Migrations

CSC309

Kian Abbasi

UNIVERSITY OF TORONTO MISSISSAUGA

Winter 2025

So far

- Next.js API handlers
- MVC and model design
- Prisma ORM and CRUD
- Auth

This session

- Migrations
- Workflow and assumptions
- Conflict resolution

The great assumption

- The state of database tables is the same as what defined in model schema

- But these two are totally independent things
  - Prisma models vs database tables

- ORM’s job to apply application’s schema to database
  - Via DDL queries

- Changes to schema's state:
  - Creation or removal of a table/model
  - Creation or removal of a column/field
  - Modification of field option/attributes

- Whenever the state changes, database should migrate to the new state

- Prisma does not do it automatically. WHY?

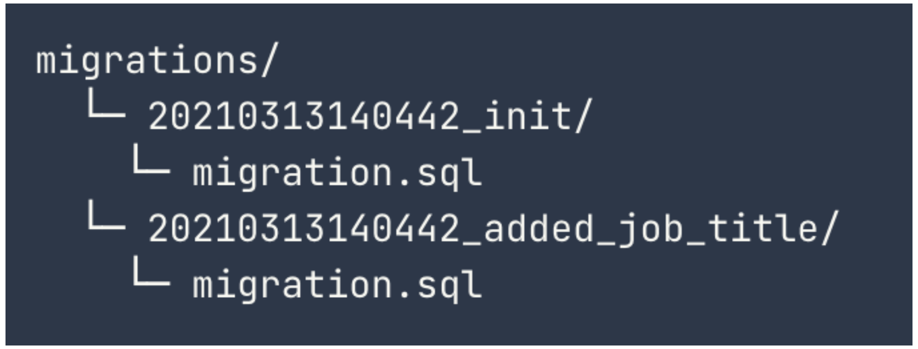
- You simply get a database exception if ORM's and database's schema do not match

Migration workflow

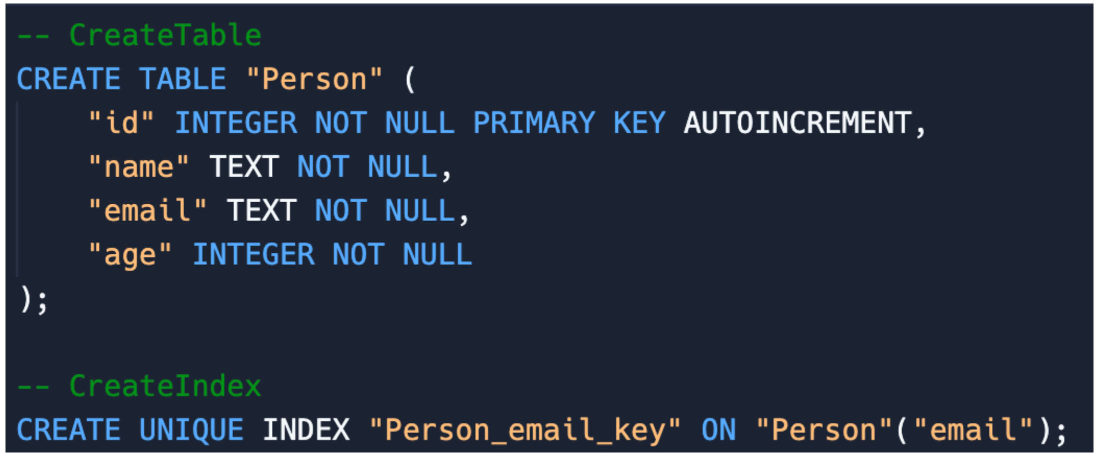
- Think about it as a git commit
    - Talks about what has changed since the last migration

- History of changes needs to be stored somewhere
    - The migrations folder

Migration workflow



```
migrations/
  └─ 20210313140442_init/
    └─ migration.sql
  └─ 20210313140442_added_job_title/
    └─ migration.sql
```



```sql
-- CreateTable
CREATE TABLE "Person" (
    "id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    "name" TEXT NOT NULL,
    "email" TEXT NOT NULL,
    "age" INTEGER NOT NULL
);

-- CreateIndex
CREATE UNIQUE INDEX "Person_email_key" ON "Person"("email");
```

New migration

- Think about it as a new commit:
  - Includes what has changed since the last commit (i.e., migration)
- Builds the old database state from previous migrations
  - Does not contact the database
- Iterates over all models to find out differences
- Creates a new folder inside the migrations directory
  - Containing the DDL queries

