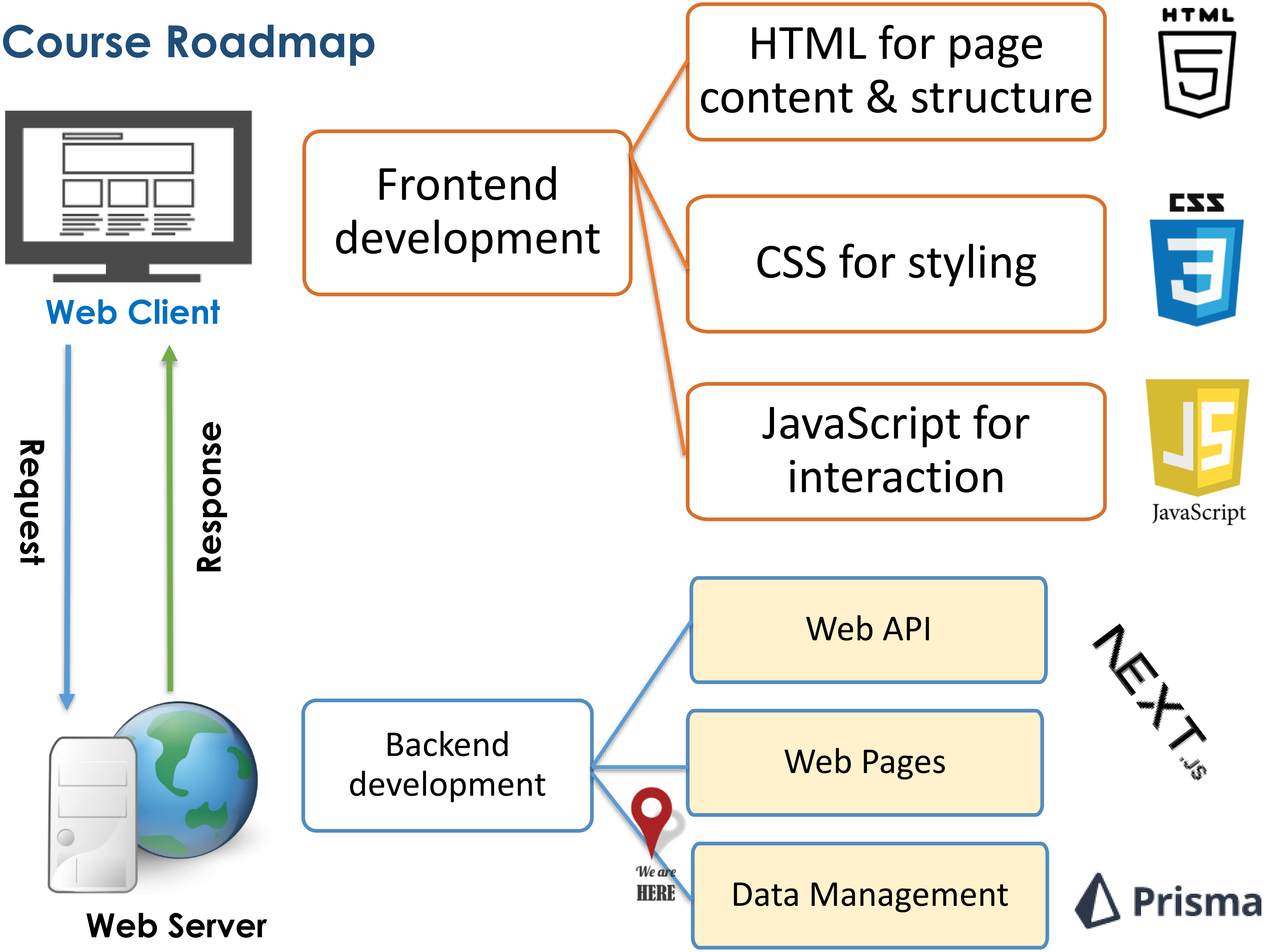
Data Management using

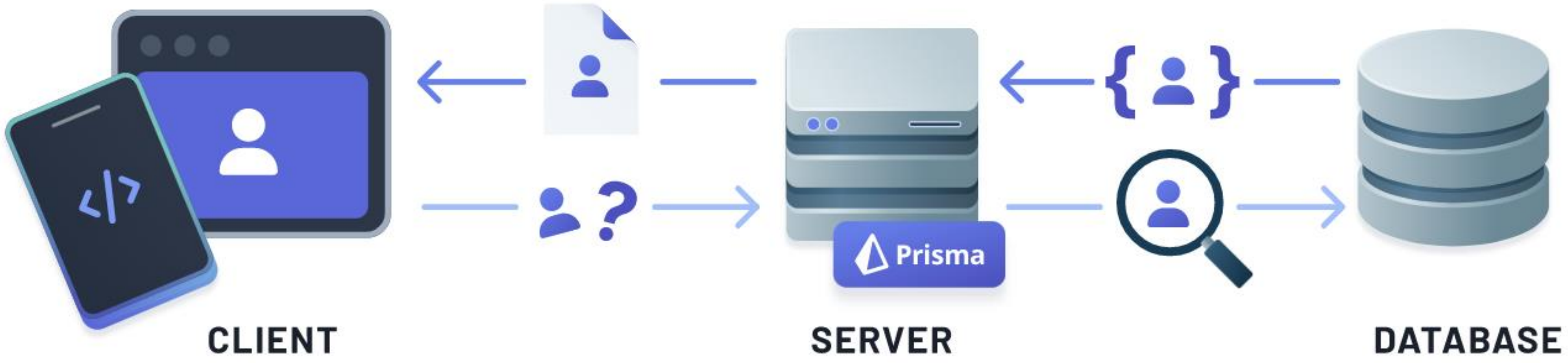
Prisma

Course Roadmap



Outline

What is Prisma ?



