Оглавление

Введение

В современном мире спорт, как профессиональный, так и любительский, становится всё более популярным. С каждым годом спортсмены стараются превзойти свои предыдущие достижения, а также установить новые мировые рекорды.

Особняком в спортивной индустрии стоят Олимпийские игры [1] – события, где сильнейшие атлеты со всего мира соренуются друг с другом, чтобы выявить сильнейшего. Зачастую именно на Олимпиаде показываются самые высокие результаты.

Было бы удобно следить за информацией о спортсменах и их местах на Олимпийских играх в одном сервесе для проведения анализа результатов.

Цель данной работы – реализовать веб-сервис, который позволит анализировать количество призовых мест, выигранных атлетами по различным характеристикам.

Для достижения поставленной цели в ходе работы требуется решить следующие задачи:

- проанализировать варианты представления данных и выбрать подходящий вариант для решения задачи;
- проанализировать системы управления базами данных и выбрать подходящую систему для хранения данных;
- проанализировать варианты анализа данных и выбрать подход агрегации данных;
- спроектировать базу данных, описать её связи, сущности;
- реализовать интерфейс для доступа к базе данных и агрегации данных;

- реализовать программное обеспечение, позволяющее анализировать базу данных по заданным критериям.

1 Аналитическая часть

В данном разделе проводится анализ способов хранения и агрегирования данных, а также систем управления базами данны. Определяется постановка задачи.

1.1 Постановка задачи

Разработать веб-сервис “Анализатор Олимпийских игр” для отображения количества призовых мест по задаваемым характеристикам (имя, возраст, пол, вес, рост спортсмена, а также страна, вида спорта и дисциплина вида спорта).

Предусмотреть роли: авторизованный пользователь, не авторизованный пользователь и администратор. Реализовать создание аккаунта и добавление комментариев. Для роли администратора реализовать удаление комментариев и пользователей. Для роли неаутентифицированны пользователь запретить использование сервиса и добавление комментариев.

1.2 Формализация данных

В базе данных должна храниться информация о следующих моделях:

- Атлеты;
- Олимпиады;
- Страны;
- Результаты;
- Виды спорта;
- Дисциплины вида спорта;
- Таблица пользователей;
- Комментарии;
- Буфер комментариев;

Иформация о категориях и данных приведены в таблице 1.1

Таблица 1.1 – Категории и сведения о данных

Категория	Сведения
Атлеты	Имя, пол, возраст, вес, рост, идентификатор страны, идентификатор вида спорта
Страны	Код страны, название страны, столица, население
Олимпиады	Город проведения, год проведения, сезон года(лето, осень)
Результаты	Идентификатор спортсмена, идентификатор Олимпиады, идентификатор вида спорта, идентификатор дисциплины вида спорта, проверка золотой медали, проверка серебренной медали, проверка бронзовой медали
Виды спорта	Название вида спорта
Дисциплины вида спорта	Идентификатор вида спорта, название дисциплины
Таблица пользователей	Имя аккаунта, почта пользователя, проверка на админа, пароль, роль пользователя, дата, время регистрации, имя, фамилия, проверка активности пользователя.
Комментарии	Идентификатор пользователя, текст, дата публикации, время публикации
Буффер комментариев	Идентификатор пользователя, текст, дата публикации, время публикации

1.3 Типы пользователей

Для доступа к основным функциям веб-сервиса, необходимо предусмотреть ролевую модель [2]. Для управления действиями пользователей принято решение ввести три роли: гость(неавторизованный пользователь), авторизованный пользователь и администратор.

Информация о типах пользователей и их возможностях представлены в таблице 1.2

Таблица 1.2 – Типы пользователей и их возможности

Тип пользователя	Возможности
Гость	Доступ к просмотру комментариев и авторизации
Авторизованный пользователь	Доступ к главной странице, возможности произвести поиск, просмотр и добавление комментариев, возможность выйти из системы
Администратор	Доступ к главной странице, возможности произвести поиск, просмотр, добавление и удаление комментариев, возможность выйти из системы, удаление пользователей.

1.4 Базы данных и системы управления базами данных

Для анализа и агригирования данных важную роль играет модель хранения данных, которая будет использоваться для хранения информации о спортсменах, результатов, стран, видов спорта, пользователей и комментариев.

Для хранения данных используются базы данных [3]. Для управления базами данных используются системы управления базами данных (сокращенно СУБД) [4]. Система управления базами данных — это совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

1.5 Хранение данных о пользователях и Олимпийских играх

Разрабатываемый в рамках курсового проекта сервис предполагает собой два приложения:

1. Для анализа и агрегирования данных
2. Для отображения результатов, полученных первым приложением, а также обработки действий пользователей в графическом интерфейсе

Каждое из двух приложений имеет права, отличные от другого. Первое должно иметь возможность читать данные, отвечающие за информацию о спортсменах и их результатах, второе же должно иметь возможность читать, записывать и удалалять данные пользователей и комментарии.

Для хранения информации о Олимпийских играх необходимо использовать строго типизированную базу данных, т.к. все данные имеют четко выраженную структуру, неподдающуюся изменениям.

1.6 Классификация баз данных по способу хранения

Способы хранения баз данных бывают двух видов: колоночные и строковые. Каждый из этих видов хранения подходит для реализации определенных задач.

1.6.1 Колоночные базы данных

Записи в базах данных колоночного типа представляются по столбцам (в памяти). Данный тип используется в аналитических система, которые характеризуется низким объемом транзакций, а запросы часто сложны и включают в себя агрегацию. Показателем эффективности системы является время отклика.

1.6.2 Строковые базы данных

Записи в базах данных строкового типа представляются построчно (в памяти). Для таких систем характерно большое количество коротких транзакций с операциями вставки, обновления и удаления данных. Зачастую их используют в транзакционных системах. Основными задачами транзакционных систем являются: поддержание целостности данных, быстрая обработка запросов. Показателем эффективности системы является количество транзакций в секунду.

1.7 Способы агрегации данных

Для сбора информации о результатах, зависящих от задаваемых характеристик, необходима агрегация данных.

1.7.1 Встроенные агрегатные функции базы данных

Современные базы данных содержат в себе механизмы анализа – агрегатные функции (агрегаторы) [5]. Агрегатные функции получают единственный результат из набора входных значений.

Значения нескольких строк или столбцов группируются вместе, образуя единое итоговое значение.

Примеры агрегатных функций:

- Нахождение среднего
- Нахождение среднего
- Подсчёт количества
- Нахождение максимума
- Нахождение минимума
- Нахождение суммы
- и другие

1.7.2 OLAP

OLAP (Online analytical processing) [6] – это технология обработки данных для выполнения многомерного анализа на высокой скорости больших объемов данных из хранилища данных.

Большое количество данных имее несколько измерений - несколько категорий, по которым можно разбить информацию для более детального представления или анализа. Например, показатели продаж могут иметь несколько измерений, связанных с местоположением (регион, страна, магазин), временем (год, месяц, неделя, день, часб секунды), продуктом (одежда, мужчины/женщины/дети, бренд), и другие.

В хранилище данных наборы данных хранятся в таблицах, каждая из которых может упорядочивать данные только по двум из этих измерений одновременно. OLAP извлекает данные из нескольких наборов реляционных данных и реорганизует их в многомерный формат, который обеспечивает очень быструю обработку и очень глубокий анализ.

Фактически, OLAP-сервер обычно является средним аналитическим уровнем решения для хранилища данных.

OLAP Cube

OLAP Cube или MOLAP (Multidimensional OLAP) [7] – это многомерная база данных на основе массивов, которая позволяет обрабатывать и анализировать несколько измерений данных намного быстрее и эффективнее, чем традиционная реляционная база данных .

Таблица реляционной базы данных структурирована как обычная таблица, в которой отдельные записи хранятся в двухмерном формате по строкам и столбцам. Каждый «факт» в базе данных находится на пересечении двух измерений - строки и столбца.

Инструменты отчетов SQL и реляционных баз данных, безусловно, могут запрашивать и анализировать многомерные данные, хранящиеся в таблицах, но производительность снижается по мере увеличения объемов данных.

