МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Базы данных»

Тема: Реализация базы данных с использованием ORM

Студент гр. 3341 _____ Шаповаленко Е.В.

Преподаватель _____ Заславский М.М.

Санкт-Петербург
2025

Цель работы.

Целью данной лабораторной работы является практическое освоение навыков создания и администрирования баз данных с использованием ORM Sequelize. Для достижения цели необходимо написать запросы для создания таблиц из предыдущей лабораторной работы и выполнения ряда действий, указанных в задании к ней.

Задание.

В данной лабораторной работе рекомендуется использовать Sequelize (Node.js). В случае, если не знаете как с ним работать, то можно воспользоваться примером из репозитория.

Вы можете использовать другой ORM по вашему выбору по согласованию с преподавателем, принимающим у вас практики.

Необходимо развернуть Sequelize на своем ПК и выполнить следующие задачи:

- Описать в виде моделей Sequelize таблицы из 1-й лабораторной работы
- Написать скрипт заполнения тестовыми данными: 5-10 строк на каждую таблицу, обязательно наличие связи между ними, данные приближены к реальности.
- Написать запросы к БД, отвечающие на вопросы из 1-й лабораторной работы с использованием ORM. Вывести результаты в консоль (или иной человеко-читабельный вывод)
- Запушить в репозиторий исходный код проекта, соблюсти .gitignore, убрать исходную базу из проекта (или иные нагенерированные данные бд если они есть).
- Описать процесс запуска: команды, зависимости

- В отчете описать цель, текст задания в соответствии с вариантом, выбранную ORM, инструкцию по запуску, скриншоты (код) моделей ORM, скриншоты на каждый запрос (или группу запросов) на

изменение/таблицы с выводом результатов (ответ), ссылку на PR в приложении, вывод

Вариант 21

Пусть требуется создать программную систему, предназначенную для продавца журналов/комиксов. Такая система должна обеспечивать хранение сведений об имеющихся в магазине журналах, о читателях журналов и списке магазинов. Для каждого журнала в БД должны храниться следующие сведения: название журнала, серия, автор (ы), издательство, год издания, число экземпляров в каждом магазине, а также ISBN и дата продажи журнала. Сведения о читателях библиотеки должны включать почту (email), ФИО покупателя, дату рождения, адрес, номер телефон. Нужно учесть, что покупатели могут делать заказы с разных магазинов, но нужно сохранять информацию, о предпочтительных магазинах (в которых чаще делают заказы). Магазин имеет несколько отделов, которые характеризуются номером, названием и кол-во журналов (вместимость). Магазин может получать новые журналы и списывать старые. ISBN может измениться в результате переклассификации, а почта в результате перерегистрации. Продавцу могут потребоваться следующие сведения о текущем состоянии библиотеки:

- Какие журналы были куплены определенным покупателем?
- Как называется журнал с заданным ISBN?
- Какой ISBN у журнала с заданным названием?
- Когда журнал был куплен?
- Кто из покупателей купил журнал более месяца тому назад?
- Найти покупателя самых редких журналов (по наличию в магазине)?
- Какое число покупателей пользуется определенным магазином?
- Сколько покупателей младше 20 лет?

Выполнение работы.

В лабораторной работе №1 была разработана следующая реляционная модель:

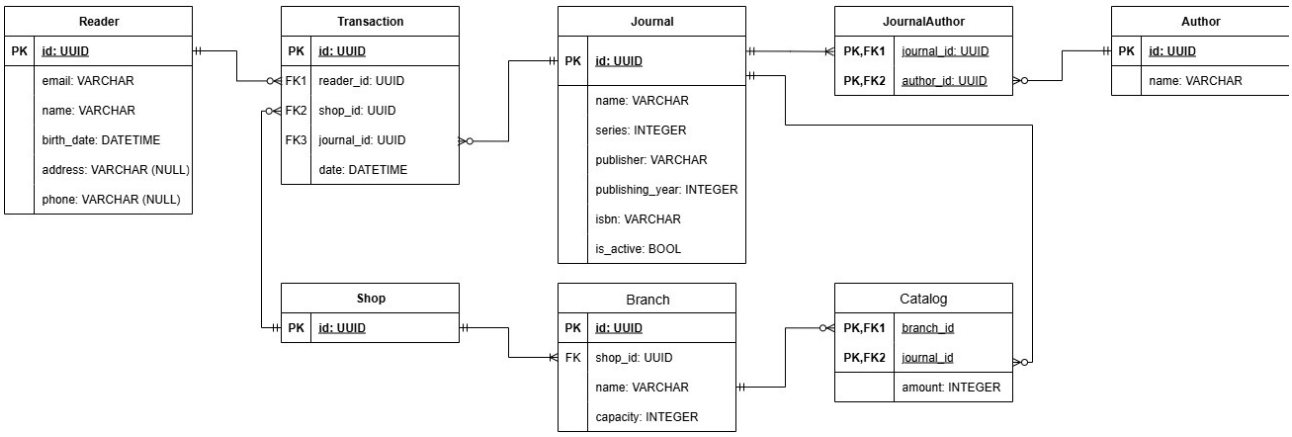

Рис. 1 — Структура базы данных

Для взаимодействия с СУБД PostgreSQL в работе использовался ORM Sequelize. Была создана новая база данных для лабораторной работы в CLI psql при помощи команды *CREATE DATABASE lab3* (с целью сохранить базу данных *comic_book_shop*).

```
postgres=# \l
                                                  Список баз данных
      Имя       | Владелец | Кодировка | Провайдер локали |     LC_COLLATE      |      LC_CTYPE       | Локаль | Правила ICU |     Права доступа
----------------+----------+-----------+------------------+---------------------+---------------------+--------+-------------+-----------------------
 comic_book_shop | postgres | UTF8      | libc             | Russian_Russia.1251 | Russian_Russia.1251 |        |             |
 lab3            | postgres | UTF8      | libc             | Russian_Russia.1251 | Russian_Russia.1251 |        |             |
```