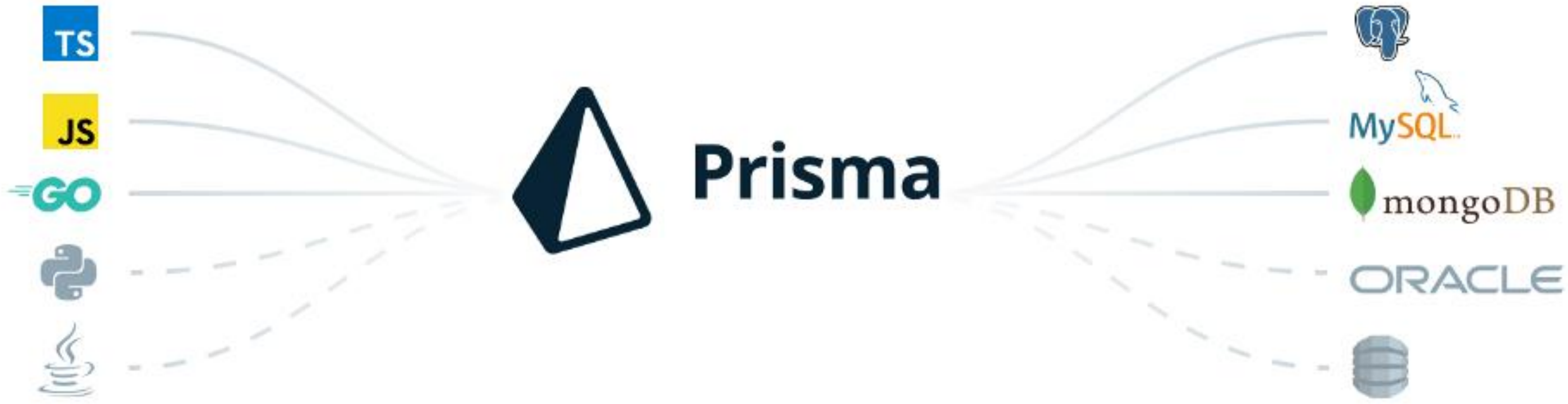
What is Prisma?

- Prisma is a server-side library that simplifies read and write data to the database in an intuitive and efficient way
- Open-source Object-Relational-Mapper (ORM), includes:
  - **Prisma Schema**: used to define the **data model** (entities and relations)
  - **Prisma Migrate**: apply schema changes to DB
  - **Prisma Client**: auto-generated to query data
  - **Prisma Studio**: GUI to view and edit data in your DB
- Why Prisma?
  - Facilitates defining the data model
  - Helps reducing the amount of code to read/write to a DB
  - Less or no SQL code to read/write to a DB
  - Abstract database-specific details => makes easier to change from one database to another

schema.prisma

- **Data Model** is defined in 1 file (**schema.prisma**)
  - Specifies the app entities and their relations
  - Syntax used is Prisma Schema Language (PSL)
- schema.prisma also specifies:
  - **Data source**: defines the data source details:
    - Database Provider (e.g., a PostgreSQL or SQLite)
    - Connection Url (e.g., postgresql://janedoe:mypassword@localhost:5432/mydb)
  - **Generator**: specifies what client should be generated based on the data model (e.g., Prisma Client)

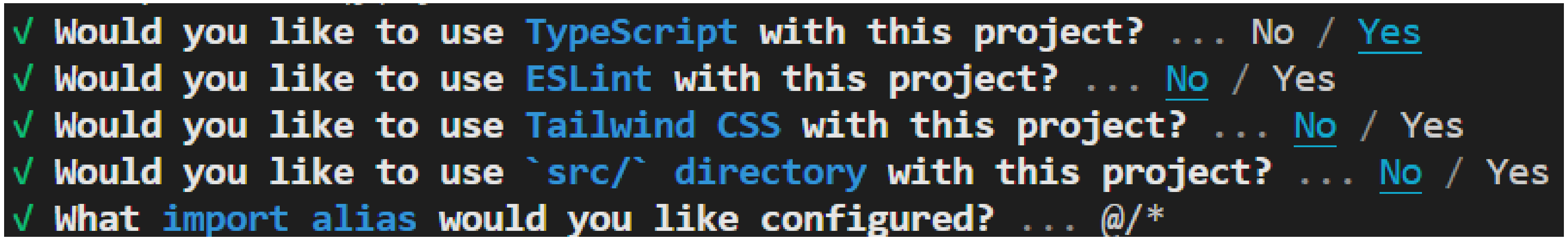
Prisma DB providers



Reminder – Next.js getting started

- Create an empty folder (with no space in the name use **dash -** instead)
- Create next.js app (select **No** for all questions except for **TypeScript select Yes**)

```
npx create-next-app@latest --experimental-app .
```



```
√ Would you like to use TypeScript with this project? ... No / Yes
√ Would you like to use ESLint with this project? ... No / Yes
√ Would you like to use Tailwind CSS with this project? ... No / Yes
√ Would you like to use `src/` directory with this project? ... No / Yes
√ What import alias would you like configured? ... @/*
```

This creates a new **`Next.js`** project and downloads all the required packages

- Run the app in dev mode: `npm run dev`

Prisma – Getting started

- Install the Prisma CLI as a development dependency in the project:

```
npm install prisma --save-dev
```

- Also install **Prisma VS Code Extension**
- Set up Prisma with the init command:

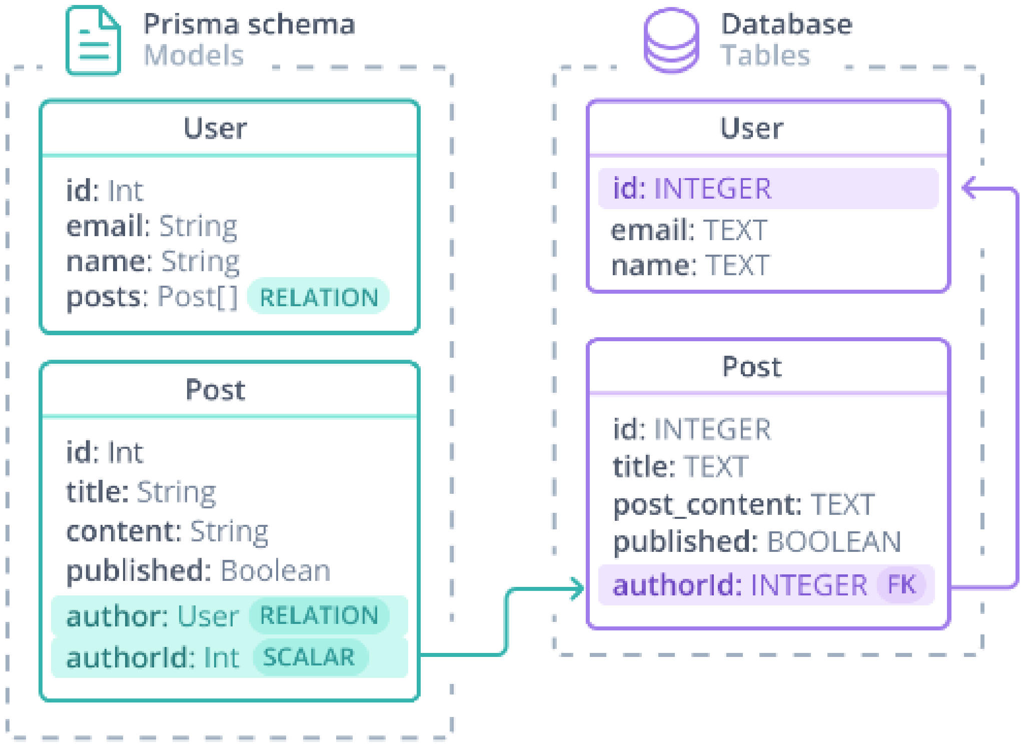
```
npx prisma init --datasource-provider sqlite
```

  - This creates a new prisma directory with your **Prisma schema** file and configures SQLite as your database
  - You can define the data model inside **Prisma schema** file

Data Model

Data Model

- Data Model (aka. Schema) have two main purposes:
  - Represent the tables in the underlying database: Data Model is used to create the database tables using **Prisma Migrate**
  - Serve as foundation to generate **Prisma Client** API

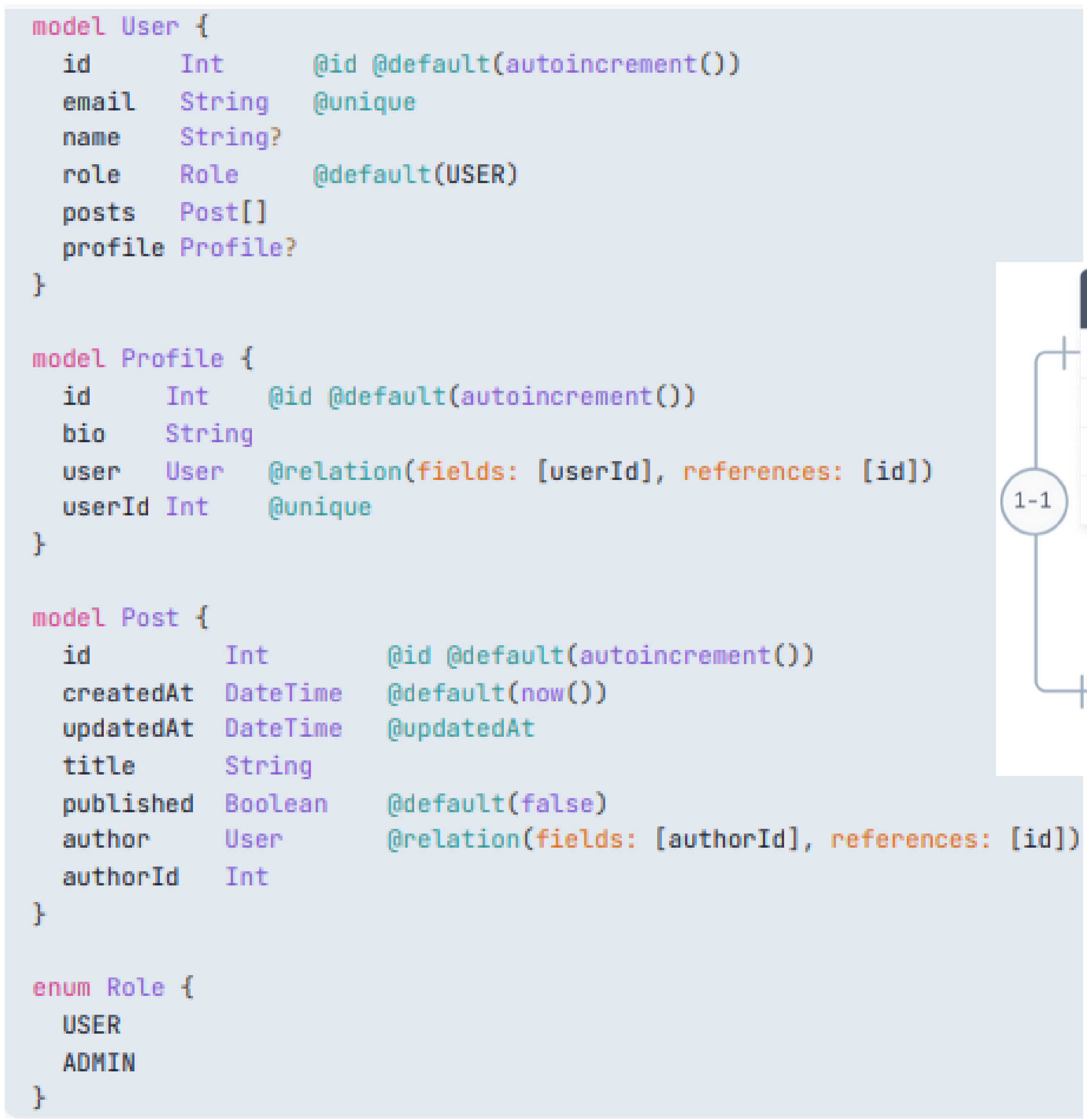


Defining fields

- Each model entity defines fields
- Each field in the model has a type, e.g. the **`id`** has the type **`Int`**
  - A field type could be scalar type such as Int, String, Boolean or could be another Model
  - Optional type modifiers: **[]** Makes a field a list

