Modelo Relacional de Base de datos

El modelo relacional es un enfoque para organizar y estructurar la información en una base de datos. En este modelo, los datos se representan en forma de tablas, donde cada tabla tiene filas que representan registros individuales y columnas que representan atributos o características de esos registros.

Cada tabla tiene una clave primaria única que identifica de manera exclusiva cada registro en la tabla. Además, las tablas se relacionan entre sí mediante claves foráneas, que son columnas que hacen referencia a la clave primaria de otra tabla.

El modelo relacional permite establecer relaciones y vínculos lógicos entre los datos, lo que facilita la consulta y manipulación de la información. Además, se enfoca en la integridad de los datos, asegurando que se cumplan las restricciones y reglas definidas en la estructura de la base de datos.

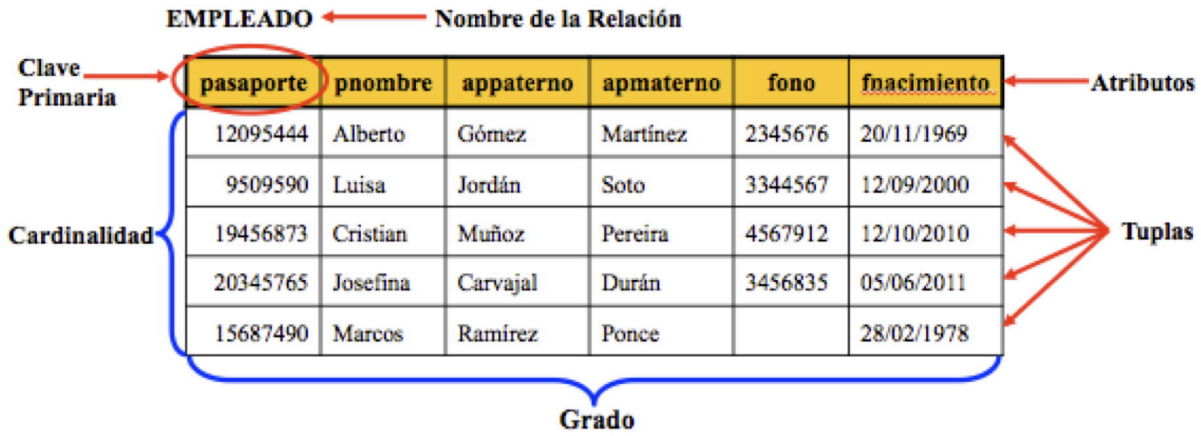
En resumen, el modelo relacional organiza los datos en tablas relacionadas entre sí mediante claves primarias y foráneas (que se definirán a continuación), lo que brinda una estructura flexible y eficiente para almacenar, consultar y manipular la información en una base de datos. Normalmente las bases de datos dentro del modelo relacional se utilizan bajo un RDBMS (Sistema de Gestión de Bases de Datos Relacionales).

RDBMS (Sistema de Gestión de Bases de Datos Relacionales)

Un RDBMS es un sistema de gestión de bases de datos que utiliza el modelo relacional para organizar y almacenar datos. El modelo relacional se basa en tablas que contienen datos y se relacionan entre sí a través de claves primarias y claves foráneas. Algunos ejemplos de marcas de RDBMS son: SQLite, MySQL, PostgreSQL, MariaDB, Microsoft SQL Server y Oracle Database.

Tablas

Las tablas son la estructura básica del modelo relacional de base de datos. Cada tabla está compuesta por columnas y filas. Las columnas representan los campos de datos que se almacenan en la tabla, y las filas contienen los valores específicos de cada campo. La intersección de una fila y una columna es un campo de dato. A continuación se muestran los conceptos utilizados en la descripción del modelo relacional asociados a una tabla:



pasaporte	pnombre	appaterno	apmaterno	fono	fnacimiento
12095444	Alberto	Gómez	Martínez	2345676	20/11/1969
9509590	Luisa	Jordán	Soto	3344567	12/09/2000
19456873	Cristian	Muñoz	Pereira	4567912	12/10/2010
20345765	Josefina	Carvajal	Durán	3456835	05/06/2011
15687490	Marcos	Ramírez	Ponce		28/02/1978

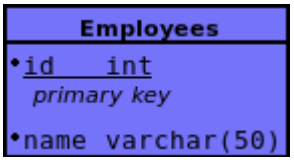
Términos:

- Relación: Corresponde con la idea general de la tabla y su relación con otras.
- Tupla/Registro: corresponde con una fila.
- Atributo: corresponde con una columna.
- Cardinalidad: es el número de tuplas.
- Grado: es el número de atributos.
- Clave primaria: identificador único, no hay dos tuplas con el mismo identificador.