Многомерная база данных создаётся путём расширения реляционной базы данных дополнительными слоями и измерениями (осями), состоящими из уровней. Например, верхний слой куба может упорядочивать продажи по местам, а подслоями могут быть страна, город, пункт выдачи. После расширения системы хранения данных куб полностью собирается и сохраняется в памяти машины.

Теоретически куб может содержать бесконечное количество слоев, но на практике аналитики данных создают OLAP кубы, содержащие только те слои, которые им нужны, для оптимального анализа и производительности.

Одними из примуществ OLAP куба является то, что нет необходимости хранить реляционную базу данных из которой сформирован куб, т.к. после сборки он сохраняется обособленно от системы хранения данных и не зависит от её типа (строкового или колоночного). Из этого вытекает главный минус: необходима дополнительная память для хранения куба.

OLAP кубы позволяют выполнять четыре основных типа операций многомерного анализа данных:

- Детализация (Drill-down)
- Свертывание (Roll up)
- Срез (Slice and dice)
- Поворот (Pivot)

На рисунке 1.1 изображён пример представления OLAP куба, содержащего 3 оси: время, продукт, место.

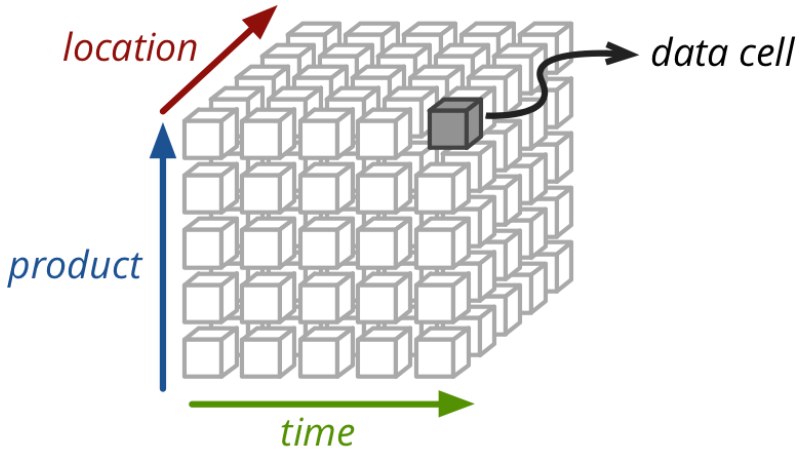


Рисунок 1.1 – Пример трёхмерного OLAP куба

На рисунке 1.2 изображён пример представления 4х основных операций над OLAP кубом.

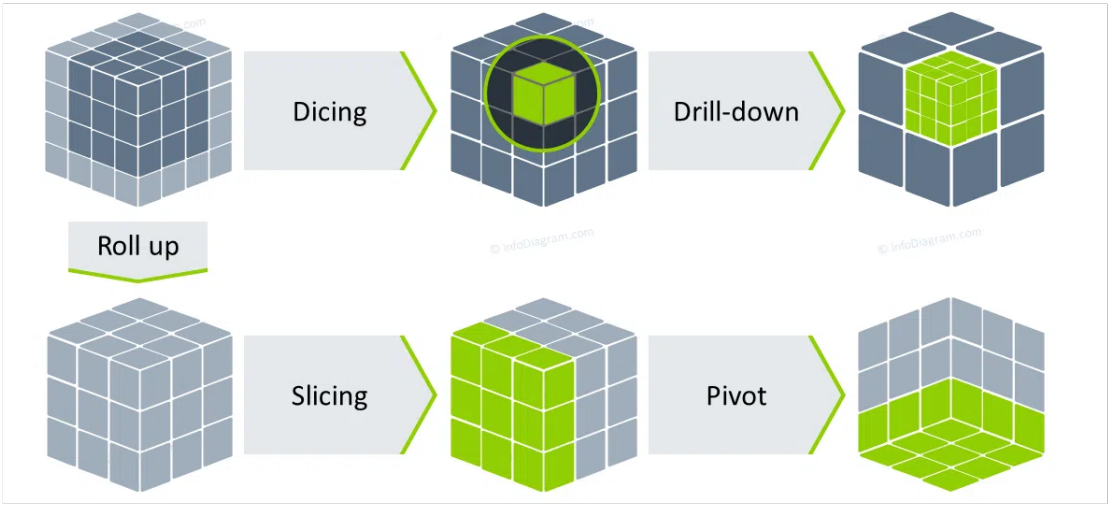


Рисунок 1.2 – Пример операций над OLAP кубом

ROLAP

ROLAP (Relational OLAP) [6] – это многомерный анализ данных, который работает непосредственно с данными в реляционных таблицах, без предварительной реорганизации данных в куб.

Как отмечалось ранее, SQL - это превосходный инструмент для многомерных запросов, отчетов и анализа. Но требуемые SQL запросы сложны, производительность может замедляться, а результирующее представление данных является статическим - его нельзя преобразовать для отображения другого представления данных. ROLAP лучше всего подходит, когда возможность работать с большими объемами данных напрямую важнее, чем производительности и гибкости.

HOLAP

При использовании HOLAP (Hybrid OLAP) [6] происходит пытака создать оптимальное разделение труда между реляционными и многомерными базами данных в рамках единой архитектуры OLAP. Реляционные таблицы содержат большие объемы данных, а кубы OLAP используются для агрегирования и спекулятивной обработки. Для HOLAP требуется сервер OLAP, поддерживающий как MOLAP, так и ROLAP.

Инструмент HOLAP может «детализировать» куб данных до реляционных таблиц, что открывает путь для быстрой обработки данных и гибкого доступа. Эта гибридная система может предложить лучшую масштабируемость, но не может избежать неизбежного замедления при доступе к реляционным источникам данных. Кроме того, его сложная архитектура обычно требует более частых обновлений и обслуживания, поскольку она должна хранить и обрабатывать все данные из реляционных баз данных и многомерных баз данных. По этой причине HOLAP может оказаться более долгим и требовательным.

1.8 Выбор способа агрегации данных для решения задачи

Для решения задачи был выбран способ создания OLAP Cube (MOLAP), т.к. он выигрывает в скорости у обычной реалиционной базы данных, после его сборки не требуется дополнительное вмешательсто со стороны пользователя, а также есть возможность размещения на сервер, в качестве отдельного приложения.

1.9 Выбор модели хранения данных для решения задачи

Для решения задачи было выбрано построчное хранение данных по нескольким причинам:

- Для реализации приложения анализа данных собирается OLAP Cube, который не зависит от модели хранения данных, а значит нет причин не использовать построчное хранение;
- Для приложения, отображающего полученные данные из первого (OLAP Cube) не предполагается выполнения аналитических запросов;
- Для второго приложения, предполагает постоянное добавление, изменение данных о коментариях и пользователях;

1.9.1 Обзор СУБД с построчным хранением

В данном подразделе буду рассмотрены популярные построчные СУБД, которые могут быть использованы для реализации хранения в разрабатываемом программном продукте.

PostgreSQL

PostgreSQL [8] – это свободно распространяемая объектно-реляционная система управления базами данных, наиболее развитая из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных.

PostgreSQL предоставляет транзакции со свойствами атомарности, согласованности, изоляции, долговечности (ACID), автоматически обновляемые представления, материализованные представления, триггеры, внешние ключи и хранимые процедуры. Данная СУБД предназначена для обработки ряда рабочих нагрузок, от отдельных компьютеров до хранилищ данных или веб-сервисов с множеством одновременных пользователей.

Рассматриваемая СУБД управляет параллелизмом с помощью технологии управления многоверсионным параллелизмом (англ. MVCC). Эта технология дает каждой транзакции «снимок» текущего состояния базы данных, позволяя вносить изменения, не затрагивая другие транзакции. Это в значительной степени устраняет необходимость в блокировках чтения и гарантирует, что база данных поддерживает принципы ACID.

MySQL

MySQL [9] – свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle.

Рассматриваемая СУБД имеет два основных движка хранения данных: InnoDB и myISAM. Движок InnoDB полностью совместим с принципами ACID, в отличии от движка myISAM. СУБД MySQL подходит для использования при разработке веб-приложений. Реализация параллелизма в СУБД MySQL реализовано с помощью механизма блокировок, который обеспечивает одновременный доступ к данным.

