

Модуль 05 - Бассейн Java

SQL/JDBC

Резюме: Сегодня вы будете использовать ключевые механизмы для работы с СУБД PostgreSQL через JDBC.

Содержание

I Предисловие

II Инструкции

III Правила дня

IV Упражнение 00: Таблицы и объекты

V Упражнение 01: Чтение/Найти

VI Упражнение 02: Создать/Сохранить

VII Упражнение 03: Обновление

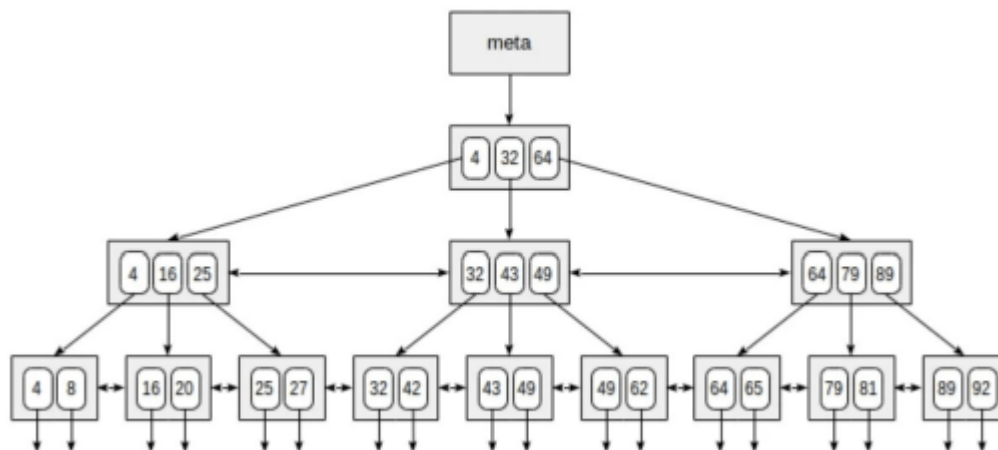
VIII Упражнение 04. Найти все

## Глава I

### Предисловие

Как известно, реляционные базы данных состоят из набора связанных таблиц. Каждая таблица имеет набор строк и столбцов. Если в таблице огромное количество строк, поиск данных с определенным значением в столбце, очевидно, может занять много времени.

Для решения этой проблемы современные СУБД используют индексный механизм. Структура данных BTree является реализацией концепции индекса.



Этот индекс может использоваться для столбца таблицы. Поскольку дерево всегда сбалансировано, поиск любого значения занимает одинаковое количество времени.

Правила, которые существенно ускоряют поиск, заключаются в следующем:

- Ключи в каждом узле упорядочены.
- Корень содержит ключи от 1 до  $t-1$ .
- Любой другой узел содержит ключи от  $t-1$  до  $2t-1$ .
- Если узел содержит  $k_1, k_2, \dots, k_n$  ключей, он имеет  $n+1$  производных классов.
- Первый производный класс и все его производные классы содержат ключи, которые меньше или равны  $k_1$ .
- Последний производный класс и все его производные классы содержат ключи, которые больше или равны  $k_n$ .

Для  $2 \leq i \leq n$   $i$ -й производный класс и все его производные классы содержат ключи в диапазоне  $(k_{i-1}, k_i)$ .

Поэтому для поиска значения нужно просто определить, к какому производному классу спускаться. Это позволяет не просматривать всю таблицу.

Этот подход, по-видимому, имеет много специфики. Например, если в таблицу постоянно поступают новые значения, СУБД всегда будет перестраивать индекс, что будет замедлять работу системы.


## Глава III

### Правила дня

- Использовать СУБД PostgreSQL во всех задачах.
- Подключить актуальную версию драйвера JDBC.
- Для взаимодействия с базой данных можно использовать классы и интерфейсы пакета `java.sql` (реализации соответствующих интерфейсов будут автоматически включены из архива с драйвером).

