Министерство образования и науки РФ
Пермский национальный исследовательский политехнический университет
Электротехнический факультет
Кафедра Информационные технологии и автоматизированные системы

Базы данных

Отчет по курсовому проекту

Тема: «Эстафеты»

Выполнил: студент группы
Миннахметов Э.Ю.
Проверил: доцент кафедры ИТАС
Петренко А.А

г. Пермь – 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....3
1 АНАЛИЗ ПОСТАВЛЕННОЙ ЗАДАЧИ.....4
1.1 Текст варианта и инструменты моделирования.....4
1.2 Выбор СУБД.....4
1.3 Выбор языка программирования.....4
1.4 Выбор среды разработки.....4
1.5 Вспомогательные инструменты.....4
2 ТЕХНОЛОГИЯ РЕАЛИЗАЦИИ.....5
2.1 Концетуальная модель.....5
2.2 Логическая модель.....7
2.3 Физическая модель.....9
2.4 Создание базы данных.....5
2.5 Написание приложения.....7
2.6 Оформление графического пользовательского интерфейса.....9
2.7 Решение задачи варианта.....9
ЗАКЛЮЧЕНИЕ.....11
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....12
ПРИЛОЖЕНИЕ А. SQL-запросы.....4
ПРИЛОЖЕНИЕ В. Исходный код программы.....4
ПРИЛОЖЕНИЕ С. Код гипертекстовой разметки.....4

ВВЕДЕНИЕ

Цель: спроектировать схемы базы данных по предметной области, назначенной преподавателем.

Задачи:

1 проанализировать вариант задания;

2 построить схему базы данных.

1 АНАЛИЗ ПОСТАВЛЕННОЙ ЗАДАЧИ

Базы данных можно использовать в приложениях, разворачиваемых хоть где — это могут быть веб-сайты, десктопные, мобильные приложения и т. д. Далее будут выполяться сравнения и выбор системы управления базами данных (СУБД), языка программирования, среды разработки и вспомогательных инструментов таких, как библиотеки, фреймворки и прочее.

Следует начать с того, с чего начинается создание приложения. После моделирования предметной области, это выбор типа приложения

1.1 Текст варианта и инструменты моделирования

Текст варианта 54:

«Разработайте реляционную базу данных Эстафета. В базе данных должны храниться сведения о спортсменах, показанных ими результатах, участие и результаты предыдущих эстафет, результаты соперников. Необходимо решать задачи формирования эстафетной команды, способной победить в новых эстафетах.»

На начальном этапе проектирования уже очевидно, что схема будет содержать следующие сущности:

- спортсмены;
- результаты спортсменов на тренировках;
- участие и результаты спортсменов в эстафетах;
- команды спортсменов.

Выделения основных сущностей на данном этапе достаточно.

Рисование схемы базы данных по нотации Питера Чена будет выполнено в бесплатном сервисе draw.io — его функционал и элементная база достаточно богаты, чтобы обходить такой инструмент стороной, а отсутствие достойных конкурентов на рынке бесплатных «рисовалок» и вовсе присуждают данному инструменту неоспоримое преимущество.

Вывод, в данном разделе была описана постановка задачи назначенного варианта и описаны необходимые сущности. Вспомогательные сущности и связи между ними будут построены в следующем разделе.

1.2 Выбор типа приложения

Такой выбор будет производиться среди десктопных, мобильный и веб-приложений.

Десктопные приложения обладают быстротой выполнения, могут иметь локальную базу данных или работать с удаленной, однако они сложнее в создании и требуют реализации многопоточности для комфортной работы без прерываний.

Мобильные приложения обладают теми же преимуществами и недостатками.

В отличии от десктопных и мобильных приложений, которые являются клиентами для работы с базой данных, веб-приложения в создании таковых не нуждаются, поскольку такими клиентами являются веб-браузеры, которые установленны на каждом современном компьютере. Эти клиенты обрающаются с запросом к серверу, а тот в свою очередь возвращает ответ в виде HTML-страницы, JavaScript-сценариев, CSS-таблиц и мультимедиа-контента. Сложность задачи сужается засчет отсутствия необходимости написания собстенного клиента. Плюс ко всему, веб-разработка является очень развитым направлением в программировании, а отсюда и множество фреймворков и богатство инструментария.

Вывод, абсолютным лидером в соотношении достоинств и недостатков является веб-приложение. Оно может работать с любыми СУБД и писаться на многих языках программирования, для которых всегда найдется множество библиотек и фреймворков, которые упростят разработку. Следующим на очереди идет выбор СУБД.

1.3 Выбор СУБД

Реляционные базы данных в своей реализации деляться на 2 основных типа:

- локальные, лидером среди которых является SQLite;
- удаленные в виде служб на локальном сервере либо веб-служб для работы с удаленным сервером — среди них особо выделяются MySQL, PostgreSQL и Oracle.

Локальные СУБД сразу отбрасываются, поскольку они не расчитаны на работу с большим объемом данных. Oracle является платной СУБД, поэтому она также не будет участвовать в сравнении.

MySQL и PostgreSQL являются бесплатными, обе обладают богатым функционалом, однако PostgreSQL богаче, но тем не менее это в текущем сравнении не является достоинством, поскольку эти возможности не будут задействованы в текущем проекте. Остается сравнивать десктопные клиенты PgAdmin и MySQL Workbench. По чисто субъективному мнению автора, MySQL Workbench визуально приятнее и удобнее в использовании.

Таким образом, выбор СУБД был сделан в пользу MySQL за удобство пользования его официальным клиентом. На очереди выбор языка программирования, как средства работы с СУБД.

1.4 Выбор языка программирования

Самые популярные языки для написания веб-приложений - это PHP, Python, C#, Java, JavaScript.

PHP — удобен для написания простых приложений, поэтому с поставленной задачей он не справится.

Python — мощный, однако очень ресурсозатратный язык, поэтому он также отбрасывается.

JavaScript вообще создан для написания скриптов для клиентской части и, не смотря на фреймворки NodeJS, React или Angular, которые позволяют

писать бэкэнд-приложения, он также подходит только для простых приложений.

C# и Java очень достойные языки и они отлично справляются с разработкой веб-приложений. Есть лишь одна разница — C# привязан к операционной системе Windows, которая в свою очередь очерь требовательна к ресурсам компьютера по сравнению с Linux, на котором отлично работают, как и на Windows, веб-приложения на Java.

Вывод, вышеописанная разница играет не на пользу C#, поэтому для разработки веб-приложения будет использоваться язык Java. Теперь же необходимо выбрать среду разработки для написания приложения.

1.5 Выбор среды разработки

Существует большое множество сред разработки на Java. Среди них:

- Visual Studio Code — самая легковесная среда, которая требует установку и настройку всех необходимых модулей для парсинга, подсветки кода и подсказок;
- Eclipse — чуть потребовательнее, подсветка и подсказки встроены, однако работают не на отлично и требует отдельного подключения компилятора;
- Intellij IDEA — самая требовательная среда из трех, подсветка кода и подсказки работают отлично, среда способна сама устанавливать необходимые компоненты и компилятор.

Вывод, абсолютный фаворит среди сред разработки — Intellij IDEA от JetBrains, требовательность к ресурсам покрывается аппаратным обеспечением, именно она и будет использоваться для разработки. Java — очень богатый язык, с богатой историей в веб-разработке в том числе. Де факто, он является №1 языком для веб-разработки в мире. Для него написано множество библиотек и фреймворков, которые упрощают веб-разработку, такие средства будут рассмотрены далее.

1.6 Вспомогательные инструменты

Java EE (Java Enterprise Edition — расширение для Java, позволяющее писать веб-приложения) дает возможность писать сервлет-контейнеры, которые обрабатывают запросы от пользователей и возвращают результат. Java EE требует подключения к веб-серверу на локальном сервере такому, как Apache Tomcat, Jboss, GlassFish и т.д.

Возвращаемый результат можно оформлять в виде HTML-документов, этим целям служит JSP (Java Server Pages — HTML-страницы со вставками Java-кода, генерирующего наполнение для страницы).

Однако существует более удобные и функциональные средства для выполнения всего того же. Например:

- Spring Boot — фреймворк, упрощающий веб-разработку. На нем пишут контроллеры, обрабатывающие запросы пользователей. В него встроен сервер Apache Tomcat, который позволяет запускать веб-приложение без самостоятельной настройки веб-сервера на локальном сервере. Полностью покрывает потребности веб-разработчика;
- Thymeleaf — шаблонизатор, альтернатива JSP, обладающая более богатыми возможностями и приятным синтаксисом. Прекрасно состыкуется с Spring Boot.

Вывод, необходимые средства разработки выбраны, среди них: СУБД MySQL, клиент СУБД MySQL WorkBench, язык программирования Java, фреймворк Spring Boot, шаблонизатор Thymeleaf. Теперь необходимо перейти непосредственно к разработке веб-приложения.

2 ТЕХНОЛОГИЯ РЕАЛИЗАЦИИ

Разбработка приложения будет включать в себя три фазы:

- создание базы данных с необходимыми таблицами, представлениями и хранимыми процедурами;
- написание исходного кода на языке Java, основная цель которого — обрабатывать запросы и формировать ответы на уровне сущностей и их массивов;
- оформление результата в виде HTML-страницы, т.е. создание графического пользовательского интерфейса.

Очевидно, что начать необходимо с создания базы данных.

2.1 Концептуальная модель

Пришло время перейти от описания необходимых сущностей к описанию вспомогательных и связей между ними вместе с основными:

1) Здесь и далее будет считаться, что один спортсмен не может самостоятельно тренироваться и вести учет, а также участвовать в эстафетах. Все это он делает, находясь в команде. Однако, за спортсменом всегда привязаны его результаты тренировок и эстафет, поэтому сущность «Команды» имеет отношение к спортсменам «один ко многим», а «Спортсмены» - к «Результатам» «один ко многим».

2) За сущностью «Результаты» остается одна неясность — результаты чего? Это могут быть 50 метров, 100 метров, 400 метров или же вообще полоса препятствий с отжиманиями, турниками, канатами и прочим. Сконкретизировать результаты поможет введение еще одной сущности - «Дисциплины» - помимо наименования, она будет содержать единицу измерения и коэффициент уравнивания с другими дисциплинами, высчитанный на основе нормативов ГТО (система спортивных нормативов «Готов к труду и обороне»).

3) Сущность «Дисциплины» будет ассоциироваться с результатами и иметь связь «один ко многим» с «Результатами» - например, Вася пробежал 100 метров (100 м. - дисциплина) за 11,5 секунд (результат). «Многие ко многим» с «Командами», поскольку команды могут относиться к разным спортивным клубам или даже видам спорта, а это определяет то, к каким дисциплинам тренируются спортсмены. «Многие ко многим» с «Эстафетами», поскольку каждое соревнование может определять полосу препятствий.

4) «Команды» не могут иметь прямой связи с «Результатами», поскольку те связаны со «Спортсменами» и семантически такая связь не имела бы смысл, поскольку результаты тренировок и эстафет должны оформляться иначе. На основании этого, необходимо ввести еще одну сущность - «Итоги» - она будет иметь одношение «один ко многим» с «Результатами».