Рис. 2 — Создана новая база данных в СУБД PostgreSQL

Для начала необходимо установить зависимости (прописаны в *package.json*) при помощи *nmp i*.

Подключение к базе данных описано в файле *config/database.js*. Для того, чтобы убрать из непосредственно кода программы пароль, была подключена библиотека *dotenv*. Перед запуском программы необходимо создать/отредактировать файл *.env*, указав в нем необходимые параметры:

```
NODE_ENV=development

DB_HOST=host
DB_PORT=port
```

```
DB_NAME=name
DB_USER=user
DB_PASS=pass

DB_LOGGING=true
```

Здесь *host* — адрес хоста (т.к. работа происходила в среде WSL Ubuntu, а сама база данных была на этой же машине в среде Windows, использовался шлюз по умолчанию 172.22.0.1), *port* — порт подключения (по умолчанию 5432), *name* — имя базы данных (*lab3*), *user* — имя пользователя (по умолчанию *postgres*), *pass* — пароль.

Модели и их связи описаны в файле *models/index.js*. Рассмотрим на примере таблицы *transaction* (рис. 3).

```
// transaction, internally called Sale due to sequelize.transaction()
const Sale = sequelize.define('Sale', {
  id: { type: DataTypes.UUID, primaryKey: true, defaultValue: Sequelize.literal('gen_random_uuid()') },
  reader_id: { type: DataTypes.UUID, allowNull: false },
  shop_id: { type: DataTypes.UUID, allowNull: false },
  journal_id: { type: DataTypes.UUID, allowNull: false },
  date: { type: DataTypes.DATE, allowNull: false },
}, {
  tableName: 'transaction',
  indexes: [
    { name: 'idx_transaction_date', fields: ['date'] },
    { name: 'idx_transaction_journal_id', fields: ['journal_id'] },
    { name: 'idx_transaction_reader_id', fields: ['reader_id'] },
    { name: 'idx_transaction_shop_id', fields: ['shop_id'] },
  ],
});
```

Рис. 3 — модель таблицы *transaction*

Создание происходит при помощи *sequelize.define()*. Внутри кода было дано имя *Sale*, т.к. можно было перепутать с функцией *sequelize.transaction()*. Однако имя самой таблицы *tableName* все еще *transaction*. На рис.3 видно, как описываются столбцы таблицы, а также как создаются индексы (используемые для ускорения поиска по часто используемым столбцам).

Далее описываются связи между моделями при помощи методов *hasMany*, *belongsTo*, *belongsToMany*.

Для заполнения таблиц данными используется скрипт *scripts/seed.js*. В нем прописаны данные из предыдущей лабораторной работы. Выводится информация о выполняемых Sequelize запросах, при их успешности выводится *Schema synced and data seeded*.

Запросы с ответами на вопросы из предыдущей лабораторной работы описаны в файле *scripts/test/js*. Так как параметр DB_LOGGING установлен в true, помимо самих ответов на вопросы выводится и отладочная информация. На рисунках ниже отображены только сами ответы.

```
1) Journals by reader email: [
  {
    reader: 'ivan.petrov@email.com',
    journal: 'The Amazing Spider-Man'
  },
  { reader: 'ivan.petrov@email.com', journal: 'X-Men' }
]
```

Рис. 4 — Какие журналы были куплены определенным покупателем?

```
2) Journal by ISBN: { isbn: '978-1-302-95001-1', journal: 'The Amazing Spider-Man' }
```

Рис. 5 — Как называется журнал с заданным ISBN?

```
3) ISBN by journal name: { journal: 'Teen Titans Go!', isbn: '978-1-7795-1789-1' }
```

Рис. 6 — Какой ISBN у журнала с заданным названием?

```
4) Purchases of journal: [ { journal: 'Teen Titans Go!', date: 2024-01-21T09:40:00.000Z } ]
```

Рис. 7 — Когда журнал был куплен?

Рис. 8 — Кто из покупателей купил журнал более месяца тому назад?

