

Resumen SQL

Elementos:

- Comando: instrucciones que se pueden crear. Tres tipos:
 - CREATE
 - DROP
 - ALTER
- Manipulación de datos:
 - SELECT: consultar filas que cumplan un criterio determinado.
 - INSERT: cargar datos en una única operación.
 - UPDATE: modificar valores de campos y filas determinados.
 - DELETE: eliminar filas.
- De control y seguridad:
 - GRANT: conceder privilegios.
 - REVOKE: eliminar privilegios.
- Clausulas:
 - FROM: determinar la tabla de la que se van a seleccionar las filas.
 - WHERE: especificar las condiciones (sueldo >2000).
 - ORDER BY: ordenar las filas seleccionadas de una manera específica (ASC|DESC).
 - GROUP BY: agrupar filas con mismos valores (Agrupar los empleados que viven en el mismo pueblo).
 - HAVING: establecer la condición que debe satisfacer cada grupo.

SI HAY HAVING TIENE QUE HABER GROUP BY SÍ O SÍ
- Operadores:
 - AND: dos condiciones (o más), solo si se cumplen ambas.
 - OR
 - NOT: valor contrario a la expresión.
 - BETWEEN
 - LIKE: comparar con un valor entre comillas simples.
 - IN

******%** => para especificar que hay algo en la posición la que este colocada (J%O, hay texto entre la primera J (primera porque no hay % delante de ella) y la última O (última porque no hay % después de ella)

- Funciones:
 - AVG: promedio.
 - COUNT: nº filas sección.
 - SUM: suma de valores de una sección.

Crear bases de datos

Crear una base de datos consiste en crear las tablas que la componen.

```
CREATE DATABASE nombreBD;
```

Crear tablas

```
CREATE TABLE [esquema.]nombreTabla (  
    Columna1 tipoDato [DEFAULT valor][restricciones],  
    ...  
    ColumnaN tipoDato [DEFAULT valor][restricciones]);
```

DEFAULT valor, será un valor por defecto.

Los elementos que estén entre corchetes son opcionales, no es obligatorio que estén.

Ejemplo:

```
CREATE TABLE Proveedores (nombre varchar2(50),  
    localidad DEFAULT 'Palencia');
```

Tipos de datos

- Textos: (se indican los tamaños entre paréntesis tras el del tipo)
 - o VARCHAR2: para textos de longitud variable, máximo 4000.
 - o CHAR: para textos de longitud fija, hasta 2000.
 - o NCHAR: para el almacenamiento de caracteres nacionales de longitud fija.
 - o NVARCHAR2: para el almacenamiento de caracteres nacionales de longitud variable.
- Números:
 - o NUMBER
 - o NUMBER(p,s): número decimal donde, p es la cantidad de números y s es el número de decimales a la derecha de la coma. NUMBER (8,3), permite representar hasta números de 8 cifras de las cuales 3 serían decimales.
 - o NUMBER(p): número entero con p dígitos.
- Fechas y horas:
 - o DATE('día/mes/año')
 - o TIMESTAMP('día/mes/año hora:minutos:segundos')

Restricciones

- CONSTRAINT

```
CREATE TABLE USUARIOS (  
    Login varchar2(15) CONSTRAINT usu_log_pk PRIMARY KEY,  
    password varchar2(8) NOT NULL,  
    fechaIngreso DATE DEFAULT SYSDATE));
```

- NOT NULL: asociado a una columna y se obliga a tener un valor.

```
password varchar2(8) NOT NULL, ó  
password varchar2(8) CONSTRAINT usu_pas_nn NOT NULL
```

- UNIQUE: asociada a una columna y se obliga a que no se puedan repetir valores.

```
CREATE TABLE USUARIOS (  
    login varchar2(15) CONSTRAINT usu_log_uk UNIQUE);  
CREATE TABLE USUARIOS (  
    login varchar2(15) UNIQUE);
```

Si queremos que la restricción englobe a varios campos lo ponemos de la siguiente forma:

```
CREATE TABLE USUARIOS (  
    login varchar2(25),  
    correo varchar2(25),  
    CONSTRAINT usu_uk UNIQUE(login, correo));
```

- PRIMARY KEY:

La clave primaria por defecto ya es NOT NULL y UNIQUE.

```
CREATE TABLE USUARIOS (  
    login varchar2(15) CONSTRAINT usu_log_pk PRIMARY KEY);  
ó  
CREATE TABLE USUARIOS (  
    login varchar2(15) PRIMARY KEY);
```

Si la clave primaria está formada por más de un campo:

```
CREATE TABLE USUARIOS (  
    Nombre varchar2(25),  
    apellidos varchar2(25),  
    fecha_nacimiento DATE,  
    CONSTRAINT usu_uk PRIMARY KEY(nombre, apellidos,  
    fecha_nacimiento));
```

- REFERENCES. FOREIGN KEY:

Ejemplo: creamos una tabla que contiene información de alquileres de películas, el cliente que la alquila y la película alquilada, estos dos atributos son FK de otras dos tablas:

```
CREATE TABLE alquiler (  
    dni varchar2(9) CONSTRAINT alq_dni_fk REFERENCES clientes(dni),  
    cod_pelicula NUMBER(5) CONSTRAINT alq_pel_fk REFERENCES  
    peliculas(cod),  
    CONSTRAINT alq_pel_pk PRIMARY KEY (dni, cod_pelicula);
```

Si el campo al que se hace referencia es la clave primaria, se puede omitir el nombre de este campo:

```
CREATE TABLE alquiler (  
    dni varchar2(9) CONSTRAINT alq_dni_fk REFERENCES clientes,  
    cod_pelicula NUMBER(5) CONSTRAINT alq_pel_fk REFERENCES  
    peliculas,  
    CONSTRAINT alq_pel_pk PRIMARY KEY (dni, cod_pelicula);
```

Cuando las claves ajenas están formadas por más de un campo se pueden definir a nivel de tabla y en este caso será necesario utilizar la expresión FOREIGN KEY:

```
CREATE TABLE existencias (  
    Tipo char2(9),  
    Modelo number(3),  
    N_almacen number(1),  
    Cantidad number(7),  
    CONSTRAINT exi_tm_fk FOREIGN KEY (tipo, modelo) REFERENCES  
    piezas,
```

```

        CONSTRAINT   exi_nal_fk   FOREIGN   KEY   (n_almacen)   REFERENCES
almacenes,
        CONSTRAINT exi_pk PRIMARY KEY (tipo, modelo, n_almacen)
);

```

- CHECK: permite comprobar que los valores introducidos cumplen las condiciones asociadas a esa columna.

Supongamos la tabla USUARIOS, en su campo crédito sólo puede tomar valores entre 0 y 2000, se especificaría:

```

CREATE TABLE USUARIOS (
    credito NUMBER(4) CHECK (credito BETWEEN 0 AND 2000));

```

Una misma columna puede tener varios CHECK asociados, para ello especificamos un CONSTRAINT para cada una de la siguiente forma:

```

CREATE TABLE ingresos (
    cod NUMBER(5) PRIMARY KEY,
    concepto VARCHAR2(50) NOT NULL,
    importe NUMBER (11,2) CONSTRAINT importe_min CHECK (importe > 0)
                                CONSTRAINT importe_max CHECK (importe
<8000));

```

En este ejemplo, se prohíbe añadir datos cuyo importe no esté entre 0 y 8000.

Si queremos hacer referencia en una validación a otras columnas, hay que incluir la restricción a nivel de tabla, independiente de la columna:

```

CREATE TABLE ingresos (
    cod NUMBER(5) PRIMARY KEY,
    concepto VARCHAR2(50) NOT NULL,
    importe_max NUMBER (11,2),
    importe NUMBER (11,2),
    CONSTRAINT importe_max CHECK (importe < importe_max));

```

Eliminar tablas

```

DROP TABLE nombreTabla [CASCADE CONSTRAINTS];

```

También podemos eliminar las filas sin eliminar la estructura de la tabla:

```

TRUNCATE TABLE nombreTabla;

```

Modificar tablas

- Cambiar nombre:

```

RENAME nombreViejo TO nombreNuevo;

```

- Añadir columnas:

```

ALTER TABLE nombreTabla ADD
(columnaNueva1 tipoDatos [propiedades],
[, columnaNueva2 tipoDatos [propiedades]] ... );

```

- Eliminar columnas:

```
ALTER TABLE nombreTabla DROP COLUMN columna1 [, columna2, ...];
```