5) «Команды» участвуют в «Эстафетах» - связь «многие ко многим».

6) «Эстафеты» формируют «Итоги» - связь «один к одному».

7) «Команды» тренируются и подводят «Итоги» - связь «один ко многим».

8) участвуя в «Эстафетах», формируются «Командные итоги» - связь «один ко многим».

9) «Командные итоги» ссылаются на, определяющие их, «Итоги» - связь «один к одному».

8) «Итоги» включают «Результаты» конкретных спортсменов — связь «один ко многим».

На основании вышеизложенных умозаключений была построена схема базы данных:

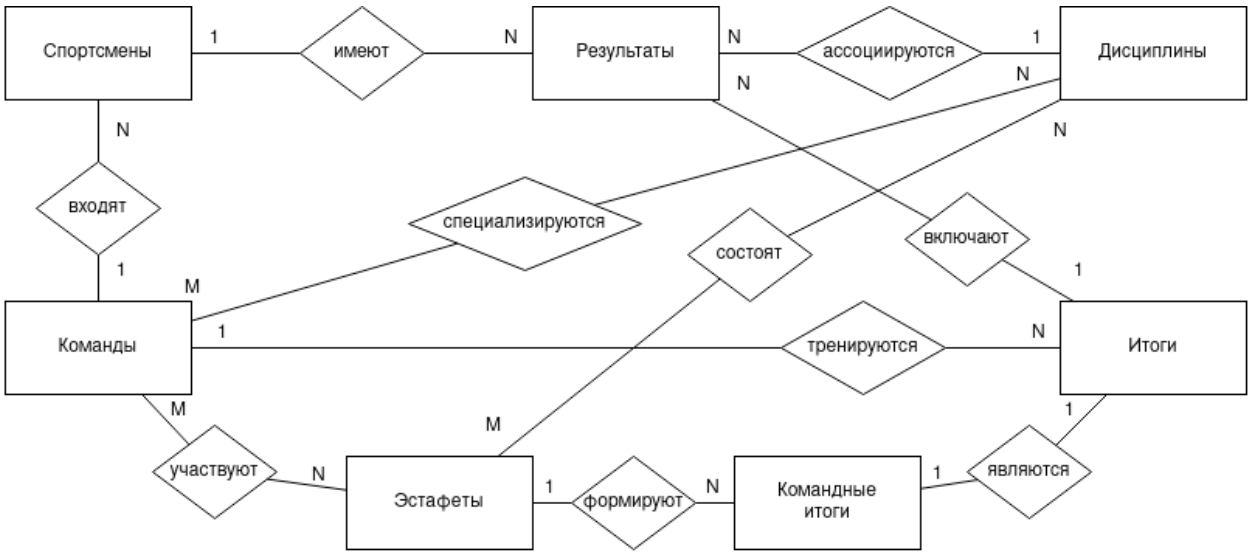


Рисунок 1 — Схема базы данных

Вывод, построение концептуальной модели завершено. Далее, на основе полученного результата, будет строиться логическая модель базы данных.

2.2 Логическая модель

Для построения логической модели, необходимо придать сущностям имена, состоящие из символов ASCII:

- Спортсмены — players;
- Результаты — results;
- Дисциплины — subjects;
- Команды — teams;
- Эстафеты — relay_races;
- Командные итоги — team_results;
- Итоги — result_lists.

Также необходимо ввести вспомогательные сущности для осуществления связи «многие ко многим»:

- между Командами и Дисциплинами — team_subjects;
- между Командами и Эстафетами — team_participation;
- между Эстафетами и Дисциплинами — relay_subjects.

Далее необходимо дополнить сущности атрибутами:

- у Спортсменов — идентификатор (player_id), команду (team_id), ФИО (player_name), «входит в сборную?» (is_chosen);

- у Дисциплин — идентификатор (subject_id), наименование (subject_name), множитель (коэффициент приведения, обсуждаемый ранее) (subject_multiplier), единицы измерения (subject_unit);

- у Команд — идентификатор (team_id), название (team_name), тренерский состав (trainers);

- у Эстафет — идентификатор Итога (result_list_id), наименование (relay_name);

- у Командных итогов — идентификатор Итога (result_list_id), эстафету (relay_id), команду (team_id), общий счёт команды (team_points);

- у Итогов — идентификатор (result_list_id), команду (team_id), общий счёт команды (team_points), дату подведения (result_date);

- у Результатов — идентификатор Итога (result_list_id), спортсмена (player_id), дисциплину (subject_id), значение результата (result_value).

- у вспомогательных сущностей, осуществляющих связь «многие ко многим» будут 2 атрибута — идентификаторы связуемых сущностей.

Теперь необходимо выполнить построение модели, основываясь на ранее сделанных умозаключениях:

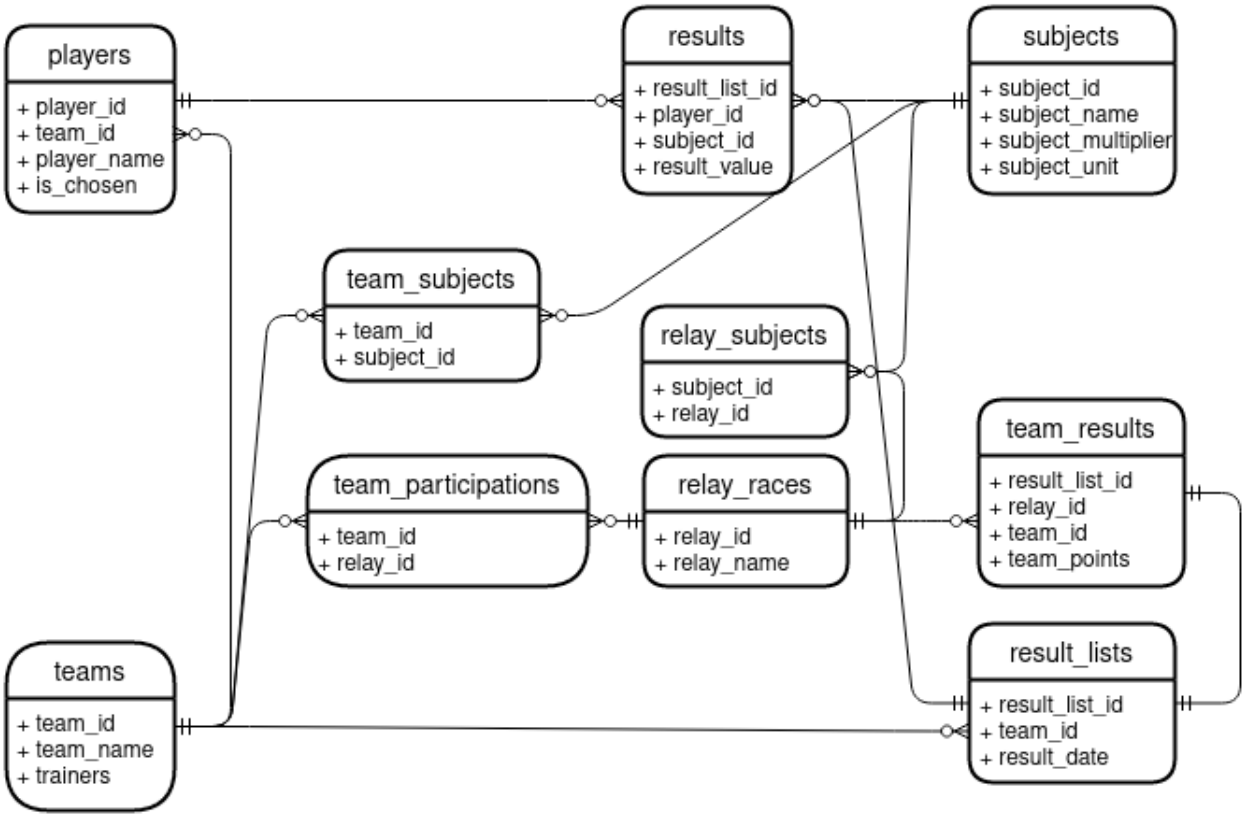


Рисунок 2 — Логическая модель

Вывод, построение логической модели завершено. Далее, на основе полученного результата, будет строиться физическая модель базы данных, которую было решено реализовать в СУБД MySQL.

2.3 Физическая модель

Физическая модель предполает использование определенной СУБД — в данном проекте это MySQL.

Теперь следует определить типы данных для всех атрибутов модели:

- все идентификаторы должны иметь тип INTEGER;

- имена спортсменов, наименования команд, тренерские составы, названия эстафет и дисциплин — будут символьными строками VARCHAR;

- предикат «входит ли спортсмен в сборную» - BOOLEAN;

- значения Результатов, множители Дисциплин и общий счёт команд в Итогах — DOUBLE;

- дата Итога — DATE.

Далее требуется дополнить логическую модель типами данных и получить, тем самым, физическую модель базы данных:

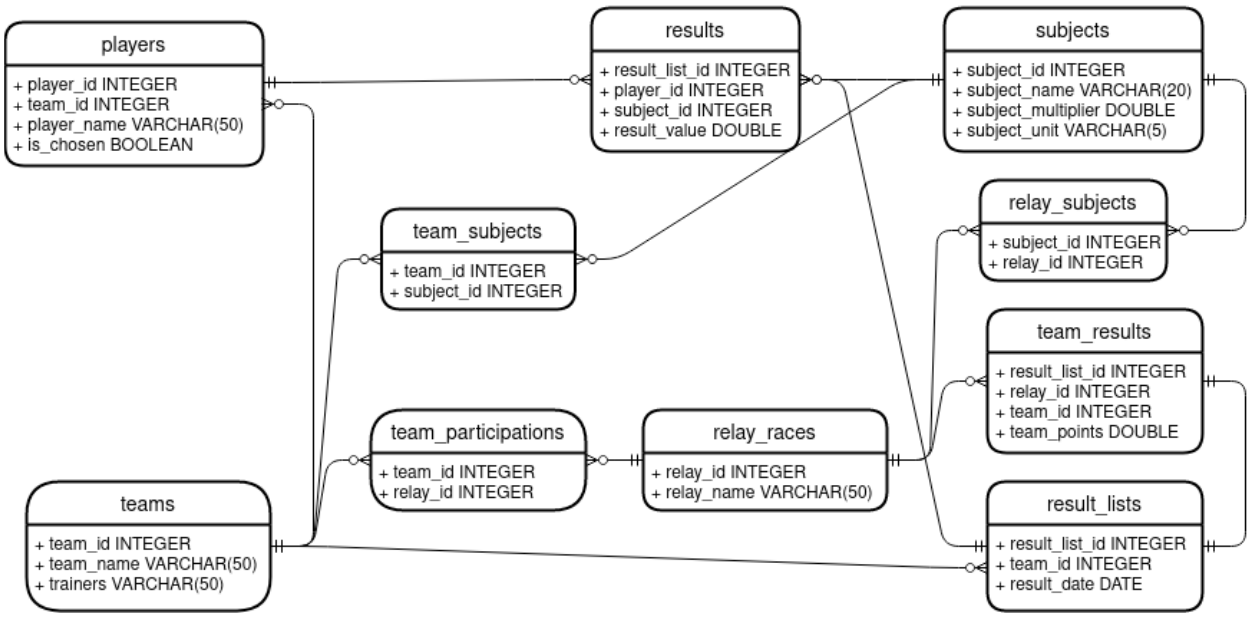


Рисунок 3 — Физическая модель

Вывод, физическая модель базы данных построена.