New migration

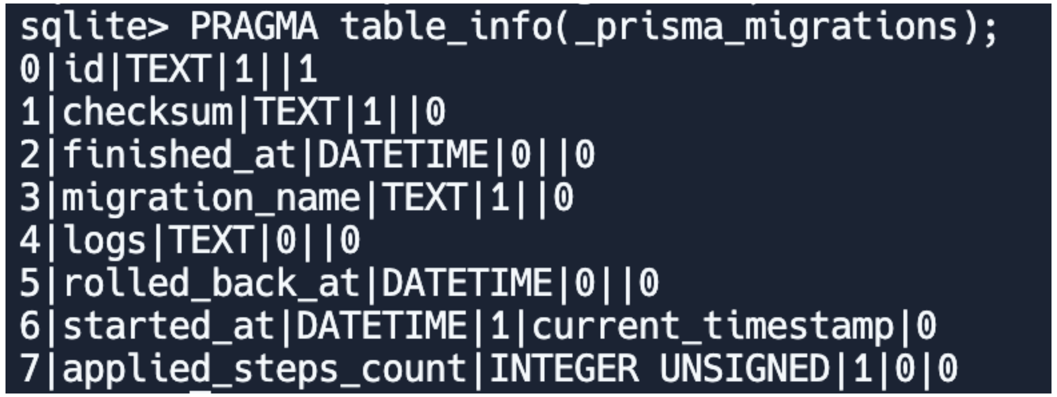
- Migrations are created and applied via
  `npx prisma migrate dev`

- But a migration should not be applied twice!
  - The same `CREATE TABLE` will not work again!
  - How is Prisma to know?

- Migrations themselves are stored in database

Migrations table

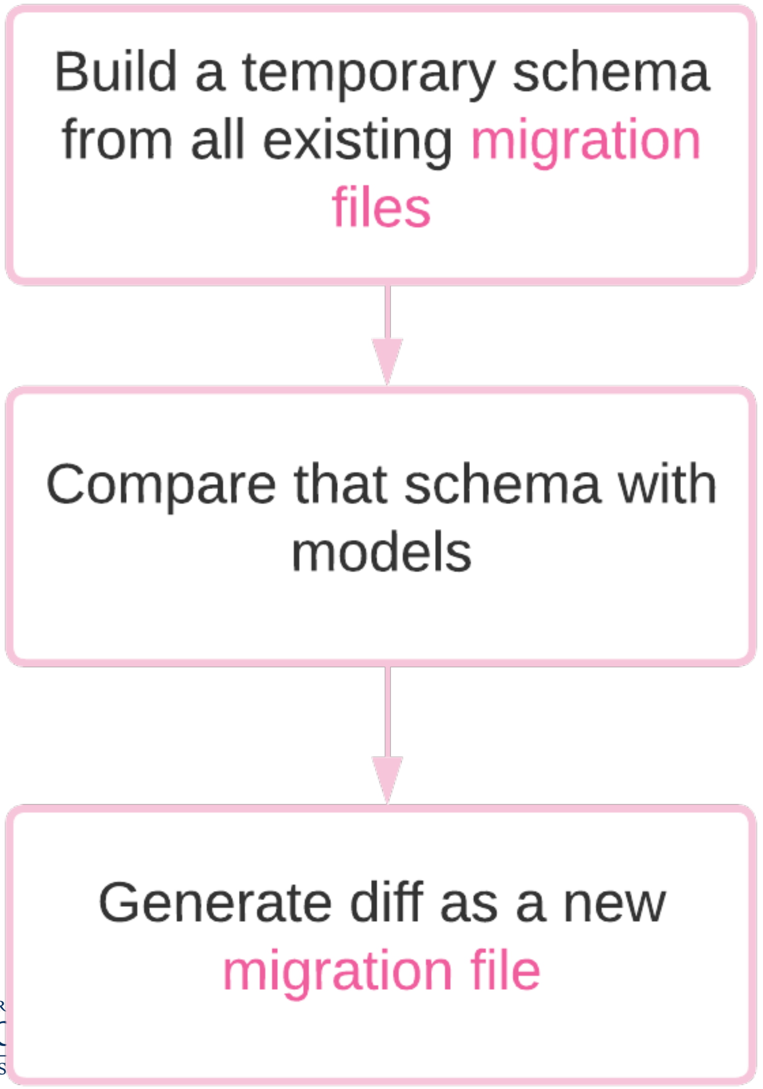
- Migrations are stored in `_prisma_migrations` table
- Stores the migrations' metadata
  - Content is only stored in the migration file
- `checksum` ensures migrations are not edited



