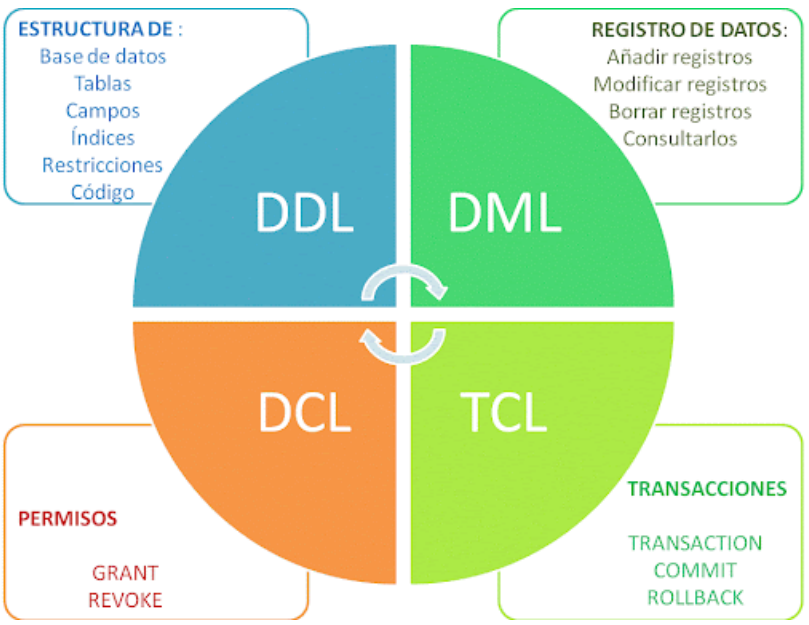
3.6. Lenguaje de definición de datos

SQL (***Structured Query Language***, lenguaje de consulta estructurado) es un lenguaje diseñado para administrar y recuperar información de SGBD relacionales.
SQL está compuesto por 4 tipos de lenguajes:

Lenguaje de definición de datos (DDL): Este lenguaje permite crear y modificar la estructura de una base de datos, los comandos principales son CREATE, ALTER, DROP
Lenguaje de manipulación de datos (DML): Permite recuperar, almacenar, modificar, eliminar, insertar y actualizar datos de una base de datos. Sus comandos principales son SELECT, INSERT, UPDATE, DELETE.
Lenguaje de control de datos (DCL): Permite crear roles, permisos e integridad referencial, así como el control al acceso a la **base de datos**. Los comandos principales son REVOKE, GRANT, CREATE USER, CREATE ROLE y CREATE PROFILE
Lenguaje de control de transacciones (TCL): se utiliza para controlar el procesamiento de transacciones en una base de datos. Los comandos básicos son COMMIT, ROLLBACK y SAVEPOINT.



3.7 Lenguaje de definición de datos (DDL)

El **lenguaje de definición de datos** se ocupa de la estructura de la base de datos donde se almacenarán los datos. **No se ocupa de los datos en sí**. Las operaciones básicas son **crear, modificar y eliminar**.

3.7.1. Manipulación de Bases de datos (NO ORACLE)

El concepto DATABASE o SCHEMA está presente en todos los gestores de BD, aunque no en todos los gestores se gestionan igual.

Los SCHEMAS o DATABASES me permiten organizar/agrupar de forma lógica o funcional el contenido de mis BBDDs. Dentro de un Gestor de BBDD podré tener tantos schemas o databases como sea necesario y podré gestionar la accesibilidad y visibilidad de unos schemas sobre los otros (a priori se crean como compartimentos independientes).

En ORACLE los schemas están asociados a usuarios y se crean cuando creas un usuario (lo uses o no). Tiene cierta similitud con la nube de Google (el drive de Google). Se genera con el usuario de correo, tu puedes tener distintos usuarios que no se ven entre si donde te organices el contenido que tienes y puedes compartir información entre los distintos drives.