Итог раздела, проектирование базы данных в соответствие с назначенным вариантом задания завершено.

2.4 Создание базы данных

Как говорилось ранее, базы данных будет содержать таблицы, представления и хранимые процедуры.

Запросы необходимые для создания необходимых таблиц, представлений и хранимых процедур находятся в Приложении А.

Теперь необходимо перейти к написанию приложения на Java.

2.5 Написание приложения

Прежде всякого написания кода, необходимо выделить модели — классы предметной области:

- Спортсмен (Player) будет хранить свой идентификатор, имя, идентификатор команды и название команды;
- Команда (Team): идентификатор, название, тренерский состав;
- Команда, участвующая в эстафете (RelayTeam) — расширение Команды полями идентификатор списка результатов и общий счёт;
- Эстафета (RelayRace): идентификатор, наименование, число команд, число спортсменов в каждой, открытость/закрытость эстафеты;
- Дисциплина (Subject): идентификатор, название, единица измерения, множитель;
- Список резальтатов (ResultList): идентификатор, название, идентификатор команды, открытость/закрытость списка, дата списка;
- Результат (Result): идентификатор списка результатов, название списка, идентификатор спортсмена, имя спортсмена, идентификатор дисциплины, название дисциплины, значение результата, единица измерения дисциплины, множитель дисциплины, дата результата;
- Счет спортсмена (PlayerScores): идентификатор списка результатов, идентификатор дисциплины, идентификатор спортсмена, название дисциплины, имя спортсмена, счёт спортсмена.

Данные классы необходимо расположить в одном пакете (рис. 2.2):

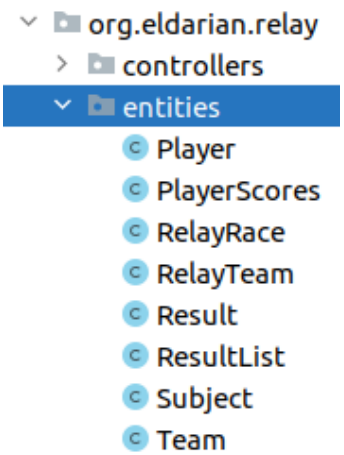


Рисунок 2.2. Пакет org.eldarian.relay.entities
со всеми классами-сущностями

Исходный код классов-сущностей приведен в Приложении В1.

Теперь же необходимо написать класс, в котором будет выполняться соединение с базой данных и выполнение запроса. Запрос будет обладать уникальными характеристиками такими, как принимаемые аргументы и возвращаемое значение. Следовательно класс должен быть обобщенным (Generic), в его конструктор будет передаваться объект с обощенным интерфейсом запроса — параметры типов этого объекта будут совпадать с параметрами типов каждного нового объекта этого класса.

Обобщенный класс, реализующий подключение к базе данных и выполняющий запрос по переданному объекту обобщенного интерфейса, будет называться DataContext.

Обобщенный интерфейс, задающий функционал запросов, будет называться ISqlQueryable.

Исходный код класса DataContext и интерфейса ISqlQueryable приведены в Приложении В2 и Приложении В3 соответственно.

Для самых запросов нужны классы-реализации интерфейса ISqlQueryable такие классы представленны в Приложении В4.

Результатом запросов могут быть объекты классов-сущностей, а также, в некоторых случаях, числовые (Integer) и булевые (Boolean) значения. Процесс считывания данных с результата запросов уровня SQL-провайдера и создания объекта класса-сущности уровня приложения будет единообразен для всех запросов, поэтому было решено вывести данное действие в класс-фабрику EntityFactory. Исходный код данного класса представлен в приложении В5.

Для запуска приложения, а также его конфигурирования, необходимо создать классы RelayApplication и WebConfig, которые будут потомками абстрактных классов запуска приложения и его конфигурирования соответственно в фреймворке Spring Boot. Исходные коды данных классов приведены в Приложении В6 и Приложении В7 соответственно.

Как говорилось ранее, приложение будет писаться с использованием фреймворка Spring Boot. Суть Spring Boot сводится к созданию контроллеров, которые обрабатывают запросы клиента, передают некоторые данные представлению, которое, в свою очередь, генерируется шаблонизатором Thymeleaf и отправляется клиенту в виде HTML-страницы.

Для обработки всех запросов потребуются контроллеры:

- главный (MainController) — обрабатывает информационные страницы со списками спортсменов, команд, дисциплин, результатов, а также конкретных их элементов, например, страница информации о конкретном спортсмене Василии Фёдорове и т.д.;
- добавления (AddController) — формы для добавления новых спортсменов, команд, дисциплин и т.д.
- изменения (EditController) — формы для редактирования конкретных спортсменов, команд, дисциплин и т.д.;
- вставки (InsertController) — формы с AddController будут направлять свои данные в этот контроллер, в нем будет выполняться вставка данных в какую-либо таблицу и перенаправлять на MainController с ранее созданным конкретным элементов. Например, в AddController регистрируется новый спортсмен, его данные направляются в InsertController, а там выполняется добавление новой записи в таблицу «Спортсмены», затем выполняется перенаправление на информационную страницу в MainController;

– обновления (UpdateController) — аналогично InsertController, но передача данных осуществляется с EditController и, вместо создания новой записи для таблицы, выполняется обновление ранее созданной, а затем выполняется перенаправление на конкретную информационную страницу в MainController обновленной записи;
– удаления (DeleteController) — удаляет запись из какой-либо таблицы, а затем выполняет перенаправление на список элементов в MainController той таблицы, с которой выполнялась работа;
– обработки ошибок (ErrorController) — обрабатывает ошибку 404 при запросе несуществующей страницы;
– абстрактный класс контроллеров AController — выполняет обработку исключений, вызванных при обработке операций — от него наследуются все классы-контроллеры во избежание дублирования кода.

Данные классы необходимо расположить в одном пакете, как это показано на рисунке 2.2:

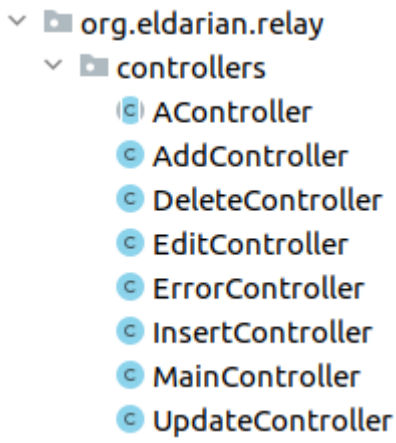


Рисунок 2.2 Пакет org.eldarian.relay.controllers
со всеми классами-контроллерами

Исходный код данных классов приведен в Приложении В8.

Вывод, написание приложения завершено. Далее необходимо оформить данными, предоставляемые в ходе обработки запросов, т. е. Создать слой представления или графический пользовательский интерфейс.

2.6 Оформление графического пользовательского интерфейса

При построении пользовательского графического интерфейса будет использоваться шаблонизатор Thymeleaf. Данный шаблонизатор позволяет разбивать код на фрагменты. Эти фрагменты могут быть статичными, а также динамичными, то есть можно передавать переменные и выражения во фрагменты или даже другие фрагменты.

Веб-страница будет состоять из левого меню и тела страницы.

Левое меню будет включать ссылку на главную страницу, на страницу авторизации, а также на основные представления в базе данных. Кроме того, в левом меню будут ссылки на запросы, специфичные для каждой страницы.

Тело страницы будет специфично для каждой странице и будет позволять выводить информацию любого типа в виде таблицы или предоставлять формы для добавления/изменения записей.

За хранение основных фрагментов будет отвечать файл fragments.html, а за специфичные запросы для каждой страницы queries.html, искодные коды гипертекста расположены в Приложении С1 и С2, соответственно.

Прежде, чем перейти дальше, стоит определить стили, оформляющие дизайн выб-страниц. Данную задачю возьмет на себя style.css. Код стилизации приложен в Приложении С3.

Гипертекстовая разметка тела страниц будет расположена на большом множестве HTML-файлов и приведена в Приложениях С4-С20.

Вывод, написание графического пользовательского интерфейса приложения, а также приложения в целом завершено.

2.7 Решение задачи варианта

При построении пользовательского графического интерфейса будет использоваться шаблонизатор Thymeleaf. Данный шаблонизатор позволяет разбивать код на фрагменты. Эти фрагменты могут быть статичными, а также динамичными, то есть можно передавать переменные и выражения во фрагменты или даже другие фрагменты.

Вывод, написание графического пользовательского интерфейса приложения, а также приложения в целом завершено.

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы были:

1) проанализирована предметная область, подобраны инструменты работы;

2) построены умозаключения относительно предметной области и нартсованна схема базы данных по нотации Питера Чена.

Вывод, цель данной лабораторной работы, а именно - проектирование схемы базы данных по предметной области.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) К. Дж. Дейт. Введение в системы баз данных, 2005 г. — 1328 с.

2) Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных: Учебник для высших учебных заведений, 2009 г. – 736 с.

3) Чен П. Модель "сущность-связь" – шаг к единому представлению данных, 1995 г. - 36 с.

