



Модуль 05 - Piscine Java

SQL/JDBC

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

Содержание

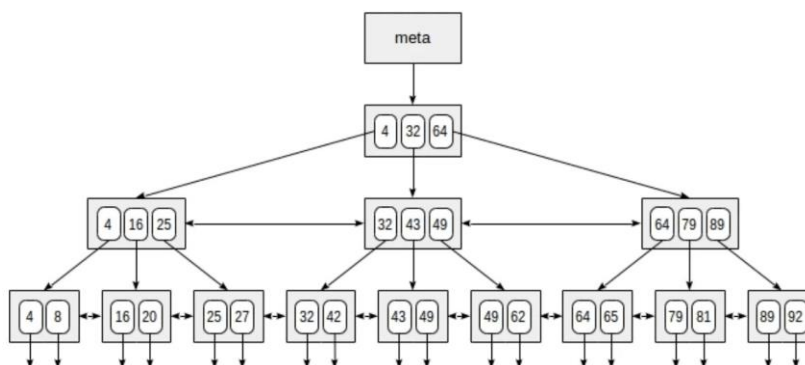
I	Предисловие	2
II	Инструкции	4
III	Правила дня	6
IV	Упражнение Таблицы и сущности 00 :	7
V	Упражнение Читать/найти 01 :	9
VI	Упражнение Создать/Сохранить 02 :	11
VII	Упражнение Обновление 03 :	12
VIII	Упражнение Найти все 04 :	13

Глава I

Предисловие

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

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



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

Правила, которые существенно ускоряют проведение обыска, следующие:

- Ключи в каждом узле упорядочены.
- Корень содержит от 1 до $t-1$ ключей.
- Любой другой узел содержит от $t-1$ до $2t-1$ ключей.
- Если узел содержит k_1, k_2, \dots, k_n ключей, то 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 достаточно определить, к какому производному классу следует спуститься. Это позволяет избежать просмотра всей таблицы.

Такой подход, очевидно, имеет много особенностей. Например, если в таблицу постоянно поступают новые значения, СУБД будет постоянно перестраивать индекс, что замедлит работу системы.

Глава II

Инструкции

- Используйте эту страницу как единственную ссылку. Не слушайте никаких слухов и домыслов о том, как приготовить раствор.
- Теперь для вас существует только одна версия Java - 1.8. Убедитесь, что компилятор и интерпретатор этой версии установлены на вашей машине.
- Вы можете использовать IDE для написания и отладки исходного кода.
- Код чаще читают, чем пишут. Внимательно прочитайте [документ](#), в котором приведены правила форматирования кода. При выполнении каждой задачи убедитесь, что вы следуете общепринятым [стандартам Oracle](#)
- Комментарии не допускаются в исходном коде вашего решения. Они затрудняют чтение кода.
- Обратите внимание на разрешения ваших файлов и каталогов.
- Чтобы ваше решение было оценено, оно должно находиться в вашем GIT-репозитории.
- Ваши решения будут оценивать ваши товарищи по аквариуму.
- Вы не должны оставлять в своем каталоге никаких других файлов, кроме тех, которые явно указаны в инструкциях к упражнению. Рекомендуется изменить свой .gitignore во избежание несчастных случаев.
- Когда вам нужно получить точный вывод в ваших программах, запрещено выводить предварительно рассчитанный вывод вместо правильного выполнения упражнения.
- У вас есть вопрос? Спросите своего соседа справа. В противном случае попробуйте поговорить с соседом слева.
- Ваше справочное пособие: товарищи / Интернет / Google. И еще кое-что. На любой ваш вопрос есть ответ на Stackoverflow. Научитесь правильно задавать вопросы.
- Внимательно прочитайте примеры. В них могут потребоваться вещи, которые не указаны в предмете.
- Используйте "System.out" для вывода

- И да пребудет с вами Сила!
- Никогда не оставляйте на завтра то, что вы можете сделать сегодня ;)