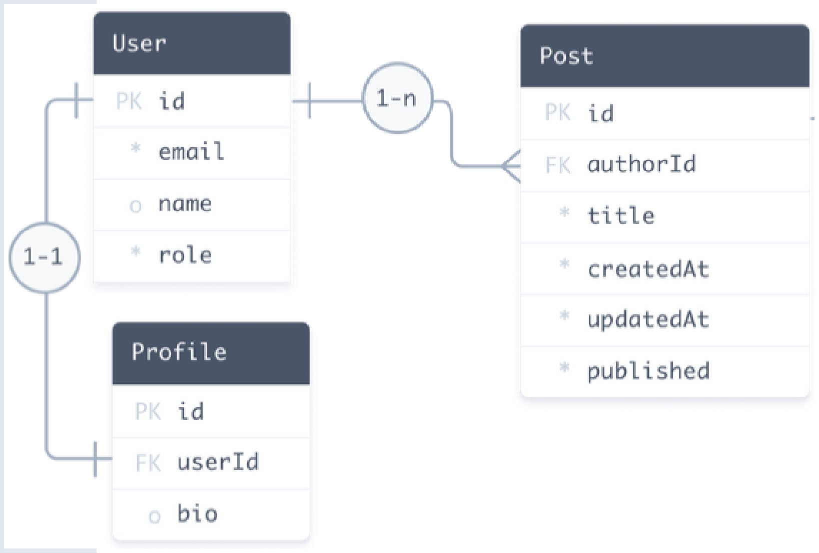
  ? Makes a field optional
- Fields may contain field attributes to define:
  - Primary keys with the **`@id`** attribute
  - Unique keys with the **@unique** attribute
  - Default values with the **@default** attribute
- More info

Data Model Example



```
model User {
  id      Int      @id @default(autoincrement())
  email   String   @unique
  name    String?
  role    Role     @default(USER)
  posts   Post[]
  profile Profile?
}

model Profile {
  id     Int    @id @default(autoincrement())
  bio    String
  user   User   @relation(fields: [userId], references: [id])
  userId Int    @unique
}

model Post {
  id        Int      @id @default(autoincrement())
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  title     String
  published Boolean  @default(false)
  author    User     @relation(fields: [authorId], references: [id])
  authorId  Int
}

enum Role {
  USER
  ADMIN
}
```

DB Schema

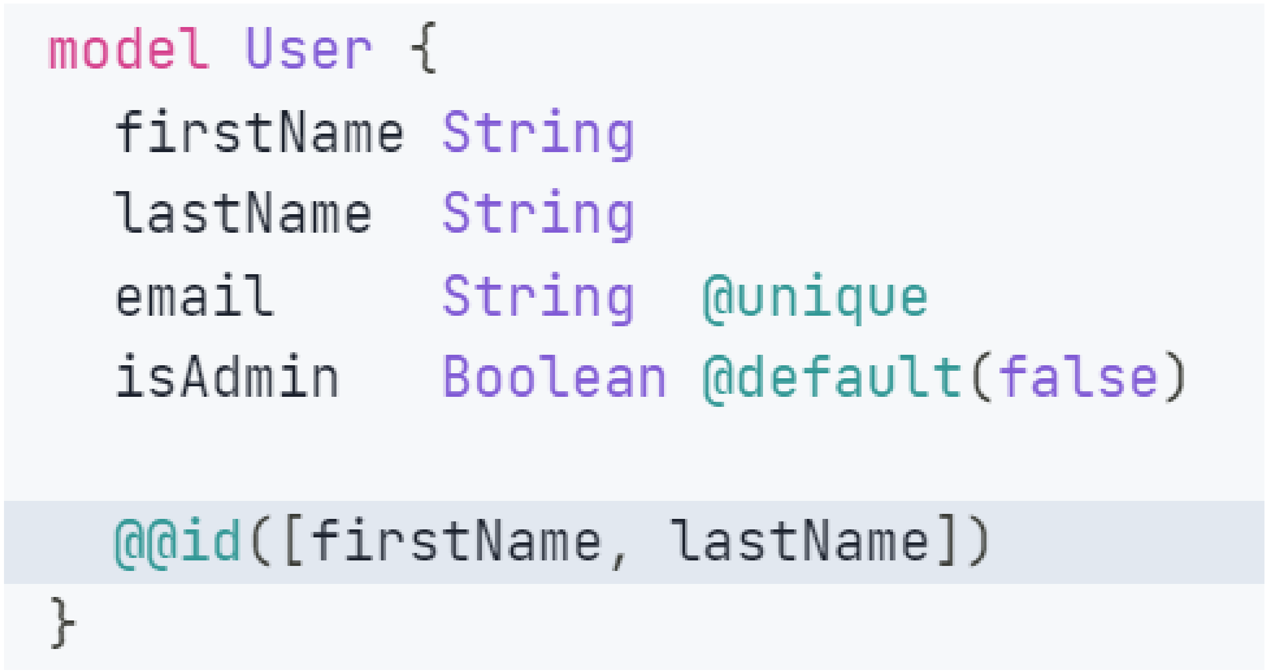


Modeling relations

- User / Post relation is made up of:
  - The scalar **authorId** field, which is referenced by the `@relation` attribute, is the foreign key that connects **Post** and **User**
  - The two relation fields: **author** and **posts** do not exist in the database table.
    - Relation fields define connections between models at the Prisma level and exist only in the Prisma schema and generated Prisma Client, where they are used to access the relations