4) апыв

ПРИЛОЖЕНИЕ А. SQL-запросы

Листинг А1. Создание таблиц

```
CREATE TABLE teams (
    team_id INT PRIMARY KEY AUTO_INCREMENT,
    team_name VARCHAR(50) NOT NULL,
    trainers VARCHAR(50) NOT NULL
);

CREATE TABLE players (
    player_id INT PRIMARY KEY AUTO_INCREMENT,
    team_id INT NULL,
    player_name VARCHAR(50) NOT NULL,
    FOREIGN KEY (team_id) REFERENCES teams (team_id)
);

CREATE TABLE subjects (
    subject_id INT PRIMARY KEY AUTO_INCREMENT,
    subject_name VARCHAR(20) NOT NULL,
    subject_multiplier DOUBLE NOT NULL,
    subject_unit VARCHAR(5) NOT NULL
);

CREATE TABLE result_lists (
    result_list_id INT PRIMARY KEY AUTO_INCREMENT,
    team_id INT NOT NULL,
    is_open BOOLEAN NOT NULL DEFAULT(TRUE),
    result_list_date DATE NOT NULL,
    FOREIGN KEY (team_id) REFERENCES teams (team_id) ON DELETE CASCADE
);

CREATE TABLE relay_races (
    relay_id INT PRIMARY KEY AUTO_INCREMENT,
    relay_name VARCHAR(50) NOT NULL,
    team_number INT NOT NULL,
    player_number INT NOT NULL,
    is_open BOOLEAN NOT NULL DEFAULT(TRUE)
);

CREATE TABLE team_participations (
    team_id INT NOT NULL,
    relay_id INT NOT NULL,
    result_list_id INT NOT NULL,
    FOREIGN KEY (team_id) REFERENCES teams (team_id) ON DELETE CASCADE,
    FOREIGN KEY (relay_id) REFERENCES relay_races (relay_id) ON DELETE CASCADE,
    FOREIGN KEY (result_list_id) REFERENCES result_lists (result_list_id) ON DELETE CASCADE
);

CREATE TABLE team_subjects (
    team_id INT NOT NULL,
    subject_id INT NOT NULL,
    FOREIGN KEY (team_id) REFERENCES teams (team_id) ON DELETE CASCADE,
    FOREIGN KEY (subject_id) REFERENCES subjects (subject_id) ON DELETE CASCADE
);

CREATE TABLE relay_subjects (
    relay_id INT NOT NULL,
    subject_id INT NOT NULL,
```

```
    subject_position INT NOT NULL,
    FOREIGN KEY (relay_id) REFERENCES relay_races (relay_id) ON DELETE CASCADE,
    FOREIGN KEY (subject_id) REFERENCES subjects (subject_id) ON DELETE CASCADE
);

CREATE TABLE player_positions (
    relay_id INT NOT NULL,
    player_id INT NOT NULL,
    player_position INT NOT NULL,
    FOREIGN KEY (relay_id) REFERENCES relay_races (relay_id) ON DELETE CASCADE,
    FOREIGN KEY (player_id) REFERENCES players (player_id) ON DELETE CASCADE
);

CREATE TABLE results (
    result_list_id INT NOT NULL,
    player_id INT NOT NULL,
    subject_id INT NOT NULL,
    result_value DOUBLE NOT NULL,
    result_date DATE NOT NULL,
    FOREIGN KEY (result_list_id) REFERENCES result_lists (result_list_id) ON DELETE CASCADE,
    FOREIGN KEY (player_id) REFERENCES players (player_id) ON DELETE CASCADE,
    FOREIGN KEY (subject_id) REFERENCES subjects (subject_id) ON DELETE CASCADE
);
```

Листинг А2. Создание представлений

```
CREATE VIEW player_views
(player_id, player_name, team_id, team_name)
AS SELECT players.player_id, players.player_name, players.team_id, teams.team_name
FROM players JOIN teams ON players.team_id = teams.team_id
UNION SELECT players.player_id, players.player_name, players.team_id, NULL AS
team_name
FROM players WHERE players.team_id IS NULL;

CREATE VIEW result_views
(result_list_id, result_list_name, player_id, player_name, subject_id, result_value,
subject_name,
    subject_unit, subject_multiplier, result_date)
AS SELECT results.result_list_id,
    CONCAT("Тренировка ", result_lists.result_list_date) AS result_list_name,
    results.player_id, players.player_name, results.subject_id, results.result_value,
    subjects.subject_name, subjects.subject_unit,
    subjects.subject_multiplier, results.result_date FROM results
JOIN players ON players.player_id = results.player_id
JOIN subjects ON subjects.subject_id = results.subject_id
JOIN result_lists ON result_lists.result_list_id = results.result_list_id
WHERE results.result_list_id NOT IN
    (SELECT team_participations.result_list_id FROM team_participations)
UNION
SELECT results.result_list_id,
    CONCAT("Эстафета ", relay_races.relay_name) AS result_list_name,
    results.player_id, players.player_name, results.subject_id, results.result_value,
    subjects.subject_name, subjects.subject_unit,
    subjects.subject_multiplier, results.result_date FROM results
JOIN players ON players.player_id = results.player_id
JOIN subjects ON subjects.subject_id = results.subject_id
JOIN result_lists ON result_lists.result_list_id = results.result_list_id
JOIN team_participations ON team_participations.result_list_id = results.result_list_id
JOIN relay_races ON relay_races.relay_id = team_participations.relay_id
WHERE results.result_list_id IN
```

```
    (SELECT team_participations.result_list_id FROM team_participations);

CREATE VIEW result_list_views
(result_list_id, result_list_name, team_id, team_name, is_open, result_list_date)
AS SELECT result_lists.result_list_id,
    CONCAT("Тренировка ", result_lists.result_list_date) AS relay_name,
    result_lists.team_id, teams.team_name, result_lists.is_open,
    result_lists.result_list_date FROM result_lists
JOIN teams ON teams.team_id = result_lists.team_id
WHERE result_lists.result_list_id NOT IN
    (SELECT team_participations.result_list_id FROM team_participations)
UNION
SELECT team_participations.result_list_id,
    CONCAT("Эстафета ", relay_races.relay_name) AS relay_name,
    team_participations.team_id, teams.team_name,
    result_lists.is_open,
    result_lists.result_list_date
FROM team_participations
JOIN result_lists ON result_lists.result_list_id = team_participations.result_list_id
JOIN teams ON teams.team_id = result_lists.team_id
JOIN relay_races ON team_participations.relay_id = relay_races.relay_id;

CREATE VIEW team_views
(team_id, player_id, player_name)
AS SELECT teams.team_id, players.player_id, players.player_name FROM teams
JOIN players ON teams.team_id = players.team_id;

CREATE VIEW team_subject_views
(team_id,subject_id, subject_name, subject_unit, subject_multiplier)
AS SELECT team_subjects.team_id, team_subjects.subject_id, subjects.subject_name,
subjects.subject_unit,
    subjects.subject_multiplier FROM team_subjects
JOIN subjects ON team_subjects.subject_id = subjects.subject_id;

CREATE VIEW relay_subject_views
(relay_id, subject_id, subject_name, subject_unit, subject_multiplier, subject_position)
AS SELECT relay_subjects.relay_id, relay_subjects.subject_id,
    subjects.subject_name, subjects.subject_unit, subjects.subject_multiplier,
relay_subjects.subject_position
FROM relay_subjects
JOIN subjects ON relay_subjects.subject_id = subjects.subject_id
ORDER BY relay_subjects.subject_position;

CREATE VIEW relay_team_views
(relay_id, relay_name, team_id, team_name, trainers, result_list_id, result_list_score)
AS SELECT team_participations.relay_id, relay_races.relay_name,
    team_participations.team_id, teams.team_name, teams.trainers,
team_participations.result_list_id,
    SUM(results.result_value*subjects.subject_multiplier) as result_list_score
FROM team_participations
JOIN relay_races ON relay_races.relay_id = team_participations.relay_id
JOIN teams ON teams.team_id = team_participations.team_id
JOIN results ON results.result_list_id = team_participations.result_list_id
JOIN subjects ON subjects.subject_id = results.subject_id
GROUP BY team_participations.relay_id, relay_races.relay_name,
team_participations.team_id, teams.team_name,
    teams.trainers, team_participations.result_list_id
UNION
SELECT team_participations.relay_id, relay_races.relay_name,
    team_participations.team_id, teams.team_name, teams.trainers,
team_participations.result_list_id,
```

```
    0 as result_list_score
FROM team_participations
JOIN relay_races ON relay_races.relay_id = team_participations.relay_id
JOIN teams ON teams.team_id = team_participations.team_id
WHERE team_participations.result_list_id NOT IN (SELECT result_list_id FROM results);

CREATE VIEW result_list_player_views
(result_list_id, team_id, team_name, player_id, player_name)
AS SELECT result_lists.result_list_id, result_lists.team_id, teams.team_name,
players.player_id, players.player_name
FROM result_lists
JOIN teams ON teams.team_id = result_lists.team_id
JOIN players ON players.team_id = result_lists.team_id;
```

Листинг А3. Создание хранимых процедур для работы с командами

```
DELIMITER //
CREATE PROCEDURE get_team_list()
    BEGIN
        SELECT * FROM teams;
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE find_team(IN arg_team_id INT)
    BEGIN
        SELECT * FROM teams WHERE team_id = arg_team_id;
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE subject_is_team(IN arg_team_id INT, IN arg_subject_id INT)
    BEGIN
        SELECT (COUNT(*) = 1) AS is_true FROM team_subjects
        WHERE team_id = arg_team_id AND subject_id = arg_subject_id;
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE add_team(IN arg_name VARCHAR(50), IN arg_trainer VARCHAR(50))
    BEGIN
        INSERT INTO teams (team_name, trainers) VALUES (arg_name, arg_trainer);
        SELECT LAST_INSERT_ID() AS 'last_insert_id';
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE add_team_subject(IN arg_team_id INT, IN arg_subject_id INT)
    BEGIN
        INSERT INTO team_subjects (team_id, subject_id) VALUES (arg_team_id,
arg_subject_id);
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE remove_team_subject(IN arg_team_id INT, IN arg_subject_id INT)
    BEGIN
        DELETE FROM team_subjects WHERE team_id = arg_team_id AND subject_id =
arg_subject_id;
    END //
```

```
DELIMITER ;

DELIMITER //
CREATE PROCEDURE get_team_player_list(IN arg_team_id INT)
   BEGIN
      SELECT * FROM players WHERE team_id = arg_team_id;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE get_team_subject_list(IN arg_team_id INT)
   BEGIN
      SELECT * FROM team_subject_views WHERE team_id = arg_team_id;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE get_not_team_subject_list(IN arg_team_id INT)
   BEGIN
      SELECT * FROM subjects
      WHERE subject_id NOT IN (
         SELECT subject_id
        FROM team_subject_views
        WHERE team_id = arg_team_id);
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE update_team(IN arg_id INT, IN arg_name VARCHAR(50), IN arg_trainer
VARCHAR(50))
   BEGIN
      UPDATE teams
      SET team_name = arg_name, trainers = arg_trainer
      WHERE team_id = arg_id;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE remove_team(IN arg_id INT)
   BEGIN
      DELETE FROM teams WHERE team_id = arg_id;
   END //
DELIMITER ;
```

Листинг А4. Создание хранимых процедур для работы со спортсменами

```
DELIMITER //
CREATE PROCEDURE get_player_list()
   BEGIN
      SELECT * FROM player_views;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE find_player(IN arg_id INT)
   BEGIN
      SELECT * FROM player_views WHERE player_id = arg_id;
   END //
```

```
DELIMITER ;

DELIMITER //
CREATE PROCEDURE add_player(IN arg_name VARCHAR(50), IN arg_team_id INT)
   BEGIN
      INSERT INTO players (player_name, team_id) VALUES (arg_name, arg_team_id);
      SELECT LAST_INSERT_ID() AS 'last_insert_id';
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE rename_player(IN arg_id INT, IN arg_name VARCHAR(50))
   BEGIN
      UPDATE players SET player_name = arg_name WHERE player_id = arg_id;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE change_player_team(IN arg_player_id INT, IN arg_team_id INT)
   BEGIN
      UPDATE players SET team_id = arg_team_id WHERE player_id = arg_player_id;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE player_results(IN arg_id INT)
   BEGIN
      SELECT * FROM result_views WHERE player_id = arg_id;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE remove_player(IN arg_id INT)
   BEGIN
      DELETE FROM players WHERE player_id = arg_id;
   END //
DELIMITER ;
```