Crear una base de datos

El comando **CREATE** se usa para **crear** una nueva base de datos, una nueva tabla, índices, restricciones, etc...

Sintaxis

```
CREATE {DATABASE | SCHEMA} nombreBBDD;
```

Los términos *DATABASE* y *SCHEMA* son sinónimos, podemos utilizar cualquiera de los dos para hacer referencia a la base de datos.
Si existiera una base de datos con ese nombre, el SGBD no nos permitiría crearla, por ello, para asegurarnos que la BBDD se crea y que no existe una con el mismo nombre usaremos la sentencia '**IF NOT EXISTS**':

Sintaxis

```
CREATE DATABASE IF NOT EXISTS nombreBBDD
```

También podemos crearla definiendo la codificación con *'CHARACTER SET'* (*set de caracteres*, es un conjunto de símbolos y codificaciones, es decir, la forma en que la base de datos guarda internamente los datos) y *'COLLATE'* (especifica el tipo de cotejamiento que vamos a utilizar en la base de datos, es decir, el criterio que vamos a seguir para ordenar las cadenas de caracteres.). Si no se especifica el set de caracteres en la creación de la base de datos, se usará latin1 por defecto. Las principales bases de datos en producción utilizan el set de caracteres utf8 o utf8mb4.

Si no especificamos ningún cotejamiento se usará el que tenga asignado por defecto el set de caracteres escogido. Por ejemplo, para el set de caracteres utf8 se usa utf8_general_ci.

Sintaxis

```
CREATE DATABASE IF NOT EXISTS nombreBBDD CHARACTER SET
juego_caracteres COLLATE nombre_cotejamiento;
```

Ejemplo

```
CREATE DATABASE IF NOT EXISTS miBaseDatos CHARACTER SET utf8mb4
COLLATE utf8mb4_general_ci;
```

En la siguiente tabla podemos ver algunas consultas sobre set de caracteres y cotejamiento:

Sentencia	
SHOW CHARACTER SET	Devuelve los juegos de caracteres que tenemos disponibles
SHOW COLLATION	Devuelve los tipos de cotejamiento disponibles
SHOW COLLATION LIKE 'utf8%'	Devuelve los tipos de cotejamiento que podemos usar con utf8

`USE database;`
`SELECT`
`@@character_set_database,`
`@@collation_database;`
	(USE ...)Seleccionamos la BBDD con la que vamos a trabajar y luego consultamos el valor de las variables @@character_set_database, @@collation_database; que nos dará el juego de caracteres y el cotejamiento que estamos utilizando en esa BBDD

El cotejamiento puede ser:

case_sensitive (_cs) → distingue las minúsculas y las mayúsculas (a y A son diferentes)
case_insensitive (_ci) → no distingue las minúsculas y las mayúsculas (a y A son iguales)
binary (_bin) → dos caracteres son iguales si su representación numérica lo es.

Eliminar una base de datos

Eliminar una base de datos implica borrar sus tablas y todo su contenido, por tanto debemos estar seguros de esta decisión. Utilizaremos el comando **DROP**

Sintaxis

```
DROP {DATABASE | SCHEMA} [IF EXISTS] nombreBBDD;
```

Modificar una base de datos

Para modificar las características de una base de datos utilizaremos el comando **ALTER**

Sintaxis

```
ALTER {DATABASE | SCHEMA} nombreBBDD
[especificaciones]
```

Ejemplo

```
ALTER DATABASE nombreBBDD CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci;
```

Consultar listado de bases de datos disponibles

Para ver un listado de las bases de datos a las que tiene acceso el usuario con el que hemos conectado a MySQL, utilizaremos el comando **SHOW:**

Sintaxis

```
SHOW DATABASES;
```

Seleccionar una base de datos

Si queremos seleccionar la base de datos con la que queremos trabajar:

Sintaxis

```
USE nombreBBDD;
```

3.7.2. Manipulación de tablas

Crear una tabla

Podemos dividir el proceso de crear una nueva tabla en 3 partes:

Definición de las columnas
Definición de las claves primarias y las ajenas (si las hay)

Restricciones de los datos

Sintaxis

```
CREATE TABLE nombre_tabla
(definición de las columnas,
definición de claves,
restricciones);
```

Definición de las columnas

Sintaxis

```
CREATE TABLE nombre_tabla
(nombreColumna1 tipoDato,
nombreColumna2 tipoDato,
…
nombreColumnaN tipoDato);
```

En la creación de la tabla estamos definiendo:

El **orden** de las columnas, que será el que determine el orden de izquierda a derecha de las columnas en la tabla.

El **nombre** de la columna, que es el que utilizaremos para referirnos a ella en las sentencias SQL. Cada columna de la tabla debe tener un nombre único, aunque sí puede repetirse en tablas distintas.

El **dominio** de los datos de la columna, que determina la clase de datos que la columna almacena.

Si la columna no puede contener **datos nulos**, por defecto se permiten valores nulos en la columna. El uso de la cláusula NOT NULL impide que aparezcan valores NULL en la columna.

Ejemplo

```
CREATE TABLE OFICINAS
(CodOficina NUMBER(6) NOT NULL,
 Ciudad VARCHAR2(15) NOT NULL,
Region VARCHAR2(10) NOT NULL,
Objetivo NUMBER(6),
Ventas NUMBER(6) NOT NULL);
```

Definición de las claves primarias y claves ajenas

Además de la definición de las columnas de una tabla, debemos especificar qué columna o columnas forman la clave primaria y si hay alguna que sea clave ajena, deberemos decir qué columnas son y de qué tabla son clave primaria. Para esto usaremos las cláusulas *PRIMARY KEY y FOREING KEY*. Hay que tener en cuenta que, si especificamos que una columna es clave primaria, mediante la cláusula *PRIMARY KEY*, deberemos asegurar también que el valor de dicha columna sea único y que exista, por lo que deberemos haber especificado que la columna es *NOT NULL*.

Sintaxis

```
CONSTRAINT FK_tablaActual_tablaReferenciada
```

```
FOREIGN KEY(columna1,columna2,...columnaN) REFERENCES
tablaReferenciada(columna1,columna2,…,columnaN);
```

Veamos un ejemplo:

Ejemplo	

Notación PK con una columna junto a la definición de la columna	`CREATE TABLE Coche`
	`(`
	`Matricula VARCHAR2 (7) NOT NULL PRIMARY KEY,`
	`Marca VARCHAR2 (20)  NOT NULL ,`
	`Modelo VARCHAR2 (20)  NOT NULL ,`
	`Precio NUMBER (7,2)  NOT NULL`
	`);`

Notación PK con una columna al final de la definición de las columnas	`CREATE TABLE Coche2`
	`(`
	`Matricula VARCHAR2(7)  NOT NULL ,`
	`Marca VARCHAR2(20)  NOT NULL ,`
	`Modelo VARCHAR2(20)  NOT NULL ,`
	`Precio NUMBER(7,2)  NOT NULL ,`
	`PRIMARY KEY (Matricula)`
	`);`

Notación PK con 2 columnas	`CREATE TABLE EjemplarLibro`
	`(`
	`ISBN varchar2 (13)  NOT NULL ,`
	`NumEjemplar NUMBER(9)  NOT NULL ,`
	`Titulo varchar2(50)  NOT NULL ,`
	`PRIMARY KEY(ISBN,NumEjemplar)`
	`);`