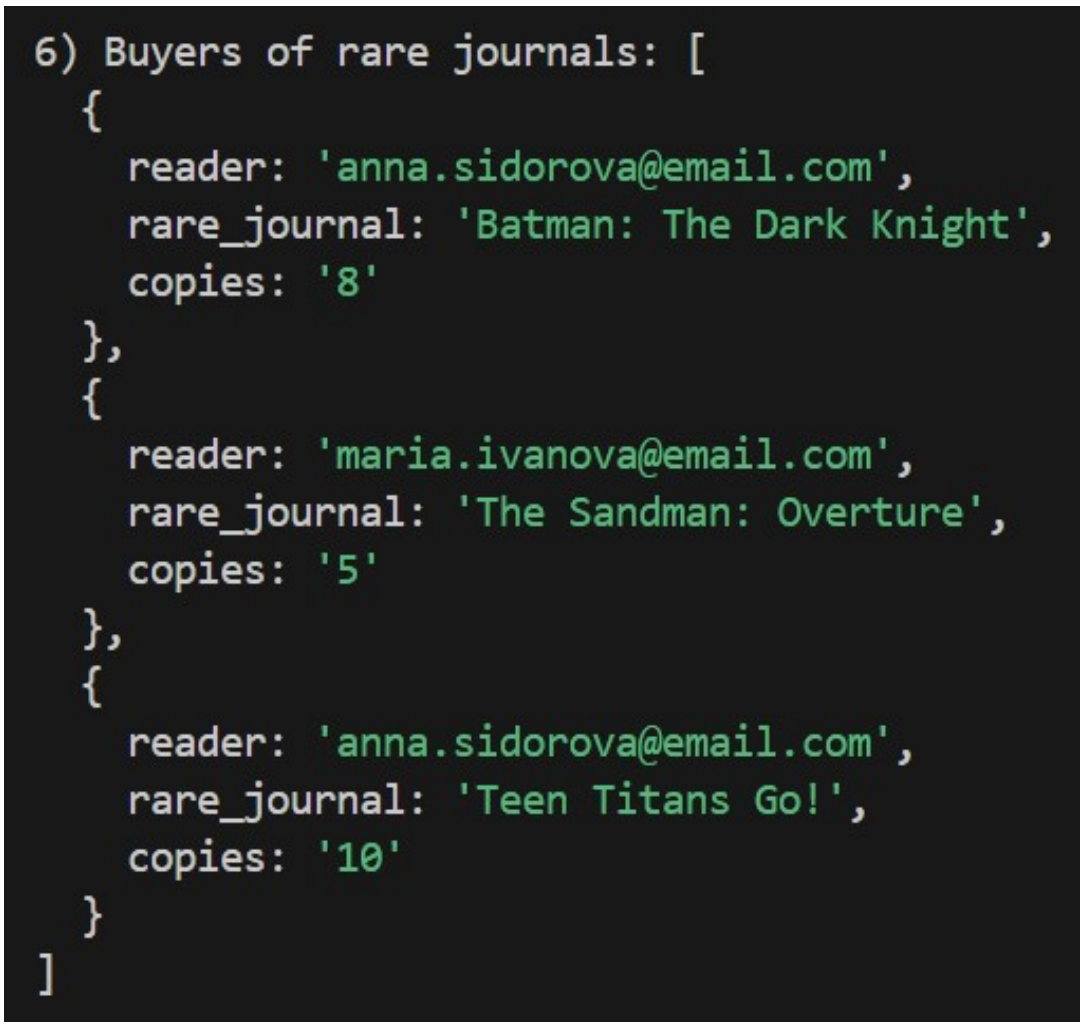

Рис. 9 — Найти покупателя самых редких журналов (по наличию в магазине)?

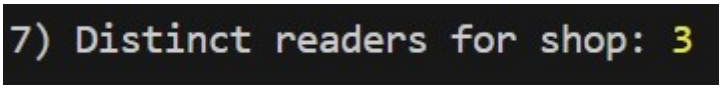

Рис. 10 — Какое число покупателей пользуется определенным магазином?

Рис. 11 — Сколько покупателей младше 20 лет?

Для удобства использования в *package.json* прописаны различные скрипты. Для заполнения базы данных и выполнения запросов необходимо всего лишь выполнить команду *npm run start*.

Написанный программный код см. в приложении А.

Выводы.

В результате выполнения лабораторной работы были освоены навыки создания и администрирования баз данных с использованием ORM Sequelize. Были написаны запросы для создания таблиц из предыдущей лабораторной работы и выполнения ряда действий, указанных в задании к ней.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Ссылка на Pull-request с результатами выполненной лабораторной работы: https://github.com/moevm/sql-2025-3341/pull/29

Файл config/database.js:

```
require('dotenv').config();
const { Sequelize } = require('sequelize');

const {
  DATABASE_URL,
  DB_HOST,
  DB_PORT,
  DB_NAME,
  DB_USER,
  DB_PASS,
  DB_LOGGING,
} = process.env;

const commonOptions = {
  dialect: 'postgres',
  logging: DB_LOGGING === 'true' ? console.log : false,
  define: {
    underscored: true,
    timestamps: false,
  },
};

const sequelize = DATABASE_URL
  ? new Sequelize(DATABASE_URL, commonOptions)
  : new Sequelize({
      host: DB_HOST,
      port: Number(DB_PORT),
```

```
      database: DB_NAME,
      username: DB_USER,
      password: DB_PASS,
      ...commonOptions,
    });

module.exports = { sequelize };
```

