1. Lenguaje de definición de datos (DDL)

- Crear Base de Datos
  - CREATE DATABASE nombre_bbdd;
  - Ejemplo:
    - CREATE DATABASE testDB;

- Borrar Base de Datos
  - DROP DATABASE nombre_bbdd;
  - Ejemplo:
    - DROP DATABASE testDB;

- Crear Tabla
  - CREATE TABLE nombre_tabla (
    columna1 datatype,
    columna2 datatype,
    columna3 datatype,
    .....
    columnN datatype,
    PRIMARY KEY (columna1, columna2,…)
    );

  - Ejemplo:

```
CREATE TABLE CUSTOMERS (
  ID   INT,
  NAME VARCHAR (20),
  AGE INT,
  ADDRESS CHAR (25) ,
  SALARY   DECIMAL (18, 2),
  PRIMARY KEY (ID))
;
```

- Borrar Tabla
  - DROP TABLE nombre_tabla;
  - Ejemplo:
    - DROP TABLE CUSTOMERS;

- Restricciones (Constraints):
  - Se pueden en usar al crear la tabla (CREATE TABLE) o cuando se modifica una tabla (ALTER TABLE)
  - Las posibles restricciones son:
    - NOT NULL

        - UNIQUE
        - PRIMARY KEY
        - FOREIGN KEY
        - CHECK
        - DEFAULT

- Alteraciones a tablas (ALTER TABLE)

    - Añadir Columna
        - ALTER TABLE nombre_tabla ADD nombre_columna DATATYPE;

    - Borrar Columna
        - ALTER TABLE nombre_tabla DROP COLUMN nombre

    - Modificar el tipo de dato de una columna
        - ALTER TABLE nombre_tabla MODIFY COLUMN nombre_columna datatype;

    - Modificar una columna indicando una restricción no NOT NULL
        - ALTER TABLE nombre_tabla MODIFY COLUMN nombre_columna datatype NOT NULL;

    - Modificar una columna indicando una restricción UNIQUE:
        - ALTER TABLE nombre_tabla ADD CONSTRAINT Constraint_name UNIQUE (column1, column2...);

    - Modificar una columna indicando una restricción CHECK:
        - ALTER TABLE nombre_tabla ADD CONSTRAINT Constraint_name CHECK (CONDITION);

    - Modificar una columna indicando una restricción PRIMARY KEY:
        - ALTER TABLE nombre_tabla ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);

    - Eliminar una restricción de la tabla:
        - ALTER TABLE nombre_tabla DROP CONSTRAINT Contraint_name;

- Ejemplos:
    - Crear una tabla con restricciones de campos no nulos y de valores distintos en una columna (UNIQUE):

```
CREATE TABLE personas (
        identificador int NOT NULL,
        nombre varchar(255) NOT NULL,
        apellido1 varchar(255) NOT NULL,
```

```
    apellido2 varchar(255),
    UNIQUE (identificador)
    );
```

- Crear una tabla con restricciones de campos no nulos y una clave primaria compuesta por 2 campos:

```
CREATE TABLE personas (
    identificador int NOT NULL,
    nombre varchar(255) NOT NULL,
    apellido1 varchar(255) NOT NULL,
    CONSTRAINT constraintPK PRIMARY KEY (identificador,
nombre)
    );
```

- Modificar table para añadir una clave primaria:

```
ALTER TABLE personas ADD PRIMARY KEY (identificador);
```

- Crear tablas con claves primarias y foráneas

```
CREATE TABLE departamentos (
    dep int NOT NULL,
    departamento varchar(255),
    PRIMARY KEY (dep)
    );

CREATE TABLE personas (
    per int NOT NULL,
    nombre varchar(255),
    apellido1 varchar(255),
    dep int NOT NULL,
    PRIMARY KEY (per),
    FOREIGN KEY (dep) REFERENCES departamentos(dep)
    );
```

- Clave externa o foránea (FOREIGN KEY) compuesta por varias columnas o queremos ponerle un nombre:

```
CONSTRAINT fkpersonas FOREIGN KEY (dep, id) REFERENCES
departamentos(dep,id);
```

- Borrar una clave foranéa:

```
ALTER TABLE personas DROP FOREIGN KEY dep;
```

- Restricción CHECK para limitar el rango de valores que puede tener una columna

```
CREATE TABLE departamentos (
        dep int NOT NULL CHECK (dep>0),
        departamento varchar(255)
        );
```