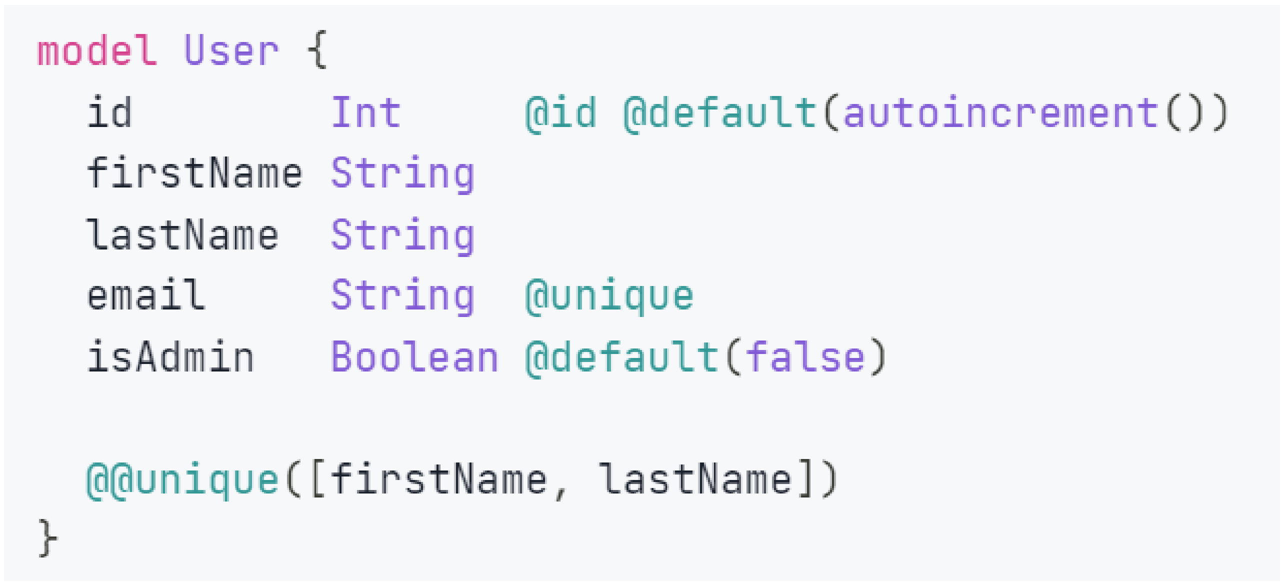
@@unique & @@id

Composite primary key



```
model User {
  firstName String
  lastName  String
  email     String  @unique
  isAdmin   Boolean @default(false)

  @@id([firstName, lastName])
}
```

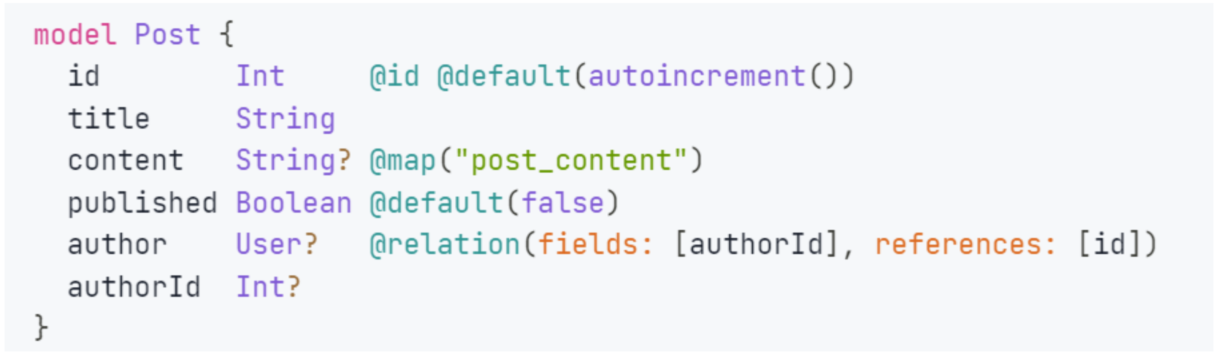
Composite Unique key



```
model User {
  id        Int     @id @default(autoincrement())
  firstName String
  lastName  String
  email     String  @unique
  isAdmin   Boolean @default(false)

  @@unique([firstName, lastName])
}
```

@map

- By default, model filed names are the same as the DB table column names
- @map attribute can be used for mapping between model fields and table columns
  - e.g., the **`content`** field maps to the `post_content` database column



```
model Post {
  id        Int      @id @default(autoincrement())
  title     String
  content   String? @map("post_content")
  published Boolean @default(false)
  author    User?    @relation(fields: [authorId], references: [id])
  authorId  Int?
}
```

Migration
(Apply changes to DB)

Migration

- Prisma Migrate auto-generates SQL migrations from the Prisma schema:
  - Keep your database schema in sync with your Prisma schema
  - Maintain existing data in your database

Workflow:

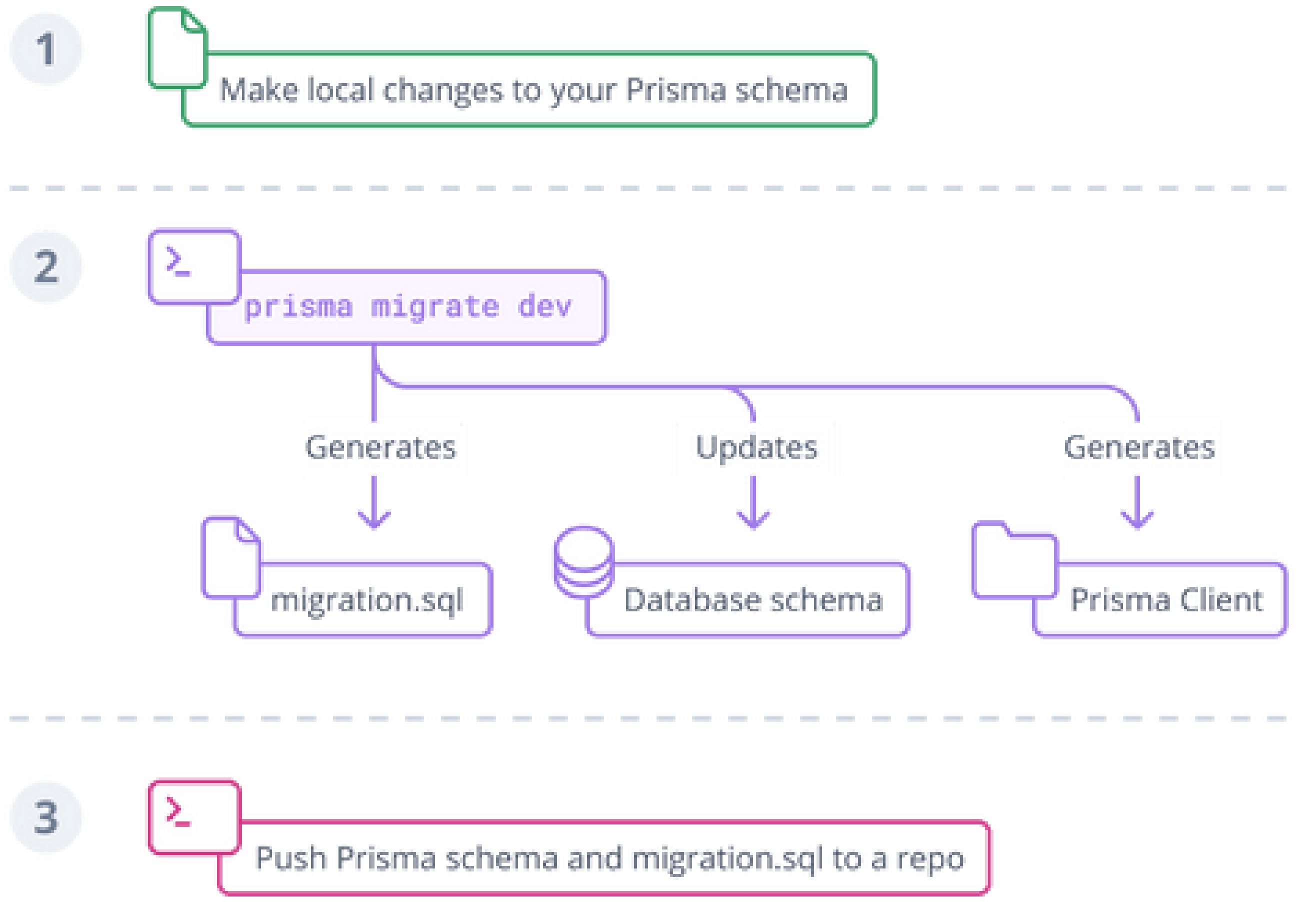
1. Update your Prisma schema
2. Run migration command: `npx prisma migrate dev`
   a. It checks your db and your schema
   b. It creates a sql file to apply the changes