Файл models/index.js:

```
const { sequelize } = require('../config/database');
const { DataTypes, Sequelize } = require('sequelize');

// Models

// author
const Author = sequelize.define('Author', {
  id: { type: DataTypes.UUID, primaryKey: true, defaultValue:
Sequelize.literal('gen_random_uuid()') },
  name: { type: DataTypes.STRING, allowNull: false },
}, {
  tableName: 'author',
});

// shop
const Shop = sequelize.define('Shop', {
  id: { type: DataTypes.UUID, primaryKey: true, defaultValue:
Sequelize.literal('gen_random_uuid()') },
}, {
  tableName: 'shop',
});

// branch
const Branch = sequelize.define('Branch', {
  id: { type: DataTypes.UUID, primaryKey: true, defaultValue:
Sequelize.literal('gen_random_uuid()') },
```

```
    shop_id: { type: DataTypes.UUID, allowNull: false },
    name: { type: DataTypes.STRING, allowNull: false },
    capacity: { type: DataTypes.INTEGER, allowNull: false },
  }, {
    tableName: 'branch',
  });

  // journal
  const Journal = sequelize.define('Journal', {
    id: { type: DataTypes.UUID, primaryKey: true, defaultValue:
Sequelize.literal('gen_random_uuid()') },
    name: { type: DataTypes.STRING, allowNull: false },
    series: { type: DataTypes.INTEGER, allowNull: false },
    publisher: { type: DataTypes.STRING, allowNull: false },
    publishing_year:  {  type:  DataTypes.INTEGER,  allowNull:
false },
    isbn: { type: DataTypes.STRING, allowNull: false },
    is_active: { type: DataTypes.BOOLEAN, allowNull: false },
  }, {
    tableName: 'journal',
    indexes: [
      { name: 'idx_journal_isbn', fields: ['isbn'] },
      { name: 'idx_journal_name', fields: ['name'] },
    ],
  });

  // reader
  const Reader = sequelize.define('Reader', {
    id: { type: DataTypes.UUID, primaryKey: true, defaultValue:
Sequelize.literal('gen_random_uuid()') },
    email: { type: DataTypes.STRING, allowNull: false, unique:
true },
    name: { type: DataTypes.STRING, allowNull: false },
    birth_date: { type: DataTypes.DATEONLY, allowNull: false },
    address: { type: DataTypes.STRING, allowNull: true },
```

```
      phone: { type: DataTypes.STRING, allowNull: true },
    }, {
      tableName: 'reader',
      indexes: [
        { name: 'idx_reader_birth_date', fields: ['birth_date'] },
      ],
    });

    // catalog
    const Catalog = sequelize.define('Catalog', {
      branch_id:  {  type:  DataTypes.UUID,  allowNull:  false,
primaryKey: true },
      journal_id:  {  type:  DataTypes.UUID,  allowNull:  false,
primaryKey: true },
      amount: { type: DataTypes.INTEGER, allowNull: false },
    }, {
      tableName: 'catalog',
      indexes: [
        { name: 'idx_catalog_amount', fields: ['amount'] },
      ],
    });

    // journal_author
    const JournalAuthor = sequelize.define('JournalAuthor', {
      journal_id:  {  type:  DataTypes.UUID,  allowNull:  false,
primaryKey: true },
      author_id:  {  type:  DataTypes.UUID,  allowNull:  false,
primaryKey: true },
    }, {
      tableName: 'journal_author',
    });

    //  transaction,  internally  called  Sale  due  to
sequelize.transaction()
    const Sale = sequelize.define('Sale', {
```

```
      id: { type: DataTypes.UUID, primaryKey: true, defaultValue:
Sequelize.literal('gen_random_uuid()') },
      reader_id: { type: DataTypes.UUID, allowNull: false },
      shop_id: { type: DataTypes.UUID, allowNull: false },
      journal_id: { type: DataTypes.UUID, allowNull: false },
      date: { type: DataTypes.DATE, allowNull: false },
    }, {
      tableName: 'transaction',
      indexes: [
        { name: 'idx_transaction_date', fields: ['date'] },
        {     name:     'idx_transaction_journal_id',     fields:
['journal_id'] },
        {     name:     'idx_transaction_reader_id',     fields:
['reader_id'] },
        { name: 'idx_transaction_shop_id', fields: ['shop_id'] },
      ],
    });

    // ASSOCIATIONS
    // Shop --> Branch
    Shop.hasMany(Branch,  {  foreignKey:  'shop_id',  onDelete:
'RESTRICT' });
    Branch.belongsTo(Shop,  {  foreignKey:  'shop_id',  onDelete:
'RESTRICT' });

    // Branch --> Catalog <-- Journal
    Branch.hasMany(Catalog, { foreignKey: 'branch_id', onDelete:
'RESTRICT' });
    Catalog.belongsTo(Branch, { foreignKey: 'branch_id', onDelete:
'RESTRICT' });
    Journal.hasMany(Catalog, { foreignKey: 'journal_id', onDelete:
'RESTRICT' });
    Catalog.belongsTo(Journal,   {   foreignKey:   'journal_id',
onDelete: 'RESTRICT' });
```

```
    // Journal --> JournalAuthor <-- Author
    Journal.belongsToMany(Author, { through: JournalAuthor, foreignKey: 'journal_id', otherKey: 'author_id', onDelete: 'RESTRICT' });
    Author.belongsToMany(Journal, { through: JournalAuthor, foreignKey: 'author_id', otherKey: 'journal_id', onDelete: 'RESTRICT' });

    // Reader --> Sale <-- Journal
    //              ^
    //              | Shop
    Reader.hasMany(Sale, { foreignKey: 'reader_id', onDelete: 'RESTRICT' });
    Sale.belongsTo(Reader, { foreignKey: 'reader_id', onDelete: 'RESTRICT' });
    Shop.hasMany(Sale, { foreignKey: 'shop_id', onDelete: 'RESTRICT' });
    Sale.belongsTo(Shop, { foreignKey: 'shop_id', onDelete: 'RESTRICT' });
    Journal.hasMany(Sale, { foreignKey: 'journal_id', onDelete: 'RESTRICT' });
    Sale.belongsTo(Journal, { foreignKey: 'journal_id', onDelete: 'RESTRICT' });

    module.exports = {
      sequelize,
      Author, Branch, Catalog, Journal, JournalAuthor, Reader, Shop, Sale,
    };
```