Claves Primarias

Una clave primaria es un campo o conjunto de campos que identifican de manera única cada fila en una tabla. La clave primaria se utiliza para garantizar la integridad de los datos en la tabla y para crear relaciones con otras tablas.

Diagrama:



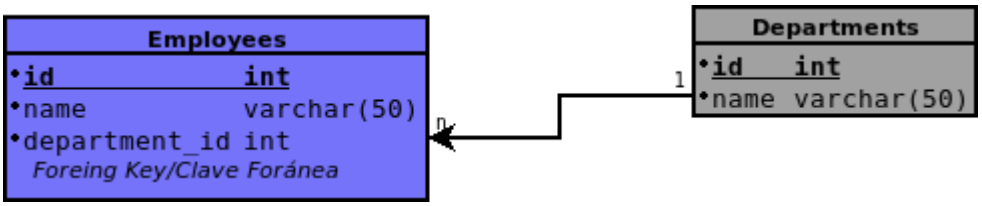
Código:

```
CREATE TABLE Employees (
  id INT PRIMARY KEY,
  name VARCHAR(50)
);
```

Claves Foráneas

Una clave foránea es un campo o conjunto de campos que se refieren a la clave primaria de otra tabla. Se utilizan para establecer relaciones entre las tablas y para garantizar la integridad de los datos.

Diagrama:



Código:

```
CREATE TABLE Departments (
  id INT PRIMARY KEY,
  nombre VARCHAR(50)
);
```

```
CREATE TABLE Employees (
  id INT PRIMARY KEY,
  nombre VARCHAR(50),
  department_id INT,
  FOREIGN KEY (department_id) REFERENCES Departments(id)
);
```

`FOREIGN KEY (department_id) REFERENCES Departments(id):` Esta línea crea la referencia con el id de la tabla departamentos mediante la clave foránea dpto_id.

Índices y Restricciones en Bases de Datos Relacionales

En una base de datos relacional, los índices y las restricciones (constraints) son fundamentales para garantizar la integridad de los datos y mejorar el rendimiento de las consultas. A continuación, se explican los diferentes tipos de índices y restricciones, junto con ejemplos en MySQL/MariaDB, SQLite y PostgreSQL.

Índices en Bases de Datos

Los índices son estructuras que optimizan el acceso a los datos en una tabla, acelerando la búsqueda y recuperación de registros. Existen varios tipos de índices:

Tipos de Índices

- **Índice primario (PRIMARY KEY)**: Se aplica a la clave primaria y garantiza la unicidad de los valores en una columna o conjunto de columnas.
- **Índice único (UNIQUE)**: Evita valores duplicados en una columna específica.
- **Índice normal (INDEX)**: Mejora la velocidad de búsqueda pero no impone restricciones de unicidad.
- **Índice de texto completo (FULLTEXT)**: Utilizado para búsqueda eficiente de texto en columnas de tipo **TEXT** o **VARCHAR**.

Creación de Índices

MySQL/MariaDB

```
CREATE INDEX idx_patient_name ON Patients(name);
CREATE UNIQUE INDEX idx_email ON Patients(email);
```

SQLite

```
CREATE INDEX idx_patient_name ON Patients(name);
```

PostgreSQL

```
CREATE INDEX idx_patient_name ON Patients(name);
CREATE UNIQUE INDEX idx_email ON Patients(email);
```

Restricciones en Bases de Datos

Las restricciones (constraints) son reglas aplicadas sobre las tablas para garantizar la integridad de los datos.

Tipos de Restricciones

- **PRIMARY KEY**: Garantiza que un campo tenga valores únicos y no nulos.
- **UNIQUE**: Asegura que no haya valores duplicados en la columna.
- **NOT NULL**: Evita valores nulos en la columna.
- **FOREIGN KEY**: Define una relación con otra tabla.
- **CHECK**: Permite validar valores en la columna.
- **DEFAULT**: Especifica un valor por defecto para una columna.

