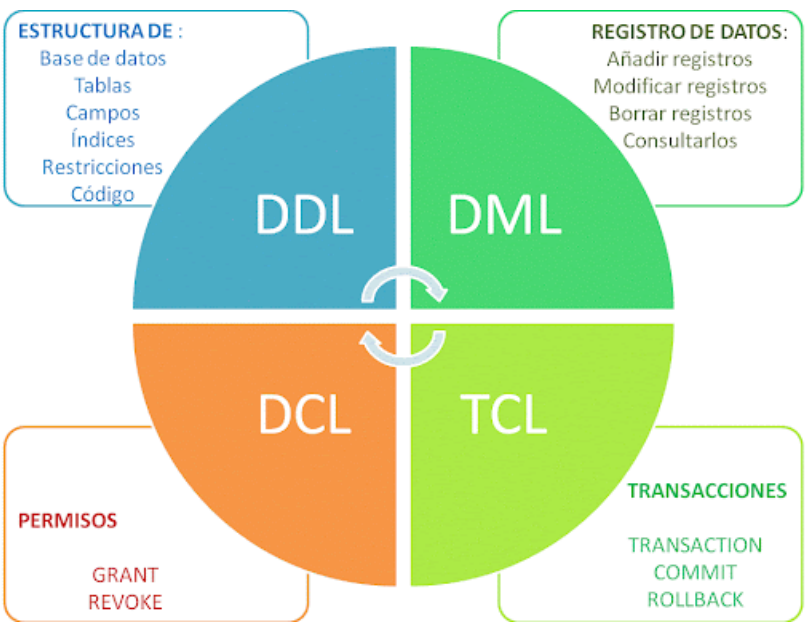
3.6. Lenguaje de definición de datos

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 (comandos CREALDRO)
Lenguaje de manipulación de datos (DML): Permite recuperar, almacenar, modificar, eliminar, insertar y actualizar datos de una base de datos. Sus comando principales son SELECT, INSERT, UPDATE, DELETE (comandos INUPDEL)
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 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 [TEMPORARY] TABLE [IF NOT EXISTS] nombre_tabla
(definición de las columnas,
definición de claves,
restricciones);
```

Definición de las columnas

Sintaxis

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] 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 DEPT (
 DEPTNO              NUMBER(2) NOT NULL,
 DNAME               CHAR(14),
 LOC                 CHAR(13),
 CONSTRAINT DEPT_PRIMARY_KEY PRIMARY KEY (DEPTNO));
```

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 VARCHAR (7) NOT NULL PRIMARY KEY,`
	`Marca VARCHAR (20)  NOT NULL ,`
	`Modelo VARCHAR (20)  NOT NULL ,`
	`Precio NUMBER (7,2)  NOT NULL`
	`);`

Notación PK con una columna al final de la definición de las columnas	`CREATE TABLE DEPT (`
	` DEPTNO  NUMBER(2) NOT NULL,`
	` DNAME  CHAR(14),`
	` LOC  CHAR(13),`
	` CONSTRAINT DEPT_PRIMARY_KEY PRIMARY KEY (DEPTNO));`

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

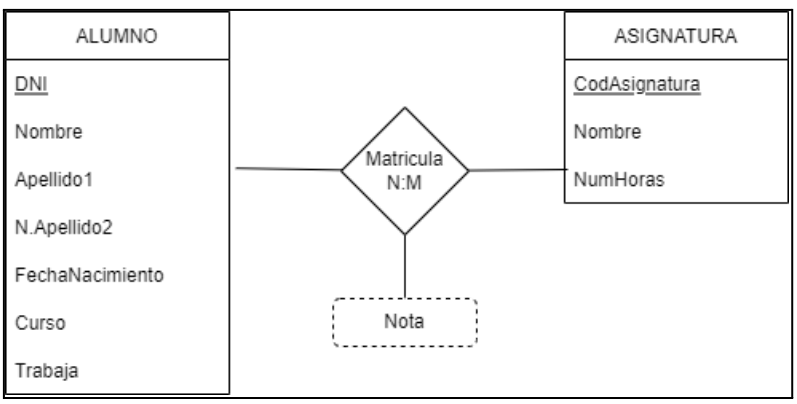
Notación FK	`CREATE TABLE Cliente`
	`( DNI VARCHAR(9) NOT NULL PRIMARY KEY,`
	`Nombre VARCHAR (50) NOT NULL,`
	`Apellido1 VARCHAR(50) NOT NULL,`
	`Apellido2 VARCHAR(50)`
	`);`
	`CREATE TABLE Mascota`
	`( Codigo INT PRIMARY KEY,`
	`Nombre VARCHAR (50),`
	`Raza VARCHAR (25),`
	`DNICliente VARCHAR (9),`
	`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:




```
create table Alumno
(
DNI varchar(9) NOT NULL PRIMARY KEY,
Nombre varchar(50) NOT NULL,
Apellido1 varchar(50) NOT NULL,
Apellido2 varchar(50),
FechaNacimiento date NOT NULL,
Curso integer NOT NULL,
Trabaja varchar(1) NOT NULL
);
create table Asignatura
(
CodAsignatura int NOT NULL PRIMARY KEY,
Nombre varchar(50) NOT NULL,
NumHoras int NOT NULL
);
create table Matricula
(
DNIAlumno varchar(9),
CodAsignatura int,
Nota decimal ,
PRIMARY KEY(DNIAlumno,CodAsignatura),
constraint FK_DNI_Matricula
FOREIGN KEY(DNIAlumno) REFERENCES Alumno(DNI),
constraint FK_CodAsignatura_Matricula
```

```
FOREIGN KEY(CodAsignatura) REFERENCES Asignatura(CodAsignatura)
);
```