Prisma migrate

```
npx prisma migrate dev --name init
```

- This command did 3 things:
  - It creates a new SQL migration file under `prisma/migrations` directory
  - It runs the SQL migration file against the database
  - Generates Prisma Client
- Because the SQLite database file didn't exist before, the command also created it inside the prisma directory with the name **dev.db** as defined via the environment variable in the .env file

Prisma migrate



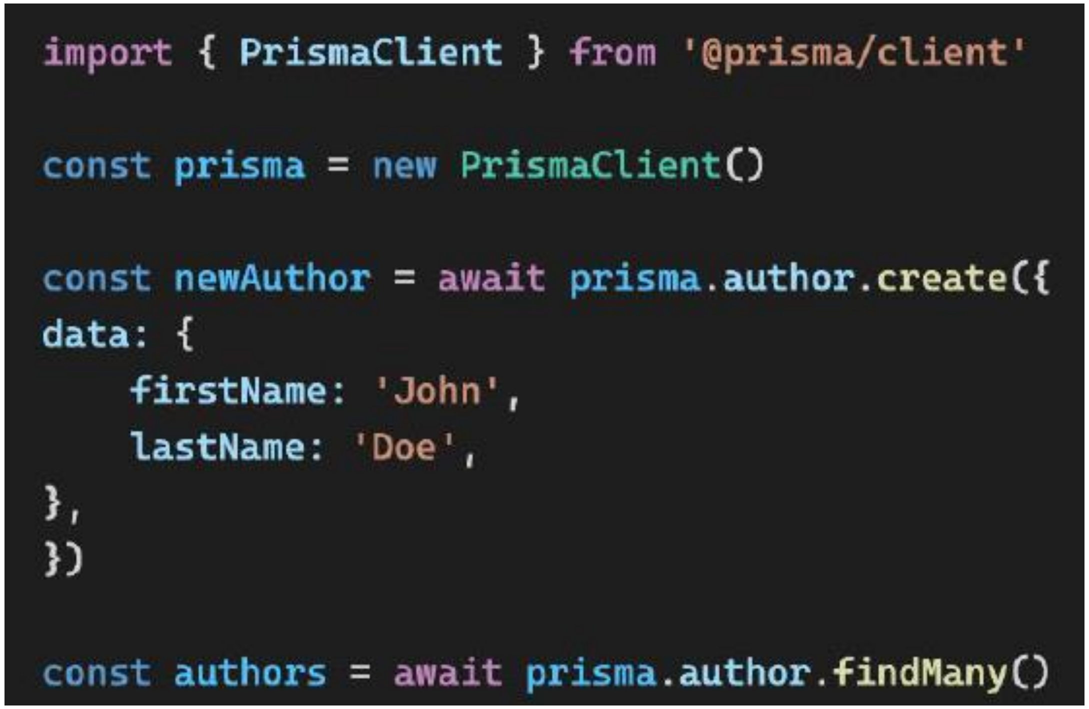
Queries
(using Prisma Client)

Prisma Client

- Run `npx prisma migrate` or `npx prisma generate`

To generate a Prisma Client that is tailored to data models defined in **schema.prisma**