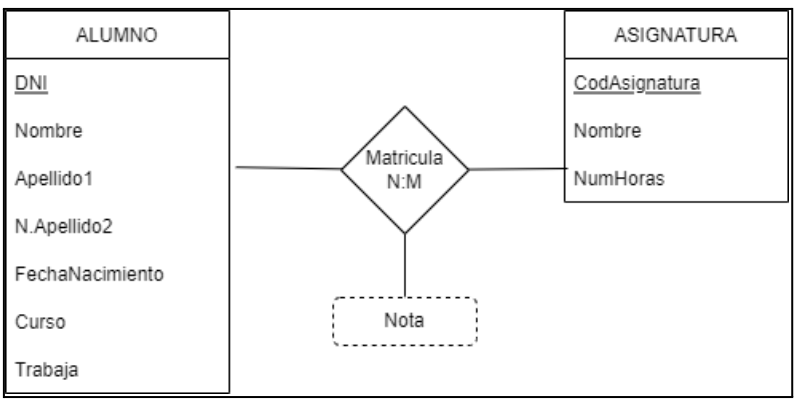
Notación FK	`CREATE TABLE Cliente`
	`( DNI VARCHAR2(9) NOT NULL PRIMARY KEY,`
	`Nombre VARCHAR2(50) NOT NULL,`
	`Apellido1 VARCHAR2(50) NOT NULL,`
	`Apellido2 VARCHAR2(50)`
	`);`
	`CREATE TABLE Mascota`
	`( Codigo NUMBER(6) PRIMARY KEY,`
	`Nombre VARCHAR2(50),`
	`Raza VARCHAR2(25),`
	`DNICliente VARCHAR2(9),`
	`CONSTRAINT FK_DNI_Cliente_Mascota`
	`FOREIGN KEY (DNICliente) REFERENCES Cliente(DNI)`
	`);`

Cuando el SGBD procesa la sentencia *CREATE TABLE*, compara cada definición de clave ajena con la clave primaria a la que referencia, asegurándose que la ambas concuerden en el número de columnas que contienen y en sus tipos de datos. La tabla referenciada debe estar ya definida en la base de datos para que esta comparación tenga éxito. Si dos o más tablas se referencian mutuamente, no se puede definir la clave ajena de la tabla que se cree en primer lugar, ya que la tabla referenciada todavía no existe.

En su lugar debe crearse la tabla sin definición de clave ajena y añadirle la clave ajena posteriormente utilizando la sentencia *ALTER TABLE* para modificar la estructura de la tabla.

Ejercicio

Vamos a crear la base de datos que representa el siguiente modelo conceptual:



Restricciones de los datos

Podemos aplicar diferentes tipos de restricciones a las columnas:

CHECK

Si queremos indicar que los datos que contienen nuestras columnas se encuentren dentro de un rango establecido o que pertenecen a un conjunto determinado, lo haremos añadiendo a la sentencia CREATE TABLE una cláusula **CHECK** sobre la columna o columnas en cuestión, indicando qué condición se debe cumplir para poder insertar un registro en dicha tabla:

Sintaxis

```
CONSTRAINT ck_NombreColumna CHECK (condición aplicada a la columna)
```

En la condición pueden emplearse:
- Operadores lógicos relacionales (menor, mayor,...)
- Operadores lógicos booleanos (and, or, not...)

Ejemplo

Supongamos que en la tabla Coche no permitimos precios menores o iguales a 0:

```
CREATE TABLE Coche3
(
Matricula VARCHAR2(7) NOT NULL PRIMARY KEY,
Marca VARCHAR2(20)  NOT NULL ,
Modelo VARCHAR2(20)  NOT NULL ,
Precio NUMBER(7,2)  NOT NULL,
```

```
CONSTRAINT ck_Precio CHECK (Precio>0)
);
```

NOT NULL o NULL

Indica si la columna permite valores nulos o no.

DEFAULT

Permite indicar un valor inicial por defecto si no especificamos ninguno en la inserción.

Sintaxis	
DEFAULT(literal|función|NULL)	
FUNCIÓN	DESCRIPCIÓN
Sysdate	Fecha actual y hora actuales

Ejemplo

Supongamos que en la tabla Coche queremos poner precio=0 si no se especifica un valor:

```
CREATE TABLE Coche4
(
Matricula VARCHAR2(7) NOT NULL PRIMARY KEY,
Marca VARCHAR2(20)  NOT NULL ,
Modelo VARCHAR2(20)  NOT NULL ,
Precio NUMBER(7,2)  DEFAULT 0.00,
Fecha Date DEFAULT sysdate
);
```

AUTO_INCREMENT (NO ORACLE)

Sirve para indicar que la columna es autonumérica. Su valor se incrementa automáticamente en cada inserción de una fila. Sólo se utiliza en campos de tipo entero.

