCHAR(5),
        TELEFONO CHAR(9)
      );
```

2º Guardamos esa consulta de creación con un nombre.

3º Desde la ventana de la base de datos ejecutamos la consulta para crear la tabla haciendo doble clic. (Borrar antes la tabla COMPRADORES para crearla de nuevo con estas especificaciones)

**4. Crear la *tabla de compradores* con la columna *cif\_comprador* como primary key y utilizando CONSTRAINT a nivel de tabla.**

- Sin nombre de CONSTRAINT:

```
SQL> CREATE TABLE compradores
      ( cif_comprador VARCHAR(11) NOT NULL,
        nombre_social VARCHAR(30),
        domicilio_social VARCHAR(30),
        localidad VARCHAR(30),
        c_postal VARCHAR(5),
        telefono CHAR(9),
        PRIMARY KEY (cif_comprador)
      );
```

- Con nombre de CONSTRAINT:

```
SQL> CREATE TABLE COMPRADORES
      ( CIF_COMPRADOR VARCHAR(11) NOT NULL,
        NOMBRE_SOCIAL VARCHAR(30),
        DOMICILIO_SOCIAL VARCHAR(30),
        LOCALIDAD VARCHAR(30),
        C_POSTAL VARCHAR(5),
        TELEFONO CHAR(9),
        CONSTRAINT PK_COMPRADORES_CIF PRIMARY KEY
        (CIF_COMPRADOR)
```

```
);
```

**En Access:**

1º Escribimos la orden (Cualquiera de las dos es válida):

```
CREATE TABLE COMPRADORES
( CIF_COMPRAADOR VARCHAR(11) NOT NULL,
  NOMBRE_SOCIAL VARCHAR(30),
  DOMICILIO_SOCIAL VARCHAR(30),
  LOCALIDAD VARCHAR(30),
  C_POSTAL VARCHAR(5),
  TELEFONO CHAR(9),
  CONSTRAINT PK_COMPRADORES_CIF PRIMARY KEY (CIF_COMPRAADOR)
);
```

2º Guardamos esa consulta de creación con un nombre.

3º Desde la ventana de la base de datos ejecutamos la consulta para crear la tabla haciendo doble clic. (Borrar antes la tabla COMPRADORES para crearla de nuevo con estas especificaciones)

5. Crear la tabla de artículos con *referencia\_articulo* como PRIMARY KEY, utilizando una CONSTRAINT a nivel de columna .

- Sin nombre de CONSTRAINT:

```
SQL> CREATE TABLE articulos
( referencia_articulo VARCHAR(6) NOT NULL PRIMARY KEY,
  descripcion_articulo VARCHAR(30),
  precio_unidad NUMBER(6),
  iva NUMBER(2),
  existencias_actuales NUMBER(5),
  stock_minimo NUMBER(4)
);
```

- Con nombre de CONSTRAINT:

```
SQL> CREATE TABLE articulos
( referencia_articulo VARCHAR(6) NOT NULL
  CONSTRAINT pk_articulos_referencia PRIMARY KEY,
  descripcion_articulo VARCHAR(30),
  precio_unidad NUMBER(6),
  iva NUMBER(2),
  existencias_actuales NUMBER(5),
  stock_minimo NUMBER(4)
);
```

**En Access:**

1º Escribimos la orden (Cualquiera de las dos es válida):

```
CREATE TABLE ARTICULOS
  ( REFERENCIA_ARTICULO VARCHAR(6) NOT NULL
    CONSTRAINT PK_ARTICULOS_REFERENCIA PRIMARY KEY,
    DESCRIPCION_ARTICULO VARCHAR(30),
    PRECIO_UNIDAD NUMBER,
    IVA NUMBER,
    EXISTENCIAS_ACTUALES NUMBER,
    STOCK_MINIMO NUMBER
  );
```

2º Guardamos esa consulta de creación con un nombre.

3º Desde la ventana de la base de datos ejecutamos la consulta para crear la tabla haciendo doble clic. (Borrar antes la tabla ARTICULOS para crearla de nuevo con estas especificaciones)

**6.** Crear la *tabla de líneas de facturas* con las columnas *factura\_no* y *referencia\_articulo* como primary key y la columna *referencia\_articulo* como foreign key, utilizando CONSTRAINT a nivel de tabla.

La CONSTRAINT para una clave compuesta sólo puede definirse a nivel de tabla.

CONSTRAINT para la foreign key a nivel de tabla.

Sin nombre de CONSTRAINT:

```
SQL> CREATE TABLE líneas_facturas
  ( factura_no NUMBER(6) NOT NULL,
    referencia_articulo VARCHAR(6) NOT NULL,
    unidades NUMBER(3),
    PRIMARY KEY(factura_no,referencia_articulo),
    FOREIGN KEY(referencia_articulo)
    REFERENCES articulos(referencia_articulo)
    ON DELETE CASCADE
  );
```

Con nombre de CONSTRAINT:

```
SQL> CREATE TABLE líneas_facturas
  ( factura_no NUMBER(6) NOT NULL,
    referencia_articulo VARCHAR(6) NOT NULL,
    unidades NUMBER(3),
    CONSTRAINT pk_lineas_factura
    PRIMARY KEY(factura_no,referencia_articulo),
    CONSTRAINT fk_lineas_referencia
    FOREIGN KEY(referencia_articulo)
    REFERENCES articulos(referencia_articulo)
    ON DELETE CASCADE
  );
```

**En Access:**

1º Escribimos la orden (Cualquiera de las dos es válida):

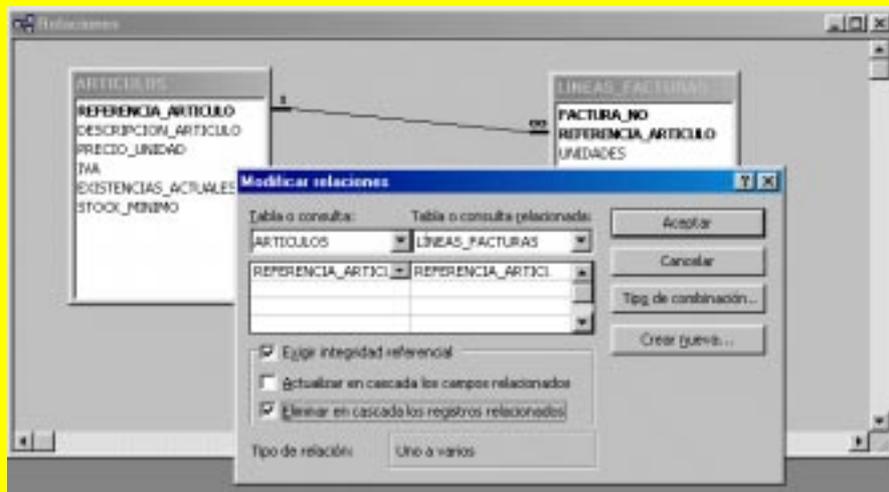
```
CREATE TABLE LÍNEAS_FACTURAS
( FACTURA_NO NUMBER(6) NOT NULL,
  REFERENCIA_ARTICULO VARCHAR(6) NOT NULL,
  UNIDADES NUMBER(3),
  CONSTRAINT PK_LÍNEAS_FACTURA
    PRIMARY KEY(FACTURA_NO,REFERENCIA_ARTICULO),
  CONSTRAINT FK_LÍNEAS_REFERENCIA
    FOREIGN KEY(REFERENCIA_ARTICULO)
    REFERENCES ARTICULOS(REFERENCIA_ARTICULO)
);
```

2º Guardamos esa consulta de creación con un nombre.

3º Desde la ventana de la base de datos ejecutamos la consulta para crear la tabla haciendo doble clic.

4º Añadir ON **DELETE CASCADE** desde la ventana relaciones. Pulsar el botón . Hacer doble clic en la línea de unión de las tablas ARTICULOS y LINEAS\_FACTURAS y seleccionar la casilla correspondiente. Ver Figura 31.

Figura 31. Añadir la restricción ON DELETE CASCADE.



No se pueden crear tablas con el mismo nombre que otras ya existentes en la misma base de datos. Si se han creado estas tablas en los ejemplos anteriores, es necesario borrarlas previamente.

**Creación de una tabla a partir de una selección de filas y columnas de otra tabla.**

En algunos gestores SQL, como el de Oracle, se puede crear una tabla a partir de las filas seleccionadas de otra tabla:

```
SQL> CREATE TABLE nombre_tabla_nueva
      AS
      SELECT [ * / lista_de_elementos]
      FROM nombre_de_tabla
      .....
      ..... ;
```

En Access podemos crear estas sentencias en la vista del diseño pulsando al botón  Vista SQL, o desde el menú Consulta / Consulta de Creación de tabla  Ver Figura 31

Figura 31. Crear una tabla a partir de otra. Consulta de Creación de tabla.



## Ejemplos.

1. Crear una tabla de vendedores seleccionando éstos de la tabla de empleados.

```
SQL> CREATE TABLE vendedores
      AS
      SELECT *
      FROM empleados
      WHERE UPPER(oficio)='VENDEDOR';
```

En Access, cambia el formato:

```
SELECT * INTO VENDEDORES
FROM EMPLEADOS
WHERE UCASE(OFICIO) = 'VENDEDOR' ;
```

2. Crear una nueva tabla sólo con los nombres y números de los departamentos a partir de la tabla ya creada con los mismos.

```
SQL> CREATE TABLE nombres_dep
AS
SELECT dep_no, dnombre
FROM departamentos;
```

En Access, cambia el formato:

```
SELECT DEP_NO, DNOMBRE INTO NOMBRES_DEP
FROM DEPARTAMENTOS ;
```

La nueva tabla creada sólo hereda las CONSTRAINTS que no tengan nombre asignado externamente.

## ***Modificación de la definición de tabla.***

Una vez que hemos creado una tabla, a menudo se presenta la necesidad de tener que modificarla. La sentencia SQL que realiza esta función es **ALTER TABLE**.

### **Formato general para la modificación de tablas**

```
ALTER TABLE nombre_de_tabla especificación_de_modificación;
```

- *nombre\_de\_tabla* que se desea modificar.
- *especificación\_de\_modificación*. Las modificaciones que pueden realizarse sobre una tabla son las siguientes:

. **Añadir una nueva columna.** La especificación de la modificación es parecida a la de la sentencia CREATE pero varía según el producto SQL del que se trate.

### **Formato para Oracle**

```
>??ADD(-?? definición_de_columna [NOT NULL]?????)?>
???????????????????? , ?????????????????????
```

La adición de una nueva columna con NOT NULL sólo será posible si la tabla está vacía. Si no lo estuviera, se añade la columna sin la cláusula NOT NULL, se actualiza (UPDATE) con datos reales o ficticios, hasta obtener los reales, y se modifica la definición de la columna (MODIFY) con cláusula NOT NULL.

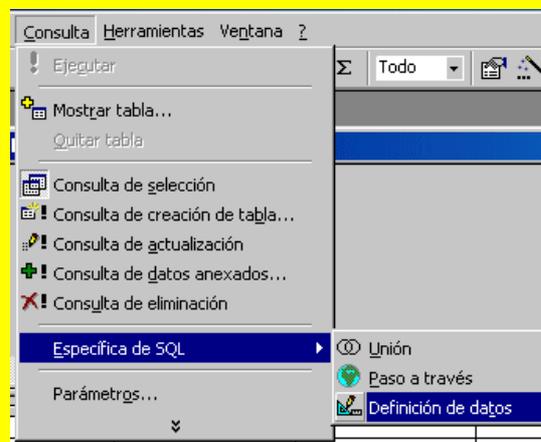
### Formato para otros gestores

```
>??ADD -? definición_de_columna [NOT NULL WITH DEFAULT]
```

La adición de una nueva columna con NOT NULL está permitida siempre que vaya con un valor por defecto, correspondiente al tipo de dato de la columna. con este tipo de formato sólo podemos añadir una columna en la misma sentencia ALTER TABLE.

En Access podemos crear estas sentencias en la vista del diseño pulsando al botón  Vista SQL, o desde el menú Consulta/Especifica de SQL/Definición de datos. Ver Figura 32

Figura 32. Consulta de definición de datos.



Lo más sencillo es utilizar la VISTA SQL.

### Ejemplos.

1. Añadir la columna *fax* a la tabla de compradores con la misma definición que el teléfono.

Comprobemos la estructura de la tabla antes de modificarla:

```
SQL> DESC compradores;
```

Name	Null?	Type
CIF_COMPRAADOR	NOT NULL	VARCHAR2(11)
NOMBRE_SOCIAL		VARCHAR2(30)
DOMICILIO_SOCIAL		VARCHAR2(30)
LOCALIDAD		VARCHAR2(30)
C_POSTAL		VARCHAR2(5)
TELEFONO		CHAR(9)

VARCHAR es el mismo tipo de dato que VARCHAR2 en Oracle.