## Глава IV

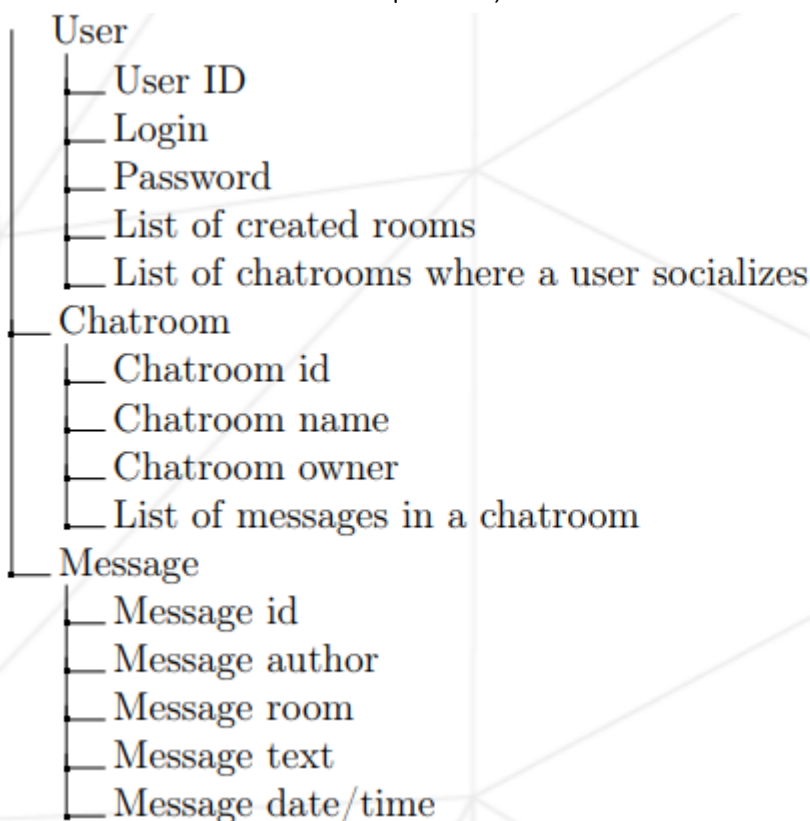
### Упражнение 00. Таблицы и объекты

	Exercise 00
Tables and Entities	
Turn-in directory : <i>ex00/</i>	
Files to turn in : Chat-folder	
Allowed functions : n/a	

В течение этой недели мы будем реализовывать функцию чата.

В этом чате пользователь может создать или выбрать существующий чат. В каждом чате может быть несколько пользователей, обменивающихся сообщениями.

Ключевые модели предметной области, для которых должны быть реализованы как таблицы SQL, так и классы Java:



Создайте файл `schema.sql`, в котором вы будете описывать операции `CREATE TABLE` для создания таблиц для проекта. Вы также должны создать файл `data.sql` с текстовыми данными `INSERTS` (не менее пяти в каждой таблице).

Важно выполнить следующее требование!

Предположим, что объект «Курс» имеет связь «один ко многим» с объектом «Урок». Тогда их объектно-ориентированное отношение должно выглядеть следующим образом:

```

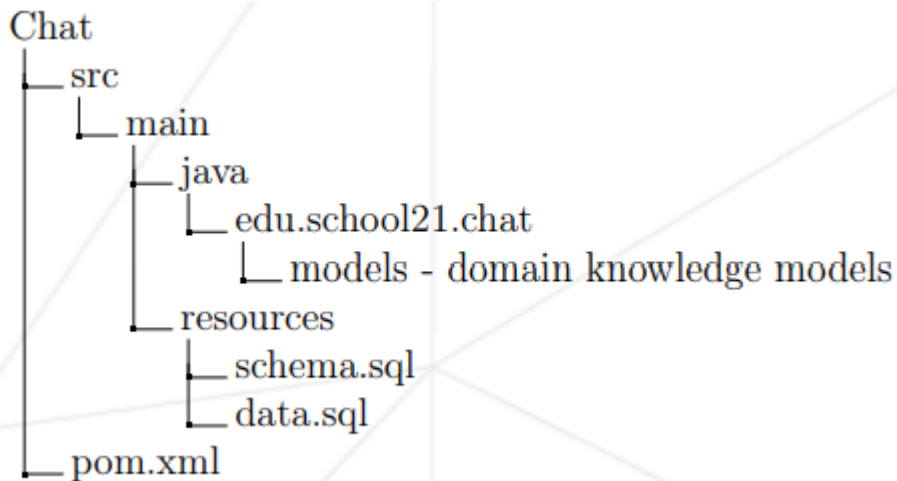
class Course {
    private Long id;
    private List<Lesson> lessons; // there are numerous lessons in the course
    ...
}
class Lesson {
    private Long id;
    private Course course; // the lesson contains a course it is linked to
    ...
}