Oracle Database

Oracle Database [10] – объектно-реляционная система управления базами данных компании Oracle. На данный момент, рассматриваемая СУБД является самой популярной в мире.

Все транзакции Oracle Database соответствуют свойствами ACID, поддерживает триггеры, внешние ключи и хранимые процедуры. Данная СУБД подходит для разнообразных рабочих нагрузок и может использоваться практически в любых задачах. Особенностью Oracle Database является быстрая работа с большими массивами данных.

Oracle Database может использовать один или более методов параллелизма. Сюда входят механизмы блокировки для гарантии монопольного использования таблицы одной транзакцией, методы временных меток, которые разрешают сериализацию транзакций и планирование транзакций на основе проверки достоверности.

1.9.2 Выбор СУБД для решения задачи

Для решения задачи была выбрана СУБД PostgreSQL, потому что данная СУБД проста в развертывании, имеет документацию на русском языке, а также легко интегрируема в приложение, написанное на языке программирования python [11] из-за большого числа библиотек для работы с ней.

Вывод

В данном разделе:

- Была проанализирована поставленная задача и описаны способы ее реализации.
- Был проведен анализ СУБД, и выбрана оптимальная СУБД для решения задачи;
- Был выбран наиболее подходящий способ анализа и агригирования данных;

2 Конструкторская часть

В данном разделе представлены этапы проектирования баз данных и OLAP Cube.

2.1 Проектирование отношений сущностей

На рисунках 2.1 и 2.2 представлены ER диаграммы сущностей, необходимых для реализации приложения.