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 Coche
(
Matricula VARCHAR (7) NOT NULL PRIMARY KEY,
Marca VARCHAR (20)  NOT NULL ,
Modelo VARCHAR (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)
```

Ejemplo
Supongamos que en la tabla Coche queremos poner precio=0 si no se especifica un valor: **CREATE TABLE** Coche (Matricula VARCHAR (7) NOT NULL PRIMARY KEY, Marca VARCHAR (20) NOT NULL , Modelo VARCHAR (20) NOT NULL , Precio FLOAT (7,2) DEFAULT 0.00);

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 INTEGER NOT NULL PRIMARY KEY, Nombre Varchar(20) NOT NULL, Apellido1 Varchar(20) NOT NULL, Apellido2 Varchar(20), Email Varchar(150), Telefono Varchar(10), UNIQUE (Email));
Notación 2	**CREATE TABLE** Clientes (CodCliente INTEGER NOT NULL PRIMARY KEY, Nombre Varchar(20) NOT NULL, Apellido1 Varchar(20) NOT NULL, Apellido2 Varchar(20), Email Varchar(150) UNIQUE, Telefono Varchar(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:

- "set null": indica que si eliminamos un registro de la tabla referenciada (TABLA2) cuyo valor existe en la tabla principal (TABLA1), dicho registro se elimine y los valores coincidentes en la tabla principal se seteen a "null".
- "cascade": indica que si eliminamos un registro de la tabla referenciada en una "foreign key" (TABLA2), los registros coincidentes en la tabla principal (TABLA1), también se eliminen; es

decir, si eliminamos un registro al cual una clave foránea referencia, dicha eliminación se extiende a la otra tabla (integridad referencial en cascada).

- "no action": es la predeterminada; indica que si se intenta eliminar un registro de la tabla referenciada por una "foreign key", Oracle no lo permita y muestre un mensaje de error. Se establece omitiendo la cláusula "on delete" al establecer la restricción.

La sintaxis completa para agregar esta restricción a una tabla es la siguiente:

```
alter table TABLA1
 add constraint NOMBRERESTRICCION
 foreign key (CAMPOCLAVEFORANEA)
 references TABLA2(CAMPOCLAVEPRIMARIA)
 on delete OPCION;
```

Ejemplo

```
alter table libros
  add constraint FK_libros_codigoeditorial
  foreign key (codigoeditorial)
  references editoriales(codigo)
  on delete cascade;
```

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;
```

Si la tabla que queremos eliminar no existiera se pararía la ejecución del código MySQL, por eso es recomendable utilizar la cláusula *IF EXISTS*:

Sintaxis

```
DROP TABLE IF EXISTS nombre_Tabla;
```

También podemos eliminar varias tablas a la vez, separando el nombre de cada una con una ',':

Sintaxis

```
DROP TABLE nombre_Tabla1, nombre_Tabla2, nombre_Tabla3,…;
```

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;
```

Añadir una nueva columna después de otra columna:

Sintaxis

```
ALTER TABLE nombre_Tabla ADD COLUMN nombre_Columna tipo_datos_columna
AFTER nombre_Columna_anterior;
```

Añadir una nueva columna en la primera posición de la tabla:

Sintaxis

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

Añadir un índice:

Sintaxis

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

Eliminar un índice:

Sintaxis

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

Modificar el valor de la columna con **propiedad autoincrement** para que empiece por el valor 100:

Sintaxis

```
ALTER TABLE nombre_Tabla AUTO_INCREMENT =100;
```

Cambiar el nombre de una columna de la tabla:

Sintaxis

```
ALTER TABLE nombre_Tabla CHANGE nombre_Columna nombre_Nuevo_Columna
tipo_dato;
```

Cambiar el nombre y el tipo de dato de una columna de la tabla:

Sintaxis

```
ALTER TABLE nombre_Tabla CHANGE nombre_Columna nombre_Nuevo_Columna
nuevo_tipo;
```

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');
```