Листинг А5. Создание хранимых процедур для работы с дисциплинами

```
DELIMITER //
CREATE PROCEDURE get_subject_list()
   BEGIN
      SELECT * FROM subjects;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE find_subject(IN arg_subject_id INT)
   BEGIN
      SELECT * FROM subjects WHERE subject_id = arg_subject_id;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE add_subject(IN arg_name VARCHAR(20), IN arg_unit VARCHAR(5), IN
arg_multiplier DOUBLE)
   BEGIN
      INSERT INTO subjects (subject_name, subject_unit, subject_multiplier) VALUES
(arg_name, arg_unit, arg_multiplier);
```

```
        SELECT LAST_INSERT_ID() AS 'last_insert_id';
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE update_subject(IN arg_id INT,
    IN arg_name VARCHAR(20), IN arg_unit VARCHAR(5), IN arg_multiplier DOUBLE)
    BEGIN
        UPDATE subjects
        SET subject_name = arg_name, subject_unit = arg_unit, subject_multiplier =
arg_multiplier
        WHERE subject_id = arg_id;
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE remove_subject(IN arg_id INT)
    BEGIN
        DELETE FROM subjects WHERE subject_id = arg_id;
    END //
DELIMITER ;
```

Листинг А6. Создание хранимых процедур для работы с результатами

```
DELIMITER //
CREATE PROCEDURE add_result(IN arg_result_id INT, IN arg_subject_id INT,
    IN arg_player_id INT, IN arg_result_value DOUBLE)
    BEGIN
        INSERT INTO results (result_list_id, subject_id, player_id, result_value, result_date)
        VALUES (arg_result_id, arg_subject_id, arg_player_id, arg_result_value,
CURRENT_DATE);
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE find_result(IN arg_result_id INT, IN arg_player_id INT, IN arg_subject_id
INT)
    BEGIN
        SELECT * FROM result_views
        WHERE result_list_id = arg_result_id AND player_id = arg_player_id AND subject_id =
arg_subject_id;
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE results_count(IN arg_result_id INT)
    BEGIN
        SELECT COUNT(*) AS number FROM results
        WHERE result_list_id = arg_result_id;
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE find_results_of_list(IN arg_result_id INT)
    BEGIN
        SELECT * FROM result_views
        WHERE result_list_id = arg_result_id ;
    END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE update_result(IN arg_result_id INT, IN prev_player_id INT, IN
next_player_id INT,
   IN prev_subject_id INT, IN next_subject_id INT, IN arg_result_value DOUBLE)
   BEGIN
      UPDATE results SET player_id = next_player_id, subject_id = next_subject_id,
result_value = arg_result_value
      WHERE result_list_id = arg_result_id AND player_id = prev_player_id AND subject_id =
prev_subject_id;
   END //
DELIMITER ;
```

Листинг А7. Создание хранимых процедур для работы со списками результатов

```
DELIMITER //
CREATE PROCEDURE start_result_list(IN arg_team_id INT)
   BEGIN
      INSERT INTO result_lists (team_id, is_open, result_list_date) VALUES (arg_team_id,
TRUE, CURRENT_DATE);
      SELECT LAST_INSERT_ID() AS 'last_insert_id';
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE close_result_list(IN arg_result_list_id INT)
   BEGIN
      UPDATE result_lists SET is_open = FALSE WHERE result_list_id = arg_result_list_id AND
is_open = TRUE;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE get_result_list(IN arg_result_list_id INT)
   BEGIN
      SELECT * FROM result_list_views WHERE result_list_id = arg_result_list_id;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE get_event_results(IN arg_result_list_id INT)
   BEGIN
      SELECT * FROM result_views WHERE result_list_id = arg_result_list_id;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE get_team_result_lists(IN arg_team_id INT)
   BEGIN
      SELECT * FROM result_list_views WHERE team_id = arg_team_id;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE get_open_result_list(IN arg_team_id INT)
   BEGIN
      SELECT * FROM result_list_views WHERE team_id = arg_team_id AND is_open = TRUE;
   END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE team_is_busy(IN arg_team_id INT)
   BEGIN
      SELECT (COUNT(*) > 0) AS is_true FROM result_list_views WHERE team_id =
arg_team_id AND is_open = TRUE;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE get_possible_player_list(IN arg_result_list_id INT)
   BEGIN
      SELECT * FROM result_list_player_views WHERE result_list_id = arg_result_list_id;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE get_possible_subject_list(IN arg_result_list_id INT)
   BEGIN
      IF (SELECT COUNT(*) AS cnt FROM team_participations WHERE result_list_id =
arg_result_list_id) = 1
      THEN
         SELECT * FROM relay_subject_views
         JOIN team_participations ON team_participations.relay_id =
relay_subject_views.relay_id
         WHERE team_participations.result_list_id = arg_result_list_id;
      ELSE
         SELECT result_lists.result_list_id, result_lists.team_id, team_subjects.subject_id,
subjects.subject_name,
            subjects.subject_unit, subjects.subject_multiplier FROM result_lists
         JOIN team_subjects ON team_subjects.team_id = result_lists.team_id
         JOIN subjects ON subjects.subject_id = team_subjects.subject_id
         WHERE result_lists.result_list_id = arg_result_list_id;
      END if;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE remove_result_list(IN arg_result_list_id INT)
   BEGIN
      DELETE FROM result_lists WHERE result_list_id = arg_result_list_id;
   END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE find_team_participation(IN arg_team_id INT)
   BEGIN
      SELECT (COUNT(team_participations.team_id) = 1) AS is_true
      FROM team_participations
      JOIN result_lists ON result_lists.result_list_id = team_participations.result_list_id
      WHERE team_participations.team_id = 1 AND result_lists.is_open = TRUE;
   END //
DELIMITER ;
```

Листинг А8. Создание хранимых процедур для работы с эстафетами

```
DELIMITER //
CREATE PROCEDURE get_relay_race_list()
   BEGIN
      SELECT * FROM relay_races;
```

```
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE find_relay_race(IN arg_id INT)
    BEGIN
        SELECT * FROM relay_races WHERE relay_id = arg_id;
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE subject_is_relay(IN arg_relay_id INT, IN arg_subject_id INT)
    BEGIN
        SELECT (COUNT(*) = TRUE) AS is_true FROM relay_subjects
        WHERE relay_id = arg_relay_id AND subject_id = arg_subject_id;
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE relay_results_count(IN arg_result_list_id INT)
    BEGIN
        SELECT relay_races.player_number*COUNT(relay_subjects.subject_id) AS number FROM
team_participations
        JOIN relay_races ON relay_races.relay_id = team_participations.relay_id
        JOIN relay_subjects ON relay_subjects.relay_id = team_participations.relay_id
        WHERE team_participations.result_list_id = arg_result_list_id
        GROUP BY relay_races.player_number;
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE add_relay_race(IN arg_relay_name VARCHAR(50), IN arg_team_number
INT, IN arg_player_number INT)
    BEGIN
        INSERT INTO relay_races (relay_name, team_number, player_number)
        VALUES (arg_relay_name, arg_team_number, arg_player_number);
        SELECT LAST_INSERT_ID() AS 'last_insert_id';
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE add_relay_team(IN arg_relay_id INT, IN arg_team_id INT)
    BEGIN
        CALL start_result_list(arg_team_id);
        INSERT INTO team_participations (relay_id, team_id, result_list_id)
        VALUES (arg_relay_id, arg_team_id, LAST_INSERT_ID());
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE add_relay_subject(IN arg_relay_id INT, IN arg_subject_id INT)
    BEGIN
        INSERT INTO relay_subjects (relay_id, subject_id, subject_position)
        VALUES (arg_relay_id, arg_subject_id, 1);
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE relay_team_list(IN arg_relay_id INT)
    BEGIN
        SELECT * FROM relay_team_views WHERE relay_id = arg_relay_id;
```

```
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE relay_subject_list(IN arg_relay_id INT)
    BEGIN
        SELECT * FROM relay_subject_views WHERE relay_id = arg_relay_id;
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE not_relay_team_list(IN arg_relay_id INT)
    BEGIN
        SELECT * FROM teams WHERE team_id NOT IN
            (SELECT team_id FROM team_participations WHERE relay_id = arg_relay_id);
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE not_relay_subject_list(IN arg_relay_id INT)
    BEGIN
        SELECT * FROM subjects WHERE subject_id NOT IN
            (SELECT subject_id FROM relay_subjects WHERE relay_id = arg_relay_id);
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE relay_is_finished(IN arg_relay_id INT)
    BEGIN
        SELECT (COUNT(team_participations.team_id) = 0) AS is_true
        FROM team_participations
        JOIN result_lists ON team_participations.result_list_id = result_lists.result_list_id
        WHERE team_participations.relay_id = arg_relay_id AND result_lists.is_open = TRUE;
    END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE close_relay_race(IN arg_relay_id INT)
    BEGIN
        UPDATE relay_races SET is_open = FALSE WHERE relay_id = arg_relay_id;
    END //
DELIMITER ;
```

ПРИЛОЖЕНИЕ В. Исходный код программы

Листинг В1. Класс org.eldarian.relay.RelayApplication

```
@SpringBootApplication
public class RelayApplication {
	public static void main(String[] args) {
		SpringApplication.run(RelayApplication.class, args);
	}
}
```

Листинг В2. Класс org.eldarian.relay.WebConfig

```
@Configuration
public class WebConfig implements
WebServerFactoryCustomizer<ConfigurableServletWebServerFactory> {
   @Override
   public void customize(ConfigurableServletWebServerFactory factory) {
      factory.addErrorPages(new ErrorPage(HttpStatus.NOT_FOUND, "/404"));
   }
}
```

Листинг В3. Класс org.eldarian.relay.DataContext

```
public class DataContext<TResult, TArgument> {
   private ISqlQueryable<TResult, TArgument> _query;

   public DataContext(ISqlQueryable<TResult, TArgument> query) {
      _query = query;
   }

   public TResult provide(TArgument argument) {
      TResult result = null;
      try{
         String url =
"jdbc:mysql://localhost/testdb?serverTimezone=Europe/Moscow&allowPublicKeyRetrieval=tr
ue&useSSL=false";
         String username = "eldarian";
         String password = "19841986";
         Class.forName("com.mysql.cj.jdbc.Driver").getDeclaredConstructor().newInstance();
         try (Connection connection = DriverManager.getConnection(url, username,
         password)){
            result = _query.execute(connection.createStatement(), argument);
         }
      }
      catch(Exception ex){
         System.out.println(ex);
      }
      return result;
   }
}
```

Листинг В4. Интерфейс org.eldarian.relay.ISqlQueryable

```
public interface ISqlQueryable<TResult, TArgument> {
   TResult execute(Statement statement, TArgument arg) throws SQLException;
}
```