- Modificar columnas:

```
ALTER TABLE nombreTabla MODIFY (columna1 tipoDato [propiedades] [, columna2 ...]);
```

- Renombrar columnas:

```
ALTER TABLE nombreTabla RENAME COLUMN nombreAntiguo TO nombreNuevo;
```

- Borrar restricciones:

```
ALTER TABLE nombreTabla DROP CONSTRAINT nombreRestricción;
```

- Modificar el nombre de una restricción:

```
ALTER TABLE nombreTabla RENAME CONSTRAINT nombreViejo TO nombreNuevo;
```

- Activar o desactivar restricciones:

```
ALTER TABLE nombreTabla DISABLE CONSTRAINT nombreRestricción  
[CASCADE];
```

La opción CASCADE desactiva las restricciones que dependan de ésta.

Para activar de nuevo la restricción:

```
ALTER TABLE nombreTabla ENABLE CONSTRAINT nombreRestricción [CASCADE];
```

Lenguaje de control de datos

- Crear usuarios:

```
CREATE USER nombreUsuario  
IDENTIFIED BY contraseña  
QUOTA n°(K/M) ON tablespace(users/system); //Limite de espacio que  
tiene el usuario para hacer tablas  
QUOTA UNLIMITED ON tablespace(users/system); //Espacio ilimitado
```

- Modificar usuarios:

```
ALTER USER nombreUsuario  
IDENTIFIED BY contraseña  
[DEFAULT TABLESPACE tablespace]  
[TEMPORARY TABLESPACE tablespace]  
[QUOTA int {k | M} ON tablespace]  
[QUOTA UNLIMITED ON tablespace]
```

- Eliminar usuarios:

```
DROP USER nombreUsuario [CASCADE];
```

Permisos

```
GRANT {privilegioObjeto [,privilegioObjeto]... |ALL [PRIVILEGES]}  
ON [usuario.]objeto  
TO {usuario1 | rol1 | PUBLIC} [,usuario2 | rol2 | PUBLIC] ...  
[WITH GRANT OPTION]
```

```
[WITH ADMIN OPTION] ;
```

Donde:

- ON , especifica la tabla sobre la que se conceden privilegios (verduras)
- TO, señala a los usuarios o roles a los que se conceden
- ALL, concede todos los privilegios sobre el objeto especificado
- [WITH GRANT OPTION], permite que el receptor del privilegio se lo asigne a otro
- [WITH ADMIN OPTION], se usa cuando se conceden privilegios de sistema y este pueda concederselos a otros usuarios
- PUBLIC, hace que el privilegio esté disponible para todos los usuarios

*Privilegios sobre tablas:

- ❖ Select
- ❖ Insert
- ❖ Update

*Privilegios de sistema:

- ❖ Create session
- ❖ Create table
- ❖ Create user
- ❖ Drop user

*Privilegios sobre una columna:

```
GRANT select/update/insert (nombreColumna) on nombreTabla  
TO usuario/public;
```

*Quitar privilegios:

Solo se pueden quitar los privilegios que se han dado.

```
REVOKE nombrePrivilegio/ALL privileges  
FROM usuario;
```

*Crear tablas:

```
SQL> connect obiwan1
Enter password:
Connected.
SQL> edit equipo1

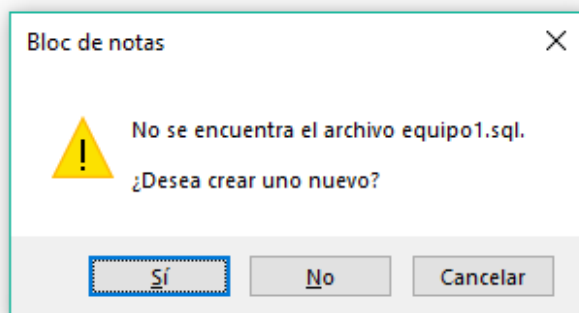
SQL> @equipo1

Table created.

SQL>
```

Intento editar con obiwan1 la tabla equipo1, como no existe me sale esta advertencia (imagen de abajo) le doy a Si, creo la tabla, guardo y cierro.