MySQL utiliza el tipo de dato autoincremental. PostGreSQL hace lo mismo con un tipo de dato que es Bigserial. En Oracle para datos auto incrementales se usan otros objetos de BBDD llamados secuencias que veremos más adelante.

Ejemplo

```
CREATE TABLE Clientes
(
CodCliente INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,
Nombre Varchar(20) NOT NULL,
Apellido1 Varchar(20) NOT NULL,
Apellido2 Varchar(20),
Email Varchar(150) NOT NULL,
Telefono Varchar(10)
);
```

UNIQUE

Lo utilizaremos si queremos indicar que el valor de una columna es único y no puede haber dos valores iguales en la misma columna.

	Ejemplo

Notación 1	Supongamos que en la tabla Clientes queremos indicar que el email debe ser único:
	`CREATE TABLE Clientes`
	`(`
	`CodCliente NUMBER(6) NOT NULL PRIMARY KEY,`
	`Nombre Varchar2(20) NOT NULL,`
	`Apellido1 Varchar2(20) NOT NULL,`
	`Apellido2 Varchar2(20),`
	`Email Varchar2(150),`
	`Telefono Varchar2(10),`
	`UNIQUE (Email)`
	`);`

Notación 2	`CREATE TABLE Clientes2`
	`(`
	`CodCliente NUMBER(6) NOT NULL PRIMARY KEY,`
	`Nombre Varchar2(20) NOT NULL,`
	`Apellido1 Varchar2(20) NOT NULL,`
	`Apellido2 Varchar2(20),`
	`Email Varchar2(150) UNIQUE,`
	`Telefono Varchar2(10)`
	`);`

Opciones en la declaración de claves ajenas (FOREIGN KEY)

Al eliminar o modificar una clave primaria, debemos definir qué hacer con las claves foráneas que la referencian. Las opciones que podemos indicar son:

NO ACTION: es la opción por defecto de Oracle (y creo que en todos los gestores de BBDD en realidad) e impide que se puedan actualizar o eliminar las filas que tienen valores referenciados por claves ajenas.

CASCADE: permite actualizar* o eliminar las filas que tienen valores referenciados por claves ajenas.
(*) La opción de eliminar está en todos los GBD que conozco. La opción de que el actualizar una PK lo haga en cascada con las FKs que haya ORACLE no la tiene, sin embargo MySQL o PostgreSQL si la tienen.

SET NULL: Asigna valor *NULL* al campo de las filas que tienen referenciadas claves ajenas. Para poder aplicarla, no debemos tener en las columnas que son clave ajena la restricción *NOT NULL.*

Ejemplo

```
CREATE TABLE Genero
(
CodGenero NUMBER(6) PRIMARY KEY,
Nombre Varchar2(50) NOT NULL
);

CREATE TABLE Pelicula
(
CodPelicula NUMBER(6) PRIMARY KEY,
Titulo Varchar2(50) NOT NULL,
Duracion NUMBER(6) NOT NULL,
CodGenero NUMBER(6),
constraint FK_CodGenero
FOREIGN KEY (CodGenero) REFERENCES Genero(CodGenero)
ON DELETE SET NULL
);
```

Si rellenamos las tablas:

Genero	
CodGenero	Nombre
1	Ciencia-ficcion
2	Comedia
3	Animación

Pelicula				
Cod Pelicula	Titulo	Duracion	Cod Genero	
1	Matrix Resurrections	148	1	
2	Spider-man: No way home	148	1	
3	Sing-2	60	3	
4	Encanto	60	3	

En este caso podremos eliminar Ciencia-ficción porque el GBD asignará a estas 2 películas el valor de null en el campo de género.