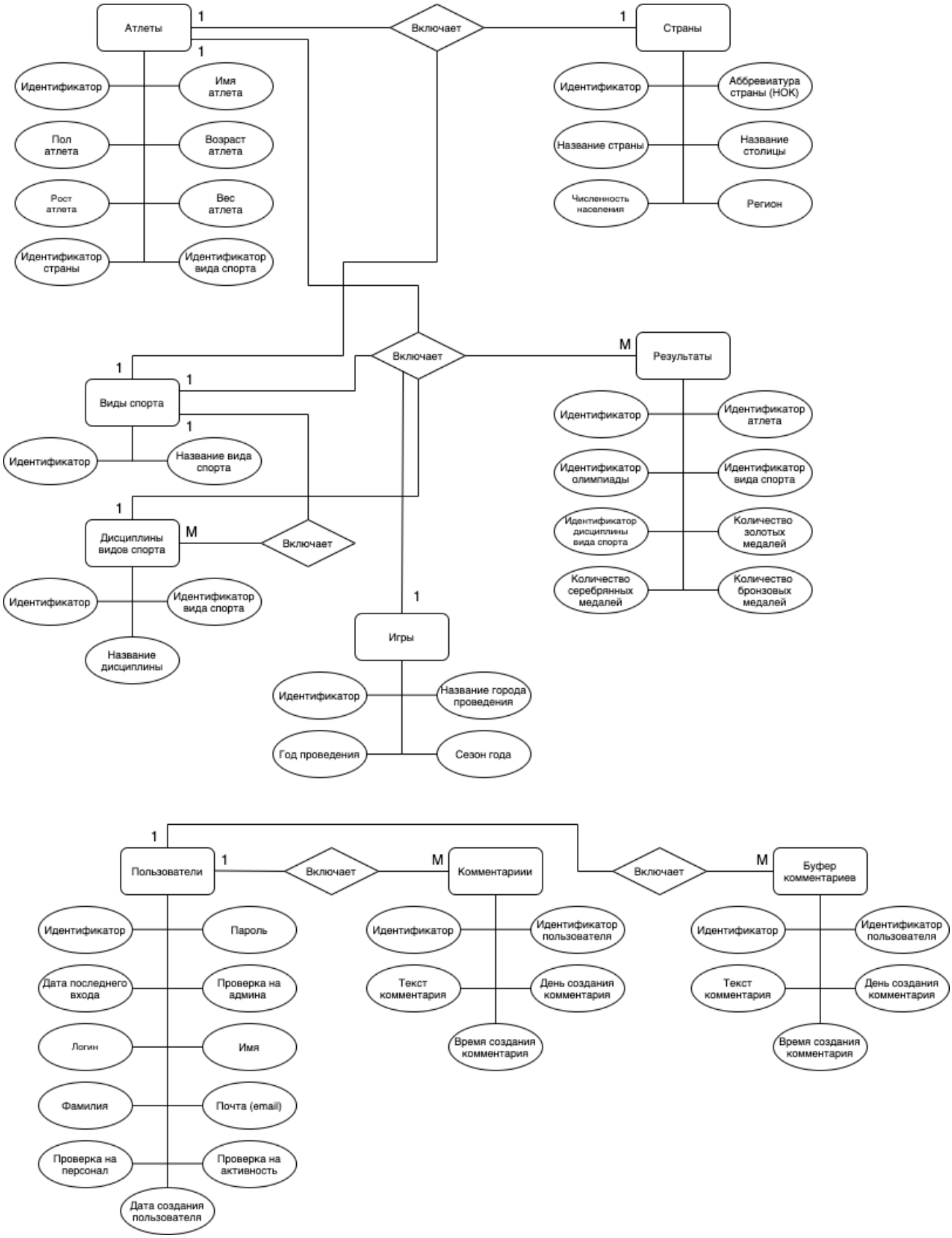


Рисунок 2.1 – ER-диаграмма сущностей базы данных в нотации Чена

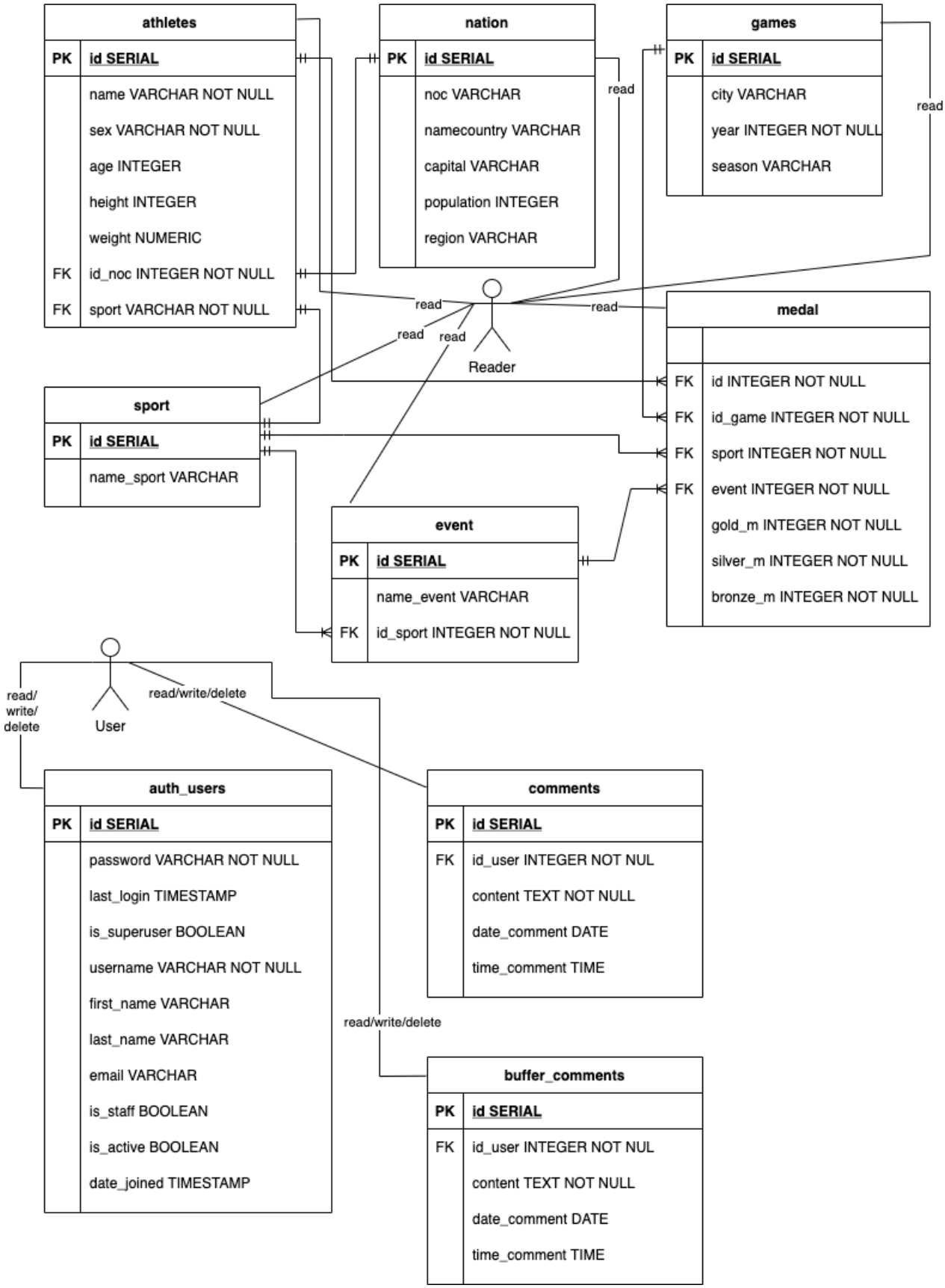


Рисунок 2.2 – ER диаграмма сервиса

2.2 Проектирование базы данных Олимпийских игр

База данных будет реализована с использованием СУБД PostgreSQL. В базе данных будет существовать 9 сущностей и 9 таблиц. ER-диаграма сущностей этой базы данных представлена на рисунке 2.2.

Поля таблицы `athletes` означают:

- `id` - уникальный идентификатор спортсмена;
- `name` - имя спортсмена;
- `sex` - пол спортсмена. Поле принимает значение M или F, если пол спортсмена мужской или женский соответсвенно;
- `age` - возраст спортсмена;
- `height` - рост спортсмена с точностью до см;
- `weight` - вес спортсмена с точностью до 0.1 кг;
- `id_noc` - идентификатор страны спортсмена;
- `sport` - идентификатор вида спорта, которым занимается спортсмен;

Поля таблицы `nation` означают:

- `id` - уникальный идентификатор страны;
- `noc` - аббревиатура страны (НОК);
- `namecountry` - полное название страны;
- `capital` - город, являющийся столицей страны;
- `population` - количество населения;

- `region` - регион (часть света), в котором расположена страна;

Поля таблицы `games` означают:

- `id` - уникальный идентификатор Олимпиады;
- `city` - город, в котором проходила Олимпиада;
- `year` - год прохождения олимпиады;
- `season` - сезон года, в котором проходила Олимпиада. Может принимать значения Summer или Winter, если Олимпиада была летом или зимой соответственно;

Поля таблицы `sport` означают:

- `id` - уникальный идентификатор вида спорта;
- `name_sport` - название вида спорта;

Поля таблицы `event` означают:

- `id` - уникальный идентификатор дисциплины вида спорта;
- `name_event` - название дисциплины вида спорта;
- `id_sport` - идентификатор вида спорта, которому принадлежит дисциплина;

Поля таблицы `medal` означают:

- `id` - идентификатор спортсмена;
- `id_game` - идентификатор Олимпиады;

- `sport` - вид спорта в котором принимал участие спортсмен;
- `event` - дисциплина а в которой соревновался спортсмен;
- `gold_m` - количество золотых медалей, которое спортсмен выиграл за участие в данной дисциплине на этой Олимпиаде (может быть 0 или 1, если не выиграл или выиграл соответственно);
- `silver_m` - количество серебрянных медалей, которое спортсмен выиграл за участие в данной дисциплине на этой Олимпиаде (может быть 0 или 1, если не выиграл или выиграл соответственно);
- `bronze_m` - количество бронзовых медалей, которое спортсмен выиграл за участие в данной дисциплине на этой Олимпиаде (может быть 0 или 1, если не выиграл или выиграл соответственно);

Поля таблицы `auth_user` означают:

- `id` - уникальный идентификатор пользователя;
- `password` - захэшированный пароль;
- `last_login` - дата, последней авторизации пользователя в сервисе;
- `is_superuser` - проверка того, что пользователь является админом (может принимать значение true, если является, иначе false);
- `username` - логин пользователя, должен быть уникальным для каждого;
- `first_name` - имя пользователя;
- `last_name` - фамилия пользователя;
- `email` - почта пользователя;
- `is_staff` - проверка того, что пользователь является одним из сотрудников сервиса (может принимать значение true, если является, иначе false);

- `is_active` - проверка того, что пользователь ещё пользуется сервисом;
- `data_joined` - дата созлания аккаунта пользователя;

Поля таблицы `comments` означают:

- `id` - уникальный идентификатор комментария;
- `id_user` - идентификатор пользователя, оставившего комментарий;
- `content` - текст комментария;
- `date_comment` - дата публикации комментария (год, месяц, день);
- `time_commen` - время публикации комментария (час, минута, секунда);

Таблица `buffer_comments` является полной копией таблицы `comments`. Данная таблица содержит все комментарии, которые когда-либо были добавлены пользователями и предназаначина только для нужд администратора. Если удалить комментарий из таблицы `comments`, то в таблице `buffer_comments` он останется без изменений.

2.3 Проектирование отношений между таблицей фактов и измерений в OLAP Cube

Развёрнутый OLAP Cube будет содержать таблицу фактов, по которой будет происходить поиск мер по заданным параметрам и их агрегирование. Операции с OLAP моделью (drilldown, slice, dice и другие) будут возможны благодаря рабиению таблицы фатков по осям (измерениям).

Компоненты проектируемого OLAP куба:

- В качестве таблицы фактов [12] принимается таблица `medal`;
- В качестве осей (измерений) [12] принимаются таблицы: `athletes`, `nation`, `games`, `sport` и `event`;
- В качестве мер агрегации [12] принимаются поля таблицы `medal`: `gold_m`, `silver_m`, `silver_m`;
- Связь измерений с таблицей фактов реализуется благодаря отношению `left-join` [13], поддерживаемого OLAP системами;

Вывод

В данном разделе были представлены этапы проектирования базы данных и OLAP Cube, а также показана ER диаграмма сервиса.

3 Технологическая часть

В данном разделе представлены архитектура приложения, средства разработки программного обеспечения, детали реализации и способы взаимодействия с программным продуктом.

3.1 Архитектура приложения

Предполагается, что разрабатываемый сервис состоит из двух приложений: анализатор (OLAP Cube) и веб-интерфейс взаимодействия с данными, полученными из первого микросервиса. Доступ к агрегированным данным будет получен с помощью GET запросов.

Общая схема архитектура сервиса представлена на рисунке 3.1.

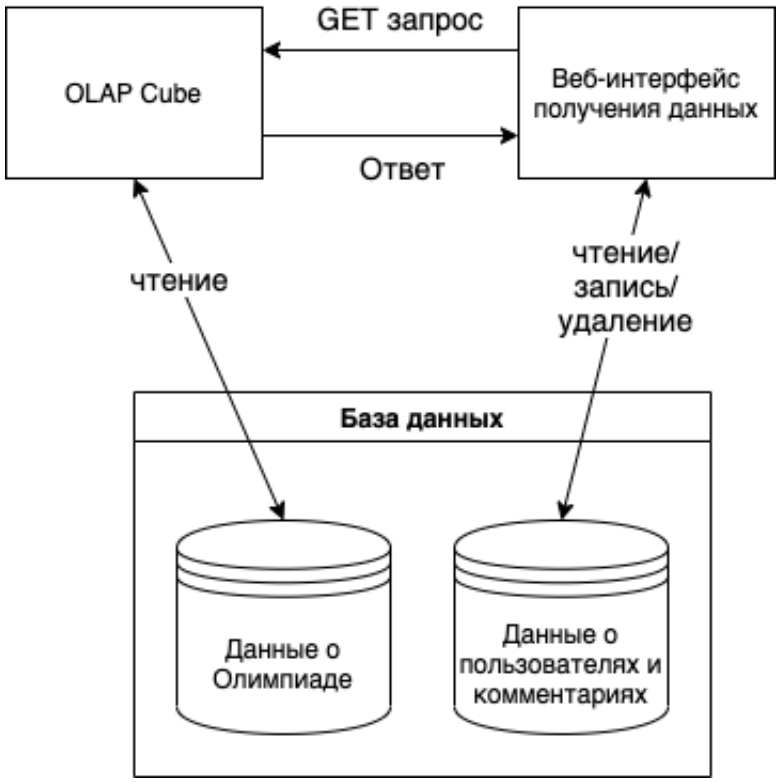


Рисунок 3.1 – Схема архитектуры приложения

3.2 Средства реализации

Для разработки приложений был выбран язык программирования Python [11]. Данный выбор обусловлен простотой языка, его интерпретируемостью, а также тесной интеграцией с СУБД PostgreSQL [11], которая будет использоваться для хранения данных.