```
SQL> ALTER TABLE compradores
      ADD (fax char(9));
```

En Access Igual.

Comprobemos la tabla después de modificarla:

```
SQL> desc compradores;
```

Name	Null?	Type
CIF_COMPRAADOR	NOT NULL	VARCHAR2(11)
NOMBRE_SOCIAL		VARCHAR2(30)
DOMICILIO_SOCIAL		VARCHAR2(30)
LOCALIDAD		VARCHAR2(30)
C_POSTAL		VARCHAR2(5)
TELEFONO		CHAR(9)
FAX		CHAR(9)

## 2. Añadir el CIF\_proveedor a la tabla de artículos.

```
SQL> ALTER TABLE articulos
      ADD (cif_proveedor VARCHAR(11));
```

En Access:

```
ALTER TABLE articulos
      ADD cif_proveedor VARCHAR(11);
```

. **Añadir restricciones de tabla.** Las restricciones se utilizan con el mismo formato que vimos en CREATE TABLE, teniendo en cuenta, aquí también, las diferencias entre los distintos productos SQL.

### Formato para Oracle

```
>??ADD -?? restricción_de_tabla ??????????????>
      ?                                     ?
      ?????????????????????? , ??????????????????
```

Toda restricción debe estar identificada con un nombre significativo.

### Formato para otros gestores

```
>??ADD -? restricción_de_tabla ??????????>
```

En otros gestores, no se necesita nombrar la restricción, pero no se permite nada más que una por cláusula ADD.

### Ejemplos.

1. Añadir en la tabla *artículos* la restricción de comprobar que la columna *precio\_unidad* contenga un valor NOT NULL y distinto de 0.

```
SQL> ALTER TABLE articulos
      ADD CONSTRAINT ck_articulos_pu
      CHECK(precio_unidad IS NOT NULL AND
           precio_unidad!=0) ;
```

En Access añadir esta restricción de validación ha de hacerse desde la vista de diseño de la tabla , en la propiedad **Regla de validación**.

Para el campo Precio\_unidad pondremos: **<>0 and not null**

2. Añadir en la tabla *compradores* la restricción UNIQUE para la columna *nombre\_social*.

```
SQL> ALTER TABLE compradores
      ADD CONSTRAINT uq_compradores_nombre
      UNIQUE(nombre_social);
```

En Access Igual.

. **Modificar la definición de una columna.** La especificación de la modificación es parecida a la de la cláusula ADD y también varía según el producto SQL del que se trate.

### Formato para Oracle

```
>??MODIFY(-?? definición_de_columna [NOT NULL]?????)??>
???????????????????? , ?????????????????????
```

En Access la palabra **MODIFY** es sustituida por **ALTER COLUMN**

La modificación de una columna con `NOT NULL` sólo podrá realizarse si la tabla está vacía o si la columna no contiene el valor nulo en ninguna de las filas de la tabla. Si no fuese así, se actualizaría (`UPDATE`) con datos reales o ficticios, hasta obtener los reales, y luego se realizaría la modificación de la definición de la columna con la cláusula `NOT NULL`.

La modificación de una columna para aumentar su tamaño siempre se puede realizar, sea de tipo carácter o de tipo numérico.

La modificación de una columna para disminuir su tamaño se puede realizar, sea de tipo carácter o de tipo numérico, siempre que dicha columna tenga `NULL` en todas las filas de la tabla.

Siempre se puede realizar la modificación de una columna de tipo numérico para aumentar o disminuir su número de decimales.

La modificación de una columna para cambiar de tipo de dato se puede realizar siempre que que dicha columna tenga `NULL` en todas las filas de la tabla.

### Formato para otros gestores

```
>??ALTER -?nombre_de_columna [SET DEFAULT valor|DROP DEFAULT]??>
```

La cláusula `ALTER` permite cambiar el valor por defecto de una columna de la tabla. con este tipo de formato sólo podemos modificar una columna en la misma cláusula `ALTER`.

### Ejemplos.

1. Modificar la tabla *líneas\_facturas* para poner 1 en la columna *unidades* como valor por defecto.

```
SQL> ALTER TABLE líneas_facturas  
      MODIFY( unidades number(3) DEFAULT 1);
```

Tabla modificada.

En Access añadir esta restricción de validación ha de hacerse desde la vista de diseño de la tabla , en la propiedad **Valor predeterminado**.  
Para el campo Unidades pondremos: 1

2. Modificar la tabla *artículos* para poner `NOT NULL` en la columna *cif\_proveedor*.

```
SQL> ALTER TABLE artículos
```

```
MODIFY(cif_proveedor VARCHAR(11) NOT NULL);
```

Tabla modificada.

En Access:

```
ALTER TABLE ARTICULOS
  ALTER COLUMN CIF_PROVEEDOR VARCHAR(11) NOT NULL;
```

```
SQL> DESC articulos
Name                               Null?      Type
-----
REFERENCIA_ARTICULO                NOT NULL  VARCHAR2(6)
DESCRIPCION_ARTICULO                VARCHAR2(30)
PRECIO_UNIDAD                       NUMBER(6)
IVA                                  NUMBER(2)
EXISTENCIAS_ACTUALES                NUMBER(5)
STOCK_MINIMO                         NUMBER(4)
CIF_PROVEEDOR                     NOT NULL VARCHAR2(11)
```

3. Modificar la tabla *compradores* para decrementar de 30 a 20 caracteres los tamaños de las columnas *domicilio\_social* y *localidad*.

```
SQL> ALTER TABLE compradores
  MODIFY(domicilio_social VARCHAR(20),
  localidad VARCHAR(20));
```

Tabla modificada.

En Access se modifican los campos de uno en uno:

```
ALTER TABLE COMPRADORES
  ALTER COLUMN
    DOMICILIO_SOCIAL VARCHAR(20);

ALTER TABLE COMPRADORES
  ALTER COLUMN
    LOCALIDAD VARCHAR(20);
```

```
SQL> desc compradores
Name                               Null?      Type
-----
CIF_COMPRADOR                      NOT NULL  VARCHAR2(11)
NOMBRE_SOCIAL                       VARCHAR2(30)
DOMICILIO_SOCIAL                   VARCHAR2(20)
LOCALIDAD                         VARCHAR2(20)
C_POSTAL                            VARCHAR2(5)
TELEFONO                            CHAR(9)
FAX                                  CHAR(9)
```

4. Modificar la tabla *articulos* para que la columna *precio\_unidad* admita un decimal.

```
SQL> ALTER TABLE articulos
      MODIFY(precio_unidad NUMBER(7,1));
```

Tabla modificada.

En Access esta modificación ha de hacerse desde la vista de diseño de la tabla , en la propiedad **Tamaño de campo** y **Lugares decimales**.

```
SQL> DESC ARTICULOS
Name                               Null?      Type
-----
REFERENCIA_ARTICULO                NOT NULL   VARCHAR2(6)
DESCRIPCION_ARTICULO                NOT NULL   VARCHAR2(30)
PRECIO_UNIDAD                       NULL       NUMBER(7,1)
IVA                                  NULL       NUMBER(2)
EXISTENCIAS_ACTUALES                NULL       NUMBER(5)
STOCK_MINIMO                        NULL       NUMBER(4)
CIF_PROVEEDOR                       NOT NULL   VARCHAR2(11)
```

### En SQLPLUS de Oracle:

Para obtener información sobre las restricciones de las tablas de usuario:

```
SQL> SELECT * FROM USER_CONSTRAINTS;
```

Para obtener información sobre las columnas con alguna restricción de las tablas de usuario:

```
SQL> SELECT * FROM USER_CONS_COLUMNS;
```

En Access no existen estas vistas para ver las constraints.

**. Borrado de restricciones de una tabla.** Permite quitar aquella restricción de la tabla que haya dejado de ser necesaria. Su utilización depende del gestor SQL que se está utilizando.

### Formato para borrar restricciones de una tabla

```
>??DROP ???UNIQUE (???nombre_de_columna?????[CASCADE]???)
?          ?????????? , ?????????? ?
??PRIMARY KEY ??????????????????????
??FOREIGN KEY nombre_de_columna???
??CONSTRAINT nombre_constraint ???
```

Este formato para el borrado de alguna restricción es general para casi todos los gestores SQL.

Si en la constraint FOREIGN KEY de la CREATE TABLE no se ha especificado, y tampoco se ha usado la cláusula CASCADE, **para cambiar o borrar una clave primaria es necesario borrar previamente las posibles claves ajenas asociadas a la misma columna, si las hubiera.**

Los siguientes ejemplos son solamente ilustrativos para no borrar las restricciones anteriores sobre las que se realizarán los ejercicios propuestos al final del tema.

### Ejemplos.

1. Borrar de la tabla *compradores* la restricción *ck\_compradores\_postal* aplicada sobre la columna *c\_postal*.

```
SQL> ALTER TABLE compradores
      DROP CONSTRAINT ck_compradores_postal;
```

2. Borrar de la tabla *compradores* la restricción *pk\_compradores\_cif* para poner como clave primaria la columna *cif\_comprador*.

```
SQL> ALTER TABLE compradores
      DROP CONSTRAINT pk_compradores_cif;
```

En Access se igual.

### Eliminación de una tabla.

Para borrar una tabla y su contenido de la base de datos se utiliza la sentencia DROP TABLE.

### Formato para eliminar una tabla.

```
DROP TABLE nombre_de_tabla [CASCADE CONSTRAINT] ;
```

La cláusula CASCADE CONSTRAINT borra las restricciones asociadas a la tabla antes de que sea eliminada.

El siguiente ejemplo es solamente ilustrativo para no borrar la tabla *articulos* que se necesitará en los ejercicios propuestos al final del tema.

**Ejemplos.****1. Borrar la tabla de articulos.**

```
SQL> DROP TABLE articulos CASCADE CONSTRAINT;
```

En access podemos borrar una tabla desde la ventana de la base de datos, seleccionando la tabla y pulsando la tecla suprimir. O bien ejecutando la orden DROP desde la vista SQL.  
En la vista SQL pondremos:

```
DROP TABLE ARTICULOS CASCADE;
```

---

## Unidad 10. SEGURIDAD EN SQL.

---

*Autor: Fernando Montero*

### ***Introducción a la seguridad en los SGBDR.***

---

Hay dos términos que se encuentran íntimamente ligados al concepto de seguridad en los sistemas de gestión de bases de datos:

- **Confidencialidad:** impedir que usuarios no autorizados accedan a información para la que no tienen permiso.
- **Disponibilidad:** garantizar que la información esté accesible para los usuarios autorizados.

Los SGBDR definen un esquema de seguridad en el que podemos encontrar básicamente tres elementos: **usuarios, objetos de la base de datos y privilegios**. Según este esquema un usuario tiene determinados privilegios o derechos de acceso a los objetos de la base de datos.

La mayoría de los fabricantes implementa este modelo aunque pueden existir algunas variaciones en los formatos para la gestión de usuarios y en algunas características avanzadas (perfiles, grupos y roles).

En este tema utilizaremos los formatos y ejemplos basados en el SGBD Oracle v. 7 y 8 que son aplicables en su mayor parte a los demás los productos comerciales. En todo caso se harán las observaciones oportunas con el fin de facilitar la compatibilidad con otros productos.

### ***Usuarios: creación.***

---

Un usuario es una entidad **conocida por la base de datos** que tiene ciertos **privilegios sobre algunos objetos de la misma y permisos para realizar determinadas acciones**.

Para que un usuario sea conocido por la base de datos tiene que haber sido creado.

Normalmente es el administrador de la base de datos quien se encarga de esta labor. El comando para crear usuarios es:

```
CREATE USER idusuario
IDENTIFIED BY palabradepaso;
```

Donde:

- *idusuario* es el identificador de usuario o nombre de usuario. Deberá ser un identificador SQL válido (comenzar por un carácter alfabético, etcétera) .
- *palabradepaso* es una contraseña, clave o password asociada al identificador.

Por ejemplo, si queremos crear el usuario ALU01 lo haremos:

```
SQL> CREATE USER alu01  
IDENTIFIED BY curso;
```

Usuario creado.

De esta forma hemos creado el usuario alu01. Ahora, el siguiente paso es concederle privilegios para poder trabajar en la base de datos.

## ***Privilegios***

---

Podemos distinguir dos tipos de privilegios:

### **Privilegios del sistema:**

**Son permisos para realizar determinadas acciones sobre algún tipo genérico de objetos.**

Por ejemplo, para poder crear usuarios se necesita tener el privilegio **CREATE USER** y para poder eliminar usuarios se requiere el privilegio **DROP USER**.

Otros privilegios del sistema son:

- **ALTER USER** para poder modificar características de otros usuarios.
- **CREATE TABLE** para poder crear tablas.
- **CREATE SESSION** para poder crear una sesión (conectar) con la base de datos.
- **CREATE PUBLIC SYNONYM** para crear sinónimos públicos.
- **DROP PUBLIC SYNONYM** para borrar sinónimos públicos.
- **GRANT ANY PRIVILEGE** para poder conceder privilegios del sistema.
- Etcétera.

Para asignar privilegios del sistema a un usuario se utiliza el comando GRANT con el siguiente formato:

```
GRANT listadeprivilegiosdelsistema TO listadeusuarios;
```

Donde:

- *listadeprivilegiosdelsistema* especifica algunos de los privilegios del sistema establecidos.
- *listadeusuarios* especifica los identificativos de los usuarios a los que se concede el privilegio.

El siguiente comando concede al usuario *alu01* los privilegios del sistema necesarios para conectarse a la base de datos (CREATE SESSION) y para crear tablas (CREATE TABLE):

```
SQL> GRANT CREATE SESSION, CREATE TABLE  
      TO alu01;
```

Concesión terminada con éxito.

### Privilegios sobre objetos:

Son permisos para realizar acciones concretas sobre objetos concretos.

Por ejemplo: permiso para consultar la tabla clientes, para insertar en la tabla de pedidos, para modificar la tabla de productos, etcétera.

La mayoría de los gestores de bases de datos admiten los siguientes privilegios de objeto **para tablas y vistas**:

- SELECT
- INSERT
- UPDATE
- DELETE

El formato genérico para la asignación de estos privilegios es:

```
GRANT listadeprivilegiosdeobjeto ON nombredeobjeto TO  
listadeusuarios;
```

Para conceder al usuario *alu01* el privilegio de consultar e insertar en la tabla *departamentos* escribiremos:

---

```
SQL > GRANT SELECT, INSERT
      ON DEPARTAMENTOS
      TO ALU01;
```

Concesión terminada con éxito.

El formato anterior se puede ampliar para el **privilegio UPDATE especificando (opcionalmente) las columnas** sobre las que se concede tal privilegio.

El siguiente ejemplo concede al usuario *alu01* el privilegio de actualizar las columnas *dnombre* y *localidad* de la tabla *departamentos*.

```
SQL > GRANT UPDATE (dnombre, localidad)
      ON DEPARTAMENTOS
      TO ALU01;
```

También se pueden **asignar todos los permisos posibles sobre un objeto utilizando la opción ALL PRIVILEGES** en lugar de la lista de privilegios.

```
SQL > GRANT ALL PRIVILEGES
      ON DEPARTAMENTOS
      TO ALU01;
```

#### **Nota: Privilegios sobre tablas usuario.**

Cuando un usuario crea una tabla ese objeto pasa a ser de su propiedad (se dice que el objeto pertenece al usuario) y dispone de todos los privilegios de acceso a la tabla. También podrá conceder privilegios sobre la tabla a otros usuarios.

---

## ***Retirada de privilegios***

**Para retirar los privilegios de que dispone un usuario se utiliza sentencia REVOKE** cuyo formato es similar al de la sentencia GRANT.

También en este caso hay dos formatos dependiendo de si se trata de privilegios del sistema o de privilegios sobre objetos.

### **Retirada de privilegios del sistema.**

Para retirar a un usuario **privilegios del sistema** se utilizará el siguiente formato:

```
REVOKE listadeproprivilegiosdelsistema FROM listadeusuarios;
```

Para retirar el privilegio CREATE TABLE a el usuario alu01 escribiremos:

```
SQL> REVOKE CREATE TABLE
      FROM alu01;
```

Denegación terminada con éxito.

### Retirada de privilegios de objeto.

Para retirar **privilegios de objeto** utilizaremos el siguiente formato:

```
REVOKE listadeprivilegiosdeobjeto ON nombredeobjeto FROM
listadeusuarios;
```

El siguiente ejemplo retira al usuario *alu01* el privilegio de inserción sobre la tabla *departamentos*.

```
SQL > REVOKE INSERT
      ON DEPARTAMENTOS
      FROM ALU01;
```

Denegación terminada con éxito.

También se puede utilizar la opción **ALL PRIVILEGES** para retirar los todos los privilegios que pueda tener un usuario sobre un determinado objeto.

```
SQL > REVOKE ALL PRIVILEGES
      ON EMPLEADOS
      FROM ALU01;
```

Nota: la instrucción **REVOKE** no admite la especificación de columnas (a diferencia de lo que ocurre con GRANT). La retirada del privilegio deberá afectar a toda la tabla.

```
SQL> REVOKE UPDATE (dnombre, localidad)
      ON DEPARTAMENTOS
      FROM ALU01;
```

```
REVOKE UPDATE (dnombre, localidad)
```

```
ERROR en línea 1:
```

```
ORA-01750: UPDATE/REFERENCES sólo se pueden denegar de toda la
tabla, no por
columna
```

Si quitamos la especificación de columnas la instrucción se hará efectiva.

```
SQL> REVOKE UPDATE
      ON DEPARTAMENTOS
      FROM ALU01;
```

Denegación terminada con éxito.

---

## ***Roles.***

---

La gestión de usuarios y privilegios tal como se ha estudiado en el apartado anterior puede resultar muy laboriosa ya que consistiría en conceder a cada uno de los usuarios los privilegios del sistema y de objeto necesarios. Por esta razón se utilizan los roles.

Simplificando, **podemos definir un rol como un conjunto de privilegios.**

Con un rol se pueden hacer las siguientes operaciones:

- Crear el rol: para poder utilizar un rol primero deberemos crearlo utilizando el siguiente formato:

```
CREATE ROLE nombrederol;
```

Donde *nombrederol* es cualquier identificador válido.

- Una vez creado podemos asignar al rol los privilegios necesarios utilizando el mismo formato que para asignar privilegios a un usuario:

```
GRANT listadeproprivilegiosdelsistema TO listaderoles;
```

O bien:

```
GRANT listadeprivilegiosdeobjeto ON nombredeobjeto TO
listaderoles;
```

- El rol se le puede asignar a un usuario o incluso a otro rol con lo que este último acumularía los privilegios correspondientes:

```
GRANT listaderoles TO listadeusuarios;
```

```
GRANT listaderoles TO listaderoles2;
```

- También se pueden retirar privilegios de uno o varios roles de manera idéntica a como se hacía con los usuarios::

```
REVOKE          listadeproprivilegiosdelsistema          FROM
listaderoles;
```

O también:

```
REVOKE listadeprivilegiosdeobjeto ON nombredeobjeto
FROM listaderoles;
```

- Para eliminar un rol utilizaremos el siguiente formato:

**DROP ROLE *listaderos*;**

### Ejemplos:

SQL> CREATE ROLE emp;	* Creamos el ROL <i>emp</i> .
Rol creado.	
SQL> GRANT CREATE SESSION, ALTER SESSION, CREATE TABLE, CREATE VIEW TO emp;	* Asignamos al rol <i>emp</i> los privilegios del sistema indicados.
Concesión terminada con éxito.	
SQL> GRANT emp TO ALU01;	* Asignamos el rol <i>emp</i> a alu01.
Concesión terminada con éxito.	Ahora alu01 tiene todos los privilegios asociados al rol <i>emp</i> (mas los que pudiera tener previamente).
SQL> REVOKE CREATE VIEW FROM emp;	* Eliminamos uno de los privilegios del rol <i>emp</i> . También le quitará a todos los usuarios que lo tuviesen concedido a través de ese rol.
Denegación terminada con éxito.	
SQL> CREATE ROLE jef;	* Creamos un nuevo rol llamado <i>jef</i> .
Rol creado.	
SQL> GRANT prog, CREATE TABLE, SELECT ANY TABLE TO jef;	* Asignamos al nuevo rol el rol <i>prog</i> (con todos sus privilegios) además de otros privilegios adicionales.
Concesión terminada con éxito.	
SQL> GRANT jef TO ALU02, ALU03;	* Asignamos el nuevo rol a los usuarios

### **Roles predefinidos**

Oracle viene con una serie de roles predefinidos que facilitan la gestión de usuarios y recursos. Podemos destacar:

Rol	Se suele utilizar para:	Incluye los siguientes privilegios:
<b>CONNECT</b>	Todos los usuarios de la base de datos que pertenecen a la compañía o entidad.	ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SESSION, CREATE SYNONYM, CREATE TABLE, CREATE VIEW
<b>RESOURCE</b>	Desarrolladores de aplicaciones.	CREATE CLUSTER, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER
<b>DBA</b>	Administradores de la base de datos	todos los privilegios del sistema WITH ADMIN OPTION.
<b>EXP_FULL_DATABASE</b>	Operadores de copias de seguridad.	SELECT ANY TABLE, BACKUP ANY TABLE.

Estos roles permiten:

- asignar privilegios a usuarios.
- asignar privilegios a otros roles
- incluir nuevos privilegios en los roles predefinidos.
- retirar aquellos privilegios que puedan considerarse inoportunos.

#### Ejemplos:

```
SQL> GRANT CONNECT TO alu01;
```

Concesión terminada con éxito.

```
SQL> GRANT RESOURCE TO jef;
```

Concesión terminada con éxito.

```
SQL> GRANT CREATE PROCEDURE TO  
CONNECT;
```

Concesión terminada con éxito.

\* Asigna el rol CONNECT con todos sus privilegios al usuario *alu01*.

\* Asigna el rol RESOURCE con todos sus privilegios al ROL *jef*, que suponemos creado.

\* Añade el privilegio CREATE PROCEDURE al rol CONNECT (y a todos los usuarios que tengan dicho rol)

<pre>SQL&gt;          REVOKE          CREATE PROCEDURE,CREATE  VIEW      FROM CONNECT;</pre> <p>Denegación terminada con éxito.</p> <pre>SQL&gt; GRANT  CREATE  VIEW  TO CONNECT;</pre> <p>Concesión terminada con éxito.</p>	<pre>* Retira los privilegios CREATE PROCEDURE y CREATE VIEW del rol CONNECT (y de todos los usuarios que tuviesen los privilegios a través de dicho rol.  * Vuelve a dar el privilegio CREATE VIEW al rol CONNECT.</pre>
---	---

Nota: aunque puede hacerse, no suele ser una buena idea asignar o retirar privilegios a los roles predefinidos por el sistema. Es preferible crear otros roles y trabajar con ellos.

## ***Privilegios con opción de administración***

---

### **Privilegios del sistema con opción de administración**

Como ya hemos mencionado, normalmente es el administrador de la base de datos quien se encarga de crear usuarios y asignar privilegios. El usuario receptor del privilegio puede utilizarlo él mismo pero no puede, en principio, conceder ese privilegio a otros usuarios. **Para que un usuario pueda administrar un del sistema** deberá:

- **Tener el privilegio del sistema GRANT ANY PRIVILEGE.**

Para asignar este privilegio a un usuario se procederá como si se tratase de cualquier otro privilegio del sistema:

```
SQL> GRANT GRANT ANY PRIVILEGE
      TO alu01;
```

Esto permite al usuario conceder cualquier privilegio del sistema a otros usuarios incluso a el propio usuario en el caso de que no tuviese alguno de los privilegios cuya administración se le concede, tal como se puede observar en la secuencia siguiente.

<pre>SQL&gt; GRANT GRANT ANY PRIVILEGE   TO alu01;  Concesión terminada con éxito.  SQL&gt; CONNECT ALU01; Introduzca su clave: ***** Conectado.  SQL&gt; CREATE ROLE PRUEBAS; CREATE ROLE PRUEBAS * ERROR en línea 1: ORA-01031: privilegios insuficientes  SQL&gt; GRANT CREATE ROLE   TO ALU01;  Concesión terminada con éxito.  SQL&gt; CREATE ROLE PRUEBAS;  Rol creado.</pre>	<pre>-&gt; El administrador concede al usuario alu01 el privilegio el sistema GRANT ANY PRIVILEGE.  - &gt; Se conecta el usuario alu01.       (El administrador ya no está       conectado)  -&gt; El usuario intenta crear un objeto denominado rol.  -&gt; El sistema le devuelve un error indicando que no tiene privilegios para crear tal objeto.  -&gt; El usuario se concede a sí mismo el privilegio requerido.  -&gt; Ahora el sistema le permite crear el rol.</pre>
---	--

*Normalmente la opción anterior solamente se concede a administradores de la base de datos.*

**- Tener un determinado privilegio del sistema con la opción de administración WITH ADMIN OPTION.**

Para conceder un privilegio del sistema con la opción de administración se añadirá dicha opción al comando GRANT como en el ejemplo siguiente:

```
SQL> GRANT CREATE PUBLIC SYNONYM
  TO ALU01 WITH ADMIN OPTION;
```

En el ejemplo anterior se asigna al usuario *alu01* el privilegio de crear sinónimos públicos con la posibilidad de que, a su vez, pueda asignar este privilegio a otros usuarios.

Para retirar la opción de administración de todos los privilegios del sistema

Las opciones anteriores son válidas para privilegios del sistema; pero existe una opción similar para los **privilegios de objeto**.

**Privilegios de objeto con opción de administración.**

Los objetos son propiedad de los usuarios que los crean, que poseen todos los privilegios sobre dichos objetos, así como la posibilidad de conceder privilegios sobre los mismos a otros usuarios.

---

Estos otros usuarios no pueden conceder a terceros los privilegios recibidos sobre el objeto (en principio). Para habilitar esta posibilidad, la concesión del privilegio deberá realizarse con la opción `WITH GRANT OPTION`.

### Ejemplo:

Supongamos que el usuario *cursosql* es el propietario de la tabla departamentos y quiere conceder los privilegios `SELECT` e `INSERT` al usuario *alu01* con la posibilidad de que este, a su vez, pueda conceder tales derechos a otros usuarios. El usuario *cursosql* deberá proceder:

```
SQL> GRANT SELECT, INSERT
      ON DEPARTAMENTOS
      TO ALU01
      WITH GRANT OPTION;
```

Ahora el usuario *alu01* podrá conceder los privilegios recibidos a otros usuarios.

También se puede conceder la opción de administración de todos los permisos sobre un objeto a un usuario como se muestra a continuación:

```
SQL> GRANT ALL PRIVILEGES
      ON DEPARTAMENTOS
      TO ALU01
      WITH GRANT OPTION;
```

### **Retirada de la opción de administración de privilegios.**

Existen las siguientes posibilidades:

- Retirar la opción de administración de todos los privilegios del sistema

```
SQL> REVOKE GRANT ANY PRIVILEGE
      FROM alu01;
```

- Retirada de la opción de administración de todos los privilegios que un usuario pueda tener sobre un objeto: Se utilizará la opción `ALL PRIVILEGES` ya que con ella también se retiran los derechos de administración.

```
SQL> REVOKE ALL PRIVILEGES
      ON departamentos
      FROM alu01;
```

- Retirar uno o varios privilegios junto con la opción de administración: Solamente habrá que retirar el privilegio ya que se retira también la opción de administración como en el ejemplo siguiente:

```
SQL> REVOKE CREATE PUBLIC SYNONYM
      FROM alu01;
```

Nota.- No se puede retirar solamente la opción de administración para un privilegio del sistema (o para un rol) , pero podemos conseguir esto retirando el privilegio del sistema y volviéndolo a asignar sin la opción de administración. A continuación se muestra cómo eliminar la opción de administración del privilegio CREATE PUBLIC SYNONYM.

```
SQL> REVOKE CREATE PUBLIC SYNONYM
      FROM alu01;
```

Denegación terminada con éxito.

```
SQL> GRANT CREATE PUBLIC SYNONYM
      TO alu01
```

Concesión terminada con éxito.

<< materiales >>

Podemos comprobar los privilegios del sistema que posee un usuario consultado la vista DBA\_SYS\_PRIVS

```
SQL> SELECT * FROM DBA_SYS_PRIVS
      WHERE GRANTEE = 'ALU01';
```

GRANTEE	PRIVILEGE	ADM
ALU01	CREATE PUBLIC SYNONYM	YES
ALU01	CREATE ROLE	NO
ALU01	CREATE SESSION	NO
ALU01	CREATE TABLE	NO

### ***Utilización de sinónimos públicos y privados.***

Un sinónimo es un nombre que significa lo mismo que otro. En una base de datos se suelen crear sinónimos con los siguientes propósitos:

- Aislar a los usuarios y a las aplicaciones de los posibles cambios de nombre de los objetos.
- Facilitar el acceso a objetos con nombres largos o *difíciles*.

- Facilitar el acceso a objetos situados en diferentes esquemas.

Los sinónimos pueden permitirnos:

- que los usuarios de una multinacional vean la tabla de clientes nombrándola cada uno en su propio idioma (CUSTOMERS, CLIENTES, ...).
- acceder a la tabla IMP\_DIC99\_BCO\_V31REV como MOROSOS.
- acceder a la tabla SYSTEM.CLIENTES como CLIENTES.
- etcétera.

Un sinónimo se crea utilizando la orden CREATE SYNONYM según el siguiente formato:

```
CREATE SYNONYM nombredelsinonimo FOR objeto;
```

Por ejemplo, podemos crear un sinónimo que nos permita referirnos a las tablas *empleados* y *departamentos* como *emp* y *dep*:

```
SQL> CREATE SYNONYM EMP FOR EMPLEADOS;
```

Sinónimo creado.

```
SQL> CREATE SYNONYM DEP FOR DEPARTAMENTOS;
```

Sinónimo creado.

Ahora podemos utilizar indistintamente el nombre o el sinónimo para referirnos a cualquiera de las dos tablas para recuperar información, insertar, borrar o modificar filas, etcétera:

```
SQL> SELECT * FROM DEP;
```

DEP_NO	DNOMBRE	LOCALIDAD
10	CONTABILIDAD	BARCELONA
20	INVESTIGACION	VALENCIA
30	VENTAS	MADRID
40	PRODUCCION	SEVILLA

Los sinónimos pueden facilitarnos el acceso a las tablas que se encuentran en otros esquemas (que son de otros usuarios). Por ejemplo, el usuario *alu01* puede crearse un sinónimo para acceder a la tabla *empleados* del usuario *cursosql*, ya que de otra forma tendría que referirse a ella como *cursosql.empleados*.

```
SQL> CONNECT ALU01
```

```
Introduzca su clave: *****
```

Conectado.

```
SQL> CREATE SYNONYM EMPLEADOS FOR CURSOSQL.EMPLEADOS;
```

Sinónimo creado.

Por supuesto, el hecho de tener un sinónimo creado sobre un objeto no sustituye a los permisos necesarios para acceder a tal objeto. De manera que si el usuario *alu01* no tuviese permiso para acceder a la tabla *empleados* del usuario *cursoSQL* el sistema le dejará crear el sinónimo (si dispone de tal privilegio) pero no accederá a la información.

### Sinónimos públicos.

Los sinónimos creados como hemos visto en el apartado anterior son operativos solamente para el usuario que lo crea. **Cuando queremos que el sinónimo sea visible y operativo para todos los usuarios de la base de datos crearemos un sinónimo público utilizando la sentencia CREATE PUBLIC SYNONYM según el siguiente formato:**

```
CREATE PUBLIC SYNONYM nombredelsinonimo FOR objeto;
```

Por ejemplo, el usuario *cursoSQL* puede crear un sinónimo público para la tabla *empleados* de su propiedad:

```
SQL> CREATE PUBLIC SYNONYM EMPLEADOS FOR CURSOSQL.EMPLEADOS;
```

Sinónimo creado.

Una vez creado el sinónimo público, cualquier usuario de la base de datos podrá referirse al objeto utilizando el sinónimo.

Para poder crear un sinónimo público hace falta tener el privilegio CREATE PUBLIC SYNONYM.

### Eliminación de sinónimos.

Para eliminar un sinónimo utilizaremos el comando DROP SYNONYM según se especifica en el siguiente formato:

```
DROP SYNONYM nombredelsinonimo;
```

O bien, si se trata de un sinónimo público:

```
DROP PUBLIC SYNONYM nombredelsinonimo;
```

Ejemplo:

```
SQL> DROP PUBLIC SYNONYM EMPLEADOS;
```

Sinónimo borrado.

Nota: Con este comando se borra el sinónimo pero no el objeto al que hace referencia.

---

## TEMA 11. SEGURIDAD EN ACCESS.

---

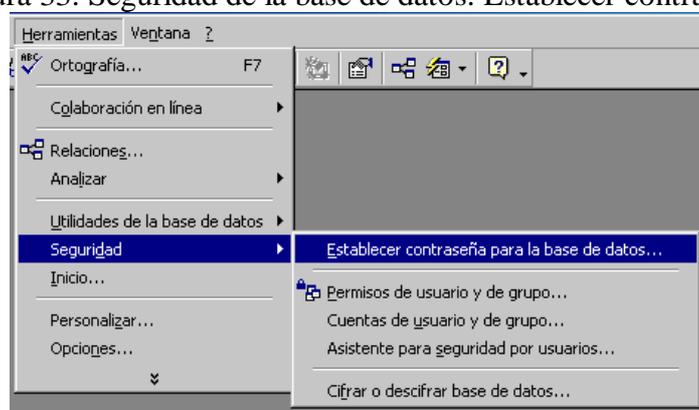
En Access podemos proteger la base de datos utilizando varios métodos.

### ***Habilitar una contraseña para abrir la base de datos.***

---

Es el método más sencillo. Para ello abrir el menú *Herramientas/Seguridad/Establecer contraseña para la base de datos*. Ver Figura 33.

Figura 33. Seguridad de la base de datos. Establecer contraseña.



A continuación se muestra un cuadro de diálogo que pide teclear la contraseña y confirmarla.

Para establecer la contraseña, la base de datos tiene que estar abierta en modo exclusivo, es decir hay que abrir la base de datos desde el menú *Archivo/Abrir*, y en la ventana de abrir desplegamos la lista que acompaña al botón *Abrir*  que aparece en la parte inferior derecha. Ver figura 34.

Figura 34. Abrir base de datos en modo exclusivo.



Una vez que se ha habilitado la contraseña, al abrir la base de datos se mostrará un cuadro de diálogo que pide teclearla. Al abrirla podremos trabajar con todos los objetos de la base de datos.

Si se desea anular la contraseña lo hacemos desde la opción *Herramientas/Seguridad/Anular la contraseña establecida para la base de datos*.

Cuando creamos una contraseña para una base de datos es importante no olvidarla pues si se olvida o se pierde no podremos abrirla, es decir no podremos recuperar los datos con lo cual perdemos nuestra información.

### ***Protección de objetos mediante la seguridad por usuarios.***

Las órdenes para la gestión de seguridad de SQL no sirven para ejecutarlas en Access en la vista SQL, sin embargo Access posee unas herramientas para proteger la base de datos que permiten crear usuarios, grupos de usuarios y dar permisos de acceso a cada uno de los objetos que forman la Base de datos.

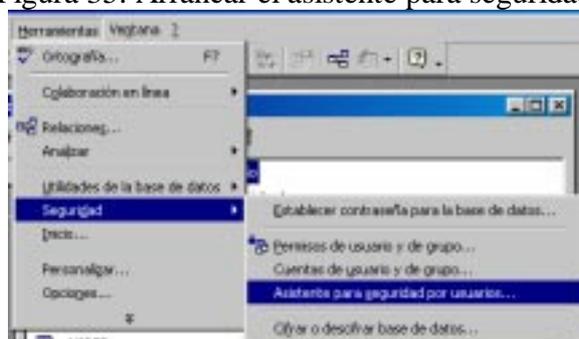
La seguridad por usuarios se utiliza para proteger los datos y la estructura de los distintos objetos que forman la base de datos ante cambios imprevistos. Se trata de impedir que los usuarios modifiquen o inutilicen una base de datos.

Para proteger una base de datos en Access se utiliza el *Asistente para seguridad por usuarios* que facilita una rápida protección de la base de datos de una manera cómoda y sencilla.

Seguiremos los siguientes pasos:

**1º** - Desde el menú *Herramientas/Seguridad/Asistente para seguridad por usuarios*. iniciamos el asistente para seguridad. Ver Figura 35.

Figura 35. Arrancar el asistente para seguridad.



2º - Si es la primera vez que iniciamos el asistente en esa base de datos aparece activada la primera opción en la que indicamos que se va a crear un archivo de información de grupo de trabajo. Este archivo va a contener la información inicial de los usuarios que comparten los datos de la base de datos y de los permisos sobre los objetos de la misma.

Si no es la 1ª vez aparece activada la 2ª opción que es la de modificar ese mismo archivo. Ver Figura 36. En esta ventana pulsamos *Siguiente*.

Figura 36. Asistente para seguridad. Paso 2.



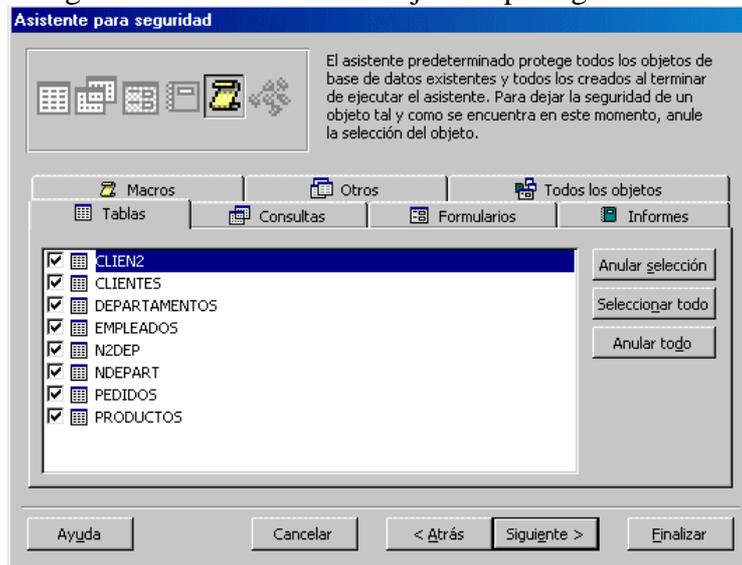
3º - En esta ventana indicamos el nombre que se desea dar al archivo de información del grupo de trabajo. Access asigna uno por defecto, que si se desea se cambia. Dejaremos activada la opción: *Crear un método abreviado para abrir la base de datos protegida*. Esto hace que la protección sea sólo para esta base de datos. Si elegimos la primera opción el grupo de trabajo generado será para todas las bases de datos. Ver Figura 37. Pulsamos *Siguiente*.

Figura 37. Asistente para seguridad. Paso 3.



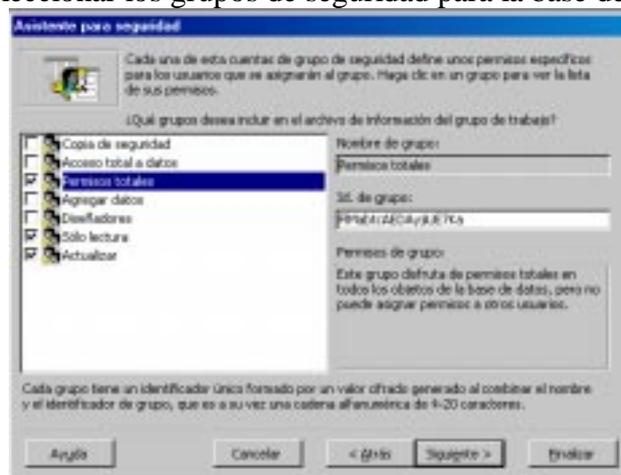
4º - En esta ventana seleccionaremos todos los objetos de la base de datos que se desea proteger. Ver Figura 38. Una vez seleccionados pulsamos *Siguiente*.

Figura 38. Seleccionar los objetos a proteger. Paso 4.



5º - A continuación aparecen todos los grupos de seguridad o roles predefinidos de Access, cada uno de ellos contiene una serie de permisos específicos. Se pueden ver los permisos cuando se selecciona el grupo en el cuadro *Permisos de grupo*. Así pues indicaremos los grupos de seguridad que vamos a utilizar para luego asignárselos a los usuarios. Ver Figura 39. Una vez seleccionados pulsamos *Siguiente*. Vamos a seleccionar todos los grupos disponibles para nuestra base de datos.

Figura 39. Seleccionar los grupos de seguridad para la base de datos. Paso 5.

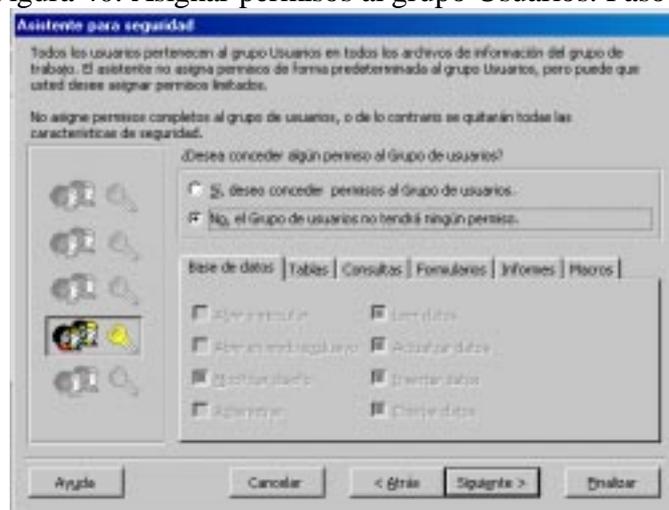


Los grupos de seguridad son los siguientes:

- ♦ **Copia de seguridad.** Este grupo puede abrir la base de datos en modo exclusivo para compactarla y repararla, pero no puede ver los objetos que contiene.
- ♦ **Acceso total a datos.** Derechos totales de acceso para modificar datos, pero no puede modificar el diseño de los objetos de la base de datos.
- ♦ **Permisos totales.** Tiene permiso sobre todos los objetos de la base de datos, pero no tiene permiso para asignar permisos a otros usuarios.
- ♦ **Agregar datos.** Este grupo puede leer e insertar datos en las tablas, pero no puede ni cambiar el diseño de los objetos, ni eliminar ni actualizar datos.
- ♦ **Diseñadores.** Este grupo tiene permiso para modificar datos y todos los objetos pero no puede modificar ni las tablas ni las relaciones entre ellas.
- ♦ **Solo lectura.** Tiene permiso para abrir la base de datos, leer todos los datos pero no puede modificar ni los datos ni el diseño de los objetos de la base de datos.
- ♦ **Actualizar.** Este grupo puede leer y actualizar datos. No puede ni insertar, ni borrar datos ni modificar el diseño de los objetos de la base de datos.

6º - En la ventana que aparece a continuación tenemos que indicar si queremos dar algún permiso al grupo *Usuarios*. El asistente no asigna permisos al grupo *Usuarios*, pero podemos indicarlos en esta ventana. De momento no asignamos ningún permiso, más adelante se pueden hacer pruebas para ver lo que ocurre. Ver Figura 40. Una vez seleccionados pulsamos *Siguiente*.

Figura 40. Asignar permisos al grupo Usuarios. Paso 6.



Existen dos grupos de usuarios iniciales:

- ♦ El grupo *Usuarios* que va a contener todas las cuentas de usuario de la base de datos. Cuando se crea un usuario este se incluye automáticamente dentro del grupo. Los usuarios del grupo *Usuarios* podrán crear objetos y tendrán todos los permisos sobre ellos, a no ser que estos sean revocados.
- ♦ El grupo *Administradores*. Este grupo tiene permiso completo sobre los objetos de la base de datos. Siempre debe de haber al menos un usuario en este grupo.

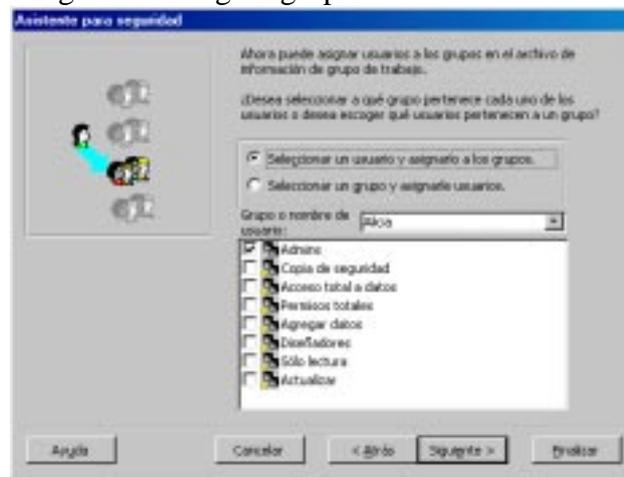
7º - En la siguiente ventana es donde creamos los usuarios que van a trabajar en esa base de datos. En *Nombre de usuario* escribimos el nombre, en *Contraseña* la clave de acceso y pulsamos al botón *Agregar usuario a la lista*. Ver Figura 41. Todos los usuarios pertenecerá al grupo *Usuarios*. Una vez creados los usuarios pulsamos *Siguiente*.

Figura 41. Crear los usuarios de la Base de datos. Paso 7.



8º - En la siguiente ventana es donde asignamos roles o grupos de seguridad a los usuarios. Recuerda que los grupos predefinidos tienen una serie de privilegios. Dependiendo de los privilegios que ha de tener el usuario le asignaremos un grupo. Podremos asignar grupos a usuarios o usuarios a grupos. Ver Figura 42. Una vez asignados los usuarios a los grupos pulsamos *Siguiente*.

Figura 42. Asignar grupos a usuarios. Paso 8.



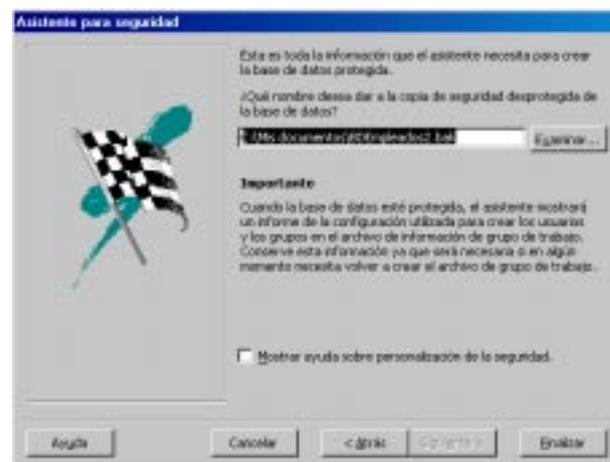
*No olvidar asignar a un usuario el grupo de seguridad Admins para que pueda ser administrador de la base de datos.*

*Si no asignamos un usuario a un grupo de seguridad este no podrá entrar en la base de datos.*

*Para que el usuario pueda abrir la base de datos y ejecutar consultas le asignaremos el grupo de seguridad: Solo lectura.*

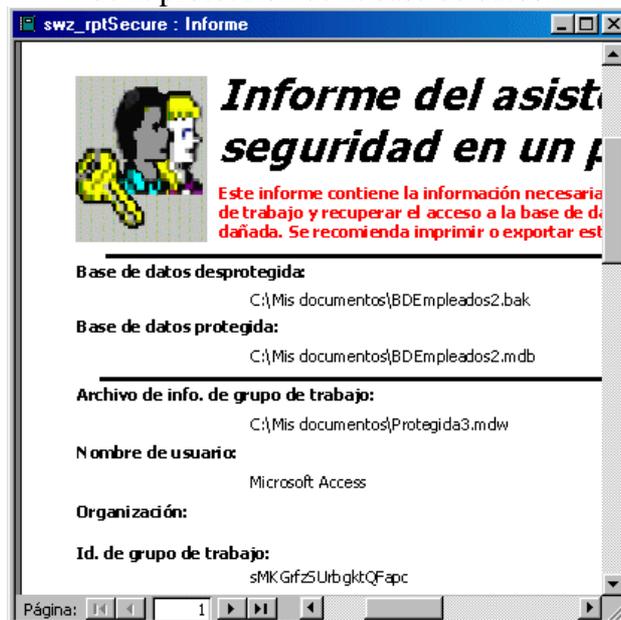
**9º** - A continuación Access hace una copia de seguridad desprotegida de la base de datos por si se necesita en algún momento. Ver Figura 43. Pulsamos *Finalizar*.

Figura 43. Copia desprotegida de la base de datos.



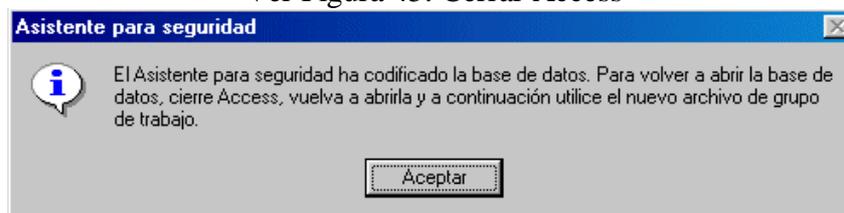
**10°** - Al finalizar la protección Access visualiza un informe con los datos de la base de datos protegida y los usuarios creados con sus permisos, conviene imprimir el informe y guardarlo para tener toda la información de la base de datos este archivo tiene la extensión .SNP. Ver Figura 44. Indicaremos que sí se desea guardar el documento..

Ver Figura 44. Informe generado con información de la protección de la base de datos.



**11°**- Por último Access crea un acceso directo a la base de datos protegida en el escritorio y va a pedir iniciar una nueva sesión con la base de datos protegida. Ver Figura 45.

Ver Figura 45. Cerrar Access



**Para entrar en la base de datos protegida se hará desde el acceso directo creado en el escritorio.**

Si miramos las propiedades de ese acceso directo vemos la línea de ejecución:

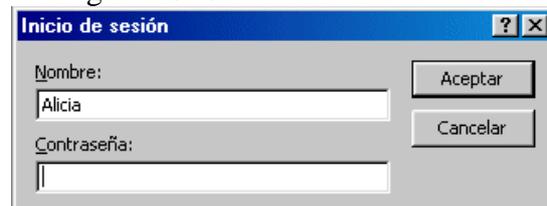
```
"C:\APP\Microsoft Office\Office\MSACCESS.EXE "  
    "C:\Mis documentos\BDEmpleados2.mdb" /WRKGRP  
    "C:\Mis documentos\Protegida3.mdw"
```

En la que se indica :

- ♦ que se va a ejecutar Access,
- ♦ la base de datos es BDEmpleados2.mdb,
- ♦ que se ejecuta en modo protegida (/WRKGRUP) y
- ♦ que el grupo de trabajo donde está la información de usuarios y permisos es Protegida3.mdw.

12º- Una vez que la base de datos está protegida si la abrimos a través de ese acceso directo. Aparece la ventana de conexión en la que teclearemos el nombre de usuario y la contraseña. Ver Figura 46

Ver Figura 46. Conexión a la base de datos.



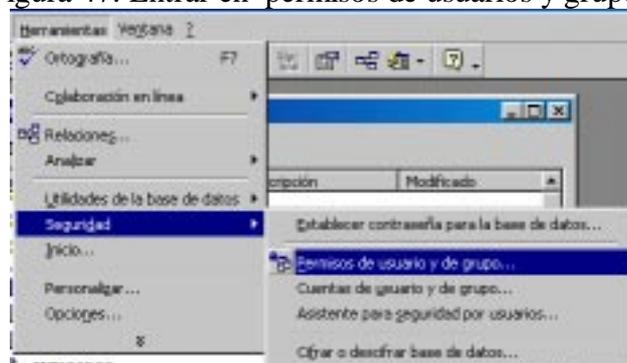
## ***Gestión de usuarios y grupos.***

Una vez que la base de datos está protegida podemos dar y revocar permisos a los usuarios o grupos sobre los objetos. Podemos crear más usuario o más grupos y asignar usuarios a grupos y grupos a usuarios.

### **Dar permisos a los usuarios y grupos sobre los objetos de la base de datos.**

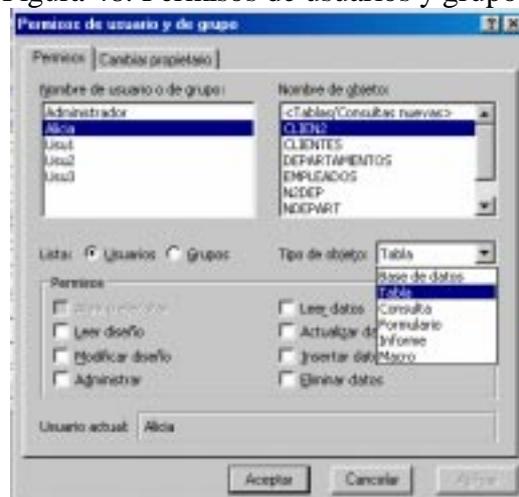
Para gestionar los permisos de los usuarios y de los grupos entramos como usuario administrador y desde el menú **Herramientas/Seguridad/Permisos de Usuario y de Grupo**. Ver Figura 47.

Figura 47. Entrar en permisos de usuarios y grupos.



Si elegimos esta opción aparece la ventana que se muestra en la Figura 48:

Figura 48. Permisos de usuarios y grupos.



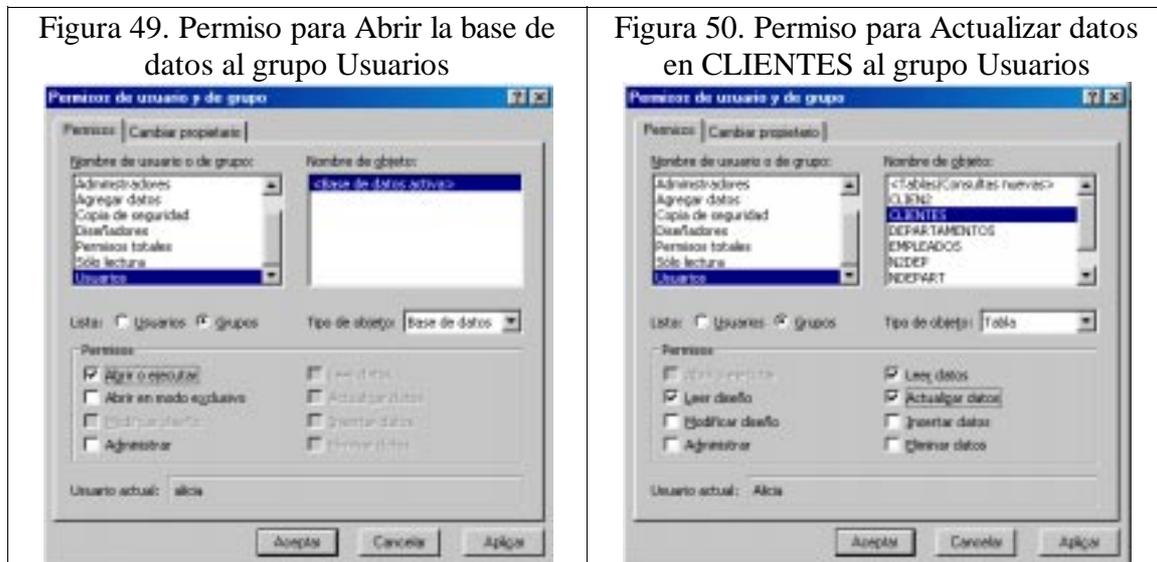
En esta ventana podemos cambiar los permisos sobre los objetos de la base de datos. Si desplegamos la lista **Tipo de objetos** vemos los objetos que se pueden seleccionar (ver figura 48), cuando seleccionamos un objeto aparecen los permisos concedidos en las casillas de selección. Para dar permisos elegimos el usuario o el grupo, elegimos el tipo de objeto y vamos eligiendo los nombres de objeto, seleccionaremos las casillas de los permisos que se desea dar.

Los permisos son los siguientes:

- ♦ **Abrir o ejecutar.** Este permiso se activa cuando seleccionamos *Base de datos* en *Tipo de objeto*. Solo permite entrar en la base de datos. Para que un usuario o grupo puede entrar en la base de datos tiene que tener activado este permiso.
- ♦ **Leer diseño.** Permite ver el diseño del objeto o los objetos seleccionados.
- ♦ **Modificar diseño.** Permite modificar el diseño del objeto o los objetos seleccionados.
- ♦ **Administrar.** Permite leer y modificar el diseño, y además leer, actualizar, insertar y eliminar datos en los objetos seleccionados.
- ♦ **Leer datos.** Permite leer el diseño y los datos de los objetos seleccionados.
- ♦ **Actualizar datos.** Permite leer el diseño, leer los datos y actualizarlos en los objetos seleccionados.
- ♦ **Insertar datos.** Permite leer el diseño, leer los datos e insertarlos en los objetos seleccionados.
- ♦ **Eliminar datos.** Permite leer el diseño, leer los datos y eliminarlos de los objetos seleccionados.

Por ejemplo damos al grupo *Usuarios* (Inicialmente todos los usuarios son del grupo *Usuarios* y no tienen ningún permiso) permiso para *Abrir o ejecutar* la base de datos,

Figura 49, y permiso para *Actualizar* la tabla CLIENTES, ver Figura 50. Así pues si iniciamos sesión con cualquier usuario podremos cambiar datos de esta tabla.



Para que un usuario pueda abrir la base de datos y ver el contenido de las tablas ha de tener permiso de *Leer diseño* y *Leer datos* en todas las tablas y de *Abrir y ejecutar* la base de datos. Estos privilegios los tiene el grupo de seguridad *Solo Lectura*, así pues si el usuario pertenece a ese grupo tiene permiso para entrar en la base de datos y ver el contenido de las tablas.

Los privilegios concedidos a cada grupo se pueden ver si se selecciona el grupo y luego el tipo de objeto. Ver Figura 51. Por ejemplo en la imagen vemos los privilegios del grupo *Solo lectura* sobre las tablas.

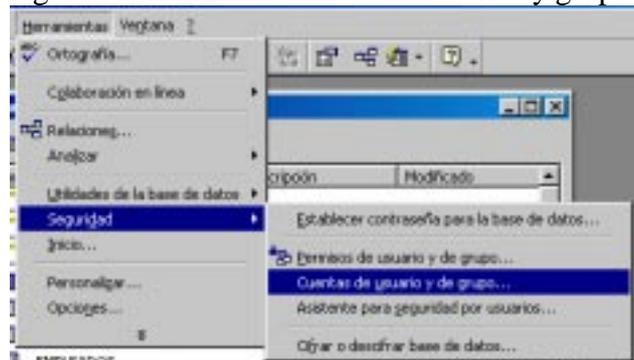
Figura 51. Grupo *Sólo lectura*.



## Asignar un grupo a un usuario.

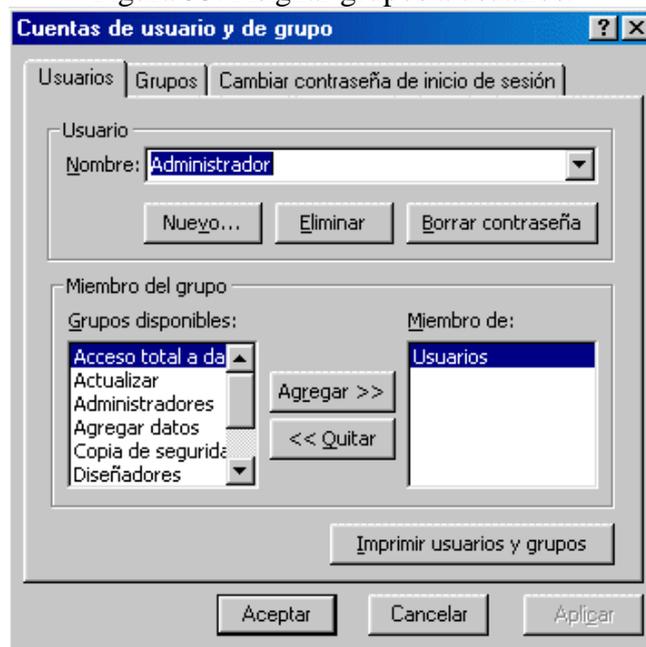
Los grupos de seguridad predefinidos tienen una serie de privilegios sobre los objetos de la base de datos cuando creamos un usuario decidimos los privilegios que va a tener y en función de esos privilegios le asignamos un grupo de trabajo. Esto lo hacemos desde el menú *Herramientas/Seguridad/Cuentas de Usuario y de Grupo*. Ver Figura 52.

Figura 52. Entrar en cuentas de usuarios y grupos.



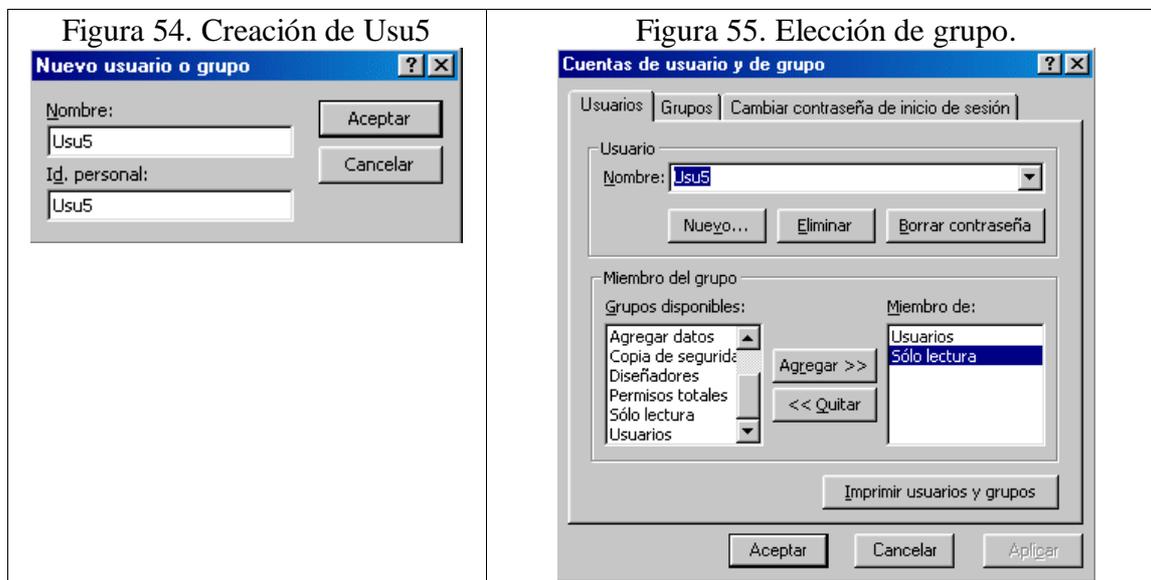
En la ventana que aparece podemos elegir la ficha *Usuarios*, elegir un usuario desplegando la lista correspondiente y podremos: borrar el usuario, crear más usuarios, cambiar la contraseña del usuario y además asignarle los grupos de la lista de grupos que aparecen en el cuadro de diálogo. Ver figura 53.

Figura 53. Asignar grupos a usuarios.



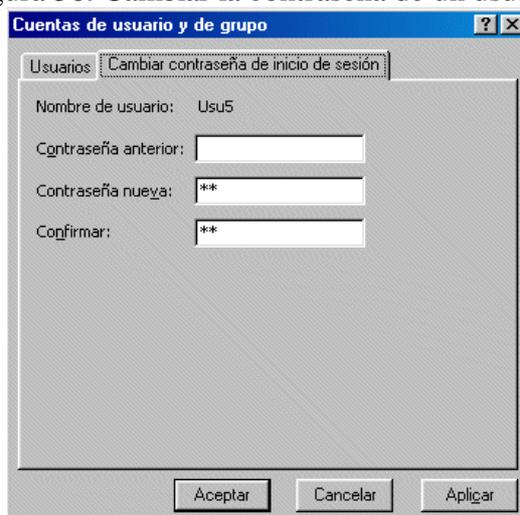
Por ejemplo vamos a crear el usuario *Usu5* y vamos a hacerle miembro del grupo de *Solo lectura*.

Pulsamos al botón *Nuevo*, y aparece la ventana para teclear el usuario, escribimos el nombre y su identificación. Ver Figura 54. Una vez creado Elegimos el grupo al de la lista de grupos disponibles. Ver Figura 55



Una vez que está creado el usuario cerramos la base de datos y entramos de nuevo con el usuario. Inicialmente el usuario no tiene contraseña, para poner una contraseña a ese usuario, iniciamos una sesión con el mismo, abrimos el menú **Herramientas/Seguridad/Cuentas de Usuario y de Grupo**, elegimos la pestaña *Cambiar contraseña de inicio de sesión*, y en la ventana que aparece tecleamos la contraseña deseada. Ver Figura 56.

Figura 56. Cambiar la contraseña de un usuario.



## Anexo

### FUNCIONES AVANZADAS Y CARACTERÍSTICAS ESPECIALES

Autor: Fernando Montero.

#### Introducción

Esta unidad trata sobre funciones y características que están disponibles en el gestor de base de datos Oracle utilizado como referencia a lo largo de este curso. Algunos otros gestores implementan algunas de estas características aunque con importantes variaciones ya que no están definidas en el estándar. El objetivo es dar a conocer algunas posibilidades adicionales sin profundizar demasiado en todas las opciones de utilización, remitiendo al alumno al manual del producto donde encontrará información más detallada.

#### Recuperación jerárquica.

Oracle y algunos gestores de bases de datos permiten la recuperación jerárquica de la información siempre y cuando exista alguna relación jerárquica establecida.

En nuestra tabla empleados existe esa relación en las columnas *emp\_no* y *director*.