Файл scripts/seed.js:

```
    const { sequelize, Author, Branch, Catalog, Journal, JournalAuthor, Reader, Shop, Sale } = require('../models');

    async function main() {
```

```
      // await sequelize.query('CREATE EXTENSION IF NOT EXISTS
pgcrypto;');

      await sequelize.sync({ force: true });

      await Shop.bulkCreate([
        { id: '4db426cb-00e3-477c-9f78-9eaf82c968b3' },
        { id: 'a8029dce-c70b-43a7-bb69-0829d1d41bca' },
        { id: 'cd707836-60d4-4213-910d-eb6dad43fe86' },
        { id: '7e0f0fc4-fcf4-4bcb-a415-e0824f1b6412' },
        { id: 'dba0ed7e-d2fc-47d7-be1e-db8a3a9d0476' },
      ]);

      await Author.bulkCreate([
        { id: '737fbeb0-a0c4-4df1-a396-e4ae45377e86', name: 'Stan
Lee' },
        { id: '008686be-9460-4b34-992c-4f2d2d75c44a', name: 'Bob
Kane' },
        { id: 'fcd98444-756e-4fff-a7ad-2f481f142994', name:
'Jerry Siegel' },
        { id: '130d813f-6f13-4d6d-8a09-8fc1475658ea', name:
'Hajime Isayama' },
        { id: 'dd4b104b-df47-4637-a3f2-a3db44418dfa', name:
'Kohei Horikoshi' },
        { id: '5a4a8e0b-95a5-4d21-be43-c8494fabd4e3', name: 'Neil
Gaiman' },
        { id: 'a8d3940a-0a00-4ab5-8af5-44b6ed46c999', name:
'Jason Latour' },
        { id: 'e01519a1-606f-4f2b-a6cd-9eb684ac1e8d', name: 'J.
Torres' },
      ]);

      await Branch.bulkCreate([
```

```
        { id: 'a025e999-5b29-4a61-892a-201b3ff173aa', shop_id:
'4db426cb-00e3-477c-9f78-9eaf82c968b3', name: 'Комиксы Marvel',
capacity: 50 },
        { id: '1448d45d-cf20-4fc7-a53e-7a455a7b8d1c', shop_id:
'4db426cb-00e3-477c-9f78-9eaf82c968b3', name: 'Комиксы DC',
capacity: 45 },
        { id: 'ca339d8a-4dd8-4cdb-85b1-f7abc942f6a1', shop_id:
'a8029dce-c70b-43a7-bb69-0829d1d41bca', name: 'Манга и аниме',
capacity: 60 },
        { id: '9bc65a0e-4deb-4952-98a3-aadec14fd6af', shop_id:
'a8029dce-c70b-43a7-bb69-0829d1d41bca', name: 'Графические романы',
capacity: 35 },
        { id: '787f74cf-4583-44eb-88cf-ac20433017a3', shop_id:
'cd707836-60d4-4213-910d-eb6dad43fe86', name: 'Новинки', capacity:
40 },
        { id: 'b415ce4b-0d5c-4e59-8920-68c39e862b4e', shop_id:
'cd707836-60d4-4213-910d-eb6dad43fe86', name: 'Раритетные издания',
capacity: 25 },
        { id: '7caf2960-6006-40c9-9455-a8d91a01f1b9', shop_id:
'7e0f0fc4-fcf4-4bcb-a415-e0824f1b6412', name: 'Детские комиксы',
capacity: 55 },
        { id: 'fc6b205a-3d6e-4862-baa0-1c5962fca291', shop_id:
'7e0f0fc4-fcf4-4bcb-a415-e0824f1b6412', name: 'Западные комиксы',
capacity: 48 },
        { id: '00943eb3-5a06-4f2b-93de-7f82ae5732a8', shop_id:
'dba0ed7e-d2fc-47d7-be1e-db8a3a9d0476', name: 'Супергерои',
capacity: 52 },
        { id: '749ced6a-5a79-46bc-984e-98f5abe5bfd4', shop_id:
'dba0ed7e-d2fc-47d7-be1e-db8a3a9d0476', name: 'Научная фантастика',
capacity: 38 },
      ]);

      await Journal.bulkCreate([
        { id: '0bc82198-02a1-4228-84fa-2c48f4c7c793', name: 'The
Amazing Spider-Man', series: 1, publisher: 'Marvel',
```

```
publishing_year: 2024, isbn: '978-1-302-95001-1', is_active: true },
        { id: 'ef947889-d288-4ded-86d2-5f26d46a9c85', name: 'X-Men', series: 7, publisher: 'Marvel', publishing_year: 2023, isbn: '978-1-302-93456-1', is_active: true },
        { id: 'f5751297-75b7-4283-a3e8-f4bd0a84168c', name: 'Batman: The Dark Knight', series: 1, publisher: 'DC Comics', publishing_year: 2024, isbn: '978-1-7795-1651-1', is_active: true },
        { id: 'd30d33f8-c0d9-4078-a4c0-86b67be587bb', name: 'Superman: Legacy', series: 3, publisher: 'DC Comics', publishing_year: 2023, isbn: '978-1-7795-1287-2', is_active: false },
        { id: '0045dc9a-8cef-4e17-976a-82a8b26ca9a4', name: 'Attack on Titan', series: 34, publisher: 'Kodansha', publishing_year: 2023, isbn: '978-1-63236-978-1', is_active: true },
