

Вот примеры простых запросов, которые демонстрируют некоторые возможности SQL (MariaDB).

|   |  |
|---|--|
| SELECT * FROM Article   | Вернуть все строки из таблицы Article в случайном порядке  |
| SELECT * FROM Article ORDER BY id<br>SELECT * FROM Article ORDER BY creationTime DESC, id           | Примеры указания порядка   |
| SELECT id, userId FROM Article  | Вернуть только столбцы id, userId для всех строк Article   |
| SELECT DISTINCT(userId) FROM Article  | Вернуть все различные userId из Article - то есть тех, кто хоть что-то написал                                     |
| SELECT * FROM User WHERE id IN (SELECT userId FROM Article) AND creationTime>NOW() - INTERVAL 7 DAY | Вернуть все строки из User, которые соответствуют тем, кто хоть что-то написал и был зареган не позже недели назад |
| SELECT MAX(creationTime) FROM User  | Вернуть время последней регистрации пользователя   |
| SELECT * FROM User WHERE openId IS NOT NULL ORDER BY creationTime DESC LIMIT 10                     | Вернуть последних 10 зарег пользователей среди тех, кто указал openId  |
| UPDATE Article SET userId=10 WHERE id=7   | Сменить автора у статьи  |
| DELETE FROM Article WHERE userId!=1 AND creationTime>NOW() - INTERVAL 7 DAY                         | Удалить все статьи не первого пользователя, которые одновременно за последнюю неделю                               |
| INSERT INTO `Article` (`userId`, `text`, `creationTime`) VALUES (4, 'VK Cup 2019', NOW());          | Вставить в таблицу Article   |

## Задания

### Задание 0 (это задание показывать преподавателю не надо).

Скачайте проект с <http://assets.codeforces.com/files/695354fbbfd96ca7/fwdb2019.7z>. Перейдите по <http://wp.codeforces.com/phpMyAdmin/> в свою базу данных (ваш логин начинается с латинской u и имеет вид u??, пароль - цифровой) и накликайте там таблицу User с полями:

- \* id (BIGINT до 18 знаков, autoincrement, primary key, not null)
- \* login (VARCHAR до 255 знаков, добавьте ключ уникальности unique\_User\_login, not null)
- \* passwordSha (VARCHAR до 255 знаков, not null)
- \* creationTime (DATETIME, индекс index\_User\_creationTime, not null)

Запустите проект (поправьте profile.properties, пересоберите с помощью mvn package), убедитесь, что всё работает - регистрация+вход+выход.

### Задание 1

На форму регистрации добавьте еще одно поле passwordConfirmation и дополнительно валидируйте, что они совпали. То есть после выполнения этого задания на форме регистрации будет три поля: login, password, passwordConfirmation. И появится доп. сообщение об ошибке.

## Задание 2

Добавьте в сущность User новое уникальное поле email. Это потребует изменение таблицы в базе данных, исходного кода репозитория и т.п. После этого добавьте поддержку поля email при регистрации. Следует проверять, что оно уникально, раньше не встречалось и выглядит как email (достаточно проверить, что переданная строка содержит ровно один символ @).

## Задание 3

Сделать, что входить в систему можно по логину или email (а не только по логину, что хочешь, то и вводишь). Не забудьте всюду переименовать поля/параметры из login в loginOrEmail.

## Задание 4

Обратите внимание, что в настоящий момент для всех action-методов в страницах требуются два обязательных параметра HttpServletRequest request, Map<String, Object> view. В ряде случаев (часто) достаточно одного из них или они вообще не нужны.

Измените исходный код FrontServlet так, чтобы в качестве action-метода подыскивался метод просто по названию (нет требования к точной сигнатуре как сейчас). При запуске надо смотреть на типы параметров метода (method.getParameterTypes()) и если параметр имеет тип Map.class, то ожидать что это view (передать), тоже самое сделать для HttpServletRequest. Таким образом, в action-методе может быть любой набор параметров типов HttpServletRequest и Map, они могут идти в любом порядке.