```

Дополнительные требования:


- Для реализации реляционных связей используйте типы ссылок «один ко многим» и «многие ко многим».
- Идентификаторы должны быть числовыми.
- Идентификаторы должны генерироваться СУБД.
- equals(), hashCode() и toString() должны корректно переопределяться внутри классов Java.

Структура проекта упражнения:



## Глава V

### Упражнение 01: Чтение/Найти

	Exercise 01
Read/Find	
Turn-in directory : <i>ex01/</i>	
Files to turn in : Chat-folder	
Allowed functions : n/a	

Объект доступа к данным (DAO, репозиторий) — это популярный шаблон проектирования, который позволяет отделить ключевую бизнес-логику от логики обработки данных в приложении.

Предположим, что у нас есть интерфейс под названием `CoursesRepository`, который обеспечивает доступ к урокам курса. Этот интерфейс может выглядеть следующим образом:

```
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 `MessagesRepository` с помощью метода `SINGLE Optional<Message> findById(Long id)` и его реализации `MessagesRepositoryJdbcImpl`.

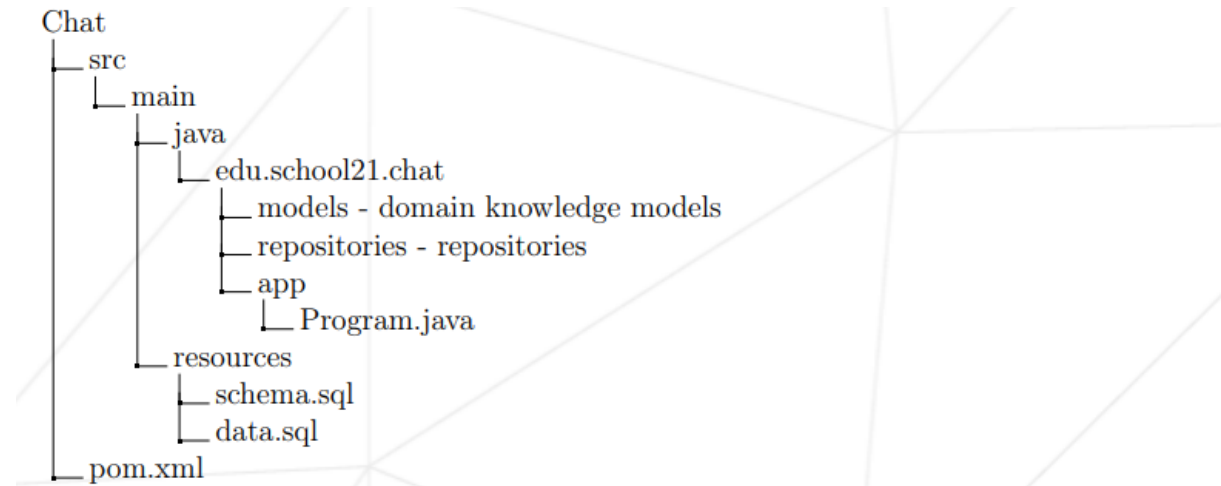
Этот метод должен возвращать объект `Message`, в котором будут указаны автор и чат.

В свою очередь НЕ НАДО вводить подсущности (список чатов, создатель чата и т.д.) для автора и чата.

Реализованный код необходимо протестировать в классе `Program.java`. Пример работы программы следующий (вывод может отличаться):

