

Day 05 - Piscine Java

SQL/JDBC

Резюме: Сегодня вы используете основные механики для работы с СУБД PostgreSQL с помощью JDBC

Contents

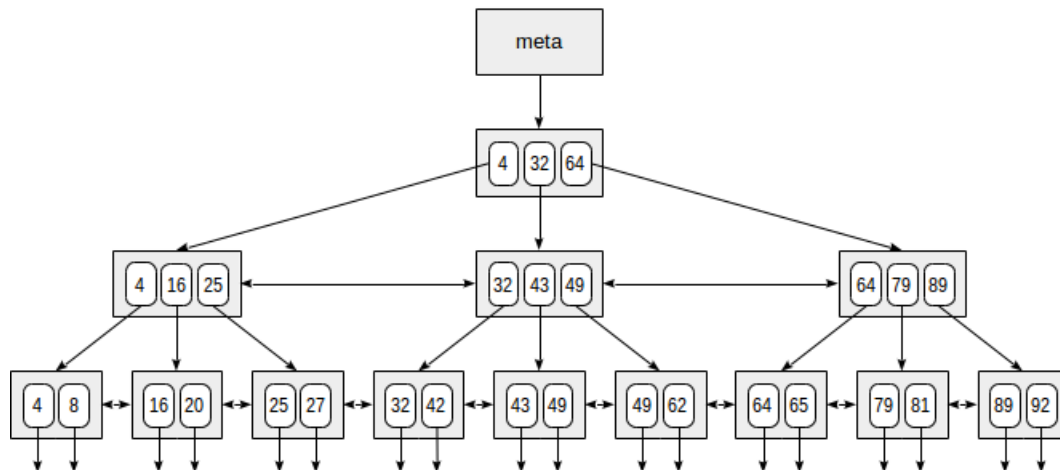
Preamble	3
General Rules	4
Exercise 00 - Tables & Entities	5
Exercise 01 - Read/Find	7
Exercise 02 - Create/Save	9
Exercise 03 - Update	10
Exercise 04 - Find All	11

Chapter I

Preamble

Как известно, реляционные базы данных состоят из набора связанных таблиц. Каждая таблица имеет набор строк и столбцов. Очевидно, что, если в таблице присутствует огромное количество строк, поиск данных с определенным значением столбца может занять длительное время.

Для решения данной проблемы в современных СУБД используется механизм индексов. Одной из реализацией индексов является структура данных BTree.



Такой индекс может быть использован для какого-либо столбца таблицы. Поскольку дерево всегда сбалансировано, поиск любого значения занимает одинаковое время.

Правила, согласно которым происходит значительное ускорение поиска, заключаются в следующем:

1. Ключи в каждом узле упорядочены.
2. Корень содержит от **1** до **t-1** ключей.
3. Любой другой узел от **t-1** до **2t-1** ключей.
4. Если узел содержит ключи **k1, k2, ... kn**, то он имеет **n+1** потомков.
5. Первый потомок и все его потомки содержат ключи, меньшие или равные **k1**.
6. Последний потомок и все его потомки содержат ключи, большие или равные **kn**.
7. Для $2 \leq i \leq n$, **i**-ый потомок и все его потомки содержат ключи из интервала (**ki-1, ki**).

Таким образом, для поиска какого-либо значения необходимо просто понять, в какой из потомков следует спуститься. Это позволяет не просматривать всю таблицу целиком.

Понятно, что данный подход имеет множество нюансов. Например, если в таблицу постоянно попадают новые значения или обновления данных, СУБД будет постоянно перестраивать индекс, замедляя работу системы.

Chapter II

General Rules

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- Сейчас для вас существует только одна версия Java - 1.8. Убедитесь, что на вашем компьютере установлен компилятор и интерпретатор данной версии.
- Не запрещено использовать IDE для написания исходного кода и его отладки.
- Код чаще читается, чем пишется. Внимательно изучите представленный [документ](https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html) с правилами оформления кода. В каждом задании обязательно придерживайтесь общепринятых стандартов Oracle - <https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html>
- Комментарии в исходном коде вашего решения запрещены. Они мешают восприятию.
- Pay attention to the permissions of your files and directories.
- To be assessed your solution must be in your GIT repository.
- Your solutions will be evaluated by your piscine mates.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google. И еще, для любых ваших вопросов существует ответ на Stackoverflow. Научитесь правильно их задавать.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- And may the Force be with you!
- Не откладывайте на завтра то, что можно было сделать вчера ;)

Chapter III

Rules of the day

- Во всех заданиях используйте СУБД PostgreSQL
- Подключите актуальную версию драйвера JDBC
- Для взаимодействия с базой данных разрешено использование классов и интерфейсов пакета `java.sql` (реализации соответствующих интерфейсов будут автоматически включены из архива с драйвером).