Offers auto-completion to help write the queries to read/write to DB



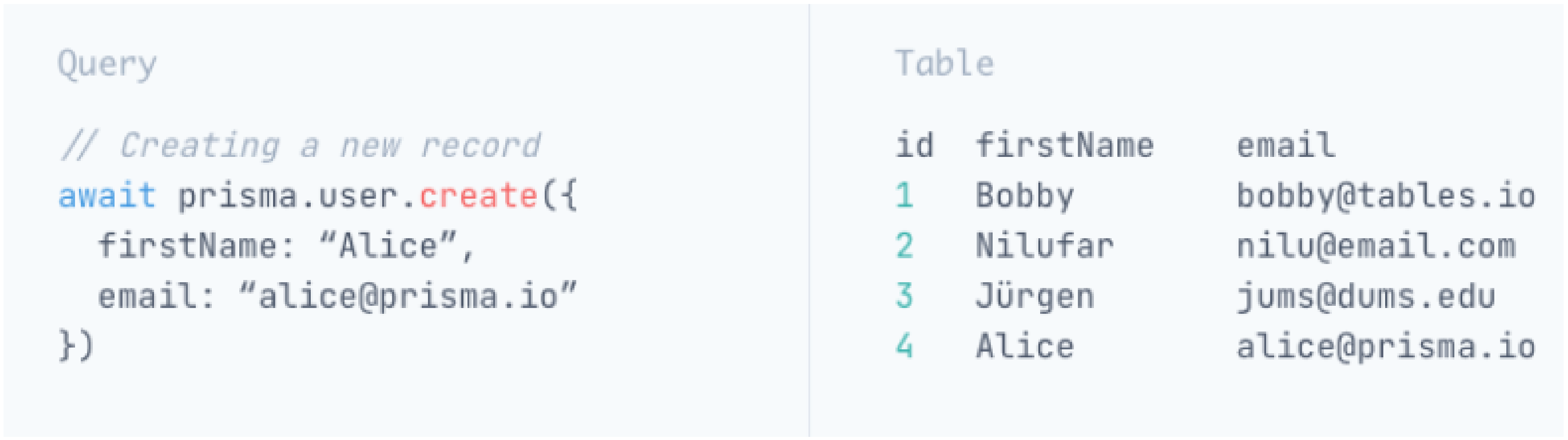
```
import { PrismaClient } from '@prisma/client'

const prisma = new PrismaClient()

const newAuthor = await prisma.author.create({
data: {
    firstName: 'John',
    lastName: 'Doe',
},
})

const authors = await prisma.author.findMany()
```

DB Operations

Prisma client offers the following operations for each model:

- `create/createMany`
- `update/updateMany`
- `delete/deleteMany`
- `findUnique/findMany/findFirst`
- `aggregate`
- `count`
- `groupBy`
- `upsert (create or update)`

Example Query

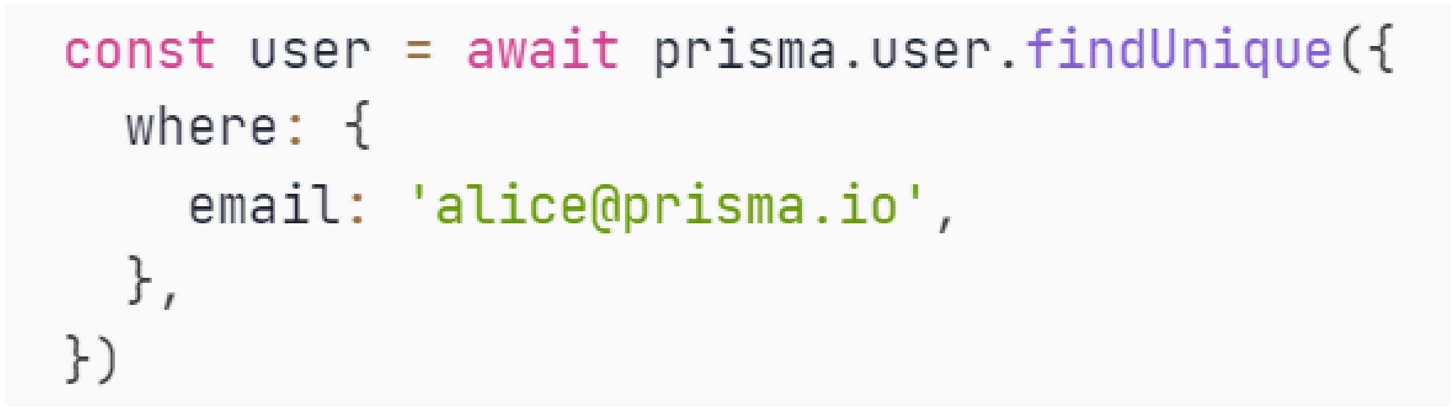


Query

```
// Creating a new record
await prisma.user.create({
  firstName: “Alice”,
  email: “alice@prisma.io”
})
```

Table

id	firstName	email
1	Bobby	bobby@tables.io
2	Nilufar	nilu@email.com
3	Jürgen	jums@dums.edu
4	Alice	alice@prisma.io

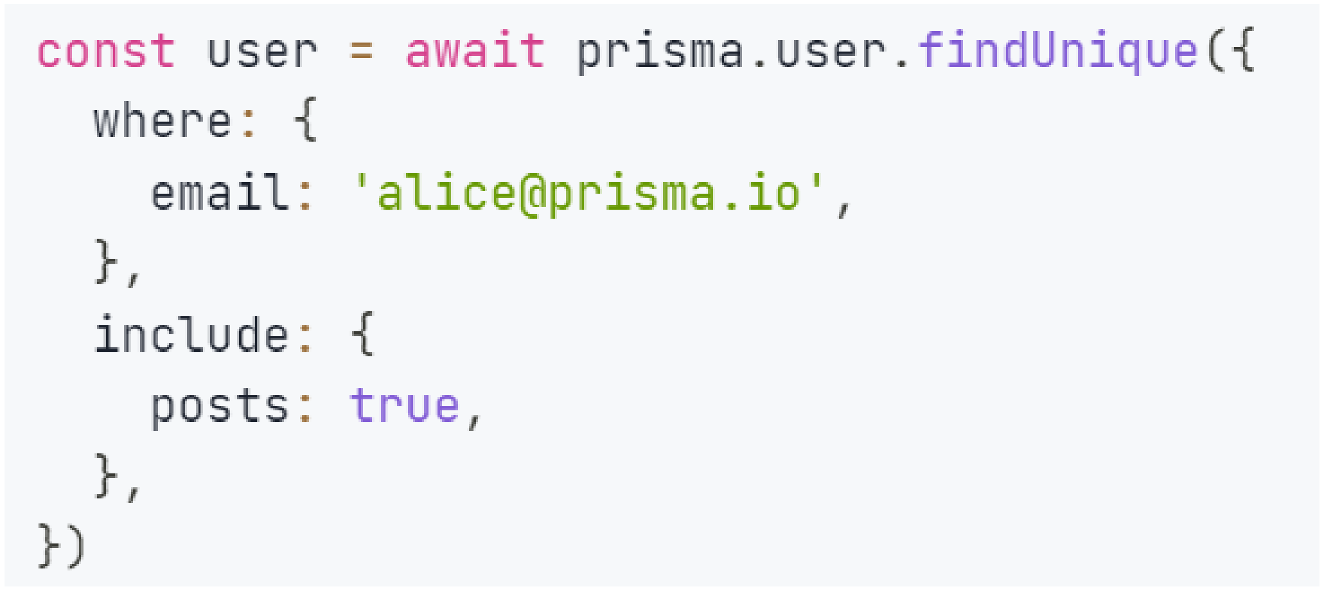


```
const user = await prisma.user.findUnique({
  where: {
    email: 'alice@prisma.io',
  },
})
```