- Restricción DEFAULT para definir un valor que se asigna por defecto a una columna

```
CREATE TABLE pedidos (
        idpedido int,
        producto int,
        cantidad int,
        fecha date DEFAULT "2012-01-01"
        );
```

- ALTER TABLE pedidos ALTER COLUMN fecha SET DEFAULT "2012-01-01";

- ALTER TABLE pedidos ALTER fecha DROP DEFAULT;

- Crear Vista
  - CREATE VIEW nombre_vista AS

```
SELECT
        columna1,
        columna2,
        …,
        columnan
FROM
        nombre_tabla
WHERE
        (condición Where);
```

- Crear indíce para una tabla
  - CREATE INDEX nombre_index ON nombre_tabla;

- Crear indíce para columnas específicas de una tabla
  - CREATE INDEX nombre_index ON nombre_tabla (columna1,…, columnan);

- Borrar Indíce
    - DROP INDEX nombre_index;

2. Lenguaje de Manipulación de datos (DML)

2.1 Sentencia SELECT

En esta sección tenemos las operaciones SQL que nos permiten hacer manipulación de datos.

Las cláusulas/instrucciones entre los paréntesis rectos ([y]) son opcionales.

- Sentencia Select
    - SELECT select_list
      FROM table_source
      [WHERE search_condition]
      [GROUP BY group_by_expression]
      [HAVING search_condition]
      [ORDER BY order_expression [ASC | DESC]];

Iremos analizar con más detalle cada una de las cláusulas.

- Cláusula Select
    - SELECT [DISTINCT] <select_list>
        - La opción distinct se usa cuando queremos que el resultado sea de la consulta los diferentes valores de una columna o las diferentes combinaciones de varias columnas

      La componente <select_list> es donde se define el listado de campos que queremos en el resultado de la consulta y tiene la siguiente forma:

      <select_list> ::= [esquema.][TABLE. | VIEW. | alias.] * | { column_name | expression } [[AS] column_alias]} [,...n]

- Cláusula From
    - [FROM {<table_source>} [,...n]]<table_source> ::= TABLE_NAME | view_name | derived_table | <joined_table>;
    - Cuando hacemos consultas con 2 o más tablas tenemos que usar la instrucción join que puede ser de distintos tipos:
        - Inner Join
        - Left Outer Join
        - Right Outer Join
        - Full Outer Join

      Tenemos también que usar una condición ON para definir la relación entre las tablas.

Ejemplo:

- Este select devuelve datos de la tabla employees, mostrando eployeed_id y last_name, junto con manager_id y last_name, haciendo un inner join. Las tablas tienen los alias e y m.

```
SELECT
        e.employee_id emp_id,
        e.last_name emp_lastname,
        m.employee_id mgr_id,
        m.last_name mgr_lastname
FROM
        employees e
  INNER JOIN
        employees m
  ON
        e.manager_id = m.employee_id;
```

- Podríamos usar diferentes tipos de join para obtener distintos conjuntos de datos de salida.

- Cláusula Where
  - La cláusula Where se usa para definir que condición o condiciones cumplen los registros que son incluidos en el resultado de la consulta.
  - Su sintaxis es

    ```
    WHERE <condicion_busqueda>
    ```
  - En la condición de búsqueda se usan los siguientes operadores:

Operador	Definición
=, !=, <>	Igual a, distinto de, distinto de
>, >=, <, <=	Mayor que, Mayor o igual que, menor que, menor o igual que
BETWEEN … AND …	Chequea un rango incluyendo los dos valores
LIKE	Compara una cadena de caracteres con un patrón o formato
IN ( ), NOT IN ( )	Comprueba que un campo o expresión tenga valores o no en un conjunto de valores
IS NULL, IS NOT NULL	Comprueba que un campo o expresión sea nula o no

Con el operador Like sn usados los símbolos:

- % - significa “cualquier cadena de cero o más caracteres”

        - Ejemplo:
          WHERE title LIKE '%computer%' **busca todos los títulos de libros que contengan la palabra 'computer' en cualquier parte del título.**
    - _ (subrayado) – Significa "cualquier carácter individual"
        - Ejemplo:
          WHERE au_fname LIKE '_ean' busca todos los nombres de cuatro letras que finalicen con ean