Глава III

Правила дня

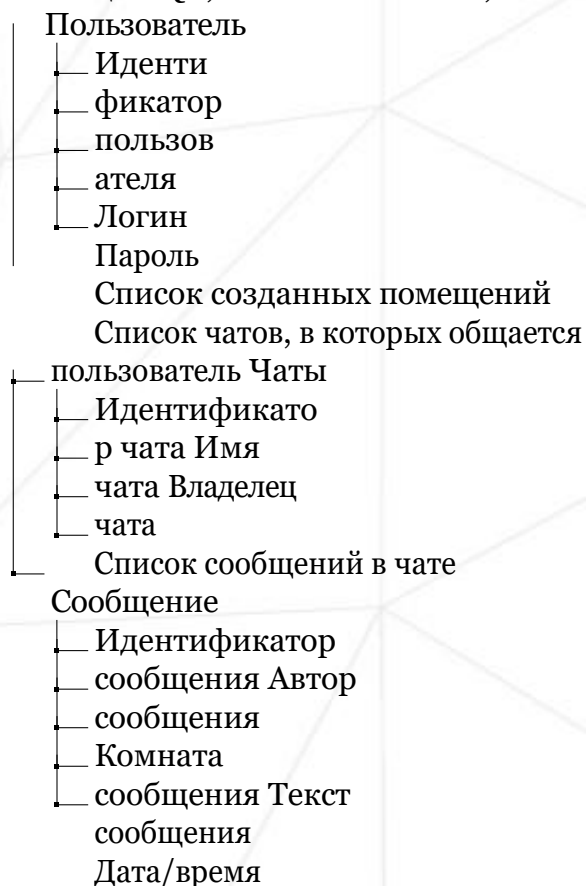
- Используйте СУБД PostgreSQL во всех задачах.
- Подключите актуальную версию драйвера JDBC.
- Для взаимодействия с базой данных вы можете использовать классы и интерфейсы пакета `java.sql` (реализации соответствующих интерфейсов будут автоматически включены из архива, содержащего драйвер).

Глава IV

Упражнение 00: Таблицы и объекты

	Упражнение 00
Таблицы и	
Каталог для сдачи : ex00/	
Файлы для сдачи : Папка с	сущности
чатом Разрешенные	
функции : n/a	

В течение этой недели мы будем реализовывать функциональность чата. В этом чате пользователь может создать или выбрать существующий чат. В каждом чате может быть несколько пользователей, обменивающихся сообщениями. Ключевыми моделями домена, для которых должны быть реализованы как таблицы SQL, так и классы Java, являются:



сообщения

Создайте файл `schema.sql`, в котором вы опишите 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 соблюдать следующее требование!

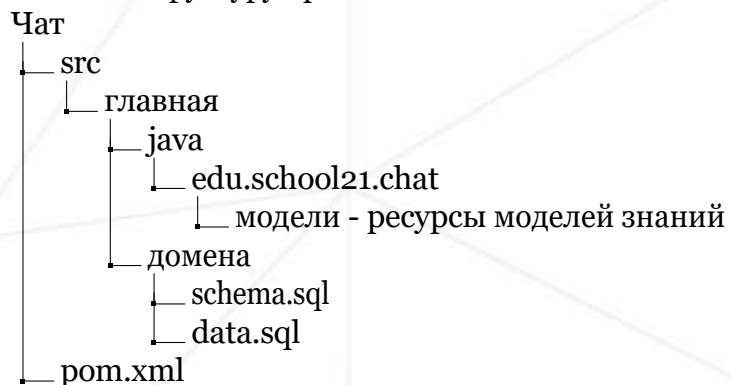
Предположим, что сущность Course имеет отношение "один ко многим" с сущностью Lesson. Тогда их объектно-ориентированное отношение должно выглядеть следующим образом:

```
class Course {  
    private Long id;  
    private List<Lesson> lessons; // в курсе есть множество уроков  
    ...  
}  
class Lesson {  
    private Long id;  
    private Course course; // урок содержит курс, с которым он связан  
    ...  
}
```

Дополнительные требования:


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

Выполните структуру проекта:



Глава V

Упражнение 01: чтение/находка

	Упражнение 01
	Читать/найти
	Каталог для сдачи : ex01/
	Файлы для сдачи : Папка с
	чатом Разрешенные
	функции : n/a

Объект доступа к данным (DAO, Repository) - это популярный шаблон проектирования, который позволяет отделить ключевую бизнес-логику от логики работы с данными в приложении.

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

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

следующим образом:

Вам необходимо реализовать `MessagesRepository` с методом `SINGLE Optional<Message> findById(Long id)` и его реализацию `MessagesRepositoryJdbcImpl`.

Этот метод возвращает объект `Message`, в котором будут указаны автор и чат. В свою очередь, для автора и чата НЕ НУЖНО вводить подсубъекты (список чатов, создатель чата и т.д.).