Creación de Restricciones

MySQL/MariaDB

```
CREATE TABLE Patients (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    age INT CHECK (age > 0)
);
```

SQLite

```sql
CREATE TABLE Patients (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    email TEXT UNIQUE,
    age INTEGER CHECK (age > 0)
);
```

PostgreSQL

```sql
CREATE TABLE Patients (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    age INT CHECK (age > 0)
);
```

Restricciones con ON DELETE y ON UPDATE

Las restricciones en claves foráneas pueden tener diferentes comportamientos al eliminar o actualizar registros:

- **CASCADE**: Propaga los cambios a las tablas relacionadas.
- **SET NULL**: Cambia el valor a NULL en las filas relacionadas.
- **RESTRICT**: Impide la eliminación o actualización si existen referencias.
- **NO ACTION**: Similar a RESTRICT pero permite diferir la verificación.

Ejemplo con Claves Foráneas y Restricciones

MySQL/MariaDB

```sql
CREATE TABLE Appointments (
    id INT PRIMARY KEY AUTO_INCREMENT,
    patient_id INT,
    appointment_date DATE NOT NULL,
```

```
    FOREIGN KEY (patient_id) REFERENCES Patients(id)
    ON DELETE CASCADE ON UPDATE CASCADE
);
```

SQLite

```
PRAGMA foreign_keys = ON;

CREATE TABLE Appointments (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    patient_id INTEGER,
    appointment_date DATE NOT NULL,
    FOREIGN KEY (patient_id) REFERENCES Patients(id)
    ON DELETE SET NULL ON UPDATE RESTRICT
);
```

PostgreSQL

```
CREATE TABLE Appointments (
    id SERIAL PRIMARY KEY,
    patient_id INT,
    appointment_date DATE NOT NULL,
    FOREIGN KEY (patient_id) REFERENCES Patients(id)
    ON DELETE NO ACTION ON UPDATE CASCADE
);
```

Los índices y restricciones son herramientas esenciales en la optimización y gestión de bases de datos. Dependiendo del motor de base de datos y de las necesidades del proyecto, se pueden usar diferentes configuraciones para garantizar la integridad referencial y mejorar el rendimiento de las consultas.

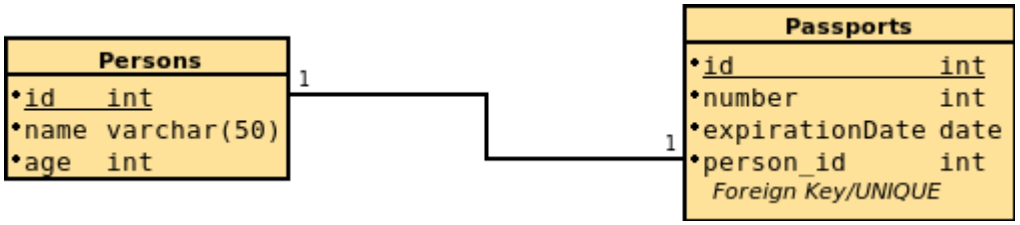
Cardinalidad

La cardinalidad se refiere a la relación entre las filas de una tabla y las filas de otra tabla. La cardinalidad puede ser uno a uno, uno a muchos o muchos a muchos. La cardinalidad se utiliza para determinar cómo se deben crear las relaciones entre las tablas.

Cardinalidad Uno a Uno (One-to-One):

En este tipo de relación, un registro en una tabla se asocia con exactamente un registro en otra tabla.

Ejemplo: Relación entre las tablas "Personas" y "Pasaportes", en el caso ideal de que una persona solo pueda tener un pasaporte (sabemos que en la realidad no es así).

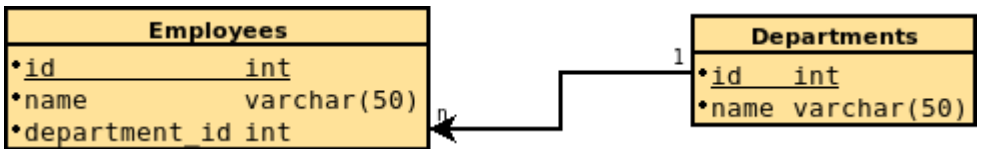


Para asegurar que la relación sea una persona y solo un pasaporte, se debe agregar una restricción (constraint) de tipo ÚNICO (UNIQUE) a la clave foránea. Concepto que se ampliará posteriormente. Lo importante es entender que de este modo no se puede repetir a la misma persona con dos pasaportes, si eso fuera necesario, se quita la restricción y la relación sería 1 persona —> n,* o muchos pasaportes.