```
$ java Program
Enter a message ID
-> 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:




- MessagesRepositoryJdbcImpl должен принимать интерфейс DataSource пакета java.sql в качестве параметра конструктора.
- Для реализации DataSource используйте библиотеку HikariCP — пул подключений к базе данных, значительно ускоряющий использование хранилища.



## Глава VI

### Упражнение 02: Создать/Сохранить

	Exercise 02
Create/Save	
Turn-in directory : <i>ex02/</i>	
Files to turn in : Chat-folder	
Allowed functions : n/a	

Теперь вам нужно реализовать метод `save(Message message)` для `MessagesRepository`.

Таким образом, нам нужно определить следующие подсущности для сохраняемой нами сущности — автора сообщения и чата. Также важно присвоить идентификаторы, существующие в базе данных, чату и автору.

Пример использования метода сохранения:


```
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 MessagesRepository.JdbcImpl(...);  
    messagesRepository.save(message);  
    System.out.println(message.getId()); // ex. id == 11  
}
```

Таким образом, метод сохранения должен присвоить значение идентификатора для входящей модели после сохранения данных в базе данных.

- Если автор и комната не имеют идентификатора, существующего в назначенной базе данных, или эти идентификаторы пусты, сгенерируйте исключение `RuntimeException NotSavedSubEntityException` (внедрите это исключение самостоятельно).
- Протестируйте реализованный код в классе `Program.java`.

## Глава VII

### Упражнение 03. Обновление

	Exercise 03
Update	
Turn-in directory : <i>ex03/</i>	
Files to turn in : Chat-folder	
Allowed functions : n/a	

Теперь нам нужно реализовать метод обновления в `MessageRepository`. Этот метод должен полностью обновить существующий объект в базе данных. Если новое значение поля в обновляемом объекте равно нулю, это значение должно быть сохранено в базе данных.


Пример использования метода обновления:

```
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 этом примере значение столбца, в котором хранится текст сообщения, будет изменено, тогда как время сообщения будет равно нулю.

## Глава VIII

### Упражнение 04. Найти все

	Exercise 04
Find All	
Turn-in directory : <i>ex04/</i>	
Files to turn in : <i>Chat-folder</i>	
Allowed functions : <i>n/a</i>	

Теперь вам нужно реализовать интерфейс `UsersRepository` и класс `UsersRepositoryJdbcImpl`, используя метод `SINGLE List<User> findAll(int page, int size)`.

Этот метод должен возвращать размер — пользователей, показанных на странице с номером страницы. Этот «кусочный» поиск данных называется разбиением на страницы. Таким образом, СУБД делит общий набор на страницы, каждая из которых содержит элементы размера. Например, если набор содержит 20 записей со страницей = 3 и размером = 4, вы получаете пользователей с 12 по 15 (нумерация пользователей и страниц начинается с 0).

Наиболее сложная ситуация при преобразовании реляционных ссылок в объектно-ориентированные возникает, когда вы извлекаете набор сущностей вместе с их подсущностями. В этой задаче каждый пользователь в результирующем списке должен иметь зависимости — список чатов, созданных этого пользователя, а также список чатов, в которых он участвует.

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

Работа реализованного метода должна быть продемонстрирована в `Program.java`.

#### Заметки

- метод `findAll(int page, int size)` должен быть реализован ОДНИМ запросом к базе данных. Не допускается использование дополнительных SQL-запросов для получения информации для каждого пользователя.
- Мы рекомендуем использовать CTE PostgreSQL. `UsersRepositoryJdbcImpl` должен принимать интерфейс `DataSource` пакета `java.sql` в качестве параметра конструктора.