Листинг В5. Класс org.eldarian.relay.EntityFactory

```
public class EntityFactory {

   private ResultSet _set;

   public EntityFactory(ResultSet set) {
      _set = set;
   }

   public Player miniPlayer() throws SQLException {
      Player item = new Player();
      item.setPlayerId(_set.getInt("player_id"));
      item.setPlayerName(_set.getString("player_name"));
      return item;
   }

   public Player player() throws SQLException {
      Player item = new Player();
      item.setPlayerId(_set.getInt("player_id"));
      item.setTeamId(_set.getInt("team_id"));
      item.setPlayerName(_set.getString("player_name"));
      item.setTeamName(_set.getString("team_name"));
      return item;
   }

   public Subject subject() throws SQLException {
      Subject item = new Subject();
      item.setSubjectId(_set.getInt("subject_id"));
      item.setSubjectName(_set.getString("subject_name"));
      item.setSubjectUnit(_set.getString("subject_unit"));
      item.setSubjectMultiplier(_set.getDouble("subject_multiplier"));
      return item;
   }

   public Team team() throws SQLException {
      Team item = new Team();
      item.setTeamId(_set.getInt("team_id"));
      item.setTeamName(_set.getString("team_name"));
      item.setTrainers(_set.getString("trainers"));
      return item;
   }

   public RelayTeam relayTeam() throws SQLException {
      RelayTeam item = new RelayTeam(team());
      item.setResultListId(_set.getInt("result_list_id"));
      item.setResultListScore(_set.getDouble("result_list_score"));
      return item;
   }

   public Result result() throws SQLException {
      Result item = new Result();
      item.setResultListId(_set.getInt("result_list_id"));
      item.setResultListName(_set.getString("result_list_name"));
      item.setPlayerId(_set.getInt("player_id"));
      item.setPlayerName(_set.getString("player_name"));
```

```
        item.setSubjectId(_set.getInt("subject_id"));
        item.setSubjectName(_set.getString("subject_name"));
        item.setResultValue(_set.getDouble("result_value"));
        item.setSubjectUnit(_set.getString("subject_unit"));
        item.setSubjectMultiplier(_set.getDouble("subject_multiplier"));
        item.setResultDate(_set.getDate("result_date"));
        return item;
    }

    public ResultList resultList() throws SQLException {
        ResultList item = new ResultList();
        item.setTeamId(_set.getInt("team_id"));
        item.setResultListId(_set.getInt("result_list_id"));
        item.setResultListName(_set.getString("result_list_name"));
        item.setResultListDate(_set.getDate("result_list_date"));
        item.setOpen(_set.getBoolean("is_open"));
        return item;
    }

    public RelayRace relayRace() throws SQLException {
        RelayRace item = new RelayRace();
        item.setRelayId(_set.getInt("relay_id"));
        item.setRelayName(_set.getString("relay_name"));
        item.setTeamNumber(_set.getInt("team_number"));
        item.setPlayerNumber(_set.getInt("player_number"));
        item.setOpen(_set.getBoolean("is_open"));
        return item;
    }

    public Boolean bool() throws SQLException {
        return _set.getBoolean("is_true");
    }

    public Integer number() throws SQLException {
        return _set.getInt("number");
    }

}
```

Листинг В6. Абстрактный класс-контроллер

```
@Controller
public abstract class AController {
    @ExceptionHandler(Exception.class)
    public String handleException(Exception e, Model model) {
        model.addAttribute("exception", e.getMessage());
        return "general/exception";
    }
}
```

