Prisma ORM

An ORM (Object-Relational Mapping) is a technique or a tool that allows developers to interact with a database using an object-oriented approach, rather than writing SQL queries.

Some examples of ORMs include:

- SQLAlchemy (Python)
- Sequelize (JavaScript)
- Hibernate (Java)
- Prisma
- Drizzle

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.
2. Set Up prisma project by creating a schema file.
3. Create your model(s).
4. Run migrations to reflect changes in your database.
5. Generate the client and use it in your project.

Install Prisma

```
npm install prisma -D or npm install prisma --save-dev
```

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
```

```
if you are using postgresql simply run:- npx prisma init
```

function of npx prisma init

- Creates Prisma folder with schema.prisma contains connection for our DB

- Creates .env used for defining the environment variables

.env comes with the below url for easy configuration:-

```
DATABASE_URL="postgresql://johndoe:randompassword@localhost:5432/mydb?
schema=public"
```

Models

Models represent the different tables in your database.

Models are made up of fields (fields correspond to the columns in our database).

Things to Note in Prisma Models

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.

Fields are made up of:-

- Field name – required
- Data type of the field (required) eg String, int etc
- Field type modifier (no necessary)
- Attributes (no necessary)

NB

Field Modifiers are used to indicate only two things:

- Whether a field is optional(no necessary), in which case we use `?`.
- Whether a field can contain multiple values in which case we use `[]`.

Field types:-

i. String – text data

ii. Int – numbers (32-bit)

iii. BigInt – numbers(64-bit)

iv. Float – fractions

v. Boolean – True or False data

vi. DateTime

Field Attributes

- @id: primary key of the model.
- @default: sets a default value for the field
- @unique: ensures that a field has unique values across all rows in a table.
- @updatedAt: automatically updates a field to the current timestamp
- @map: maps the field to a column with a different name in the database
- @relation: defines relationships between models.
- @@id: it is a model attribute, and it defines a composite primary key using multiple fields
- @@unique: it is a model attribute, and it ensures a unique constraint on a combination of multiple fields.
- @@map: it is a model attribute, and it maps the model to a table with a different name in the database.

Migrations

are a way to manage and apply changes to your database in a controlled and consistent way.

How to run migrations:- `npx prisma migrate dev --name MIGRATION-NAME`

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.

Use this command to generate a client:- `npx prisma generate`

if `@prisma/client not installed automattically use this command:-`
`npm install @prisma/client`

CRUD Operations - Create

Creating single record using create()

```
import { PrismaClient } from '@prisma/client';
const client = new PrismaClient();

const createUser = async () => {
    const newUser = await client.User.create({
        data: {
            userName: "Cyruson",
            ethnicity: "Kamba",
```

```
            age: 23
        }
    })

    console.log(newUser);
}
```

CRUD operations - Read

Get all records using findMany()

```
import { PrismaClient } from "@prisma/client";
const client = new PrismaClient();

const getUser = async () => {
  const User = await client.User.findMany();
  console.log(User)
};
getUser();
```

comparison operators in Prisma

Example- consider Students height in (feets)

- equals: matches exactly eg `{ height: { equals: 4 } }`.
- not: matches 'not equal to the specified value' eg `{ height: { not: 4 } }`
- lt: less than eg `{ height: { lt: 5 } }`.
- lte: less than or equal to eg `{ height: { lte: 6 } }`.
- gt: greater than eg `{ height: { gt: 6 } }`.
- gte: greater than or equal to eg `{ height: { gte: 5 } }`.
- equals: matches exactly (case-sensitive) eg `{ name: { equals: "John" } }`
- contains: matches if the string contains the specified substring
- startsWith: matches if the string starts with the specified string
- endsWith: matches if the string ends with the specified string
- not: negates the condition

Relationships

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

- **one-to-one relationship (1-1)**:- one record in a table is associated to only one record in another table.
- **one-to-many relationship (1-n)**:- one record in a table can be associated with multiple other records in another table.

- **many-to-many relationship (m-n)**:- multiple records in one table can be associated to multiple records in another table.

one-to-one relationships

```
model Employee {
  id        String @id @default(uuid())
  name      String
  position  String
  workstation Workstation?

  @@map("employees")
}
model Workstation {
  computerId String @unique
  location   String
  employeeId String @unique
  employee   Employee @relation(fields: [employeeId], references: [id])

  @@map("workstations")
}
```

one-to-many relationships

```
model Department {

  id       String    @id @default(uuid())

  name     String

  employees Employee[]

  @@map("departments")

}

model Employee {

  id           String     @id @default(uuid())

  name         String

  position     String

  departmentId String

  department   Department @relation(fields: [departmentId], references: [id])

  @@map("employees")

}
```