После этого следует удалить из всех страниц из всех action-методов лишние параметры и убедиться в работоспособности.

## Задание 5

Добавьте в футер информацию об общем кол-ве зарегистрированных пользователей. Для этого сделайте методы findCount в UserRepository/UserService. Добавьте в таско для страницы (commons.ftl) использование переменной шаблона \${userCount}.

После этого вам надо как-то во view во всех страницах всегда класть по ключу "userCount" значение userService.findCount().

Для этого сделайте общий базовый класс для всех страниц Page (то есть все страницы будут унаследованы от Page) и пару методов у него:

- void before(HttpServletRequest request, Map<String, Object> view)
- void after(HttpServletRequest request, Map<String, Object> view)

В потомках (всех страницах) можно будет переопределять эти методы (не забывайте вызывать super.before()/super.after()).

Добавьте в код FrontServlet поиск и запуск before/after-методов до и после запуска action-метода соответственно.

После этого у базового класса Page в before (или after) следует добавить помещение во view нужного значения userCount и теперь всюду в футере будет нужная информация.

Перенесите в Page#before из FrontServlet установку аутентифицированного пользователя user. Еще перенесите в Page#before установку message из IndexPage - после этого message можно устанавливать в

любую страницу: достаточно положить его в сессию и сделать редирект. Сделайте в page метод `setMessage`, который будет добавлять в сессию сообщение (не надо будет каждый раз руками писать `request.getSession()...`). Аналогично, сделайте в Page пару методов `setUser/getUser`, которые устанавливают в сессию аутентифицированного пользователя и возвращают его из сессии. Обратите внимание, что вам понадобится для этого `request`, но его вы можете сохранить в поле в Page во время `Page#before`.

Можно в Page добавить пустой метод с названием `action`, чтобы не писать в каждой странице пустой `action` (если нужен именно такой).

## Задание 6

Поддержать новую сущность Event - события от пользователя с полями `id`, `userId`, `type`, `creationTime`. Поле `userId` надо сделать внешним ключом на User: `ALTER TABLE `Event` ADD CONSTRAINT `fk_Event_userId` FOREIGN KEY (`userId`) REFERENCES `User` (`id`)`. Поле `type` должно быть enum с пока двумя значениями `ENTER`, `LOGOUT`. Вставлять записи в таблицу Event на каждый удачный вход/выход. Для этой сущности нужен и класс Event в пакете `domain` и свой репозиторий `EventRepository` (там будет лишь один метод `save`).

## Задание 7

Сделать сущность Talk (`id`, `sourceUserId`, `targetUserId`, `text`, `creationTime`) - сообщение от одного пользователя другому. Сделайте страницу `/talks` (только для аутентифицированных, если пользователь не аутентифицирован - отсылайте на `IndexPage` с соответствующим сообщением). На `TalksPage` должна быть простая форма с 2 полями "Send Message" и список всех сообщений, где заданный пользователь автор или адресат в порядке от более поздних к более новым. Таким образом, на сайте появится система внутренней переписки.

Переписку можно уложить в типичную datatable (колонки: `id`, от кого, кому, текст, когда), а вот форма должна быть с полем типа `select` (выпадающий список всех зарегистрированных пользователей) и большим полем типа `textarea` для сообщения.

Форма должна быть сверху, под ней - сообщения в табличке.

Не переживайте, если для отображения списка сообщений на каждое сообщение вам понадобится делать SQL-запрос для поиска отправителя/получателя (просто сделайте `UserRepository#find(id)` и норм). Это учебное задание, в реальной системе за счёт кэширования или более подробных абстракций такой бы проблемы не было.

## Задание 8

Обратите внимание, что сейчас в классах `XxxRepositoryImpl` очень много похожего кода. Проведите рефакторинг (сами придумайте какой), чтобы уменьшить размер кода, переиспользовав его. Возможно, вам понадобится создавать дополнительные удобные методы в `DatabaseUtils` или сделайте базовый класс для всех `XxxRepositoryImpl` назвав его `BasicRepositoryImpl` и вынесите туда общий код.