- All queries return plain old JavaScript objects

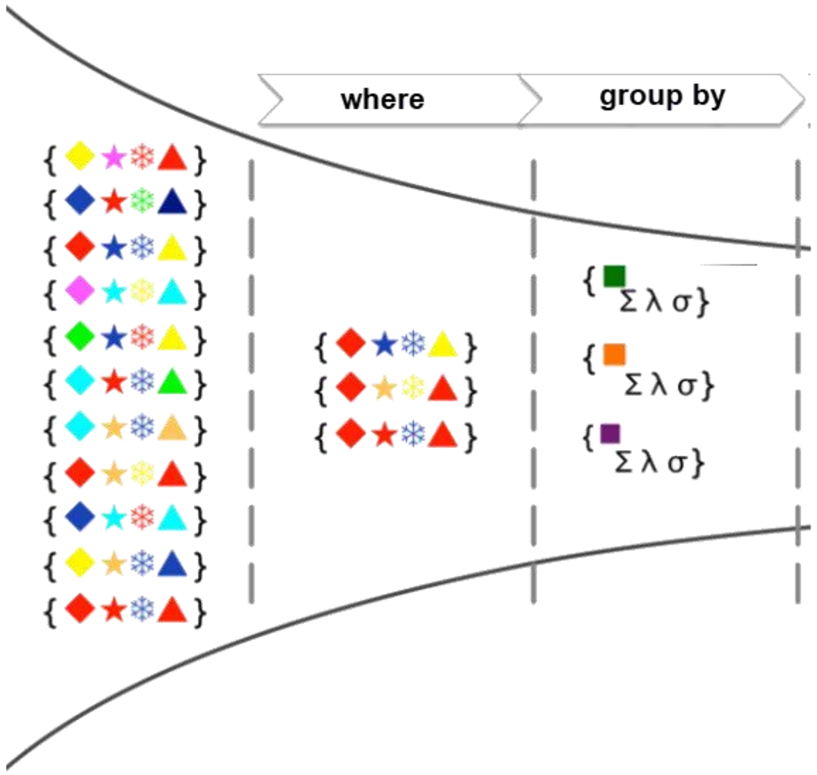
Fetching relations

- By default, Prisma will return all the scalar fields of a model
- Fetch relations with Prisma Client is done with the **include** option. For example, to fetch a user and their posts would be done as follows:



```
const user = await prisma.user.findUnique({
  where: {
    email: 'alice@prisma.io',
  },
  include: {
    posts: true,
  },
})
```

Aggregation Queries



Aggregation Queries

- Summarize data typically for reports
- How would we solve this in SQL?

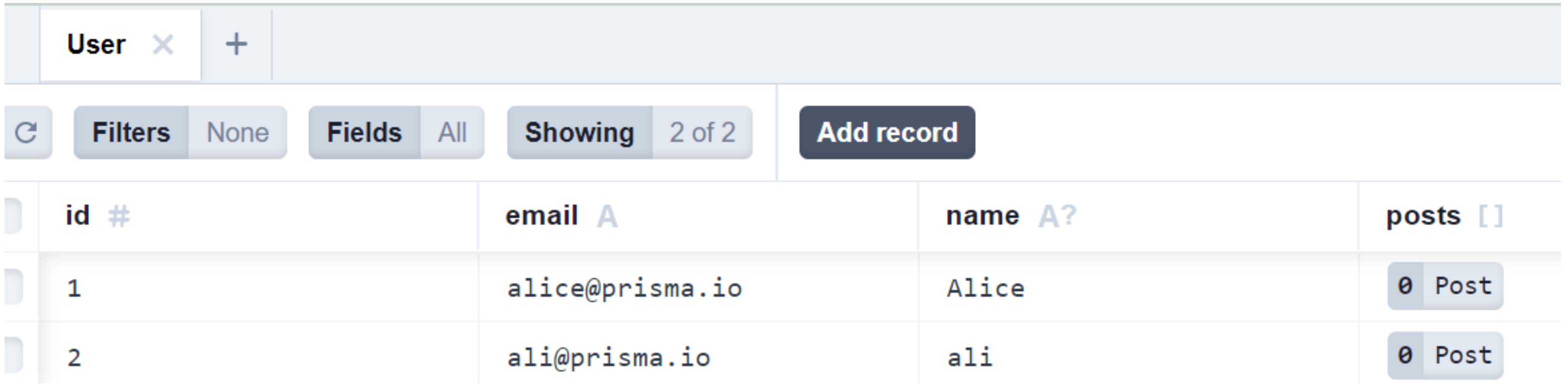
```
SELECT GROUP BY HAVING
```

- **To do** – more info

Prisma Studio

- GUI to view, explore and edit the data in the DB
  - Browse across tables, filter, paginate, traverse relations and edit data

```
npx prisma studio
```



id #	email A	name A?	posts []
1	alice@prisma.io	Alice	0 Post
2	ali@prisma.io	ali	0 Post

DB Seeding

- Allows initialing the database with some data
  - Add DB init code seed.js file
  - Run it using: **`npx prisma db seed`**
- **ToDo** – more info

Resources

- Prisma Documentation

https://www.prisma.io/docs/getting-started/quickstart

- Prisma Playground

https://playground.prisma.io/examples/

- Prisma Examples

https://github.com/prisma/prisma-examples

- Aggregation Queries

https://www.prisma.io/docs/concepts/components/prisma-client/aggregation-grouping-summarizing