Luego pongo @equipo y ya esta



```
SQL> create user usu2
2 identified by usu2
3 quota 100k on users;

User created.
```

```
SQL> create role rol1;

Role created.
```

```
SQL> grant select on verduras to rol1;

Grant succeeded.
```

```
SQL> grant rol1 to usu1;

Grant succeeded.
```

```
SQL> grant create session to rol1;

Grant succeeded.
```

```
SQL> grant select on verduras to usu4;
Grant succeeded.

SQL> grant insert on actores to usu4;
Grant succeeded.

SQL> grant update on peliculas to usu4;
Grant succeeded.
```

```
SQL> grant create session to usu4
2   with admin option
3   ;

Grant succeeded.
```

CREAR USUARIOS

CREAR TABLAS

MODIFICAR TABLAS

DAR ROLES Y PERMISOS

UNIDAD 5

Consultas

CADENAS

INITCAP(cad) -> devuelve la cadena cad con la primera letra en mayúsculas.

LPAD(cad1, n, cad2) -> devuelve la cadena cad1 de longitud n rellena por la izquierda con cad2. LPAD('M', 5, '*') : *****

RPAD(cad1, n, cad2) -> devuelve la cadena cad1 de longitud n rellena por la derecha con cad2. RPAD('M', 5, '*') : M*****

REPLACE(cad, ant, nue) -> reemplaza en cad las cadenas ant por las nue.

LENGTH(cad) -> longitud de cadena.

FECHAS

SYSDATE

SYSTIMESTAMP -> fecha y hora actuales

MONTHS_BETWEEN(fecha1, fecha2)

LAST_DAY(fecha) -> último día del mes al que pertenece fecha

EXTRACT(valor FROM fecha)

OTRAS

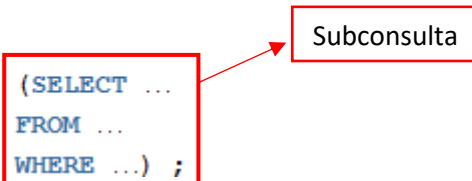
DECODE (expr1, cond1, valor1 [,cond2 ,valor2, ...] default) -> Esta función evalúa una expresión **expr1**, si se cumple la primera condición **cond1** devuelve el valor **valor1**, en caso contrario se pasa a la siguiente condición y así hasta que se cumpla una de ellas o se devuelve un valor por defecto expresado por **default** si no se cumple ninguna. Es como el if en programación.

UNIDAD 6

Subconsultas

Primero se realiza la subconsulta, y con el resultado de esta, se realiza la consulta inicial.

```
SELECT ...  
FROM ...  
WHERE columna condiciónBúsqueda (SELECT ...  
FROM ...  
WHERE ...) ;
```



Ejemplo: obtener el APELLIDO de los empleados con el mismo oficio que 'GIL'.

```
SELECT APELLIDO  
FROM EMPLE  
WHERE OFICIO = (SELECT OFICIO  
FROM EMPLE  
WHERE APELLIDO = 'GIL');
```

CONDICIONES DE BÚSQUEDA

Test de comparación en subconsultas (>, <, <>, <=, >=, =). Compara el valor de una expresión con un valor único producido por una subconsulta.

Test de pertenencia a un conjunto devuelto por una subconsulta (IN). Ejemplo: obtener los apellidos de los empleados cuyo oficio sea alguno de los oficios que hay en el departamento 20.

```
SELECT APELLIDO FROM EMPLE  
WHERE OFICIO IN (SELECT OFICIO FROM EMPLE  
WHERE DEPTNO=20);
```

Test de existencia (EXISTS, NOT EXISTS). Examina si una subconsulta produce alguna fila de resultados. El test será TRUE si devuelve filas y FALSE en caso contrario. Visualizar los APELLIDOS de los empleados si existe alguno llamado 'ARROYO'. Se utilizará la cláusula NOT EXISTS si queremos comprobar si la subconsulta no devuelve ninguna fila.

```
SELECT APELLIDO FROM EMPLE
WHERE EXISTS (SELECT * FROM EMPLE
              WHERE APELLIDO LIKE '%ARROYO%');
```

Test de comparación cuantificada (ANY y ALL). Se utilizan junto a los operadores de comparación (>, <, <>, <=, >=, =).