        { id: 'eb7446ff-4ee3-410b-b41a-8d5ed0521cc6', name: 'My Hero Academia', series: 32, publisher: 'Shueisha', publishing_year: 2024, isbn: '978-1-9747-2345-6', is_active: true },
        { id: '655f827b-c785-4b43-979d-47b6f11d31d4', name: 'The Sandman: Overture', series: 1, publisher: 'Vertigo', publishing_year: 2023, isbn: '978-1-4012-6552-1', is_active: true },
        { id: '664969a2-0804-4390-9400-7c24f8eb2066', name: 'Spider-Gwen: Ghost-Spider', series: 1, publisher: 'Marvel', publishing_year: 2024, isbn: '978-1-302-95678-1', is_active: true },
        { id: '3497d265-40a6-43fc-8858-01a8e555fa5e', name: 'The Fantastic Four', series: 1, publisher: 'Marvel', publishing_year: 1961, isbn: '978-1-302-90001-1', is_active: false },
        { id: '1bb5c6ac-b86a-4d33-9e0a-f6a3e3dce2d5', name: 'Teen Titans Go!', series: 15, publisher: 'DC Comics', publishing_year: 2024, isbn: '978-1-7795-1789-1', is_active: true },
      ]);
```

```
      await JournalAuthor.bulkCreate([
        {   journal_id:   '0bc82198-02a1-4228-84fa-2c48f4c7c793',
author_id: '737fbeb0-a0c4-4df1-a396-e4ae45377e86' },
        {   journal_id:   'ef947889-d288-4ded-86d2-5f26d46a9c85',
author_id: '737fbeb0-a0c4-4df1-a396-e4ae45377e86' },
        {   journal_id:   '3497d265-40a6-43fc-8858-01a8e555fa5e',
author_id: '737fbeb0-a0c4-4df1-a396-e4ae45377e86' },
        {   journal_id:   'f5751297-75b7-4283-a3e8-f4bd0a84168c',
author_id: '008686be-9460-4b34-992c-4f2d2d75c44a' },
        {   journal_id:   'd30d33f8-c0d9-4078-a4c0-86b67be587bb',
author_id: 'fcd98444-756e-4fff-a7ad-2f481f142994' },
        {   journal_id:   '0045dc9a-8cef-4e17-976a-82a8b26ca9a4',
author_id: '130d813f-6f13-4d6d-8a09-8fc1475658ea' },
        {   journal_id:   'eb7446ff-4ee3-410b-b41a-8d5ed0521cc6',
author_id: 'dd4b104b-df47-4637-a3f2-a3db44418dfa' },
        {   journal_id:   '655f827b-c785-4b43-979d-47b6f11d31d4',
author_id: '5a4a8e0b-95a5-4d21-be43-c8494fabd4e3' },
        {   journal_id:   '664969a2-0804-4390-9400-7c24f8eb2066',
author_id: 'a8d3940a-0a00-4ab5-8af5-44b6ed46c999' },
        {   journal_id:   '1bb5c6ac-b86a-4d33-9e0a-f6a3e3dce2d5',
author_id: 'e01519a1-606f-4f2b-a6cd-9eb684ac1e8d' },
      ]);

      await Reader.bulkCreate([
        {   id:   '66c8c0fd-d849-43b4-ae97-5a7cf04a1d4e',   email:
'ivan.petrov@email.com', name: 'Иван Петров',  birth_date: '1990-
05-15', address: 'г. Москва, ул. Ленина, д. 10, кв. 25', phone:
'+7-915-123-45-67' },
        {   id:   '5740d88c-701c-4137-a4ba-91f78e993467',   email:
'anna.sidorova@email.com', name: 'Анна Сидорова',   birth_date:
'1985-12-03', address: 'г. Санкт-Петербург, Невский пр-т, д. 50',
phone: null },
```

```
        { id: '9477efca-4b5e-418b-8983-22c70e74eb2b', email:
'sergey.kozlov@email.com', name: 'Сергей Козлов', birth_date:
'1998-08-20', address: null, phone: '+7-916-987-65-43' },
        { id: '5d10d4b5-9045-4aa0-956d-65c56b204140', email:
'maria.ivanova@email.com', name: 'Мария Иванова', birth_date:
'2010-03-10', address: null, phone: null },
        { id: '11309de7-2b40-4ee2-ae96-c891faf5a9d1', email:
'dmitry.sokolov@email.com', name: 'Дмитрий Соколов', birth_date:
'2011-11-28', address: 'г. Казань, ул. Баумана, д. 15', phone:
null },
      ]);

      await Catalog.bulkCreate([
        { branch_id: 'a025e999-5b29-4a61-892a-201b3ff173aa',
journal_id: '0bc82198-02a1-4228-84fa-2c48f4c7c793', amount: 12 },
        { branch_id: 'a025e999-5b29-4a61-892a-201b3ff173aa',
journal_id: 'ef947889-d288-4ded-86d2-5f26d46a9c85', amount: 10 },
        { branch_id: '1448d45d-cf20-4fc7-a53e-7a455a7b8d1c',
