all prisma commands with examples

Sure! Prisma is a popular Node.js ORM (Object-Relational Mapping) tool that helps you interact with databases using a type-safe API. Prisma has a **CLI** (`prisma`) with multiple commands for different tasks: database schema management, migrations, generating client code, and more. I’ll provide a **full list of Prisma commands with examples**.

---

1. `prisma init`

Initializes a new Prisma project.

Example:

bash

```bash
npx prisma init
```

Copy code

This creates:

- `prisma/schema.prisma` – The database schema
- `.env` – Environment variables for database connection

---

2. `prisma generate`

Generates the Prisma Client based on your schema.

Example:

bash

```bash
npx prisma generate
```

Copy code

This must be run whenever you update `schema.prisma`.

---

3. `prisma migrate`

Manages database migrations. It has several subcommands:

a) `prisma migrate dev`

Applies migrations in development and updates the database.

Example:

bash

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

Copy code

b) `prisma migrate deploy`

Applies all pending migrations in production.

Example:

bash

```
npx prisma migrate deploy
```

Copy code

c) `prisma migrate reset`

Resets the database (drops all data, re-applies migrations).

Example:

bash

```bash
npx prisma migrate reset
```

Copy code

d) `prisma migrate status`

Shows the status of migrations.

Example:

bash

```bash
npx prisma migrate status
```

Copy code

---

4. `prisma db`

Handles database-related tasks. Subcommands:

a) `prisma db pull`

Introspects your database and updates `schema.prisma` .

Example:

bash

```
npx prisma db pull
```

Copy code

b) `prisma db push`

Pushes the schema changes to the database **without a migration**.

Example:

bash

```
npx prisma db push
```

Copy code

c) `prisma db seed`

Runs your seed script (defined in `package.json` or `prisma/seed.ts`).

Example:

bash

```
npx prisma db seed
```

Copy code

---

5. `prisma studio`

Opens a GUI to view and edit your database.

Example:

bash

```
npx prisma studio
```

Copy code

---

6. `prisma validate`

Validates your `schema.prisma` file.

Example:

bash

```
npx prisma validate
```

Copy code

---

7. `prisma format`

Formats the `schema.prisma` file according to Prisma conventions.

Example:

bash

```
npx prisma format
```

Copy code

---

8. `prisma version`

Displays the installed Prisma CLI and client versions.

Example:

bash

```bash
npx prisma version
```

Copy code

---

9. `prisma migrate` vs `db push`

- Use **migrate** if you want **versioned migrations** (recommended for production).
- Use **db push** if you want **quick schema changes** without migration history (good for dev/testing).

---

10. Using Prisma Client in Code

After `prisma generate`, you can use Prisma Client like this:

ts

```ts
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

async function main() {
  const users = await prisma.user.findMany();
  console.log(users);
}

main()
  .catch(e => console.error(e))
  .finally(async () => await prisma.$disconnect());
```

Copy code

💡 Quick command cheat sheet:

Command	Description
`prisma init`	Initialize project
`prisma generate`	Generate Prisma client
`prisma migrate dev`	Apply dev migrations
`prisma migrate deploy`	Apply prod migrations
`prisma migrate reset`	Reset database
`prisma migrate status`	Show migration status
`prisma db pull`	Introspect DB
`prisma db push`	Push schema to DB
`prisma db seed`	Run seed script
`prisma studio`	GUI for DB

Command	Description
`prisma validate`	Validate schema
`prisma format`	Format schema
`prisma version`	Show version

If you want, I can also make a **visual diagram showing the workflow of all Prisma commands** so you can see how `db push` , `migrate` , and `generate` relate. It's a huge time-saver for remembering what does what.

Do you want me to make that diagram?

am talking of funtions such as create()

Ah! Now I understand — you're asking about **Prisma Client methods** like `create()` , `findMany()` , etc., not the CLI commands. These are the **functions you use in your code** to interact with the database. I'll give a **full overview with examples** for each main method.

---

1. `create()`

Creates a new record in the database.

ts

```ts
const newUser = await prisma.user.create({
  data: {
    name: "Alice",
    email: "alice@example.com",
    age: 25,
  },
});
console.log(newUser);
```

Copy code

---

2. `createMany()`

Creates multiple records at once.

ts

```ts
const users = await prisma.user.createMany({
  data: [
    { name: "Bob", email: "bob@example.com", age: 30 },
    { name: "Charlie", email: "charlie@example.com", age: 28 },
  ],
});
console.log(users); // { count: 2 }
```

Copy code

---

3. `findMany()`

Fetches multiple records, optionally with filters.