Листинг В7. Классы-контроллеры

```
@Controller
public class MainController extends AController {

    @GetMapping("/")
    public String home() {
        return "general/home";
    }
```

```
    @GetMapping("/teams")
    public String teams(Model model) {
        Collection<Team> teams = (Collection<Team>)(new DataContext(new
TeamListQuery()).provide(null));
        model.addAttribute("teams", teams);
        return "general/teams";
    }

    @GetMapping("/players")
    public String players(Model model) {
        Collection<Player> players = (Collection<Player>)(new DataContext(new
PlayerListQuery()).provide(null));
        model.addAttribute("players", players);
        return "general/players";
    }

    @GetMapping("/subjects")
    public String subjects(Model model) {
        Collection<Subject> subjects = (Collection<Subject>)(new DataContext(new
SubjectListQuery()).provide(null));
        model.addAttribute("subjects", subjects);
        return "general/subjects";
    }

    @GetMapping("/team")
    public String team(@RequestParam(name = "id") String id, Model model) {
        Team team = (Team)(new DataContext(new TeamQuery()).provide(id));
        ResultList resultList = (ResultList)(new DataContext(new
OpenedResultListQuery()).provide(id));
        Collection<Player> players = (Collection<Player>)(new DataContext(new
TeamPlayerListQuery()).provide(id));
        Collection<Subject> subjects = (Collection<Subject>)
                (new DataContext(new IncludedTeamSubjectQuery()).provide(id));
        Collection<ResultList> resultLists = (Collection<ResultList>)
                (new DataContext(new ResultListsQuery()).provide(id));
        model.addAttribute("team", team);
        model.addAttribute("players", players);
        model.addAttribute("subjects", subjects);
        model.addAttribute("resultLists", resultLists);
        model.addAttribute("resultListId", resultList != null ? resultList.getResultListId() : 0);
        return "general/team";
    }

    @GetMapping("/player")
    public String player(@RequestParam(name = "id") String id, Model model) {
        Player player = (Player)(new DataContext(new PlayerQuery()).provide(id));
        Collection<Result> results = (Collection<Result>)(new DataContext(new
PlayerResultListQuery()).provide(id));
        model.addAttribute("player", player);
        model.addAttribute("results", results);
        return "general/player";
    }

    @GetMapping("/subject")
    public String subject(@RequestParam(name = "id") String id, Model model) {
        Subject subject = (Subject)(new DataContext(new SubjectQuery()).provide(id));
        model.addAttribute("subject", subject);
        return "general/subject";
    }
```

```
    @GetMapping("/result_list")
    public String resultList(@RequestParam(name = "id") String id, Model model) {
        ResultList resultList = (ResultList)(new DataContext(new
ResultListQuery()).provide(id));
        Collection<Result> results = (Collection<Result>)(new DataContext(new
EventResultsQuery()).provide(id));
        model.addAttribute("resultList", resultList);
        model.addAttribute("results", results);
        return "general/result_list";
    }

    @GetMapping("/relay_race")
    public String relayRace(@RequestParam(name = "id") String id, Model model) {
        RelayRace relayRace = (RelayRace)(new DataContext(new
RelayRaceQuery()).provide(id));
        Collection<RelayTeam> teams = (Collection<RelayTeam>)(new DataContext(new
IncludedRelayTeamQuery())
                .provide(id));
        Collection<Subject> subjects = (Collection<Subject>)(new DataContext(new
IncludedRelaySubjectQuery())
                .provide(id));
        model.addAttribute("teams", teams);
        model.addAttribute("subjects", subjects);
        model.addAttribute("relayRace", relayRace);
        return "general/relay_race";
    }

    @GetMapping("/relay_races")
    public String relayRaces(Model model) {
        Collection<RelayRace> relayRaces = (Collection<RelayRace>)(new DataContext(new
RelayRaceListQuery())
                .provide(null));
        model.addAttribute("relayRaces", relayRaces);
        return "general/relay_races";
    }

    @GetMapping("/authorization")
    public String authorization() {
        return "general/authorization";
    }

}

@Controller
public class AddController extends AController {

    @GetMapping("/add_team")
    public String addTeam() {
        return "addition/add_team";
    }

    @GetMapping("/add_player")
    public String addPlayer(Model model) {
        Collection<Team> teams = (Collection<Team>)(new DataContext(new
TeamListQuery()).provide(null));
        model.addAttribute("teams", teams);
        return "addition/add_player";
    }

    @GetMapping("/add_subject")
    public String addSubject() {
```

```
        return "addition/add_subject";
    }

    @GetMapping("/add_team_subject")
    public String addTeamSubject(@RequestParam(name = "id") String id, Model model) {
        Collection<Subject> subjects = (Collection<Subject>)
                (new DataContext(new NotIncludedTeamSubjectQuery()).provide(id));
        model.addAttribute("teamId", id);
        model.addAttribute("subjects", subjects);
        return "addition/add_team_subject";
    }

    @GetMapping("/add_relay_race")
    public String addRelayRace() {
        return "addition/add_relay_race";
    }

    @GetMapping("/add_relay_team")
    public String addRelayTeam(@RequestParam(name = "id") String relayId, Model model) {
        Collection<Team> teams = (Collection<Team>)(new DataContext(new
NotIncludedRelayTeamQuery())
                .provide(relayId));
        model.addAttribute("teams", teams);
        model.addAttribute("relayId", relayId);
        return "addition/add_relay_team";
    }

    @GetMapping("/add_relay_subject")
    public String addRelaySubject(@RequestParam(name = "id") String id, Model model) {
        Collection<Subject> subjects = (Collection<Subject>)(new DataContext(new
NotIncludedRelaySubjectQuery())
                .provide(id));
        model.addAttribute("subjects", subjects);
        model.addAttribute("relayId", id);
        return "addition/add_relay_subject";
    }

    @GetMapping("/add_result")
    public String addResult(@RequestParam(name = "id") String resultListId, Model model) {
        Collection<Subject> subjects = (Collection<Subject>)(new DataContext(new
PossibleSubjectListQuery())
                .provide(resultListId));
        Collection<Player> players = (Collection<Player>)(new DataContext(new
PossiblePlayerListQuery())
                .provide(resultListId));
        model.addAttribute("subjects", subjects);
        model.addAttribute("players", players);
        model.addAttribute("resultListId", resultListId);
        return "addition/add_result";
    }

}

@Controller
public class InsertController extends AController {

    @PostMapping("/insert_team")
    public String insertTeam(@RequestParam(name = "team_name") String teamName,
                             @RequestParam(name = "trainers") String trainers) {
        int id = (Integer)new DataContext(new AddTeamQuery()).provide(new String[]
{teamName, trainers});
```

```
        return "redirect:/team?id=" + id;
    }

    @PostMapping("/insert_player")
    public String insertPlayer(@RequestParam(name = "player_name") String name,
                               @RequestParam(name = "team_id") String team) {
        int id = (Integer)new DataContext(new AddPlayerQuery()).provide(new String[]{name,
team});
        return "redirect:/player?id=" + id;
    }

    @PostMapping("/insert_subject")
    public String insertSubject(@RequestParam(name = "subject_name") String name,
                                @RequestParam(name = "subject_unit") String unit,
                                @RequestParam(name = "subject_multiplier") String multiplier) {
        int id = (Integer)new DataContext(new AddSubjectQuery()).provide(new String[]{name,
unit, multiplier});
        return "redirect:/subject?id=" + id;
    }

    @PostMapping("/insert_team_subject")
    public String insertTeamSubject(@RequestParam(name = "id") String teamId,
                                    @RequestParam(name = "subject_id") String subjectId) throws
Exception {
        boolean subjectIsTeam = (Boolean) new DataContext(new SubjectIsTeamQuery())
                .provide(new String[]{teamId, subjectId});
        if(subjectIsTeam)
            throw new Exception("Команда уже тренерует данную дисциплину");
        new DataContext(new AddTeamSubjectQuery()).provide(new String[]{teamId,
subjectId});
        return "redirect:/team?id=" + teamId;
    }

    @GetMapping("/start_workout")
    public String addWorkout(@RequestParam(name = "id") String teamId) throws Exception
{
        boolean teamIsBusy = (Boolean) new DataContext(new
TeamIsBusyQuery()).provide(teamId);
        if(teamIsBusy)
            throw new Exception("Команда уже занята");
        int id = (Integer)new DataContext(new AddResultListQuery()).provide(teamId);
        return "redirect:/result_list?id=" + id;
    }

    @PostMapping("/insert_relay_race")
    public String insertRelayRace(@RequestParam(name = "relay_name") String relayName,
                                  @RequestParam(name = "team_number") String teamNumber,
                                  @RequestParam(name = "player_number") String playerNumber) {
        int id = (Integer)new DataContext(new AddRelayRaceQuery())
                .provide(new String[]{relayName, teamNumber, playerNumber});
        return "redirect:/relay_race?id=" + id;
    }

    @PostMapping("/insert_relay_team")
    public String insertRelayTeam(@RequestParam(name = "id") String relayId,
                                  @RequestParam(name = "team_id") String teamId) throws Exception {
        boolean teamIsBusy = (Boolean) new DataContext(new
TeamIsBusyQuery()).provide(teamId);
        if(teamIsBusy)
            throw new Exception("Команда уже занята");
        new DataContext(new AddRelayTeamQuery()).provide(new String[]{relayId, teamId});
```

```
        return "redirect:/relay_race?id=" + relayId;
    }

    @PostMapping("/insert_relay_subject")
    public String insertRelaySubject(@RequestParam(name = "id") String relayId,
                                     @RequestParam(name = "subject_id") String subjectId) throws
Exception {
        boolean subjectIsRelay = (Boolean) new DataContext(new SubjectIsRelayQuery())
                .provide(new String[]{relayId, subjectId});
        if(subjectIsRelay)
            throw new Exception("Эстафета уже включает данную дисциплину");
        new DataContext(new AddRelaySubjectQuery()).provide(new String[]{relayId,
subjectId});
        return "redirect:/relay_race?id=" + relayId;
    }

    @PostMapping("/insert_result")
    public String insertResult(@RequestParam(name = "id") String resultListId,
                               @RequestParam(name = "subject_id") String subjectId,
                               @RequestParam(name = "player_id") String playerId,
                               @RequestParam(name = "result_value") String resultValue) throws
Exception {
        boolean resultIsExist = new DataContext(new ResultQuery())
                .provide(new String[]{resultListId, playerId, subjectId}) != null;
        if(resultIsExist)
            throw new Exception("Данный результат уже зафиксирован");
        new DataContext(new AddResultQuery())
                .provide(new String[]{resultListId, subjectId, playerId, resultValue});
        return "redirect:/result_list?id=" + resultListId;
    }

}

@Controller
public class EditController extends AController {

    @GetMapping("/edit_team")
    public String editTeam(@RequestParam(name = "id") String id, Model model) {
        Team team = (Team)(new DataContext(new TeamQuery()).provide(id));
        model.addAttribute("team", team);
        return "edition/edit_team";
    }

    @GetMapping("/rename_player")
    public String renamePlayer(@RequestParam(name = "id") String id, Model model) {
        Player player = (Player)(new DataContext(new PlayerQuery()).provide(id));
        model.addAttribute("player", player);
        return "edition/rename_player";
    }

    @GetMapping("/change_player_team")
    public String changePlayerTeam(@RequestParam(name = "id") String id, Model model) {
        Player player = (Player)(new DataContext(new PlayerQuery()).provide(id));
        Collection<Team> teams = (Collection<Team>)(new DataContext(new
TeamListQuery()).provide(null));
        model.addAttribute("player", player);
        model.addAttribute("teams", teams);
        return "edition/change_player_team";
    }

    @GetMapping("/edit_subject")
```

```
    public String editSubject(@RequestParam(name = "id") String id, Model model) {
        Subject subject = (Subject)(new DataContext(new SubjectQuery()).provide(id));
        model.addAttribute("subject", subject);
        return "edition/edit_subject";
    }

    @GetMapping("/edit_result")
    public String editResult(Model model,
                             @RequestParam(name = "result_list_id") String resultListId,
                             @RequestParam(name = "player_id") String playerId,
                             @RequestParam(name = "subject_id") String subjectId) {
        Result result = (Result)(new DataContext(new ResultQuery())
                .provide(new String[]{resultListId, playerId, subjectId}));
        Collection<Subject> subjects = (Collection<Subject>)(new DataContext(new
PossibleSubjectListQuery())
                .provide(resultListId));
        Collection<Player> players = (Collection<Player>)(new DataContext(new
PossiblePlayerListQuery())
                .provide(resultListId));
        model.addAttribute("result", result);
        model.addAttribute("subjects", subjects);
        model.addAttribute("players", players);
        return "edition/edit_result";
    }

    @GetMapping("/edit_relay_race")
    public String editRelayRace() {
        return "edition/add_relay_race";
    }

}

@Controller
public class UpdateController extends AController {

    @PostMapping("/update_team")
    public String updateTeam(@RequestParam(name = "id") String team,
                             @RequestParam(name = "team_name") String name,
                             @RequestParam(name = "trainers") String trainers) {
        new DataContext(new UpdateTeamQuery()).provide(new String[]{team, name,
trainers});
        return "redirect:/team?id=" + team;
    }

    @PostMapping("/update_player_name")
    public String updatePlayerName(@RequestParam(name = "id") String player,
                                   @RequestParam(name = "player_name") String name) {
        new DataContext(new RenamePlayerQuery()).provide(new String[]{player, name});
        return "redirect:/player?id=" + player;
    }

    @PostMapping("/update_player_team")
    public String updatePlayerTeam(@RequestParam(name = "id") String player,
                                   @RequestParam(name = "team_id") String team) {
        new DataContext(new ChangePlayerTeamQuery()).provide(new String[]{player,
team});
        return "redirect:/player?id=" + player;
    }

    @GetMapping("/leave_player_team")
    public String leavePlayerTeam(@RequestParam(name = "id") String player,
```

```
                        @RequestParam(name = "from", defaultValue = "") String from) {
        new DataContext(new ChangePlayerTeamQuery()).provide(new String[]{player,
"NULL"});
        return "redirect:/" + (from.equals("") ? "player?id=" + player : from);
    }

    @PostMapping("/update_subject")
    public String updateSubject(@RequestParam(name = "id") String subject,
                        @RequestParam(name = "subject_name") String name,
                        @RequestParam(name = "subject_unit") String unit,
                        @RequestParam(name = "subject_multiplier") String multiplier) {
        new DataContext(new UpdateSubjectQuery()).provide(new String[]{subject, name,
unit, multiplier});
        return "redirect:/subject?id=" + subject;
    }

    @PostMapping("/update_result")
    public String updateResult(@RequestParam(name = "result_list_id") String resultListId,
                        @RequestParam(name = "prev_player_id") String prevPlayerId,
                        @RequestParam(name = "next_player_id") String nextPlayerId,
                        @RequestParam(name = "prev_subject_id") String prevSubjectId,
                        @RequestParam(name = "next_subject_id") String nextSubjectId,
                        @RequestParam(name = "result_value") String resultValue) throws
Exception {
        boolean resultIsExist = new DataContext(new ResultQuery())
                .provide(new String[]{resultListId, nextPlayerId, nextSubjectId}) != null;
        if(resultIsExist && (!nextPlayerId.equals(prevPlayerId) || !
nextSubjectId.equals(prevSubjectId)))
            throw new Exception("Данный результат уже зафиксирован");
        new DataContext(new UpdateResultQuery()).provide(
                new String[]{resultListId, prevPlayerId, nextPlayerId, prevSubjectId, nextSubjectId,
resultValue});
        return "redirect:/result_list?id=" + resultListId;
    }

    @GetMapping("/close_result_list")
    public String closeResultList(@RequestParam(name = "id") String resultListId) throws
Exception {
        boolean isRelayTeam = (Boolean) new DataContext(new
TeamParticipationQuery()).provide(resultListId);
        if(isRelayTeam) {
            int resultsCount = (Integer) new DataContext(new
ResultsCountQuery()).provide(resultListId);
            int relayResultsCount = (Integer) new DataContext(new
RelayResultsCountQuery()).provide(resultListId);
            if(resultsCount != relayResultsCount)
                throw new Exception("Данная команда не прошла все испытания");
        }
        new DataContext(new CloseResultListQuery()).provide(resultListId);
        return "redirect:/result_list?id=" + resultListId;
    }

    @GetMapping("/close_relay_race")
    public String closeRelayRace(@RequestParam(name = "id") String id) throws Exception {
        boolean isNotFinished = !(Boolean) new DataContext(new
RelayIsFinishedQuery()).provide(id);
        if(isNotFinished)
            throw new Exception("Не все команды завершили эстафету");
        new DataContext(new CloseRelayRaceQuery()).provide(id);
        return "redirect:/relay_race?id=" + id;
    }
```

```
}

@Controller
public class DeleteController extends AController {

    @GetMapping("/remove_team")
    public String deleteTeam(@RequestParam(name = "id") String id) {
        new DataContext(new RemoveTeamQuery()).provide(id);
        return "redirect:/teams";
    }

    @GetMapping("/remove_player")
    public String deletePlayer(@RequestParam(name = "id") String id) {
        new DataContext(new RemovePlayerQuery()).provide(id);
        return "redirect:/players";
    }

    @GetMapping("/remove_subject")
    public String deleteWorkout(@RequestParam(name = "id") String id) {
        new DataContext(new RemoveSubjectQuery()).provide(id);
        return "redirect:/subjects";
    }

    @GetMapping("/exclude_team_subject")
    public String deleteWorkout(@RequestParam(name = "id") String teamId,
                                @RequestParam(name = "subject_id") String subjectId) {
        new DataContext(new ExcludeTeamSubjectQuery()).provide(new String[]{teamId,
subjectId});
        return "redirect:/team?id=" + teamId;
    }

    @GetMapping("/remove_result_list")
    public String removeResultList(@RequestParam(name = "id") String teamId,
                                   @RequestParam(name = "result_list_id") String resultListId) {
        new DataContext(new RemoveResultListQuery()).provide(resultListId);
        return "redirect:/team?id=" + teamId;
    }

}

@Controller
public class ErrorController extends AController {

    @GetMapping("/404")
    public String notFound(Model model) {
        model.addAttribute("exception", "Данная страница не существует...");
        return "general/exception";
    }
}
```