Sin embargo si ponemos:

```
CREATE TABLE Pelicula2
(
CodPelicula NUMBER(6) PRIMARY KEY,
Titulo Varchar2(50) NOT NULL,
Duracion NUMBER(6) NOT NULL,
CodGenero NUMBER(6),
```

```
constraint FK_CodGenero2
FOREIGN KEY (CodGenero) REFERENCES Genero(CodGenero)
ON DELETE CASCADE
);
```

Si eliminamos Ciencia-ficción, también eliminaremos las películas 1 y 2 porque son las que hacen referencia a este género.

Eliminar una tabla

Utilizando la sentencia *DROP TABLE* podemos eliminar rápidamente una tabla de una base de datos siempre que tengamos permisos para hacerlo:

Sintaxis

```
DROP TABLE nombre_Tabla;
```

Modificar una tabla

Muchas veces es necesario modificar las propiedades de una tabla, agregar o eliminar columnas, crear o eliminar índices, modificar el tipo de columnas existentes, renombrar columnas o renombrar la propia tabla.
Si la tabla no tiene datos podemos eliminar la tabla y volver a crearla, pero si la tabla ya contiene datos tenemos que utilizar la sentencia **ALTER TABLE.**

Si queremos **cambiar el nombre** de la tabla:

Sintaxis

```
ALTER TABLE nombre_Tabla RENAME nombre_nuevo_Tabla;
```

Si queremos **eliminar una columna** de la tabla:

Sintaxis

```
ALTER TABLE nombre_Tabla DROP COLUMN nombre_Columna;
```

Eliminar varias columnas de la tabla:

Sintaxis

```
ALTER TABLE nombre_Tabla DROP COLUMN nombre_Columna1,DROP COLUMN
nombre_Columna2,…;
```

Eliminar la clave primaria de la tabla:

Sintaxis

```
ALTER TABLE nombre_Tabla DROP PRIMARY KEY;
```

Añadir clave primaria:

Sintaxis

```
ALTER TABLE nombre_Tabla ADD PRIMARY KEY (nombre_Columna);
```

Eliminar una **clave foránea** de la tabla:

Sintaxis

```
ALTER TABLE nombre_Tabla DROP FOREIGN KEY nombre_Columna;
```

Añadir una **clave foránea** de la tabla:

Sintaxis

```
ALTER TABLE nombre_Tabla ADD CONSTRAINT nombre_Restriccion FOREIGN KEY
nombre_Columna REFERENCES ...;
```

Añadir una restricción a la tabla:

Sintaxis

```
ALTER TABLE nombre_Tabla ADD CONSTRAINT nombre_Restriccion sintaxis de
la restricción …;
```

Eliminar una restricción de la tabla:

Sintaxis

```
ALTER TABLE nombre_Tabla DROP CONSTRAINT nombre_Restriccion ;
```

Añadir una nueva columna al final de la tabla:

Sintaxis

```
ALTER TABLE nombre_Tabla ADD ~~COLUMN~~ nombre_Columna1 tipo_datos_columna;
```

sin column

Añadir un índice:

Sintaxis

```
ALTER TABLE nombre_Tabla ADD INDEX nombre_Columna;
```

Eliminar un índice:

Sintaxis

```
ALTER TABLE nombre_Tabla DROP INDEX nombre_Indice;
```

Cambiar el nombre de una columna de la tabla:

Sintaxis

```
ALTER TABLE nombre_Tabla RENAME COLUMN nombre_Columna TO
nombre_Nuevo_Columna;
```

Cambiar tipo de dato de una columna

Sintaxis

```
ALTER TABLE nombre_Tabla MODIFY nombre_Columna nuevo_tipo;
```

Vaciar el contenido de una tabla

Si queremos eliminar los registros de una tabla sin eliminarla la tabla, basta con hacer uso de la cláusula **TRUNCATE**:

Sintaxis

```
TRUNCATE TABLE nombre_Tabla;
```

Crear una vista

Una vista es una tabla virtual cuyo contenido está definido por una consulta.

