**PRISMA ORM**

**ORM** (Object-Relational Mapping) refers to a programming technique that simplifies database interaction by allowing developers to work with objects in their code instead of writing SQL queries.

Example; SQLAlchemy (python), Sequelize (Javascript), Hibernate (Java) and Prisma

**Advantages of Using ORM Tools**

* It speeds up development time for teams.
* Decreases the cost of development.
* Handles the logic required to interact with databases.
* Improves security. ORM tools are built to eliminate the possibility of SQL injection attacks.
* You write less code when using ORM tools than with SQL.

**Disadvantages of Using ORM Tools**

* Learning how to use ORM tools can be time consuming.
* They are likely not going to perform better when very complex queries are involved.
* ORMs are generally slower than using SQL.

**PRISMA**

Prisma is a modern ORM (Object-Relational Mapper) for Node.js and TypeScript.

**Steps to use Prisma﻿**

1. **Install Prisma as a dev dependency.**

**npm install prisma -D**

or

npm install prisma --save-dev

1. **Set Up prisma**

npx prisma init --datasource-provider DATABASE

Replace DATABASE with the database that you are using, assuming you are using postgresql:

npx prisma init --datasource-provider postgresql

Prisma uses postgresql by default, so if your database is postgresql, you can run the command without passing --datasource-provider flag.

npx prisma init

1. **Create your model(s).**

Models represent the different tables in your database.  
**Basic Prisma Model Structure**

model User {

id Int @id @default (autoincrement ())

name String

email String @unique

posts Post [ ]

createdAt DateTime @default (now ())}

**Key concepts of Prisma Model**

1. Data Models: Define the tables and their columns (fields) in your database.
2. Field Types: Specify the data types for each field (e.g., String, Int, Boolean).
3. Relations: Describe how models relate to each other using relation fields.
4. Attributes: Add additional metadata to fields and models using attributes like @id, @default, @relation.
5. **Run migrations**

In Prisma, migrations are a way to manage and apply changes to your database in a controlled and consistent way.  
npx prisma migrate dev --name MIGRATION-NAME #To run Migration

1. **Generate the client**

The Prisma Client is an auto-generated and type-safe database client that you use to interact with your database in a Node.js or TypeScript application.

npx prisma generate

If it is not already installed, run the command:

npm install @prisma/client

After we create the client, Prisma gives us a set of APIs to help us perform **CRUD operations.**

**CRUD Operations**

* 1. **Create**

In Prisma, the create operation is used to insert a new record into the database.

**Single Record Creation**

(create ()) - The following query creates a single user with two fields

**Multiple Record Creation**

createMany () - The following query creates multiple users.

* 1. **Read**

**Get record by ID or unique identifier**

(findUnique()) - The following queries return a single record by unique identifier or ID

**Get all records**

**findMany() -** The following queries returns all User records

**Get the first record that matches a specific criteria**

findFirst() - The following queries returns the most recently created user.

* 1. **Update**

update() – The following query updates a single user record

updateMany() – The following query updates all user records

updateManyAndReturn() – The following query updates many records and returns the resulting objects

* 1. **Delete**

delete() – the following query deletes a single user record

deleteMany() – The following query deletes all user records

**RELATIONSHIPS**

A relation is a connection between two models in the Prisma schema

Relationships define how different models (tables) in a database are connected to each other.

The types of relationships include:

* one-to-one relationship (1-1): means that one record in a table is associated to only one record in another table.  
  example: A User has **one** Profile, and each Profile belongs to **one** User.
* one-to-many relationship (1-n): means that one record in a table can be associated with multiple other records in another table.  
  example: A User has **many** posts, and each post belongs to **one** User.
* many-to-many relationship (m-n): means that multiple records in one table can be associated to multiple records in another table.  
  example: A student can enroll in **many** courses, and each course can have **many** students.