Листинг В8. Класс-сущности

```
public class Team {
    private int _teamId;
    private String _teamName;
    private String _trainers;

    public int getTeamId() {
        return _teamId;
```

```
    }

    public void setTeamId(int teamId) {
        _teamId = teamId;
    }

    public String getTeamName() {
        return _teamName;
    }

    public void setTeamName(String teamName) {
        _teamName = teamName;
    }

    public String getTrainers() {
        return _trainers;
    }

    public void setTrainers(String trainers) {
        _trainers = trainers;
    }
}

public class Player {
    private int _playerId;
    private int _teamId;
    private String _playerName;
    private String _teamName;

    public int getPlayerId() {
        return _playerId;
    }

    public void setPlayerId(int playerId) {
        _playerId = playerId;
    }

    public int getTeamId() {
        return _teamId;
    }

    public void setTeamId(int teamId) {
        _teamId = teamId;
    }

    public String getPlayerName() {
        return _playerName;
    }

    public void setPlayerName(String playerName) {
        _playerName = playerName;
    }

    public String getTeamName() {
        return _teamName;
    }

    public void setTeamName(String teamName) {
        _teamName = teamName;
    }
}
```

```
public class Subject {
    private int _subjectId;
    private String _subjectName;
    private String _subjectUnit;
    private double _subjectMultiplier;

    public int getSubjectId() {
        return _subjectId;
    }

    public void setSubjectId(int subjectId) {
        _subjectId = subjectId;
    }

    public String getSubjectName() {
        return _subjectName;
    }

    public void setSubjectName(String subjectName) {
        _subjectName = subjectName;
    }

    public String getSubjectUnit() {
        return _subjectUnit;
    }

    public void setSubjectUnit(String subjectUnit) {
        _subjectUnit = subjectUnit;
    }

    public double getSubjectMultiplier() {
        return _subjectMultiplier;
    }

    public void setSubjectMultiplier(double subjectMultiplier) {
        _subjectMultiplier = subjectMultiplier;
    }
}

public class Result {
    private int _resultListId;
    private String _resultListName;
    private int _playerId;
    private String _playerName;
    private int _subjectId;
    private String _subjectName;
    private double _resultValue;
    private String _subjectUnit;
    private double _subjectMultiplier;
    private Date _resultDate;

    public int getPlayerId() {
        return _playerId;
    }

    public void setPlayerId(int playerId) {
        _playerId = playerId;
    }

    public String getPlayerName() {
```

```
    return _playerName;
}

public void setPlayerName(String playerName) {
    _playerName = playerName;
}

public int getResultListId() {
    return _resultListId;
}

public void setResultListId(int resultListId) {
    _resultListId = resultListId;
}

public String getResultListName() {
    return _resultListName;
}

public void setResultListName(String resultListName) {
    _resultListName = resultListName;
}

public int getSubjectId() {
    return _subjectId;
}

public void setSubjectId(int subjectId) {
    _subjectId = subjectId;
}

public String getSubjectName() {
    return _subjectName;
}

public void setSubjectName(String subjectName) {
    _subjectName = subjectName;
}

public double getResultValue() {
    return _resultValue;
}

public void setResultValue(double resultValue) {
    _resultValue = resultValue;
}

public String getSubjectUnit() {
    return _subjectUnit;
}

public void setSubjectUnit(String subjectUnit) {
    _subjectUnit = subjectUnit;
}

public double getSubjectMultiplier() {
    return _subjectMultiplier;
}

public void setSubjectMultiplier(double subjectMultiplier) {
    _subjectMultiplier = subjectMultiplier;
```

```
    }

    public Date getResultDate() {
        return _resultDate;
    }

    public void setResultDate(Date resultDate) {
        _resultDate = resultDate;
    }
}

public class ResultList {
    private int _resultListId;
    private String _resultListName;
    private int _teamId;
    private boolean _isOpen;
    private Date _resultListDate;

    public int getResultListId() {
        return _resultListId;
    }

    public void setResultListId(int resultListId) {
        _resultListId = resultListId;
    }

    public String getResultListName() {
        return _resultListName;
    }

    public void setResultListName(String resultListName) {
        _resultListName = resultListName;
    }

    public int getTeamId() {
        return _teamId;
    }

    public void setTeamId(int teamId) {
        _teamId = teamId;
    }

    public boolean isOpen() {
        return _isOpen;
    }

    public void setOpen(boolean isOpen) {
        _isOpen = isOpen;
    }

    public Date getResultListDate() {
        return _resultListDate;
    }

    public void setResultListDate(Date resultListDate) {
        _resultListDate = resultListDate;
    }
}

public class RelayTeam extends Team{
    private int _resultListId;
```

```
    private double _resultListScore;

    public RelayTeam() {}

    public RelayTeam(Team team) {
        setTeamId(team.getTeamId());
        setTeamName(team.getTeamName());
        setTrainers(team.getTrainers());
    }

    public int getResultListId() {
        return _resultListId;
    }

    public void setResultListId(int resultListId) {
        _resultListId = resultListId;
    }

    public double getResultListScore() {
        return _resultListScore;
    }

    public void setResultListScore(double resultListScore) {
        _resultListScore = resultListScore;
    }
}

public class RelayRace {
    private int _relayId;
    private String _relayName;
    private int _teamNumber;
    private int _playerNumber;
    private boolean _isOpen;

    public int getRelayId() {
        return _relayId;
    }

    public void setRelayId(int relayId) {
        _relayId = relayId;
    }

    public String getRelayName() {
        return _relayName;
    }

    public void setRelayName(String relayName) {
        _relayName = relayName;
    }

    public int getTeamNumber() {
        return _teamNumber;
    }

    public void setTeamNumber(int teamNumber) {
        _teamNumber = teamNumber;
    }

    public int getPlayerNumber() {
        return _playerNumber;
    }
```

```
    public void setPlayerNumber(int playerNumber) {
        _playerNumber = playerNumber;
    }

    public boolean isOpen() {
        return _isOpen;
    }

    public void setOpen(boolean isOpen) {
        _isOpen = isOpen;
    }
}
```

Листинг В9. Абстрактные классы запросов

```
public abstract class AItemQuery<TResult, TArgument> implements ISqlQueryable<TResult,
TArgument> {

    protected abstract String query(TArgument arg);
    protected abstract TResult item(EntityFactory builder) throws SQLException;

    @Override
    public TResult execute(Statement statement, TArgument arg) throws SQLException {
        ResultSet set = statement.executeQuery(query(arg));
        if(set.next())
            return item(new EntityFactory(set));
        else
            return null;
    }
}

public abstract class AListQuery<TResult, TArgument> implements
ISqlQueryable<Collection<TResult>, TArgument> {

    protected abstract String query(TArgument arg);
    protected abstract TResult item(EntityFactory builder) throws SQLException;

    @Override
    public Collection<TResult> execute(Statement statement, TArgument arg) throws
SQLException {
        ResultSet set = statement.executeQuery(query(arg));
        Collection<TResult> list = new ArrayList<>();
        EntityFactory builder = new EntityFactory(set);
        while(set.next())
            list.add(item(builder));
        return list;
    }
}

public abstract class AInsertQuery<TArg> implements ISqlQueryable<Integer, TArg> {

    protected abstract String query(TArg arg);

    @Override
    public Integer execute(Statement statement, TArg arg) throws SQLException {
        ResultSet set = statement.executeQuery(query(arg));
        set.next();
        return set.getInt("last_insert_id");
```

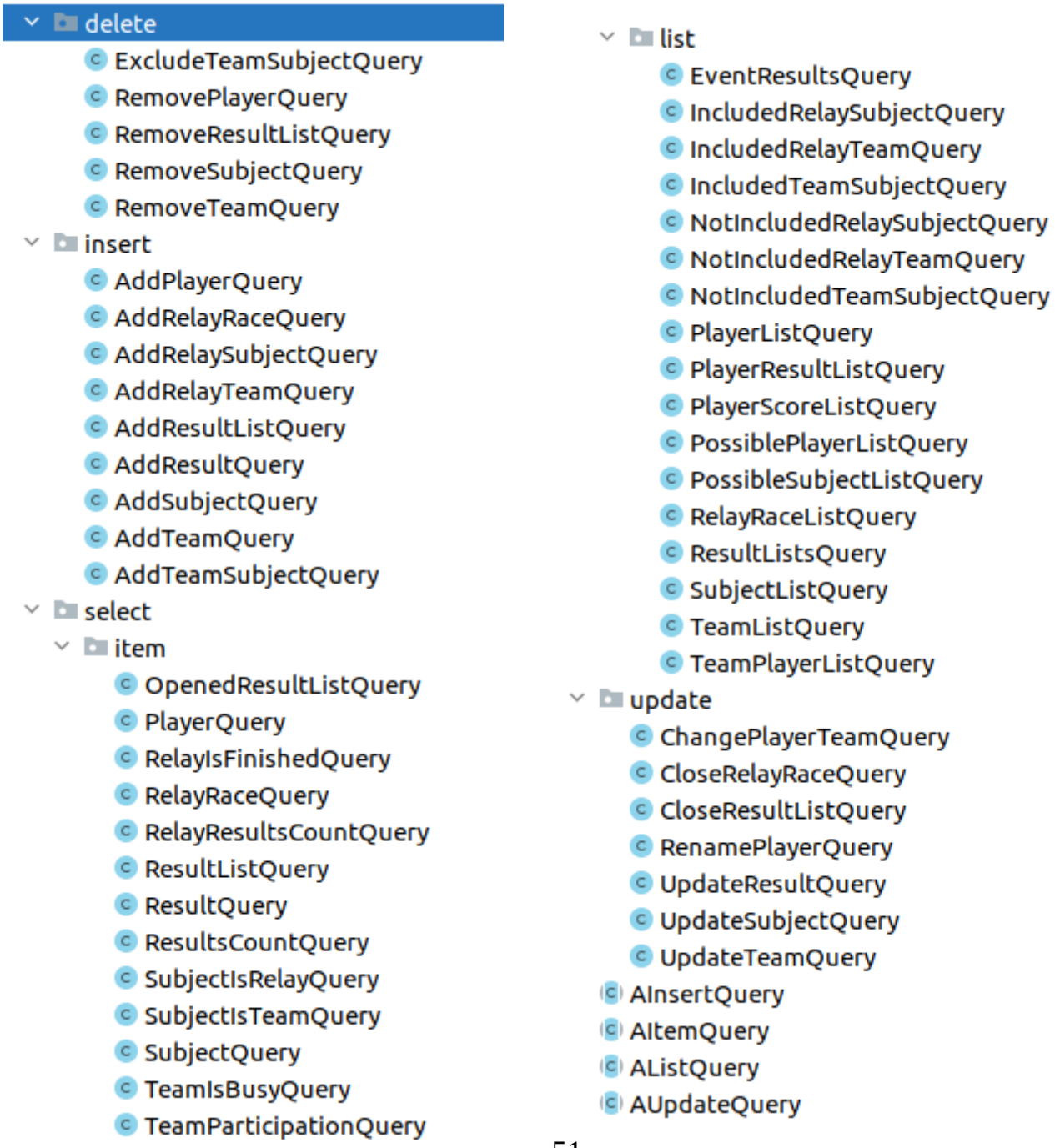
```
    }
}

public abstract class AUpdateQuery<TArgument> implements ISqlQueryable<Void,
TArgument> {

    protected abstract String query(TArgument arg);

    @Override
    public Void execute(Statement statement, TArgument arg) throws SQLException {
        statement.execute(query(arg));
        return null;
    }
}
```

Листинг B10. Классы запросов



ПРИЛОЖЕНИЕ С. Код гипертекстовой разметки

Листинг С1.