ANY compara el valor de una expresión con cada uno del conjunto de valores devuelto por la subconsulta, si alguna de las comparaciones individuales da resultado true, ANY devolverá true, si la subconsulta devuelve false, ANY devolverá false. Ejemplo, obtener los datos de los empleados cuyo salario sea igual a algún salario de los empleados del departamento 30.

```
SELECT * FROM EMPLE
WHERE SALARIO = ANY (SELECT SALARIO FROM EMPLE
                    WHERE DEPTNO=30);
```

ALL compara el valor de una expresión con cada uno del conjunto de valores devuelto por la subconsulta, si todas las comparaciones individuales da como resultado true, ALL devuelve true, en caso contrario devuelve false. Ejemplo: obtener los datos de los empleados cuyo salario sea menor a cualquier salario de los empleados del departamento 30.

```
SELECT * FROM EMPLE
WHERE SALARIO < ALL (SELECT SALARIO FROM EMPLE
                    WHERE DEPTNO = 30);
```

GROUP BY

Para averiguar el salario medio de cada departamento necesitaremos realizar un agrupamiento por departamento y utilizaremos la cláusula **GROUP BY**. La consulta quedaría de la siguiente forma:

```
select deptno, avg(salario)
from emple
group by deptno;
```

```
WHERE Selecciona las filas
GROUP BY Agrupa estas filas
HAVING Filtra los grupos
ORDER BY Clasifica la salida, ordena los grupos
```

Los datos seleccionados en la sentencia SELECT que lleva el GROUP BY deben ser: una constante, una función de grupo (SUM, AVG, COUNT, ...), una columna expresada en el GROUP BY.

Unión, Intersección y Diferencia

OPERADOR UNION

Este operador combina resultados de dos consultas. Permite añadir el resultado de un SELECT a otro SELECT. Las filas duplicadas que aparecen se reducen a una única fila.

Ejemplo:

```
SELECT nombre FROM provincias
UNION
SELECT nombre FROM comunidades
```

El resultado será una tabla que contendrá nombres de provincias y de comunidades. Se crea una sola tabla con registros incluidos en cualquiera de las consultas. Se pueden mostrar los registros duplicados especificando **UNION ALL**.

OPERADOR INTERSECT

Devuelve las filas que son iguales en ambas consultas. Todas las filas duplicadas serán eliminadas antes de la generación del resultado final.

Ejemplo: visualizar los tipos y modelos de piezas que se encuentran en los almacenes 1 y 2.

```
SELECT tipo,modelo FROM existencias WHERE n_almacen=1
INTERSECT
SELECT tipo,modelo FROM existencias WHERE n_almacen=2
```

OPERADOR MINUS

Devuelve las filas que estén en la primera SELECT y no en la segunda. Las filas duplicadas del primer conjunto se reducirán a una única fila antes de que empiece la comparación con el otro conjunto.

Ejemplo: tipos y modelos de piezas que se encuentren en el almacén 1 y no en el 2.

```
SELECT tipo,modelo FROM existencias WHERE n_almacen=1
MINUS
SELECT tipo,modelo FROM existencias WHERE n_almacen=2
```

Ejemplos:

Obtener los nombres y ciudades de todos los proveedores y clientes de Alemania

```
SELECT nombrePro, ciudad FROM PROVEEDORES WHERE pais = 'Alemania'
UNION
SELECT nombreCli, ciudad FROM CLIENTES WHERE pais= 'Alemania';
```

Una academia de idiomas da clases de inglés, francés y portugués y guarda los datos de los alumnos en tres tablas cuyo nombre coincide con el idioma. Averiguar el nombre y domicilio de todos los alumnos que cursan los tres idiomas para enviarles información sobre los exámenes.

```
SELECT nombre, domicilio FROM ingles
INTERSECT
SELECT nombre, domicilio FROM frances
INTERSECT
SELECT nombre, domicilio FROM portugues;
```

Averiguar los nombres y domicilios de los alumnos que cursan inglés y no portugués.

```
SELECT nombre, domicilio FROM ingles
MINUS
SELECT nombre, domicilio FROM portugues;
```

UNIDAD 7

Insert, Update y Delete

ORDEN INSERT

```
INSERT INTO HISTORIAL_LABORAL VALUES (13893893, 34567, '01/01/1987', NULL, 4, 51520002);
INSERT INTO HISTORIAL_LABORAL VALUES (51520002, 15280, NULL, NULL, 30, 13893893);
```