Chapter IV

Exercise 00 - Tables & Entities

Exercise 00: Tables & Entities	
Turn-in directory	ex00
Files to turn-in	Chat-folder

На протяжении этой недели мы реализуем функциональность Чата. В таком чате пользователь может создать или выбрать уже созданную комнату. В каждой комнате может быть несколько пользователей, обменивающихся сообщениями.

Основные модели предметной области, для которых необходимо реализовать как SQL-таблицы, так и java-классы:

- Пользователь
 - Идентификатор пользователя
 - Логин
 - Пароль
 - Список созданных комнат
 - Список комнат, в которых общается пользователь
- Комната
 - Идентификатор комнаты
 - Название комнаты
 - Создатель комнаты
 - Список сообщений в комнате
- Сообщение
 - Идентификатор сообщения
 - Автор сообщения
 - Комната сообщения
 - Текст сообщения
 - Дата/время отправки

Создайте файл `schema.sql`, в котором вы опишете операции `CREATE TABLE` для создания необходимых для проекта таблиц. Также необходимо создать файл `data.sql` с `INSERT`-ами тестовых данных (не менее 5 в каждую таблицу).

Важно соблюдать следующее требование!

Пусть сущность `Course` связана с сущностью `Lesson` реляционным отношением один-ко-многим. Тогда их объектно-ориентированное отношение должно выглядеть следующим образом:

```

class Course {
    private Long id;
    private List<Lesson> lessons; // внутри курса - много уроков
    ...
}
class Lesson {
    private Long id;
    private Course course; // урок содержит курс, к которому
    он привязан
    ...
}

```

Дополнительные требования:

- Для реализации реляционных отношений используйте типы связей один-ко-многим, многие-ко-многим.
- Идентификаторы должны быть числовыми.
- Генерацию идентификаторов должна выполнять СУБД.
- Внутри java-классов следует корректно переопределить equals(), hashCode() и toString().

Структура проекта упражнения:

- Chat
 - src
 - main
 - java
 - edu.school21.chat
 - models - модели предметной области
 - resources
 - schema.sql
 - data.sql
 - pom.xml

Chapter V

Exercise 01 - Read/Find

Exercise 01: Read/Find	
Turn-in directory	ex01
Files to turn-in	Chat-folder

Data Access Object (DAO, Repository) - популярный шаблон проектирования, позволяющий разделить основную бизнес-логику приложения от логики работы с данными.

Таким образом, пусть есть некоторый интерфейс `CoursesRepository`, который предоставляет доступ к урокам какого-либо курса. Данный интерфейс может иметь следующий вид:

```
public interface CoursesRepository {
    Optional<Course> findById(Long id);
    void delete(Course course);
    void save(Course course);
    void update(Course course);

    List<Course> findAll();
}
```

Вам необходимо реализовать интерфейс `MessagesRepository` ТОЛЬКО с одним методом `Optional<Message> findById(Long id)` и его имплементацию `MessagesRepositoryJdbcImpl`.

Данный метод должен возвращать объект `Message`, в котором также будут указаны автор и комната. В свою очередь, у автора и комнаты НЕ НУЖНО заполнять подсушности - список комнат, создателя комнаты и т.д.

В классе `Program.java` протестировать реализованный код. Пример работы программы (вывод может отличаться):

```
$ java Program
```

```
В в е д и т е  i d  с о о б щ е н и я
```

```
-> 5
```

```
Message : {
    id=5,
    author={id=7,login="user",password="user",createdRooms=null,rooms=null},
    room={id=8,name="room",creator=null,messages=null},
    text="message",
```



```
        dateTime=01/01/01 15:69
    }
```

Структура проекта упражнения:

- Chat
 - src
 - main
 - java
 - edu.school21.chat
 - models - модели предметной области
 - repositories - репозитории
 - app
 - Program.java
 - resources
 - schema.sql
 - data.sql
 - pom.xml
 - MessagesRepositoryJdbcImpl должен принимать интерфейс DataSource пакета java.sql как параметр конструктора.
 - В качестве реализации DataSource используйте библиотеку HikariCP, представляющую собой пул соединений с базой данных, что значительно ускоряет работу с хранилищем.