Для развёртыания OLAP куба был выбрана фремворк cubes [14], позволяющая развернуть куб на отдельном сервере и общаться с базой данных на основе СУБД PostgreSQL [11] без дополнительных коннекторов или библиотек.

Для получения данных посредством GET запросов была выбрана библиотека request [15].

3.3 Детали реализации

В приложении на листингах 5.1 – 5.3 приведён код взаимодействия приложений с базой данных, OLAP кубом, а также инициализации базы данных.

3.4 Взаимодействие с приложением

На рисунках 3.2 – 3.4 представлены примеры работы приложения.

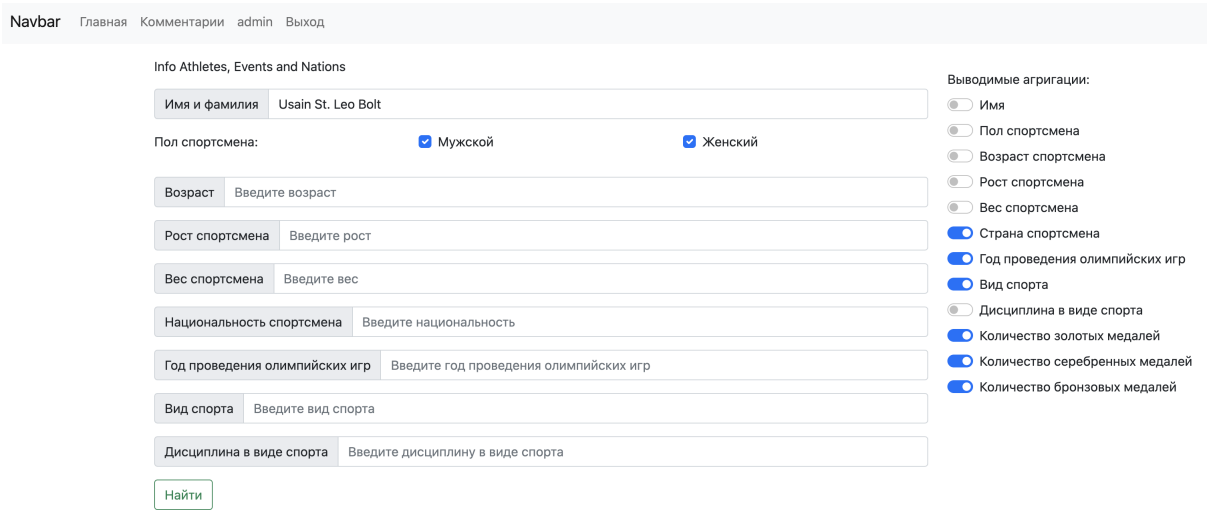
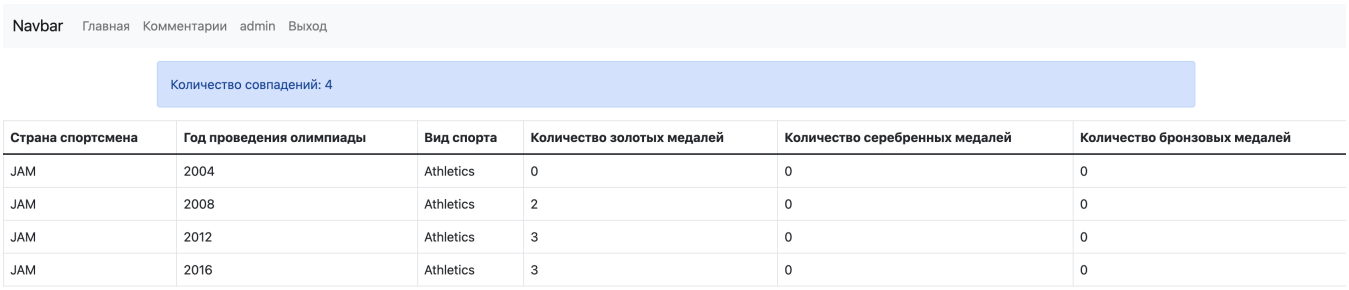


Рисунок 3.2 – Пример взаимодействия с интерфейсом



Страна спортсмена	Год проведения олимпиады	Вид спорта	Количество золотых медалей	Количество серебренных медалей	Количество бронзовых медалей
JAM	2004	Athletics	0	0	0
JAM	2008	Athletics	2	0	0
JAM	2012	Athletics	3	0	0
JAM	2016	Athletics	3	0	0

Рисунок 3.3 – Пример выдачи

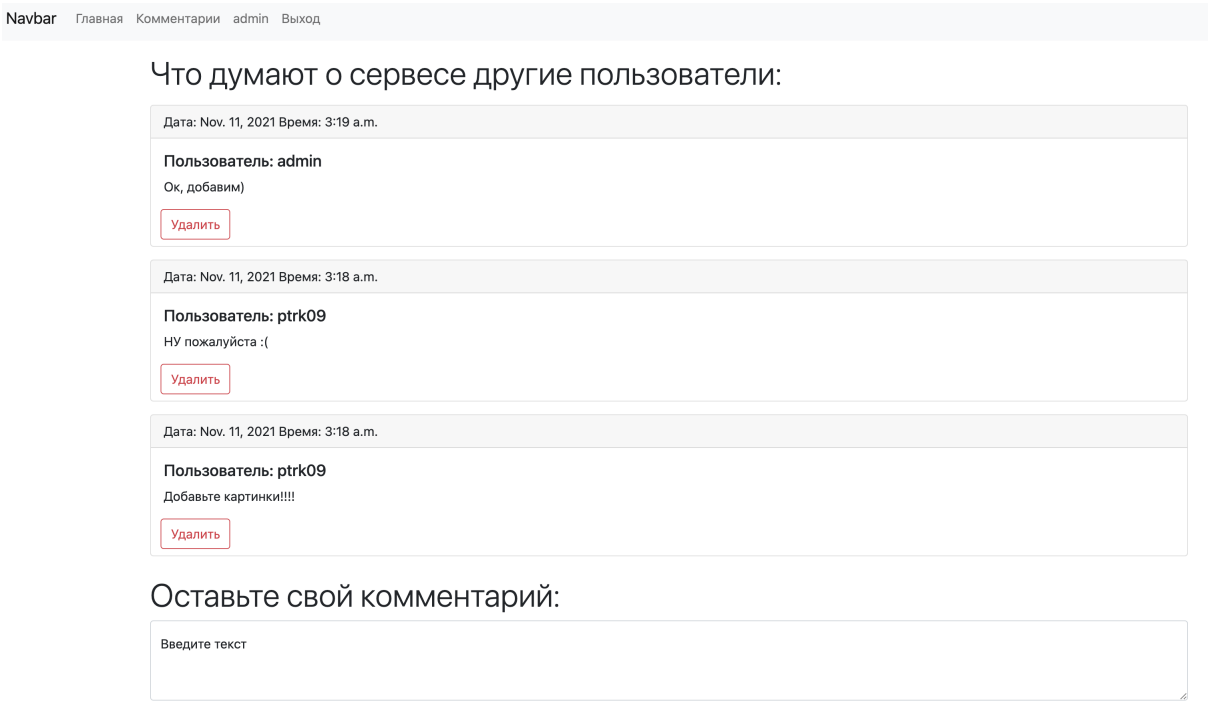


Рисунок 3.4 – Пример работы с комментариями

Вывод

В данном разделе была представлена архитектура и рассмотрены средства реализации веб-сервиса, листинги ключевых компонентов системы и пример работы приложения.