ORDEN UPDATE

```
UPDATE EMPL SET SALARIO = SALARIO + 100,
COMISION = COMISION + 10 WHERE DEPTNO = 10;
```

A partir de la tabla EMPL, cambia el salario a la mitad y la comisión a 0, a aquellos empleados que pertenezcan al departamento con mayor número de empleados.

```
SQL> UPDATE EMPL SET SALARIO = SALARIO/2, COMISION = 0 WHERE DEPT_NO =
(SELECT DEPT_NO FROM EMPL GROUP BY DEPT_NO HAVING COUNT(*) =
(SELECT MAX(COUNT(*)) FROM EMPL GROUP BY DEPT_NO));
```

Para todos los empleados de la tabla EMPL y del departamento de 'CONTABILIDAD', cambiamos su salario al doble del salario de 'SÁNCHEZ' y su apellido, a minúscula.

```
SQL> UPDATE EMPL SET APELLIDO = LOWER(APELLIDO),
SALARIO = (SELECT SALARIO*2 FROM EMPL WHERE APELLIDO = 'SANCHEZ')
WHERE DEPT_NO = (SELECT DEPT_NO FROM DEPART
WHERE Dnombre = 'CONTABILIDAD');
```

ORDEN DELETE

Borramos todas las filas de la tabla CENTROS: SQL> DELETE FROM CENTROS;

Igualmente, podríamos haber puesto: DELETE CENTROS;

Borramos todas las filas de la tabla LIBRERIA cuyos EJEMPLARES no superen la media de ejemplares en su ESTANTE:

```
SQL> DELETE FROM LIBRERIA L WHERE EJEMPLARES <
(SELECT AVG(EJEMPLARES) FROM LIBRERIA WHERE ESTANTE = L.ESTANTE
GROUP BY ESTANTE);
```

Borramos los departamentos de la tabla DEPART con menos de cuatro empleados.

```
SQL> DELETE FROM DEPART WHERE DEPT_NO IN
(SELECT DEPT_NO FROM EMPL GROUP BY DEPT_NO HAVING COUNT(*) < 4);
```

Commit, Autocommit y Rollback

ORDEN COMMIT

Validar cambios realizados en la base de datos:

```
commit;
```

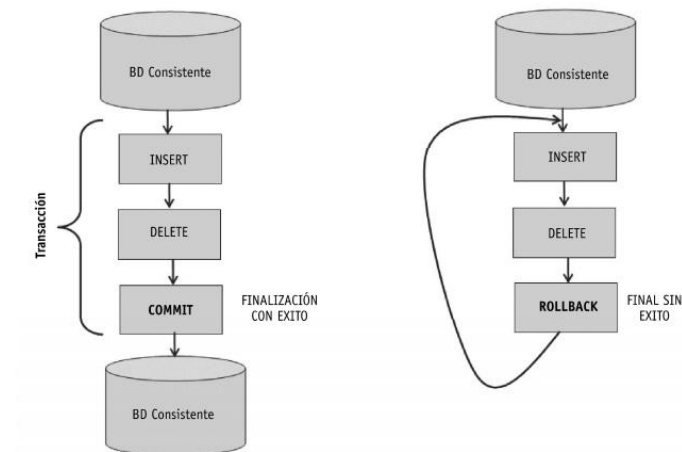
ORDEN AUTOCOMMIT

Validar automáticamente sin necesidad de poner commit todo el rato. Para ver el estado de auto commit, pondremos "SHOW AUTOCOMMIT;" y para activarlo, pondremos "SET AUTOCOMMIT ON;"

ORDEN ROLLBACK

Ignora los cambios hechos después del último commit:

rollback;



Creación de tablas a partir de otras

Para crear la tabla **EJEMPLO_AS** a partir de los datos de la tabla **EJEMPLO** se procede así:

```
CREATE TABLE EJEMPLO_AS  
AS SELECT * FROM EJEMPLO;
```

La tabla se crea con los mismos nombres de columnas e idéntico contenido de filas que la tabla **EJEMPLO**.

En el siguiente ejemplo, asignamos un nombre a las columnas de la tabla **EJEMPLO_AS2** (COL1, COL2, COL3 y COL4):

```
CREATE TABLE EJEMPLO_AS2 (COL1, COL2, COL3, COL4)  
AS SELECT * FROM EJEMPLO;
```

El contenido de la tabla es el mismo que el que tiene la tabla **EJEMPLO**.