Chapter VI

Exercise 02 - Create/Save

Exercise 02: Create/Save	
Turn-in directory	ex02
Files to turn-in	Chat-folder

Сейчас вам необходимо реализовать метод `save(Message message)` для `MessagesRepository`.

Таким образом, для сохраняемой сущности должны быть определены подсущности - автор сообщения и комната. При этом важно, чтобы у комнаты и автора были заданы `id`, существующие в базе данных.

Пример использования метода `save`:

```
public static void main(String args[]) {
    ...
    User creator = new User(7L, "user", "user", new ArrayList(), new
ArrayList());
    User author = creator;
    Room room = new Room(8L, "room", creator, new ArrayList());
    Message message = new Message(null, author, room, "Hello!",
LocalDateTime.now());
    MessagesRepository messagesRepository = new
MessagesRepositoryJdbcImpl(...);
    messagesRepository.save(message);
    System.out.println(message.getId()); // ex. id == 11
}
```

Таким образом, метод `save` должен задавать значение `id` для входящей модели после сохранения данных в БД.

В случае, если у сущности `author` и `room` не заданы `id`, существующие в бд, либо эти `id` имеют значение `null` - выбросить `RuntimeException` `NotSavedSubEntityException` (исключение реализовать самостоятельно).

В классе `Program.java` протестировать реализованный код.

Chapter VII

Exercise 03 - Update

Exercise 03: Update	
Turn-in directory	ex03
Files to turn-in	Chat-folder

Сейчас необходимо реализовать метод `update` в `MessageRepository`. Данный метод должен полностью обновлять существующую сущность в базе данных. Если новое значение какого-либо из полей обновляемой сущности имеет значение `null` - такое значение должно быть сохранено в базе данных.

Пример использования метода `update`:

```
public static void main(String args[]) {
    MessagesRepository messagesRepository = new
MessagesRepositoryJdbcImpl(...);
    Optional<Message> messageOptional = messagesRepository.findById(11);
    if (messageOptional.isPresent()) {
        Message message = messageOptional.get();
        message.setText("Bye");
        message.setDateTime(null);
        messagesRepository.update(message);
    }
    ...
}
```

В данном примере значение колонки для хранения текста сообщения будет изменено, а значение времени отправки сообщения будет иметь значение `null`.

Chapter VIII

Exercise 04 - Find All

Exercise 04: Find All	
Turn-in directory	ex04
Files to turn-in	Chat-folder

Сейчас вам необходимо реализовать интерфейс `UsersRepository` и класс `UsersRepositoryJdbcImpl` с ОДНИМ методом `List<User> findAll(int page, int size)`.

Данный метод должен возвращать `size`-пользователей, размещенных на странице с номером `page`. Такое “кусочное” получение данных называется пагинацией. Таким образом, все результирующее множество СУБД разбивает на страницы, каждая из которых содержит `size`-записей. Например, если множество содержит 20 записей, то при `page = 3` и `size = 4` вы получаете пользователей с 12-го по 15-й (нумерация страниц и пользователей начинается с 0).

Наиболее сложная ситуация при работе с конвертацией реляционных отношений в объектно-ориентированные связи - это получение набора сущностей вместе с их подсущностями. В текущем задании необходимо, чтобы каждый пользователь из результирующего списка имел включенные зависимости - список комнат созданных пользователем, а также список комнат, в которых участвует пользователь.

Каждая подсущность пользователя НЕ ДОЛЖНА включать свои зависимости, т.е. список сообщений внутри каждой комнаты пользователя должен быть пустым.

В `Program.java` необходимо продемонстрировать работу реализуемого метода.

Примечания

- Метод `findAll(int page, int size)` должен быть реализован ОДНИМ запросом в БД, недопустимо использование дополнительных SQL-запросов для получения информации по каждому пользователю.
- Рекомендуется использование CTE PostgreSQL.
- `UsersRepositoryJdbcImpl` должен принимать интерфейс `DataSource` пакета `java.sql` как параметр конструктора.

CHECKLIST

https://docs.google.com/document/d/1EEHF5NHP3h1_V1kGdROppeyiN5YlYsFivleDREL-ipo/edit?usp=sharing