4 Исследовательская часть

В данном разделе представлено сравнение времени агрегации данных в OLAP Cube при разном количестве результатов, показанных на Олимпийских играх.

4.1 Постановка эксперимента

Время обработки замерялось при помощи библиотеки time. Время замерялось на протяжении 100 запусков и брался средний показатель.

4.1.1 Цель эксперимента

Целью эксперимента является сравнение времени, требуемого для агригации полученных данных.

4.1.2 Результат эксперимента

В таблице 4.1 представлены результаты поставленного эксперимента.

Таблица 4.1 – Результаты сравнения времени, необходимого для агрегации данных Олимпиад

Количество результатов	**Количество срезов (slice) по осям**	**Время , мс**
100	0	1490
100	1	989
100	2	300
500	0	1561
500	1	1000
500	2	490
1500	0	1763
1500	1	1100
1500	2	621
3000	0	2167
3000	1	1700
3000	2	1121
3000	3	1000
5000	0	2727
5000	1	1500
5000	2	1100
5000	3	570
10000	0	3476
10000	1	2056
10000	2	1302
10000	3	811

На рисункt 4.1 представлены графики зависимости времени агрегации от количества результатов, показанных на Олимпийских играх.

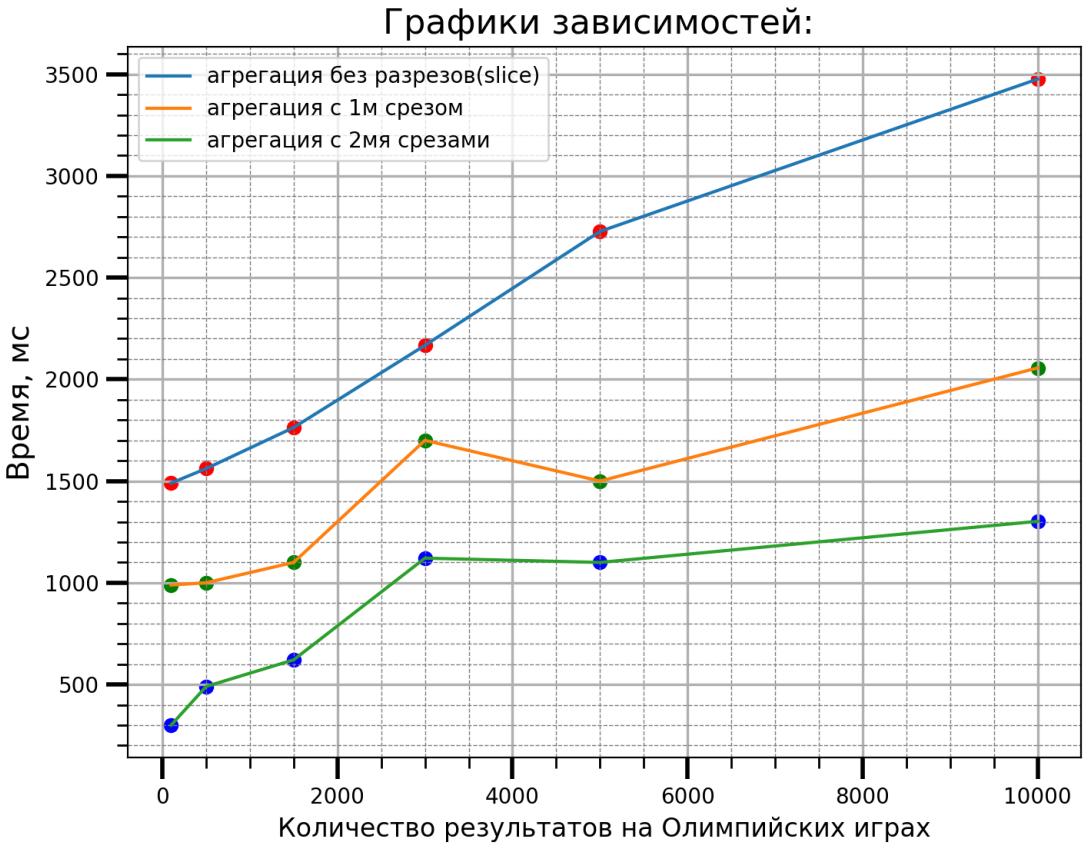


Рисунок 4.1 – Зависимость времени агригации данных от количества результатов, при отсутствии, одном и двух срезах.

Вывод

В результате сравнения времени, необходимого для агрегации данных об результатах Олимпиад, были получены следующие результаты:

- С увелечением количества результатов Олимпиады увеличивается время агрегации;
- С увелечением количества срезов время агрегации уменьшается;
- С увелечением количества срезов скорость роста времени агрегации уменьшается при увелечении количества данных;

Выяснено, что эффективность агрегации с ростом объёма данных уменьшается при отсутствии операций среза. В тоже время при увелечении количества срезов время агрегации на тех же объёмах данных значительно сокращается и замедляет свой рост.

Можно сделать вывод, что способ анализа и агригирования данных с помощью OLAP куба показывает максимальную эффективность при введении некоторого числа срезов. Возможно, другие OLAP модели или иные технологии анализа могут дать лучшие результаты и без введения срезов.

5 Заключение

В ходе курсового проекта было разработано программное обеспечение, для хранения и анализа результатов, показанных спортсменами на Олимпийских играх.

В процессе выполнения данной работы были выполнены следующие задачи:

- проанализированы варианты представления данных и выбран подходящий вариант для решения задачи;
- проанализированы системы управления базами данных и выбрана подходящая система для хранения данных;
- проанализированы варианты анализа данных и выбран подходящий подход агригации данных;
- спроектирована база данных, описаны её связи, сущности;
- реализован интерфейс для доступа к базе данных и агрегации данных;
- реализовано программное обеспечение, позволяющее анализировать базу данных по заданным критериям.

В процессе исследовательской работы было выяснено, что, для более эффективной по времени агрегации, необходимо ввести дополнительные параметры поиска.

Литература

[1] Olypic games [Электронный ресурс]. Режим доступа: https://olympics.com/ru/ (дата обращения: 01.06.2021).

[2] Implementing Role Based Security in a Web App [Электронный ресурс]. Режим доступа: https://medium.com/bluecore-engineering/implementing-role-based-security-in-a-web-app-89b66d1410e4 (дата обращения: 01.06.2021).

[3] Что такое база данных | Oracle Россия и СНГ [Электронный ресурс]. Режим доступа: https://www.oracle.com/ru/database/what-is-database/ (дата обращения: 07.06.2021).

[4] Что такое СУБД - RU-CENTER [Электронный ресурс]. Режим доступа: https://www.nic.ru/help/chto-takoe-subd_8580.html (дата обращения: 07.06.2021).

[5] Агрегатные функции (Transact-SQL) [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/sql/t-sql/functions/aggregate-functions-transact-sql?view=sql-server-ver15 (дата обращения: 07.06.2021).

[6] What is OLAP? | IBM [Электронный ресурс]. Режим доступа: https://www.ibm.com/cloud/learn/olap (дата обращения: 07.06.2021).

[7] Общие сведения о кубах OLAP в Service Manager для расширенной аналитики. Режим доступа: https://docs.microsoft.com/ru-ru/system-center/scsm/olap-cubes-overview?view=sc-sm-2019 (дата обращения: 07.06.2021).

[8] PostgreSQL: Документация. [Электронный ресурс]. Режим доступа: https://postgrespro.ru/docs/postgresql/ (дата обращения: 07.06.2021).

[9] MySQL Database Service is a fully managed database service to deploy

cloud-native applications. [Электронный ресурс]. Режим доступа: https://www.mysql.com/ (дата обращения: 07.06.2021).

[10] SQL Language | Oracle[Электронный ресурс]. Режим доступа: https://www.oracle.com/database/technologies/appdev/sql.html (дата обращения: 07.06.2021).