- Cláusula Group By
    - La cláusula Group By se usa cuando queremos agrupar registros que tengan el mismo valor en una columna y pretendemos procesarlos de la misma manera.
    - Ejemplo:
        - Queremos sumar los importes por cliente (identificados por C_CLIENTE) en la tabla FACTURA.

```
SELECT
        C_CLIENTE,
        SUM(IMPORTE)
FROM
        FACTURA
GROUP BY
        C_CLIENTE;
```

    - Se suelen usar las distintas funciones agregadas:
        - AVG – Media aritmética
            - Sintaxis: AVG(expresión)
        - COUNT – Conteo de registros
            - Sintaxis: Count(expresión)
        - MAX y Min – Valores máximos y mínimos de un campo
            - Sintaxis: Max(expresión) y Min(expresión)
        - SUM – Suma los valores
            - Sintaxis: SUM (expresión)
        - STDEV | STDEVP | VAR | VARP – Desviación típica y varianza

- Cláusula Having
    - Esta cláusula se usa para definir una condición de búsqueda para un grupo definido por la cláusula GROUP BY. Si es usada sin un GROUP BY funciona como una cláusula WHERE pero es menos eficiente.
    - Sintaxis:
        - HAVING <condicion_busqueda_grupo>

- Ejemplo:

```
SELECT
    cCmnMtr
FROM
    porte
GROUP BY
    cCmnMtr
HAVING
    count(*) > =1;
```

- Cláusula Order By
  - Esta cláusula se usa para ordenar los registros de la tabla de salida de la consulta.
  - Sintaxis:

```
[ORDER BY {order_by_expression [ ASC | DESC ] } [,...n] ]
```

2.2 Sentencia INSERT

En la siguiente sección del documento vamos a ver la función INSERT que se usa para añadir registros a una tabla.

Podemos insertar registros especificando los valores (caso 1) o a partir de una consulta (caso 2).

Caso 1 – INSERT INTO con cláusula VALUES

La sintaxis de la instrucción INSERT INTO en que se definen los valores a insertar es:

```
INSERT INTO nombre_tabla (columna1,…,columnan)
VALUES (valor1,…,valorn);
```

- Ejemplo:

```
INSERT INTO Store_Information (Store_Name, Sales, Txn_Date)
VALUES ('Los Angeles', 900, '10-Jan-1999');
```

Caso 2 – INSERT INTO con sentencia SELECT

La sintaxis en este caso es diferente ya que incluye cláusulas de una sentencia SELECT para definir que registros se van a añadir a una tabla.

Se podrán usar todas las cláusulas opcionales en las sentencias SELECT (GROUP BY, WHERE, etc)

```
INSERT INTO nombre_tabla1 (columna1,…,columnan)
SELECT columnaselect1,…,columnaselectn.
FROM nombre_tabla2;
```

- Ejemplo:

```
INSERT INTO Store_Information (Store_Name, Sales, Txn_Date)
SELECT  Store_Name, Sales, Txn_Date
FROM Sales_Information
WHERE Year(Txn_Date)=1998;
```

2.3 Sentencia UPDATE

La sentencia UPDATE es usada para actualizar valores de registros ya existentes en una tabla.

Su sintaxis es:

```
UPDATE nombre_tabla
    SET
        columna_1 = nuevovalor1
        …
        columna_n=nuevovalorn
     WHERE
        condiciones_registros_actualizar;
```

Ejemplo:

```
UPDATE
        empleados
 SET
        salario = salario * 1.02
        precio_hora = precios_hora * 1.01
 WHERE
        salario < 3000;
```

2.4 Sentencia DELETE y TRUNCATE

La sentencia DELETE se usa para borrar registros de una tabla que cumplan las condiciones especificadas en una condición WHERE.

Su sintaxis es:

```
DELETE FROM
        nombre_tabla
WHERE
        condiciones_registros_actualizar;
```

Ejemplo:

```
DELETE FROM
        peliculas
WHERE
        Duracion >= 115;
```

En el caso de se pretende borrar la totalidad de los registros de una tabla se usa la sentencia TRUNCATE que tiene la siguiente sintaxis:

```
TRUNCATE TABLE nombre_tabla;
```

Ejemplo:

```
TRUNCATE TABLE peliculas;
```