Stitch

Symfony

Other

Create a password

For security, each database password can connect to one branch.

The password **main-2023-09-21-pqf1v9** has been created

Username

```
yp8e38nz1zwfgba98g8s
```

Password

```
pscale_pw_Bgvqz0q9MkbHtrmqZnIgqeoR02cf1m6u0ZbuaQkS7go
```

Be sure to copy your password. It cannot be displayed again after initial creation.

```
DATABASE_URL='mysql://yp8e38nz1zwfgba98g8s:
pscale_pw_Bgvqz0q9MkbHtrmqZnIgqeoR02cf1m6u0ZbuaQkS7go@aws.connect.psdb.cloud/ecommerce-ad
```

Configure your Prisma application

To connect to PlanetScale from Prisma, a TypeScript and JavaScript ORM, you can use the `@prisma/client` package.

Installation

First, you will need to install the `prisma` package:

Command line

```
npm install prisma
```

Next, set up Prisma in your application with the following command:

Command line

```
npx prisma init
```

Add credentials to .env

For local development, you can place your credentials in a `.env` file. For production, we recommend setting your credentials as environment variables wherever your application is deployed.

.env

```
DATABASE_URL='mysql://:@/?sslaccept=strict'
```

Connecting and querying

Creating your schema

Next, update your `prisma/schema.prisma` file to use the `mysql` provider and set the relation mode type to `prisma`:

prisma/schema.prisma

```
datasource db {
  provider     = "mysql"
  url          = env("DATABASE_URL")
  relationMode = "prisma"
}

generator client {
```

```
  provider = "prisma-client-js"
}
```

Now, you can write your Prisma models or modify the existing ones. See Prisma documentation on Prisma schema to learn more.

Finally, once you are ready to push your schema to PlanetScale, run `prisma db push` against your PlanetScale database to update the schema in your database:

Command line

```
npx prisma db push
```

Using the Prisma client

Once you have created a schema and are ready to connect and query from your JavaScript or TypeScript application, it is time to use the Prisma client. When you run `prisma db push`, it generates your Prisma client so you can use it in your code.

JavaScript

src/index.js

```
const { PrismaClient } = require('@prisma/client')

const prisma = new PrismaClient()

async function main() {
  // ... you will write your Prisma Client queries here
}

main()
  .then(async () => {
    await prisma.$disconnect()
  })
  .catch(async (e) => {
    console.error(e)
    await prisma.$disconnect()
    process.exit(1)
  })
```

TypeScript

src/index.ts

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

const prisma = new PrismaClient()

async function main() {
  // ... you will write your Prisma Client queries here
}

main()
  .then(async () => {
    await prisma.$disconnect()
  })
  .catch(async (e) => {
    console.error(e)
    await prisma.$disconnect()
    process.exit(1)
  })
```

See the Prisma documentation for using the Prisma Client to read and write data.

Learn more

There are some best practices to follow when using Prisma and PlanetScale together; see our Prisma best practices documentation for more info. Some examples include creating indexes when using `@relation` in your models and using `prisma db push` instead of `prisma migrate dev`.

Go to your database overview