[11] The official home of the Python Programming Language. [Электронный ресурс]. Режим доступа: https://www.python.org/ (дата обращения: 07.06.2021).

[12] Cube, Dimensions, Facts and Measures. [Электронный ресурс]. Режим доступа: https://cubes.readthedocs.io/en/v1.0.1/introduction.html#cube-dimensions-facts-and-measures (дата обращения: 07.06.2021).

[13] Schemas and Models. [Электронный ресурс]. Режим доступа: https://cubes.readthedocs.io/en/v1.0.1/schemas.html (дата обращения: 07.06.2021).

[14] Cubes - OLAP Framework. [Электронный ресурс]. Режим доступа: https://cubes.readthedocs.io/en/v1.0.1/index.html (дата обращения: 07.06.2021).

[15] Requests. [Электронный ресурс]. Режим доступа: https://docs.python-requests.org/en/latest/ (дата обращения: 07.06.2021).

Приложение

Листинг 5.1 – Листинг модуля взаимодействия с OLAP Cube.

```
1  import requests
2  import analyzer.file_converter as fc
3  from itertools import groupby
4  from operator import itemgetter
5  import analyzer.olap_exceptions as olapexc
6  import collections
7
8
9  class OlapData:
10     def __init__(self, url=None, model=None, outputs_fields=None, aggregates=None) ->
           None:
11
12         if url is not None:
13             self.olap_data = requests.get(url).json()
14         elif model is not None:
15             self.olap_data = model
16
17         self.outputs_fields = outputs_fields
18         self.aggregates = aggregates
19         self.__measures = list()
20         for fields in self.outputs_fields:
21             if fields in self.aggregates:
22                 self.__measures.append(fields)
23
24         for measure in self.__measures:
25             self.outputs_fields.remove(measure)
26
27
28     def get_cells(self):
29         return self.olap_data["cells"][:]
30
31
32     def filter_data(self, outputs_fields):
33         model = self.olap_data.copy()
34
35         if outputs_fields:
36             model["cells"] = self.__groupby_data_structs(model["cells"], outputs_fields)
37
38         return OlapData(
39             url=None,
40             model=model,
41             outputs_fields=self.outputs_fields + self.__measures,
42             aggregates=self.aggregates
```

```
        )


    def __groupby_data_structs(self, data, sequence_keys):
        list_structs = list()

        data = sorted(data, key=itemgetter(*sequence_keys))
        for keys, values in groupby(data, key = itemgetter(*sequence_keys)):
            if not isinstance(keys, collections.Iterable) or isinstance(keys, str):
                keys = [keys]

            struct = self.__create_struct(list(sequence_keys), tuple(keys))
            self.__sum_measures(struct, values)
            list_structs.append(struct)

        return list_structs


    def __create_struct(self, keys, values):
        struct = {}

        for i in range(len(keys)):
            struct[keys[i]] = values[i]
        for measure in self.__measures:
            struct[measure] = 0

        return struct


    def __sum_measures(self, struct, data):
        for item in data:
            for measure in self.__measures:
                struct[measure] += item[measure]


    def json(self):
        return self.olap_data


    def table(self):
        data = self.olap_data["cells"]
        result_data = list()

        if data:
            list_headers = list(data[0].keys())
            result_data.append(list_headers)

        for item in data:
```

```
91              temp = [item[header] for header in list_headers]
92              result_data.append(temp)
93
94
95          return result_data
96
97
98  class OlapModel:
99
100     def __init__(self, data_model_settings):
101         self.data_settings = None
102         self.aggregations_field = None
103         self.connect(data_model_settings)
104
105
106     def connect(self, data_model_settings):
107         self.data_settings = data_model_settings
108         self.aggregations_field = self.data_settings["aggregates"]
109
110
111     def close(self):
112         self.data = None
113         self.aggregations_field = None
114
115
116     def aggregations(self, dict_cuts_options, list_outputs_fields) -> OlapData:
117         if not list_outputs_fields:
118             raise olapexc.OlapInputDataException(
119                 "The␣list␣list_outputs_fields␣is␣empty!" +
120                 "␣The␣list␣list_outputs_fields␣must␣contain␣at␣least␣1␣output␣field"
121             )
122
123         cuts_url = self.get_cuts(dict_cuts_options)
124         drilldown_url = self.get_drilldown(list_outputs_fields, dict_cuts_options)
125         url = self.__get_aggregations_model_url(cuts_url, drilldown_url)
126
127         return OlapData(
128             url=url,
129             model=None,
130             outputs_fields=list_outputs_fields,
131             aggregates=self.aggregations_field
132         )
133
134
135     def __get_base_url(self):
136         return self.data_settings["metadata"]["base_url"]
137
138
```

```
139     def __get_aggregations_model_url(self, cuts_url:str=None, drilldown_url:str=None):
140         base_url_aggregations = self.__get_base_url() + "/aggregate?"
141         url_options = self.__get_url_options_model(cuts_url, drilldown_url)
142         return base_url_aggregations + url_options
143 
144 
145     def __get_url_options_model(self, cuts_url=None, drilldown_url=None):
146         url_cuts = "&" + cuts_url if cuts_url else ""
147         url_drilldown = "&" + drilldown_url if drilldown_url else ""
148 
149         return url_cuts + url_drilldown
150 
151 
152     def get_cuts(self, dict_cuts):
153         list_cuts = self.__create_list_cuts(self.data_settings["dimensions"], dict_cuts)
154         return self.__get_url_format_cuts(list_cuts)
155 
156 
157     def __get_url_format_cuts(self, list_cuts: list) -> str:
158         return "cut=" + "|".join(list_cuts) if list_cuts else ""
159 
160 
161     def __create_list_cuts(self, dimensions_list, dict_cuts):
162         list_cuts = list()
163 
164         for dim in dimensions_list:
165             fields_curr_dim = dim["fields"]
166 
167             for field in fields_curr_dim:
168                 hierarchy_val = self.__create_hierarchy_val_url(field,
                        dict_cuts[dim["name"]])
169 
170                 if hierarchy_val:
171                     list_cuts.append(dim["name"] + hierarchy_val)
172 
173         return list_cuts
174 
175 
176     def __create_hierarchy_val_url(self, dict_field, list_dicts_fields):
177         for item in list_dicts_fields:
178             if item["name"] == dict_field["name"]:
179                 return "@" + dict_field["hierarchy"] + ":" +
                        self.__format_url_value(str(item["value"]))
180 
181         return ""
182 
183 
184     def __format_url_value(self, str_value):
```

```
185         parts = str_value.split()
186         return "+".join(parts)
187 
188 
189     def get_drilldown(self, list_field, dict_cuts):
190         dict_all_fields = self.__create_dict_all_field(list_field, dict_cuts)
191         list_drilldowns_pair = self.__create_list_drilldowns_pair(dict_all_fields)
192         return self.__get_url_format_drilldown(list_drilldowns_pair)
193 
194 
195     def __create_dict_all_field(self, list_field, dict_cuts):
196         # add field from dict cuts field
197         dict_all_fields = {key : [item["name"].split('.')[1] for item in value] for key,
                value in dict_cuts.items()}
198 
199         # add field from list outputs field
200         for field in list_field:
201             if field not in self.aggregations_field:
202                 parts_field = field.split('.')
203                 dict_all_fields[parts_field[0]].append(parts_field[1])
204 
205         # leave only unique fields
206         dict_all_fields = {key : list(set(value)) for key, value in
                dict_all_fields.items()}
207 
208         return dict_all_fields
209 
210 
211     def __create_list_drilldowns_pair(self, dict_all_fields):
212         # the list_drilldowns_pair must contain sublists containing two elements: axis
                and last level
213         list_drilldowns_pair = list()
214         # the list_drilldowns_pair must contain dicts containing struct:
215         # {"name" : str, fields: list, "base_drilldown" : str, "levels" : list }
216         list_dims_model = self.data_settings["dimensions"]
217 
218         for dim in list_dims_model:
219             key, reverse_levels = dim["name"], dim["levels"][::-1],
220             for level in reverse_levels:
221                 if level in dict_all_fields[key]:
222                     list_drilldowns_pair.append([key, level])
223                     break
224 
225         return list_drilldowns_pair
226 
227 
228     def __get_url_format_drilldown(self, list_drilldowns_pair):
229         result_url = "drilldown="
```

```
230         # the list contains lines that characterize the axis and the required last level
                for this axis
231         list_pair = [":".join(pair) for pair in list_drilldowns_pair]
232         # return combining all necessary axes into one request for drilldown
233         return result_url + "|".join(list_pair) if list_pair else ""
234
235
236 def connect_olap_model(filename_model: str) -> OlapModel:
237     model = fc.get_json_from_file(filename_model)
238     cube = OlapModel(model)
239     return cube
```

