Лекция 10. ORM. Миграции

Корытов Павел

9 ноября 2023 г.

Варианты использования БД

- SQL

```
1  SELECT * FROM users WHERE name = 'Pavel' ORDER BY id
```

- DAL / DBAL (Database Abstraction Layer)

```
1  (org-roam-db-query
2   [:select [id name surname]
3   :from users
4   :where (= name "Pavel")])
```

```
((1 "Pavel" "Korytov"))
```

- ORM (Object Relational Mapping)

```
1  users = User.findAll(where: { name: 'Pavel' }})
```

```
[<User id="1" name="Pavel" surname="Korytov">]
```

ORM

Object-Relational Mapping (объектно-реляционное отображение) - технология, связывающая БД и объекты в объектно-ориентированных парадигмах.

Т.е. дает возможность работать с данными в БД как с объектами в языке программирования.

Как правило, отображение имеет такой вид:

- Таблицы в БД становятся классами
- Строчки (кортежи) в БД становятся объектами

ORM. Варианты

Некоторые варианты ORM:

Язык программирования	Фреймворки
Python	SQLAlchemy, Django ORM
Go	gorm
Common Lisp	mito
Elisp	closql
JavaScript / TypeScript	Sequelize, TypeORM
Java	Hibernate
PHP	Doctrine
.NET	Entity Framework
C++/Qt	QxOrm

Разных ORM-фреймворков очень много!

ORM. Варианты (2)

SQLAlchemy - конструкции ORM напоминают SQL запросы:

```
results = db.execute(
    sa.select(User, Department.title) \
      .join(Department, Department.id == User.departmentId) \
      .where(User.surname == 'Корытов')
).one()
```

Sequelize:

```
const user = await User.findOne({
    where: { surname: 'Корытов' },
    include: [
        { model: Department, attributes: ['title'], required:
↪   true }
    ]
})
```

ORM. Преимущества

Немного оценочных суждений:

- Экономия времени (программиста)
  - Безопасность (транзакции, санитизация - см. лекцию 12)
  - Меньше кода
    - Части, которые можно автоматизировать, автоматизированы
    - Проще переиспользование кода
  - Слабее связь бизнес-логики приложения и модели данных
- Гибкость
  - Привязка к диалекту SQL менее сильная или отсутствует
  - Легче внести изменения в структуру БД
  - (Обычно) можно использовать концепции ORM, например наследование

ORM. Недостатки

- Сложность изучения
- Всё равно нужно знать SQL!
- Всё равно нужно знать, что ORM делает ”за кадром”
- Сложности с производительностью

Sequelize

Время демонстрации

`https://gist.github.com/SqrtMinusOne/6c9a2cb81918c285ce25ead12e81acbc`

Миграции

Миграция - инкрементальное, обратимое изменение схемы БД.

Как правило, состоит из двух логических частей:

- Часть, применяющая изменения (создание таблиц, добавление столбцов, т.п.)
- Часть, откатывающая изменения

Миграции применяются по очереди, следующая зависит от предыдущей.

Миграции. Продолжение

Миграции используются, когда необходимо произвести изменения в структуре БД работающего приложения.

Например, изменились требования к системе, и нужно сделать что-нибудь из следующего:

- добавить новые столбцы в БД
- удалить старые столбцы
- изменить формат хранения данных

Или:

- исправить ошибки / недочеты, допущенные при проектировании
- …

Миграции особенно полезны при использовании гибких методологий разработки.

Идеальный вариант - структура БД полностью создается запуском миграций с первой до последней.

sequelize-cli

sequelize-cli - пакет для работы с миграциями в sequelize.

```
npm i -g sequelize-cli
```

Конфигурация: файл `.sequelizerc` в корне проекта (возможны другие варианты):

```
const path = require('path')

module.exports = {
  config: path.resolve('./src/config',
  ↪  'sequelize-cli-config.js'),
  // 'models-path': path.resolve('./src', 'models'),
  // 'seeders-path': path.resolve('./src', 'seeders'),
  'migrations-path': path.resolve('./src', 'migrations')
}
```

sequelize-cli (2)

Файл `sequlize-cli-config.js` может выглядеть так:

```
1  /* eslint-disable @typescript-eslint/no-var-requires */
2  const dotenv = require('dotenv');
3  const fs = require('fs');
4
5  if (fs.existsSync('.env')) {
6    dotenv.config({ path: '.env' });
7  } else {
8    dotenv.config({ path: '.env.example' });
9  }
10
11 module.exports = {
12   development: {
13     username: process.env.DB_USER,
14     password: process.env.DB_PASS,
15     database: process.env.DB_NAME,
16     host: process.env.DB_HOST || 'localhost',
17     port: 5432,
18     dialect: 'postgres',
19   },
20 };
```

sequelize-cli (3)

После настройки использование следующее:

- `sequelize-cli migration-generate --name <имя>` - создает пустую миграцию
- `sequelize-cli db:migrate` - применяет доступные миграции
- `sequelize-cli db:migrate:undo` - отменяет одну миграцию

Одна миграция выглядит так:

```
1  module.exports = {
2    up: async (queryInterface, Sequelize) => {
3      // применение изменений (создание таблиц, добавление
   ↪  атрибутов и т.п.)
4    },
5    down: async (queryInterface, Sequelize) => {
6      // отмена этих изменений
7    }
8  }
```

sequelize-cli (4)

Что может `queryInterface`, можно посмотреть здесь.

```
1  await queryInterface.addColumn(
2    'Groups',
3    'isFake',
4    {
5      type: Sequelize.BOOLEAN,
6      allowNull: false,
7      defaultValue: false,
8    },
9    { transaction: t },
10 );
```

sequelize-cli (5)

Если в миграции выполняется несколько шагов, действия в ней нужно обернуть в транзакцию. В sequelize её можно получить так:

```
1  module.exports = {
2    up: async (queryInterface, Sequelize) => {
3      await queryInterface.sequelize.transaction((t) => {
4        // действия
5      })
6    },
7    down: async (queryInterface, Sequelize) => {
8      await queryInterface.sequelize.transaction((t) => {
9        // действия
10     })
11   }
12 }
```