Cardinalidad Uno a Muchos (One-to-Many):

En este tipo de relación, un registro en una tabla (departament) se asocia con varios registros en otra tabla (employee).

Ejemplo: Relación entre las tablas "Departamentos" y "Empleados".

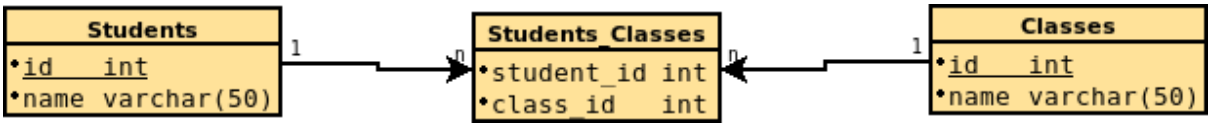


Un departamento puede pertenecer a muchos empleados.

Cardinalidad Muchos a Muchos (Many-to-Many):

En este tipo de relación, varios registros en una tabla se asocian con varios registros en otra tabla. Se logra a través de una "**tabla de unión" (Student_Classes).**

Ejemplo: Relación entre las tablas "Estudiantes" y "Clases".



Consultas DDL (Lenguaje de Definición de Datos)

El lenguaje de definición de datos (DDL - Data Definition Language) se utiliza para definir la estructura de la base de datos. Las operaciones DDL incluyen CREATE, ALTER y DROP.

- CREATE: Crea una nueva tabla, vista, índice u otro objeto en la base de datos.
- ALTER: Modifica la estructura de una tabla existente, como agregar o eliminar columnas.
- DROP: Elimina una tabla, vista, índice u otro objeto de la base de datos.
- TRUNCATE: Elimina todos los datos de una tabla sin afectar su estructura.
- RENAME: Cambia el nombre de una tabla u otro objeto en la base de datos.

- **Ejemplos de Comandos DDL en diferentes motores de base de datos:**

(usamos nombres en inglés tanto para la columna como par los campos)

Crear una tabla Users con los campos con id (clave primaria), nombre y edad:
CREATE

SQLite:

```
CREATE TABLE Users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT,
  age INTEGER
);
```

MySQL y MariaDB:

```
CREATE TABLE Users (
  id INT PRIMARY KEY AUTOINCREMENT,
  name VARCHAR(50),
  age INT
);
```

PostgreSQL:

```
CREATE TABLE Users (
  id SERIAL PRIMARY KEY,
  name TEXT,
  age INTEGER
);
```

Modificar la tabla Users y agregar un campo/columna email: ALTER

SQLite, MySQL, MariaDB y PostgreSQL:

```
ALTER TABLE Users
ADD COLUMN email VARCHAR(50);
```

Eliminar la tabla users: DROP

SQLite, MySQL y PostgreSQL:

```
DROP TABLE Users;
```

Eliminar todos los datos de la tabla Users sin eliminar la estructura: TRUNCATE

SQLite:

```
DELETE FROM Users;
```

MySQL:

```
TRUNCATE TABLE Users;
```

PostgreSQL:

```
TRUNCATE TABLE Users;
```

Cambiar el nombre de la tabla Users: RENAME

(En realidad es una cláusula que se usa con ALTER)

SQLite, MySQL y PostgreSQL:

```
ALTER TABLE Users RENAME TO NewUsers;
```

Consultas DML (Lenguaje de Manipulación de Datos)

El lenguaje de manipulación de datos (DML - Data Manipulation Language) se utiliza para realizar operaciones en los datos almacenados en las tablas. Las operaciones DML incluyen SELECT, INSERT, UPDATE y DELETE.

- SELECT: Recupera registros o datos de una o varias tablas.
- INSERT: Inserta nuevos registros en una tabla.
- UPDATE: Actualiza los valores de uno o varios registros en una tabla.
- DELETE: Elimina uno o varios registros de una tabla.