Листинг 5.2 – Листинг модуля взаимодействия с базой данных при работе с пользователями и комментариями.

```
1  import psycopg2 as pg
2  from datetime import datetime
3
4  class AbstractDataBase:
5      def __init__(self, dbname=None, user=None, password=None, host=None) -> None:
6          check = dbname is None or user is None or password is None or host is None
7          if check: raise ValueError()
8          self.__dbname = dbname
9          self.__user = user
10         self.__password = password
11         self.__host = host
12         self.conn = None
13         self.cursor = None
14
15
16     def connect(self, dbname=None, user=None, password=None, host=None):
17         self.conn = pg.connect(
18             dbname=self.__dbname if dbname is None else dbname,
19             user=self.__user if user is None else user,
20             password=self.__password if password is None else password,
21             host=self.__host if host is None else host
22         )
23         self.cursor = self.conn.cursor()
24
25
26     def close(self):
27         self.cursor.close()
28         self.conn.close()
29
30
31 class OlympicDataBase(AbstractDataBase):
32     def __init__(self, dbname=None, user=None, password=None, host=None) -> None:
33         super().__init__(dbname=dbname, user=user, password=password, host=host)
```

```
34
35      def get_commets(self):
36          list_data = list()
37          self.cursor.execute(
38              'select␣comments.id,␣auth_user.username,␣comments.content,␣' +
39              'comments.date_comment,␣comments.time_comment␣' +
40              'from␣comments␣' +
41              'join␣auth_user␣on␣auth_user.id␣=␣comments.id_user␣' +
42              'order␣by␣comments.date_comment␣desc,␣comments.time_comment␣desc;'
43          )
44
45          for row in self.cursor:
46              list_data.append({
47                  'id' : row[0],
48                  "username" : row[1],
49                  "content" : row[2],
50                  "date" : row[3],
51                  "time" : row[4]
52              })
53
54          return list_data
55
56      def append_comment(self, user_name, text):
57          date = datetime.now()
58          day = "-".join(list(map(str, [date.day, date.month, date.year])))
59          time = ":".join(list(map(str, [date.hour, date.minute, date.second])))
60          print(day, time)
61
62          self.cursor.execute(
63              "select␣auth_user.id␣" +
64              "from␣auth_user␣" +
65              "where␣auth_user.username␣=␣%s;", (str(user_name),)
66          )
67
68          print(self.cursor.query)
69
70          user_id = self.cursor.fetchall()[0][0]
71
72          self.cursor.execute(
73              "INSERT␣INTO␣comments␣(id_user,␣content)␣" +
74              "VALUES␣(%s,␣%s);", (str(user_id), str(text))
75          )
76          self.conn.commit()
77
78
79      def delete_comment(self, id_comment):
80          self.cursor.execute(
81              "DELETE␣FROM␣comments␣" +
```

```
            "where id= %s;", (str(id_comment))
        )
        self.conn.commit()


    def users_info(self):
        list_data = list()

        self.cursor.execute(
            "SELECT id, last_login, username, email, " +
            "is_staff, is_active, date_joined " +
            "from auth_user " +
            "where is_superuser is false;"
        )

        for row in self.cursor:
            list_data.append({
                'id' : row[0],
                "last_login" : row[1],
                "username" : row[2],
                "email" : row[3],
                "is_staf" : row[4],
                "is_active" : row[5],
                "date_joined" : row[6]
            })

        return list_data


    def delete_user(self, id):
        self.cursor.execute(
            "DELETE FROM auth_user " +
            "where id= %s;", (str(id))
        )
        self.conn.commit()
```

Листинг 5.3 – Инициализация базы данных.

```
CREATE TABLE games(
    id INTEGER NOT NULL,
    city VARCHAR,
    year INTEGER NOT NULL CHECK(year > 1895),
    season VARCHAR CHECK(season = 'Summer' OR season = 'Winter'),
    PRIMARY KEY(id),
    UNIQUE(id)
);


CREATE TABLE nation(
```

```
    id INTEGER NOT NULL,
    noc VARCHAR,
    namecountry VARCHAR,
    capital VARCHAR,
    population INTEGER,
    region VARCHAR,
    PRIMARY KEY(id)
);


CREATE TABLE athletes(
    id INTEGER NOT NULL UNIQUE,
    name VARCHAR NOT NULL,
    sex VARCHAR(1) NOT NULL,
    age INTEGER,
    height INTEGER,
    weight NUMERIC(4, 1),
    id_noc INTEGER NOT NULL,
    sport VARCHAR NOT NULL,
    PRIMARY KEY(id),
    FOREIGN KEY(id_noc) REFERENCES nation(id)
);


CREATE TABLE sport(
    id INTEGER NOT NULL,
    name_sport VARCHAR,
    PRIMARY KEY(id)
);


CREATE TABLE event(
    id INTEGER NOT NULL,
    name_event VARCHAR,
    id_sport INTEGER NOT NULL,
    PRIMARY KEY(id),
    FOREIGN KEY(id_sport) REFERENCES sport(id)
);


CREATE TABLE medal(
    id INTEGER NOT NULL,
    id_game INTEGER NOT NULL,
    sport INTEGER NOT NULL,
    event INTEGER NOT NULL,
    gold_m INTEGER NOT NULL CHECK(gold_m < 2),
    silver_m INTEGER NOT NULL CHECK(silver_m < 2),
    bronze_m INTEGER NOT NULL CHECK(bronze_m < 2),
```

```
    FOREIGN KEY(id) REFERENCES athletes(id),
    FOREIGN KEY(id_game) REFERENCES games(id),
    FOREIGN KEY(sport) REFERENCES sport(id),
    FOREIGN KEY(event) REFERENCES event(id)
);


CREATE TABLE comments(
    id SERIAL,
    id_user INTEGER NOT NULL,
    content TEXT NOT NULL,
    date_comment DATE,
    time_comment TIME,
    PRIMARY KEY(id),
    FOREIGN KEY(id_user) REFERENCES auth_user(id),
    UNIQUE(id)
);


CREATE TABLE buffer_comments(
    id SERIAL,
    id_user INTEGER NOT NULL,
    content TEXT NOT NULL,
    date_comment DATE,
    time_comment TIME,
    PRIMARY KEY(id),
    FOREIGN KEY(id_user) REFERENCES auth_user(id),
    UNIQUE(id)
);


create or replace function logs_buffer_comment()
returns trigger
language plpgsql as
$$
begin
    new.date_comment := current_date;
    new.time_comment := localtime;
    insert into buffer_comments(id_user, content, date_comment, time_comment)
    values(new.id_user, new.content, new.date_comment, new.time_comment);
    return new;
end;
$$;


create trigger insert_comment before insert
on comments
for each row
```

```
execute procedure logs_buffer_comment();


create or replace function delete_user_comments()
returns trigger
language plpgsql as
$$
begin
    delete from comments
    where id_user = old.id;

    delete from buffer_comments
    where id_user = old.id;

    return old;
end;
$$;


create trigger delete_user before delete
on auth_user
for each row
execute procedure delete_user_comments();

create user user_serv with password 'admin';
create user reader with password 'reader';

grant select on athletes, games, nation, medal, sport, event to reader;
REVOKE ALL ON auth_user, comments, buffer_comments event from reader;

grant select, insert, delete on auth_user, comments, buffer_comments to user_serv;
REVOKE ALL ON athletes, games, nation, medal, sport, event from user_serv;
```