journal_id: 'f5751297-75b7-4283-a3e8-f4bd0a84168c', amount: 8 },
        { branch_id: 'ca339d8a-4dd8-4cdb-85b1-f7abc942f6a1',
journal_id: '0045dc9a-8cef-4e17-976a-82a8b26ca9a4', amount: 15 },
        { branch_id: 'ca339d8a-4dd8-4cdb-85b1-f7abc942f6a1',
journal_id: 'eb7446ff-4ee3-410b-b41a-8d5ed0521cc6', amount: 12 },
        { branch_id: '9bc65a0e-4deb-4952-98a3-aadec14fd6af',
journal_id: '655f827b-c785-4b43-979d-47b6f11d31d4', amount: 5 },
        { branch_id: '787f74cf-4583-44eb-88cf-ac20433017a3',
journal_id: '664969a2-0804-4390-9400-7c24f8eb2066', amount: 8 },
        { branch_id: '7caf2960-6006-40c9-9455-a8d91a01f1b9',
journal_id: '1bb5c6ac-b86a-4d33-9e0a-f6a3e3dce2d5', amount: 10 },
        { branch_id: 'fc6b205a-3d6e-4862-baa0-1c5962fca291',
journal_id: 'ef947889-d288-4ded-86d2-5f26d46a9c85', amount: 7 },
        { branch_id: '00943eb3-5a06-4f2b-93de-7f82ae5732a8',
journal_id: '664969a2-0804-4390-9400-7c24f8eb2066', amount: 6 },
      ]);
```

```
      await Sale.bulkCreate([
        { id: '9a34773e-2d46-4a73-b099-ee745a62a3f6', reader_id:
'66c8c0fd-d849-43b4-ae97-5a7cf04a1d4e', shop_id: '4db426cb-00e3-
477c-9f78-9eaf82c968b3', journal_id: '0bc82198-02a1-4228-84fa-
2c48f4c7c793', date: '2024-01-15 10:30:00+03' },
        { id: 'c163b0ca-cd16-4585-ad0e-dc831927b090', reader_id:
'5740d88c-701c-4137-a4ba-91f78e993467', shop_id: '4db426cb-00e3-
477c-9f78-9eaf82c968b3', journal_id: 'f5751297-75b7-4283-a3e8-
f4bd0a84168c', date: '2024-01-16 14:20:00+03' },
        { id: 'c1f985bc-71cb-4719-bc64-c01ebc6d48d9', reader_id:
'9477efca-4b5e-418b-8983-22c70e74eb2b', shop_id: 'a8029dce-c70b-
43a7-bb69-0829d1d41bca', journal_id: '0045dc9a-8cef-4e17-976a-
82a8b26ca9a4', date: '2024-01-17 16:45:00+03' },
        { id: 'cabb4e87-f218-40f3-82ca-6b27b179c9ae', reader_id:
'5d10d4b5-9045-4aa0-956d-65c56b204140', shop_id: 'a8029dce-c70b-
43a7-bb69-0829d1d41bca', journal_id: '655f827b-c785-4b43-979d-
47b6f11d31d4', date: '2024-01-18 11:15:00+03' },
        { id: '008a3ef4-9786-4f28-9731-b581045cf338', reader_id:
'11309de7-2b40-4ee2-ae96-c891faf5a9d1', shop_id: 'cd707836-60d4-
4213-910d-eb6dad43fe86', journal_id: '664969a2-0804-4390-9400-
7c24f8eb2066', date: '2024-01-19 13:30:00+03' },
        { id: '5404b7d2-989d-4241-933e-3fe6492c291d', reader_id:
'66c8c0fd-d849-43b4-ae97-5a7cf04a1d4e', shop_id: '7e0f0fc4-fcf4-
4bcb-a415-e0824f1b6412', journal_id: 'ef947889-d288-4ded-86d2-
5f26d46a9c85', date: '2024-01-20 15:20:00+03' },
        { id: 'c2fb3ebd-361b-40a0-b060-bff05282bb84', reader_id:
'5740d88c-701c-4137-a4ba-91f78e993467', shop_id: '7e0f0fc4-fcf4-
4bcb-a415-e0824f1b6412', journal_id: '1bb5c6ac-b86a-4d33-9e0a-
f6a3e3dce2d5', date: '2024-01-21 12:40:00+03' },
        { id: '0f8aaeb5-8470-4723-97b9-3371f2f9cbea', reader_id:
'9477efca-4b5e-418b-8983-22c70e74eb2b', shop_id: 'a8029dce-c70b-
43a7-bb69-0829d1d41bca', journal_id: 'eb7446ff-4ee3-410b-b41a-
8d5ed0521cc6', date: '2024-01-22 17:10:00+03' },
        { id: '00a0bd29-88ee-4cbc-8503-76f9f2ea6499', reader_id:
'5d10d4b5-9045-4aa0-956d-65c56b204140', shop_id: 'dba0ed7e-d2fc-
```

```
47d7-be1e-db8a3a9d0476',   journal_id:   'ef947889-d288-4ded-86d2-
5f26d46a9c85', date: '2024-01-23 09:50:00+03' },
        { id: '47e71442-9308-402c-8bd9-a4c1c1b8b786',  reader_id:
'11309de7-2b40-4ee2-ae96-c891faf5a9d1',  shop_id:  '4db426cb-00e3-
477c-9f78-9eaf82c968b3',   journal_id:   '0bc82198-02a1-4228-84fa-
2c48f4c7c793', date: '2024-01-24 18:30:00+03' },
      ]);

      console.log('Schema synced and data seeded');
      await sequelize.close();
    }

    main().catch(err => {
      console.error(err);
      process.exit(1);
    });
```