Se crea la tabla **EMPLOYDEPART** a partir de las tablas **EMPLE** y **DEPART**. Esta tabla contendrá el apellido y el nombre del departamento de cada empleado:

```
CREATE TABLE EMPLOYDEPART  
AS SELECT APELLIDO, DNOMBRE  
FROM EMPL, DEPART WHERE EMPL.DEPT_NO = DEPART.DEPT_NO;
```

```
CREATE TABLE COPIAEMPLE  
AS SELECT * FROM EMPL;
```

Creación de vistas

Sirven para obtener los datos de una consulta compleja. Si se suprime una tabla la vista se invalida.

ES ACONSEJABLE PONER SIEMPRE CREATE OR REPLACE VIEW

```
CREATE VIEW DEP30  
AS SELECT APELLIDO, OFICIO, SALARIO FROM EMPL  
WHERE DEPTNO = 30;
```

```
SELECT * FROM DEP30;
```

Visualizamos la
vista

```
CREATE TABLE EMPLE20 (nombre, salario, oficio)  
AS SELECT APELLIDO, salario, oficio FROM EMPL  
WHERE DEPTNO = 20;
```

Ponemos nombre
nuevo a las columnas

INSERTAR VALORES EN VISTAS

```
INSERT INTO DEP10 VALUES (1010, 'PEPE', 'EMPERADOR', 7369, SYSDATE, 2000, NULL, 20);
```

OPCIONES DE CREACIÓN DE VISTA

WITH CHECK OPTION

Antes de insertar una nueva fila o de actualizar la vista, se chequea que se cumplen las condiciones de la vista:

```
CREATE OR REPLACE VIEW DEP10  
AS SELECT * FROM EMPL  
WHERE DEPTNO = 10 AND SALARIO > 1200  
WITH CHECK OPTION;
```

WITH READ ONLY

Solo nos permite hacer SELECT, no podremos modificarla:

```
CREATE OR REPLACE VIEW ANTIGUOS (CODIGO, NOMBRE, SALARIO)  
AS SELECT EMPNO, APELLIDO, SALARIO  
FROM EMPL  
WHERE TRUNC((SYSDATE-FECHAALT))/365 > 5  
with read only;
```

VISTAS COMPLEJAS

Son las que se crean a partir de mas de una tabla

```
CREATE OR REPLACE VIEW VMEDIA  
AS SELECT E.DEPTNO, D.DNOMBRE, AVG(SALARIO) MEDIA, MAX(SALARIO) MAXIMO  
FROM EMPL E, DEPART D  
WHERE E.DEPTNO = D.DEPTNO  
GROUP BY E.DEPTNO, D.DNOMBRE;
```

Estas vistas no se podrán modificar.

ELIMINAR VISTAS

Para eliminar vistas utilizaremos el siguiente comando:

```
DROP VIEW DEP10;
```

Creación de secuencias

Sirven para generar números automáticamente.

Se crea una tabla llamada **FRUTAS** con dos columnas: **CODIGO** y **NOMBRE**. La columna **CODIGO** se define como clave primaria:

```
CREATE TABLE FRUTAS (CODIGO NUMBER(2) NOT NULL PRIMARY KEY, NOMBRE VARCHAR2(15));
```

Ahora se crea una secuencia llamada **CODIGOS** que generará números empezando por el valor 1, con incremento 1 y cuyo máximo valor para la secuencia será 99: **CREATE SEQUENCE** CODIGOS START WITH 1 INCREMENT BY 1 MAX-VALUE 99;

Se insertan filas en la tabla **FRUTAS** usando la secuencia **CODIGOS** para generar el **CODIGO** de cada fila de la tabla:

```
INSERT INTO FRUTAS VALUES (CODIGOS.NEXTVAL, 'MANZANAS');  
INSERT INTO FRUTAS VALUES (CODIGOS.NEXTVAL, 'NARANJAS');  
INSERT INTO FRUTAS VALUES (CODIGOS.NEXTVAL, 'PERAS');
```

Para consultar el valor actual de la secuencia escribimos: **SELECT CODIGOS.CURRVAL FROM DUAL;**