ts

```ts
const users = await prisma.user.findMany({
  where: { age: { gt: 20 } },  // users older than 20
  orderBy: { age: "desc" },    // sort by age descending
  take: 5,                     // limit to 5 users
});
console.log(users);
```

Copy code

---

4. `findUnique()`

Fetches a single record by a unique field (like `id` or `email`).

ts

```ts
const user = await prisma.user.findUnique({
  where: { email: "alice@example.com" },
});
console.log(user);
```

Copy code

> ⚠️ Use `findUniqueOrThrow()` if you want it to throw an error when not found.

ts

```ts
const user = await prisma.user.findUniqueOrThrow({
  where: { id: 1 },
});
```

Copy code

---

5. `findFirst()`

Fetches the first record matching a filter.

ts

```ts
const user = await prisma.user.findFirst({
  where: { age: { gte: 18 } },
  orderBy: { name: "asc" },
});
```

Copy code

---

6. `update()`

Updates a single record identified by a unique field.

ts

```ts
const updatedUser = await prisma.user.update({
  where: { email: "alice@example.com" },
  data: { age: 26 },
});
console.log(updatedUser);
```

Copy code

---

7. `updateMany()`

Updates multiple records at once.

ts

```ts
const result = await prisma.user.updateMany({
  where: { age: { lt: 30 } },
  data: { isActive: true },
});
console.log(result); // { count: X }
```

Copy code

---

8. `upsert()`

Creates a record if it doesn’t exist, or updates it if it does.

ts

```ts
const user = await prisma.user.upsert({
  where: { email: "alice@example.com" },
  update: { age: 27 },
  create: { name: "Alice", email: "alice@example.com", age: 27 },
});
```

Copy code

---

9. `delete()`

Deletes a single record by unique field.

ts

```ts
const deletedUser = await prisma.user.delete({
  where: { email: "alice@example.com" },
});
console.log(deletedUser);
```

Copy code

---

10. `deleteMany()`

Deletes multiple records at once.

ts

```ts
const result = await prisma.user.deleteMany({
  where: { age: { lt: 18 } },
});
console.log(result); // { count: X }
```

Copy code

---

11. `count()`

Counts records matching a filter.

ts

```ts
const totalUsers = await prisma.user.count({
  where: { isActive: true },
});
console.log(totalUsers);
```

Copy code

12. `aggregate()`

Performs aggregations like min, max, avg, sum.

ts

```ts
const stats = await prisma.user.aggregate({
  _avg: { age: true },
  _max: { age: true },
  _min: { age: true },
});
console.log(stats);
```

Copy code

---

13. `groupBy()`

Groups records by a field and applies aggregates.

ts

```ts
const result = await prisma.user.groupBy({
  by: ["isActive"],
  _count: { id: true },
  _avg: { age: true },
});
console.log(result);
```

Copy code

✅ Quick Cheat Sheet of Prisma Client Functions

Function	Purpose
`create()`	Insert one record
`createMany()`	Insert multiple records
`findMany()`	Get multiple records
`findUnique()`	Get a single record by unique key
`findFirst()`	Get first matching record
`update()`	Update one record
`updateMany()`	Update multiple records
[illegible]	Update if exists, else create
[illegible]	[illegible]
`deleteMany()`	Delete multiple records
`count()`	Count matching records
`aggregate()`	Aggregate data (sum, avg, min, max)
`groupBy()`	Group records and aggregate

If you want, I can make a **diagram showing all Prisma Client methods visually**, including which ones are for **single records vs multiple records**, which makes it super easy to remember.

Do you want me to make that?

which website can i get more information

For official and detailed information about Prisma, the **best resource is the Prisma documentation website**. It's very well organized, includes examples for every method, and has guides for beginners and advanced users. Here are the main links:

---

1. Prisma Official Docs

- **Website:** https://www.prisma.io/docs ↗
- Covers everything: setup, schema, client methods (`create()` , `findMany()` , etc.), migrations, relations, and more.

---

2. Prisma Client API Reference

- **Website:** https://www.prisma.io/docs/reference/api-reference/prisma-client ↗
- Direct reference for all Prisma Client methods with full examples.

---

3. Prisma Examples

- **Website:** https://www.prisma.io/docs/guides ↗
- Guides and practical examples for real-world use cases: CRUD, authentication, relations, aggregation, and more.

---

4. GitHub Repository

- **Website:** https://github.com/prisma/prisma ↗
- For source code, issues, and community discussions.

💡 **Tip:** If you're learning Prisma Client functions like `create()`, `findMany()`, `update()`, the **Client API Reference** link above is the most direct resource—it lists every function with syntax and examples.

If you want, I can **make a shortcut list of links for all the Prisma Client functions** so you can jump directly to each one. It's like a cheat sheet with live docs links.

Do you want me to do that?