```
SQL> SELECT * FROM EMPLEADOS;
```

EMP_NO	APELLIDO	OFICIO	DIRECTOR	FECHA_AL	SALARIO	COMISION	DEP_NO
7839	REY	PRESIDENTE		17/11/81	600000		10
7698	GARRIDO	DIRECTOR	7839	01/05/81	385000		30
7782	MARTINEZ	DIRECTOR	7839	09/06/81	245000		10
7499	ALONSO	VENDEDOR	7698	20/02/81	140000	40000	30
7521	LOPEZ	EMPLEADO	7782	08/05/81	135000		10
7654	MARTIN	VENDEDOR	7698	28/09/81	150000	160000	30
7844	CALVO	VENDEDOR	7698	08/09/81	180000	0	30
7876	GIL	ANALISTA	7782	06/05/82	335000		20
7900	JIMENEZ	EMPLEADO	7782	24/03/83	140000		20

Podemos observar que:

- REY es el director de GARRIDO y MARTINEZ
- GARRIDO es el director de ALONSO, MARTIN y CALVO.
- MARTINEZ es el director de LOPEZ, GIL y JIMENEZ.

Para recuperar la información en forma jerárquica utilizaremos la instrucción SELECT junto con algunas cláusulas que hacen posible dicha estructuración:

**START WITH *condición*** Es una condición que determina la fila o filas que serán la raíz o punto de partida de la jerarquía.

**CONNECT BY** especifica el criterio que define la relación. Habitualmente tendrá el formato:

```
CONNECT BY PRIOR columna operador_comparacion columna
```

Ejemplo:

```
SQL> SELECT APELLIDO, EMP_NO, OFICIO, DIRECTOR
      FROM EMPLEADOS
      CONNECT BY PRIOR EMP_NO = DIRECTOR
      START WITH APELLIDO = 'REY';
```

APELLIDO	EMP_NO	OFICIO	DIRECTOR
REY	7839	PRESIDENTE	
GARRIDO	7698	DIRECTOR	7839
ALONSO	7499	VENDEDOR	7698
MARTIN	7654	VENDEDOR	7698
CALVO	7844	VENDEDOR	7698
MARTINEZ	7782	DIRECTOR	7839
LOPEZ	7521	EMPLEADO	7782
GIL	7876	ANALISTA	7782
JIMENEZ	7900	EMPLEADO	7782

9 filas seleccionadas.

### La pseudocolumna LEVEL.

Cuando utilizamos cláusulas de recuperación jerárquica está disponible una pseudocolumna denominada LEVEL que devuelve el nivel al que se encuentra una fila en el árbol jerárquico. Este nivel comienza en el 1 para *la raíz*, continúa en el 2 para los *hijos de la raíz*, el tres para los *hijos* de estos, y así sucesivamente.

Si al ejemplo anterior le añadimos la pseudocolumna LEVEL obtendremos:

```
SQL> SELECT APELLIDO, EMP_NO, OFICIO, DIRECTOR, LEVEL
      FROM EMPLEADOS
      CONNECT BY PRIOR EMP_NO = DIRECTOR
      START WITH APELLIDO = 'REY';
```

APELLIDO	EMP_NO	OFICIO	DIRECTOR	LEVEL
REY	7839	PRESIDENTE		1
GARRIDO	7698	DIRECTOR	7839	2
ALONSO	7499	VENDEDOR	7698	3
MARTIN	7654	VENDEDOR	7698	3

---

CALVO	7844	VENDEDOR	7698	3
MARTINEZ	7782	DIRECTOR	7839	2
LOPEZ	7521	EMPLEADO	7782	3
GIL	7876	ANALISTA	7782	3
JIMENEZ	7900	EMPLEADO	7782	3

9 filas seleccionadas.

Habitualmente esta pseudocolumna se utiliza para mejorar la presentación:

```
SQL> SELECT LPAD(' ',4*(LEVEL)) || LEVEL || '.' || APELLIDO
Organigrama
      FROM EMPLEADOS
      CONNECT BY PRIOR EMP_NO = DIRECTOR
      START WITH APELLIDO = 'REY';
```

ORGANIGRAMA

```
-----
1.REY
  2.GARRIDO
    3.ALONSO
    3.MARTIN
    3.CALVO
  2.MARTINEZ
    3.LOPEZ
    3.GIL
    3.JIMENEZ
```

9 filas seleccionadas.

## ***La función DECODE.***

---

La función DECODE permite realizar conversiones de valores según se especifica en el siguiente formato:

<b>DECODE <i>expresion</i>, <i>lista_de_asignaciones</i>, [<i>valor_por_defecto</i>]</b>
--

Donde:

- *expresion* es la expresión SQL que va a ser evaluada. Normalmente se tratará de una columna o de una expresión de columna.
- *lista\_de\_asignaciones* es una lista con la siguiente estructura:  
*valor1, resultado1, valor2, resultado2*, etcétera.  
 En esta lista se indicará cada uno de los valores de la expresión que queremos transformar (*valorn*), y seguidamente el resultado (*resultadon*) que queremos obtener en caso de que se produzca ese valor.

Ejemplo:

```
DECODE (DEP_NO, 10, 'CON', 20, 'INV', 30, 'VEN', 40, 'PRO');
```

Puesto que el formato anterior puede inducir a errores, normalmente se utilizan saltos de línea:

```
DECODE (DEP_NO, 10, 'CON',
        20, 'INV',
        30, 'VEN',
        40, 'PRO')
```

La función DECODE formará parte de una sentencia de recuperación de datos:

```
SQL> SELECT APELLIDO, SALARIO, DECODE (DEP_NO, 10, 'CON',
                                     20, 'INV',
                                     30, 'VEN')
        FROM EMPLEADOS WHERE SALARIO > 300000;
```

APELLIDO	SALARIO	DEC
REY	600000	CON
GARRIDO	385000	VEN
GIL	335000	INV

- La opción *valor\_por\_defecto* permite especificar un valor que se devolverá en el caso de que el valor de la expresión no corresponda con ninguno de los indicados en la lista.

```
SQL> SELECT APELLIDO, SALARIO, DECODE (DEP_NO, 10, 'CON',
                                     20, 'INV',
                                     '----')
        FROM EMPLEADOS WHERE SALARIO > 300000;
```

APELLIDO	SALARIO	DEC
REY	600000	CON
GARRIDO	385000	---
GIL	335000	INV

**Nota:** si no se especifica el valor por defecto y el valor que toma la expresión no coincide con ninguno de los valores indicados, la función retornará NULL.

---

## **Disparadores.**

---

Podemos definir un disparador como un programa (procedimiento almacenado) asociado a una tabla que se ejecuta o dispara automáticamente cuando ocurren ciertos sucesos o eventos que modifican los datos de la tabla (INSERT, UPDATE o DELETE).

Se suelen utilizar para:

- ♦ Implementar reglas administrativas y/o restricciones complejas de seguridad o integridad.
- ♦ Auditar actualizaciones y prevenir transacciones erróneas.

A continuación mostramos un ejemplo de un disparador *auditar\_salario* que se disparará automáticamente después de cada modificación del salario de un empleado insertando en una tabla de auditoria ( *T\_auditar1* ) los datos del cambio realizado.