Sintaxis

```
CREATE VIEW nombre_Vista [(listaColumnas)] AS (Consulta)
[WITH CHECK OPTION];
```

Dado que la creación de una vista se basa en una consulta, las estudiaremos cuando conozcamos el Lenguaje de Manipulación de Datos (DML)

Crear una Secuencia

Una secuencia (sequence) se emplea para generar valores enteros secuenciales únicos y asignarlos a campos numéricos; se utilizan generalmente para las claves primarias de las tablas garantizando que sus valores no se repitan.

Una secuencia es una tabla con un campo numérico en el cual se almacena un valor y cada vez que se consulta, se incrementa tal valor para la próxima consulta.

Sintaxis

```
create sequence NOMBRESECUENCIA
  start with VALORENTERO
  increment by VALORENTERO
  maxvalue VALORENTERO
  minvalue VALORENTERO
  cycle | nocycle;
```

- La cláusula "start with" indica el valor desde el cual comenzará la generación de números secuenciales. Si no se especifica, se inicia con el valor que indique "minvalue".
- La cláusula "increment by" especifica el incremento, es decir, la diferencia entre los números de la secuencia; debe ser un valor numérico entero positivo o negativo diferente de 0. Si no se indica, por defecto es 1.
- "maxvalue" define el valor máximo para la secuencia. Si se omite, por defecto es 999999999999999999999999999.
- "minvalue" establece el valor mínimo de la secuencia. Si se omite será 1.
- La cláusula "cycle" indica que, cuando la secuencia llegue a máximo valor (valor de "maxvalue") se reinicie, comenzando con el mínimo valor ("minvalue") nuevamente, es decir, la secuencia vuelve a utilizar los números. Si se omite, por defecto la secuencia se crea "nocycle".

Si no se especifica ninguna cláusula, excepto el nombre de la secuencia, por defecto, comenzará en 1, se incrementará en 1, el mínimo valor será 1, el máximo será 999999999999999999999999999 y "nocycle".

Dijimos que las secuencias son tablas; por lo tanto se accede a ellas mediante consultas, empleando "select". La diferencia es que utilizamos pseudocolumnas para recuperar el valor actual y el siguiente de la secuencia. Estas pseudocolumnas pueden incluirse en el "from" de una consulta a otra tabla o de la tabla "dual".

Para recuperar los valores de una secuencia empleamos las pseudocolumnas "currval" y "nextval".

Primero debe inicializarse la secuencia con "nextval". La primera vez que se referencia "nextval" retorna el valor de inicio de la secuencia; las siguientes veces, incrementa la secuencia y nos retorna el nuevo valor:

NOMBRESECUENCIA.NEXTVAL;

Se coloca el nombre de la secuencia seguido de un punto y la pseudocolumna "nextval" (que es una forma abreviada de "next value", siguiente valor).

Para recuperar el valor actual de una secuencia usamos:

NOMBRESECUENCIA.CURRVAL;

Es decir, el nombre de la secuencia, un punto y la pseudocolumna "currval" (que es una forma abreviada de "current value", valor actual).

Los valores retornados por "currval" y "nextval" pueden usarse en sentencias "insert" y "update".

Creamos una secuencia para el código de la tabla "libros", especificando el valor máximo, el incremento y que no sea circular:

```
create sequence sec_codigolibros
 maxvalue 999999
 increment by 1
 nocycle;
```

Luego inicializamos la secuencia. Recuerde que la primera vez que se referencie la secuencia debe emplearse "nextval" para inicializarla.

Insertamos un registro en "libros", almacenando en el campo "codigo" el valor siguiente de la secuencia:

```
insert into libros values
 (sec_codigolibros.nextval,'Matematica estas ahi', 'Paenza',Nuevo siglo');
```