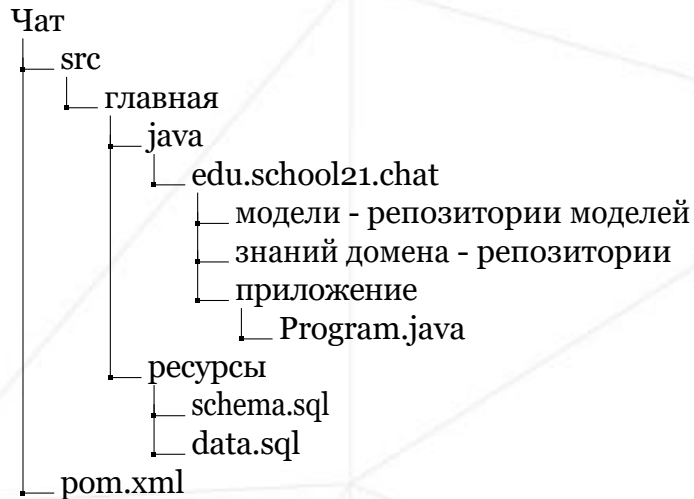
Реализованный код должен быть протестирован в классе `Program.java`. Пример работы программы следующий (вывод может отличаться):

```
$ java Программа  
Введите  
идентификатор  
сообщения  
-> 5  
Сообщение : {
```



```
room={id=8,name="room",creator=null,messages=null},
text="сообщение",
dateTime=01/01/01 15:69
}
```


Выполните структуру проекта:



- MessagesRepositoryJdbcImpl должен принимать интерфейс DataSource из пакета java.sql в качестве параметра конструктора.
- Для реализации DataSource используйте библиотеку HikariCP - пул соединений с базой данных, который значительно ускоряет использование хранилища.

Глава VI

Упражнение 02 : Создать/сохранить

	Упражнение 02
	Создание/сохранен
Каталог для сдачи : ex02/	
Файлы для сдачи : Папка с	ие
чатом Разрешенные	
функции : n/a	

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

Таким образом, нам нужно определить следующие сущности для сохраняемой нами сущности - автор сообщения и чат. Также важно присвоить чату и автору идентификаторы, существующие в базе данных.


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

- Таким образом, метод сохранения должен присвоить значение ID для входящей модели после сохранения данных в базе данных.
- Если автор и комната не имеют ID, существующих в базе данных, или эти ID равны null, бросьте `RuntimeException NotSavedSubEntityException` (реализуйте это исключение самостоятельно).
- Протестируйте реализованный код в классе `Program.java`.

Глава VII

Упражнение 03: Обновление

	Упражнение 03
Каталог для сдачи : <i>ex03/</i>	Обновление
Файлы для сдачи : Папка с	е
чатом Разрешенные	
функции : н/а	

Теперь нам нужно реализовать метод `update` в `MessageRepository`. Этот метод должен полностью обновить существующую сущность в базе данных. Если новое значение поля в обновляемой сущности равно `null`, то это значение будет 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, хранящего текст сообщения, будет изменено, а время сообщения будет равно нулю.

Глава VIII

Упражнение 04: Найти все

	Упражнение 04
Каталог для сдачи : <i>ex04/</i>	Найти все
Файлы для сдачи : Папка с	
чатом Разрешенные	
функции : п/а	

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

Этот метод возвращает пользователей `size`, показанных на странице с номером страницы. Такой "кусочный" поиск данных называется **постраничным**. Таким образом, СУБД делит общий набор на страницы, каждая из которых содержит записи `size`. Например, если набор содержит 20 записей со страницей = 3 и размером = 4, вы извлекаете пользователей с 12 по 15 (нумерация пользователей и страниц начинается с 0).

Самая сложная ситуация при преобразовании реляционных ссылок в объектно-ориентированные возникает, когда вы получаете набор сущностей вместе с их подсущностями. В этой задаче каждый пользователь в полученном списке должен содержать зависимости - список чатов, созданных этим пользователем, а также список чатов, в которых он участвует.

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

Реализованная работа метода должна быть продемонстрирована в

`Program.java`. Примечания

- метод `findAll(int page, int size)` должен быть реализован ОДНОЙ базой данных запрос. Не допускается использование дополнительных SQL-запросов для получения информации для каждого пользователя.

- Мы рекомендуем использовать CTE PostgreSQL.

- `UsersRepositoryJdbcImpl` должен принимать интерфейс `DataSource` из пакета `java.sql` в качестве параметра конструктора.