- **Ejemplos de Comandos DML en diferentes motores de base de datos:**

Obtener/Seleccionar todos los datos de la tabla Users: SELECT

SQLite, MySQL y PostgreSQL:

```
SELECT * FROM Users;
```

Insertar valores en la tabla Users: INSERT

SQLite, MySQL y PostgreSQL:

```
INSERT INTO Users (name, age)
VALUES ('John', 25);
```

Actualizar valores en la tabla Users: UPDATE

(se combina con la cláusula WHERE, se va a actualizar la edad a 30 años donde el id del usuario sea igual a 1)

SQLite, MySQL y PostgreSQL:

```
UPDATE Users
SET age = 30
WHERE id = 1;
```

Borrar filas en la tabla Users: UPDATE

(se combina con la cláusula WHERE, se va a borrar los usuario cuya edad sea mayor a 40 años)

SQLite, MySQL y PostgreSQL:

```
DELETE FROM Users
WHERE age > 40;
```

- **Ejemplos de cláusulas para comandos SQL**

Cláusula WHERE (condicional):

SELECT con WHERE:

```
SELECT * FROM Users WHERE age > 30;
```

UPDATE con WHERE:

```
UPDATE Users SET active = 0 WHERE id = 1;
```

DELETE con WHERE:

```
DELETE FROM Users WHERE name = 'Juan';
```

Cláusula ORDER BY (ordenamiento):

SELECT con ORDER BY:

```
SELECT * FROM Users ORDER BY name ASC;
```

Cláusula LIMIT (límite de resultados):

SELECT con LIMIT:

```
SELECT * FROM Users LIMIT 10;
```

Cláusulas MIN y MAX (valores mínimo y máximo):

SELECT con MIN:

```
SELECT MIN(age) FROM Users;
```

SELECT con MAX:

```
SELECT MAX(age) FROM Users;
```

Cláusula LIKE (búsqueda de patrones):

SELECT con LIKE:

```
SELECT * FROM Users WHERE name LIKE 'J%';
```

Cláusula IN (coincidencia en una lista de valores):

SELECT con IN:

```
SELECT * FROM Users WHERE id IN (1, 2, 3);
```

Cláusula GROUP BY (agrupamiento):

SELECT con GROUP BY:

```
SELECT Departments, COUNT(*) FROM Employees GROUP BY Departments;
```

Cláusula HAVING (condicional para grupos):

SELECT con HAVING:

```
SELECT Departments, COUNT(*) FROM Employees GROUP BY Departments
HAVING COUNT(*) > 5;
```

Cláusula JOIN (unión de tablas):

SELECT con JOIN:

```
SELECT Users.name, Orders.date FROM Users JOIN Orders ON Users.id
= Orders.user_id;
```

Tipos de datos en Bases de Datos

Es algo muy curioso cómo la industria de software tiene una relación ambivalente con los tipos de datos, en los lenguajes de programación en ciertos períodos se promovió el tipado estático y por otros el tipado dinámico. “Hay frecuentemente conflictos entre aquellos que prefieren la tipificación fuerte y/o estática y aquellos que se inclinan por la tipificación dinámica, libre o débil. El primer grupo aboga por la detección temprana de errores durante la compilación y el aumento de rendimiento en tiempo de ejecución, mientras que el segundo grupo aboga por los prototipos rápidos que son posibles con un sistema de tipificación dinámico.” (https://es.wikipedia.org/wiki/Sistema_de_tipos).

Los motores de base de datos relacionales extienden esa curiosidad, porque si en un sistema se elige usar un RDMBS por más que trabajemos con lenguajes dinámicos en otras capas de la arquitectura del software al final, en la persistencia de esos datos final aparecerán los tipos que se trataron de ocultar anteriormente:

Tipos de Datos en SQLite

Tipo de dato	Descripción
NULL	El valor es un valor NULL (NULO).
INTEGER	El valor es un entero con signo, almacenado en 1, 2, 3, 4, 6 u 8 bytes, según la magnitud del valor.
REAL	El valor es un valor de punto flotante, almacenado como un número de punto flotante IEEE de 8 bytes.
TEXT	El valor es una cadena de texto, almacenada usando la codificación de la base de datos (UTF-8, UTF-16BE o UTF-16LE)
BLOB	El valor es una blob (gota) de datos, almacenada exactamente como se ingresó.

Ejemplos de nombres de afinidad:

La siguiente tabla muestra cuántos nombres de tipos de datos comunes de implementaciones SQL más tradicionales se convierten en afinidades mediante las cinco reglas de la sección anterior. Esta tabla muestra solo un pequeño subconjunto de los nombres de tipos de datos

que SQLite aceptará. Tenga en cuenta que SQLite ignora los argumentos numéricos entre paréntesis que siguen al nombre del tipo (por ejemplo, "VARCHAR(255)"); SQLite no impone ninguna restricción de longitud (excepto el gran límite global SQLITE_MAX_LENGTH) sobre la longitud de cadenas, BLOB o valores numéricos.