Файл scripts/test.js:

```
    const  {  sequelize,  Reader,  Journal,  Sale,  Shop  }  =
require('../models');
    const { Op, Sequelize } = require('sequelize');

    async function run() {
      // 1) Какие журналы были куплены определенным покупателем?
      {
        const email = 'ivan.petrov@email.com';
        const rows = await Sale.findAll({
          include: [
            { model: Reader, attributes: [] },
            { model: Journal, attributes: [] },
          ],
          where: { '$Reader.email$': email },
          attributes: [
            [sequelize.col('Reader.email'), 'reader'],
            [sequelize.col('Journal.name'), 'journal'],
```

```
    ],
    raw: true,
  });
  console.log('\n1) Journals by reader email:', rows);
}

// 2) Как называется журнал с заданным ISBN?
{
  const isbn = '978-1-302-95001-1';
  const row = await Journal.findOne({
    attributes: ['isbn', ['name', 'journal']],
    where: { isbn },
    raw: true,
  });
  console.log('\n2) Journal by ISBN:', row);
}

// 3) Какой ISBN у журнала с заданным названием?
{
  const name = 'Teen Titans Go!';
  const row = await Journal.findOne({
    attributes: [['name', 'journal'], 'isbn'],
    where: { name },
    raw: true,
  });
  console.log('\n3) ISBN by journal name:', row);
}

// 4) Когда журнал был куплен? (по названию)
{
  const name = 'Teen Titans Go!';
  const rows = await Sale.findAll({
    include: [{ model: Journal, attributes: [] }],
    where: { '$Journal.name$': name },
    attributes: [
```

```
        [sequelize.col('Journal.name'), 'journal'],
        'date',
      ],
      order: [['date', 'ASC']],
      raw: true,
    });
    console.log('\n4) Purchases of journal:', rows);
  }

  // 5) Кто из покупателей купил журнал более месяца назад?
  {
    const rows = await Sale.findAll({
      include: [
        { model: Journal, attributes: [] },
        { model: Reader, attributes: [] },
      ],
      where: Sequelize.where(
        Sequelize.col('date'),
        '<',
        Sequelize.literal(`NOW() - INTERVAL '1 month'`)
      ),
      attributes: [
        'date',
        [sequelize.col('Journal.name'), 'journal'],
        [sequelize.col('Reader.email'), 'reader'],
      ],
      order: [['date', 'ASC']],
      raw: true,
    });
    console.log('\n5) Readers who bought > 1 month ago:',
rows);
  }

  // 6) Найти покупателя самых редких журналов (по наличию в
магазине).
```

```
    // raw SQL
    {
      const [rows] = await sequelize.query(`
        WITH rare_journal AS (
          SELECT j.id, j.name, COALESCE(SUM(c.amount), 0) AS
copies
          FROM journal j
          LEFT JOIN catalog c ON j.id = c.journal_id
          GROUP BY j.id, j.name
          HAVING COALESCE(SUM(c.amount), 0) > 0
          ORDER BY copies ASC
          LIMIT 3
        )
        SELECT r.email AS reader, rj.name AS rare_journal,
rj.copies
        FROM transaction t
        JOIN reader r ON t.reader_id = r.id
        JOIN rare_journal rj ON t.journal_id = rj.id;
      `);
      console.log('\n6) Buyers of rare journals:', rows);
    }

    // 7) Какое число покупателей пользуется определенным
магазином?
    {
      const shopId = '4db426cb-00e3-477c-9f78-9eaf82c968b3';
      const count = await Sale.count({
        where: { shop_id: shopId },
        distinct: true,
        col: 'reader_id',
      });
      console.log('\n7) Distinct readers for shop:', count);
    }

    // 8) Сколько покупателей младше 20 лет?
```

```
  {
    const [row] = await sequelize.query(`
      SELECT COUNT(DISTINCT id) AS readers
      FROM reader
      WHERE birth_date > (CURRENT_DATE - INTERVAL '20 years');
    `);
    console.log('\n8) Readers younger than 20:', row[0]);
  }

  await sequelize.close();
}

run().catch(err => {
  console.error(err);
  process.exit(1);
});
```