```
CREATE TRIGGER auditar_salario
  AFTER UPDATE OF salario
  ON empleados
  FOR EACH ROW
BEGIN
  insert INTO T_auditar1
    VALUES ( 'MODIFICACIÓN SALARIO: ' || :old.emp_no ||
             ' VARIACIÓN: ' || :new.salario - :old.salario ||
             ' FECHA: ' || SYSDATE );
END;
```

Podemos observar en el trigger dos partes:

- ♦ **La cabecera del trigger** donde se indica:
  - ♦ El nombre del trigger
  - ♦ El evento que lo dispara (BEFORE o AFTER INSERT, UPDATE, DELETE) y la tabla (ON *nombretabla*) a la que está asociado.
  - ♦ El nivel de disparo: columna (FOR EACH ROW) u orden (FOR EACH STATEMENT).
  - ♦ Una restricción o condición (opcional).
- ♦ **El cuerpo del trigger:**
  - ♦ Es un bloque de código (en el caso de Oracle se trata de código PL/SQL) que especifica las acciones a realizar.

Nota: un trigger forma parte de la operación de actualización que lo disparó de manera que si el trigger falla Oracle dará por fallida la actualización completa, deshaciendo cualquier cambio que se pudiese haber producido en la base de datos.

---

Existen restricciones en la utilización de los triggers:

- En el bloque PL/SQL no se pueden utilizar comandos DDL, ni sentencias de control de transacciones como COMMIT, ROLLBACK .
- Un trigger no puede contener instrucciones que consulten o modifiquen **tablas mutantes** (aquellas que están siendo modificadas, en la misma sesión, por una sentencia UPDATE, DELETE o INSERT).
- Tampoco se pueden modificar valores de las columnas que sean claves primaria, única o ajena de **tablas de restricción** (aquellas tablas que deben ser consultadas o actualizadas, en la misma sesión, por el comando que disparó el trigger -normalmente debido a una restricción de integridad referencial-).

Nota: Los disparadores son una herramienta muy útil pero **su uso indiscriminado puede degradar el comportamiento de la base de datos**. Para implementar restricciones de integridad se utilizarán preferentemente las restricciones ya disponibles: PRIMARY KEY, FOREIGN KEY, CHECK, NOT NULL, UNIQUE, etcétera.

### ***Utilización del diccionario de datos.***

---

El diccionario de datos o catálogo del sistema es un conjunto de tablas y vistas donde el gestor de la base de datos guarda toda la información sobre la base de datos:

- ♦ Nombres y localizaciones.
- ♦ Restricciones de integridad.
- ♦ Privilegios.
- ♦ Información para auditoría.
- ♦ Etcétera.

Estas tablas son propiedad del usuario SYS y se crean en el tablespace SYSTEM. No está permitida su actualización mediante comandos de manipulación de datos (INSERT, UPDATE o DELETE). El gestor de la base de datos las actualizará automáticamente cada vez que se produzca un comando DML (CREATE, ALTER, DROP, ...)

Las vistas y los sinónimos del diccionario de datos se crean para facilitar el acceso a las tablas. Respecto a las primeras, podemos distinguir tres tipos de prefijos que determinan las filas a seleccionar:

- ♦ USER : Las vistas que comienzan con USER son accesibles por cualquier usuario de la BD. Proporcionan información relativa a el usuario que la solicita y los objetos que son de su propiedad. En realidad retornan la misma información que las vistas cuyo

prefijo es ALL pero seleccionando solamente aquellas filas en las que la columna OWNER es igual al usuario que solicita la información.

Por ejemplo, la siguiente instrucción utiliza la vista USER\_TABLES para visualizar las tablas que son propiedad del usuario actual (CURSOSQL) y el tablespace (espacio físico) donde se encuentran:

```
SQL> SELECT TABLE_NAME, TABLESPACE_NAME FROM USER_TABLES;
```

TABLE_NAME	TABLESPACE_NAME
CLIEN2	USER_DATA
CLIENTES	USER_DATA
DEPARTAMENTOS	USER_DATA
EMPLEADOS	USER_DATA
N2DEPT	USER_DATA
NDEPART	USER_DATA
PEDIDOS	USER_DATA
PRODUCTOS	USER_DATA

8 filas seleccionadas.

- ♦ ALL : las vistas que comienzan con ALL proporcionan información a cerca de los objetos a los cuales tiene acceso el usuario por ser propietario, o por tener privilegios concedidos explícitamente o a través de PUBLIC. Tienen las mismas columnas que las vistas USER\_... y otra adicional que hace referencia al propietario del objeto (OWNER).  
Si introducimos la instrucción anterior pero indicando ALL\_TABLES en lugar de USER\_TABLES obtendremos todas las tablas a las que tiene acceso el usuario. Si además incluimos la cláusula WHERE OWNER = `CURSOSQL` la salida será idéntica a la obtenida con USER\_TABLES (siempre que el usuario actual sea CURSOSQL).
- ♦ DBA: las vistas que comienzan con DBA incluyen información de todos los objetos de la base de datos. Se necesita el privilegio SELECT ANY TABLE o el rol DBA para tener acceso a estas vistas.

En nuestra instalación de Oracle 8 encontramos 286 tablas y vistas en el diccionario de datos (sin contar los sinónimos). No es nuestro objetivo estudiarlas todas, ni siquiera una pequeña parte de ellas pero sí mencionaremos que existe una herramienta muy útil que podemos consultar para saber donde encontrar la información que buscamos:

**DICTIONARY** o su sinónimo **DICT** proporciona información sobre todas las tablas y vistas del diccionario de datos.

```
SQL> DESCRIBE DICT;
```

Name	Null?	Type
TABLE_NAME		VARCHAR2(30)
COMMENTS		VARCHAR2(4000)

La primera columna (TABLE\_NAME) indica el nombre de la tabla o vista, y la segunda (COMMENTS) contiene un resumen a cerca de la información que contiene dicha tabla o vista.

Utilizando las posibilidades de búsqueda que brinda SQL podemos realizar consultas para saber dónde encontrar la información que necesitamos.

Por ejemplo, si queremos saber qué tablas o vistas del diccionario de datos contienen información sobre privilegios podemos lanzar la siguiente consulta:

```
SQL> SELECT TABLE_NAME, SUBSTR(COMMENTS,1,50) FROM DICT
WHERE UPPER(COMMENTS) LIKE '%PRIVILEGES%';
```

TABLE_NAME	SUBSTR(COMMENTS,1,50)
DBA_PRIV_AUDIT_OPTS	Describes current system privileges being audited
DBA_SYS_PRIVS	System privileges granted to users and roles
USER_SYS_PRIVS	System privileges granted to current user
ROLE_SYS_PRIVS	System privileges granted to roles
ROLE_TAB_PRIVS	Table privileges granted to roles
SESSION_PRIVS	Privileges which the user currently has set

6 filas seleccionadas.

La siguiente consulta proporciona información sobre las vistas del diccionario de datos que contienen información sobre índices:

```
SQL> SELECT TABLE_NAME, SUBSTR(COMMENTS,1,50) FROM DICT
WHERE UPPER(COMMENTS) LIKE '%INDEXES%';
```

TABLE_NAME	SUBSTR(COMMENTS,1,50)
ALL_INDEXES	Descriptions of indexes on tables accessible to th
ALL_IND_COLUMNS	COLUMNS comprising INDEXes on accessible TABLES
DBA_INDEXES	Description for all indexes in the database
DBA_IND_COLUMNS	COLUMNS comprising INDEXes on all TABLES and CLUST
USER_INDEXES	Description of the user's own indexes
USER_IND_COLUMNS	COLUMNS comprising user's INDEXes or on user's TAB
IND	Synonym for USER_INDEXES

7 filas seleccionadas.

Una vez que encontramos la tabla que contiene la información que necesitamos tendremos que realizar la consulta según los datos requeridos.

Podemos encontrar una completa descripción de las columnas de las tablas del diccionario de datos en la tabla DICT\_COLUMNS;

```
SQL> DESCRIBE DICT_COLUMNS;
```

Name	Null?	Type
TABLE_NAME		VARCHAR2(30)
COLUMN_NAME		VARCHAR2(30)
COMMENTS		VARCHAR2(4000)

La siguiente instrucción visualiza la descripción de las columnas de la vista DBA\_TABLES (por razones de espacio hemos restringido la longitud de la columna COMMENTS a las cincuenta primeras posiciones):

```
SQL> SELECT COLUMN_NAME, SUBSTR(COMMENTS,1,50)
FROM DICT_COLUMNS
WHERE TABLE_NAME = 'DBA_TABLES'
```

COLUMN_NAME	SUBSTR(COMMENTS,1,50)
MAX_EXTENTS	Maximum number of extents allowed in the segment
PCT_INCREASE	Percentage increase in extent size
FREELISTS	Number of process freelists allocated in this segm
FREELIST_GROUPS	Number of freelist groups allocated in this segmen
LOGGING	Logging attribute
BACKED_UP	Has table been backed up since last modification?
NUM_ROWS	The number of rows in the table
BLOCKS	The number of used blocks in the table
EMPTY_BLOCKS	The number of empty (never used) blocks in the tab
AVG_SPACE	The average available free space in the table
CHAIN_CNT	The number of chained rows in the table
AVG_ROW_LEN	The average row length, including row overhead
AVG_SPACE_FREELIST_BLOCKS	The average freespace of all blocks on a freelist
NUM_FREELIST_BLOCKS	The number of blocks on the freelist
DEGREE	The number of threads per instance for scanning th
INSTANCES	The number of instances across which the table is
CACHE	Whether the table is to be cached in the buffer ca
TABLE_LOCK	Whether table locking is enabled or disabled
SAMPLE_SIZE	The sample size used in analyzing this table
LAST_ANALYZED	The date of the most recent time this table was an
PARTITIONED	Is this table partitioned? YES or NO
IOT_TYPE	If index-only table, then IOT_TYPE is IOT or IOT_O
TEMPORARY	Can the current session only see data that it plac
NESTED	Is the table a nested table?
BUFFER_POOL	The default buffer pool to be used for table block
OWNER	Owner of the table
TABLE_NAME	Name of the table
TABLESPACE_NAME	Name of the tablespace containing the table
CLUSTER_NAME	Name of the cluster, if any, to which the table be
IOT_NAME	Name of the index-only table, if any, to which the
PCT_FREE	Minimum percentage of free space in a block
PCT_USED	Minimum percentage of used space in a block
INI_TRANS	Initial number of transactions
MAX_TRANS	Maximum number of transactions
INITIAL_EXTENT	Size of the initial extent in bytes
NEXT_EXTENT	Size of secondary extents in bytes
MIN_EXTENTS	Minimum number of extents allowed in the segment

37 filas seleccionadas.

## BIBLIOGRAFÍA E INFORMACIÓN COMPLEMENTARIA

### BIBLIOGRAFÍA

ABELLATIF, A. y ZEROUAL, A., *Informix, la base de datos relacional para Unix*, Editorial Diaz de Santos, Madrid.

ABBEY, y otros. ORACLE 8 Guia de Aprendizaje, Editorial McGraw Hill, Madrid.

ABBEY, y otros. Puesta a Punto de ORACLE 8, Editorial McGraw Hill, Madrid

BARKER, R., CASE *\*Metod, Modelo entidad- relación, Oracle*, Editorial Addison Wesley

DATE, *Introducción a los sistemas de bases de datos*, Editorial Addison Wesley.

FERNÁNDEZ BAIZÁN, C. *El modelo relacional de datos*, Editorial Díaz de Santos, Madrid.

FISHER, ALAN S., *Teología CASE. Herramientas de desarrollo de software*, Editorial Anaya Multimedia, Madrid.

GROFF, *Guia de SQL*. Editorial McGraw Hill, Madrid

JACKSON, G., *Introducción al diseño de bases de datos relacionales*, Editorial paraninfo, Madrid. 1988.

KOCH, ORACLE 7 Manual de referencia, Editorial McGraw Hill, Madrid

LONEY. ORACLE 8 Guia del Administrador, Editorial McGraw Hill, Madrid

MARTIN, J. *organización de las bases de datos. 4ª Edición*. Editorial prentice-Hall Hispanoamericana. Madrid.

MARTIN, T. y HARTLEY, T. *DBS/S.Q.L. Manual para programadores*. Editorial Mc Graw Hill, Madrid.

MULLER. *Manual de ORACLE Developer/2000*, Editorial McGraw Hill, Madrid

URMAN. *Oracle 8 Programación en PL/SQL*, Ed. Mc.Graw Hill.

RIVERO CORNELIO. *S.Q.L para programadores*, Editorial paraninfo, Madrid.

RODRÍGUEZ ALMEIDA, M. A. *Bases de datos*. Editorial McGraw Hill, Madrid.

Se pueden encontrar referencias bibliográficas actualizadas sobre bases de datos en:

[http://www.mcgraw-hill.es/McGrawHill/catalogo.nsf/\(INFORMATICA/Bases+de+datos\)?OpenView&CollapseView](http://www.mcgraw-hill.es/McGrawHill/catalogo.nsf/(INFORMATICA/Bases+de+datos)?OpenView&CollapseView)

En la dirección <http://clubs.yahoo.com/clubs/structuredquerylanguage> se encuentra la página principal del CLUB SQL de YAHOO.

A continuación se muestran algunos sitios donde podrá encontrar otros cursos tutoriales sobre SQL y SQL\*PLUS:

<http://w3.one.net/~jhoffman/sqltut.htm>

<http://www.uwp.edu/academic/mis/baldwin/sqlplus.htm>

<http://sirius.cs.ucdavis.edu/teaching/sqltutorial/>

---