Ejemplos de nombres de tipos de la declaración CREATE TABLE o la expresión CAST	**Afinidad resultante**	**Regla utilizada para determinar la afinidad**
INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER	Si el tipo declarado contiene la cadena "INT", se le asigna afinidad INTEGER.
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB	TEXT	Si el tipo declarado de la columna contiene cualquiera de las cadenas "CHAR", "CLOB" o "TEXT", entonces esa columna tiene afinidad TEXT. Observe que el tipo VARCHAR contiene la cadena "CHAR" y, por lo tanto, se le asigna la afinidad TEXT.
BLOB *no datatype specified*	BLOB	Si el tipo declarado para una columna contiene la cadena "BLOB" o si no se especifica ningún tipo, entonces la columna tiene afinidad BLOB.
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL	Si el tipo declarado para una columna contiene cualquiera de las cadenas "REAL", "FLOA" o "DOUB", entonces la columna tiene afinidad REAL.
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC	En caso contrario, la afinidad es NUMÉRICA.

Mayor detalle de los tipos de SQLite en la documentación oficial:

https://www.sqlite.org/datatype3.html

Tipos de Datos en MySQL/MariaDB ´

Mayor detalle de MariaDB en documentación oficial: https://mariadb.com/kb/en/data-types/

Tipos de Datos en PostgreSQL

Mayor detalle en documentación oficial: https://www.postgresql.org/docs/8.1/datatype.html

Concepto de Esquema (Schema) en base de datos relacionales

Un esquema en una base de datos es una estructura organizativa o un plan lógico que define cómo se organiza y almacena la información dentro de la base de datos.
El esquema incluye la estructura y relación de todos los elementos dentro de la base de datos, como:

- Tablas (y sus columnas).
- Vistas.
- Índices.
- Procedimientos almacenados.
- Funciones.
- Triggers (o disparadores).

En esencia, un esquema es como un mapa o plano de la base de datos que especifica dónde y cómo se almacena cada tipo de dato, así como las reglas de organización y las relaciones entre los diferentes elementos.

Ejemplo de Esquema:

Imagina una base de datos que gestiona información para una librería. Un posible esquema podría definir las siguientes tablas:

- **books** (para almacenar información de los libros).
- **authors** (para guardar datos de los autores).
- **customers** (para registrar datos de los clientes).
- **orders** (para gestionar las órdenes de compra).

Esquemas en Diferentes Sistemas de Bases de Datos:

La forma en la que se manejan los esquemas puede variar dependiendo del sistema de base de datos (DBMS):

- SQLite: No tiene un concepto formal de esquema separado. Todo está contenido en una sola base de datos sin división en esquemas.
- MySQL: Soporta esquemas, pero en MySQL, un "esquema" es sinónimo de una "base de datos". En otras palabras, cada base de datos tiene su propio esquema.
- PostgreSQL: Utiliza esquemas dentro de una misma base de datos, lo que permite crear múltiples esquemas dentro de una misma base de datos. Esto es útil para organizar datos y usuarios en diferentes "subconjuntos" dentro de una sola base de datos.

Ventajas del Uso de Esquemas:

1. Organización: Los esquemas ayudan a organizar datos de forma lógica, facilitando la gestión de la base de datos.
2. Seguridad y permisos: Se pueden otorgar permisos específicos para cada esquema, lo que permite controlar quién puede ver o modificar ciertos datos.
3. Separación de datos: Permite organizar y separar datos dentro de la misma base de datos, lo que puede ser útil para mantener datos de diferentes departamentos, aplicaciones o versiones de un sistema.

Ejemplo en Código SQL (PostgreSQL)

En PostgreSQL, puedes crear un esquema y luego agregar tablas dentro de él. Aquí un ejemplo:

```sql
-- Crear un esquema llamado "library"
CREATE SCHEMA library;

-- Crear una tabla en el esquema "library"
CREATE TABLE library.books (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255),
    author VARCHAR(255),
    published_date DATE
);
```

En este ejemplo:

- El esquema library actúa como un contenedor que agrupa todas las tablas relacionadas con la librería.
- La tabla books está organizada dentro del esquema library.

Para consultar esta tabla, tendríamos que referirnos a ella como library.books.

Resumen

Un esquema en una base de datos es la estructura organizativa que define cómo se almacenan y organizan los datos. Sirve para agrupar tablas y otros objetos, facilitando la organización, seguridad, y acceso a los datos. La implementación y uso del concepto de esquema pueden variar según el sistema de base de datos.