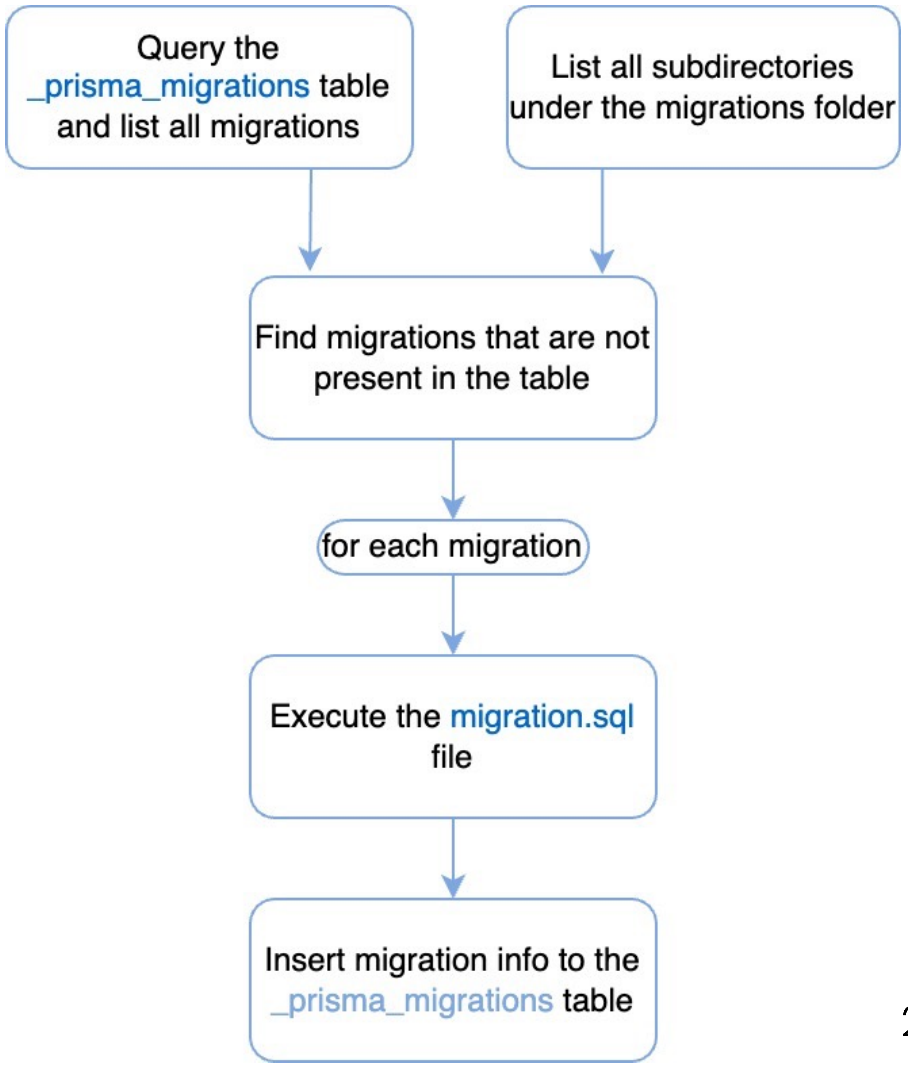
```
sqlite> PRAGMA table_info(_prisma_migrations);
0|id|TEXT|1||1
1|checksum|TEXT|1||0
2|finished_at|DATETIME|0||0
3|migration_name|TEXT|1||0
4|logs|TEXT|0||0
5|rolled_back_at|DATETIME|0||0
6|started_at|DATETIME|1|current_timestamp|0
7|applied_steps_count|INTEGER UNSIGNED|1|0|0
```

Migration workflow

Generate a new migration file


Build a temporary schema from all existing migration files
Compare that schema with models
Generate diff as a new migration file

Apply the migrations to the database


Query the _prisma_migrations table and list all migrations
List all subdirectories under the migrations folder
Find migrations that are not present in the table
for each migration
Execute the migration.sql file
Insert migration info to the _prisma_migrations table

Migration assumptions

- For this system to work, you must
  - Never directly change the database tables
    - e.g., manually running an `ALTER TABLE …`
  - Never edit/delete a migration file

- Migration files must be the same everywhere
  - Always push the migration files into git

- Migration errors can take hours to resolve!
  - Be cautious!

Migration commands

```
npx prisma generate
```

- Generates JavaScript code of the schema

```
npx prisma migrate dev
```

- Identifies schema changes since last migration
- Generates a new migration
- Applies unapplied migrations
- Should only be used in development (WHY?)

```
npx prisma migrate deploy
```

- Applies unapplied migrations (without creating new ones)
- Suitable for production

Migration errors

- Common scenarios:
  - You and your teammate added the same or conflicting migrations independently
  - Someone manually updated the database tables
  - Someone created a failing migration
    - e.g., marking a column with `NULL` values as `NOT NULL`
  - Someone edited a migration file

- Very tricky:
  - Potential for data loss is high. This should be avoided at all costs!

Migration error solutions

Visit https://www.prisma.io/docs/orm/prisma-migrate/workflows/patching-and-hotfixing

- Resolve a migration

```
npx prisma migrate resolve --applied "migration_name"
npx prisma migrate resolve --rolled-back "migration_name"
```

- Will only update the migrations table, without executing the queries

- Manually sync database schema with migrations

The last resort

- Reset the entire database

```
npx prisma db reset
```

- Deletes all table's data
  - Applies the migrations on an empty database
- Definitely NOT an option in production
  - So be careful about migrations

The very last resort

- Delete the entire database
  - Just delete the dev.db file!
- Delete the migrations directory afterwards
- Restart with a fresh schema and generate new migrations!
- Definitely NOT an option in production
  - So be careful about migrations

Next session

- Begin (or resume) our front-end journey

- Modern client-side JavaScript
  - React, JSX

- React application
  - Props
  - Events
  - State