Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

Пояснительная записка

к курсовому проекту

на тему

**РЕАЛИЗАЦИЯ АРХИТЕКТУРЫ БД ДЛЯ АНАЛИЗА СПОРТИВНЫХ ДАННЫХ.**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Студент |  | Ю. П. Плахотникова |
| Руководитель |  | А. В. Давыдчик |

Минск 2022

Министерство образования Республики Беларусь

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Учреждение образования БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ | | | | | | | | | |
|  | | | | | | | | | |
| Факультет | КС и С | | | | Кафедра | | Информатики | | |
| Специальность | 1-40 04 01 | | | | Специализация | | |  | |
| ЗАДАНИЕ | | | | | | | | | |
| по курсовому проекту студента | | | | | | | | | |
| **Плахотниковой Юлии Павловны** | | | | | | | | | |
| (фамилия, имя, отчество) | | | | | | | | | |
| 1. Тема проекта: | | **Реализация архитектуры БД для анализа данных** | | | | | | | |
|  | | | | | | | | | |
|  | | | | | | | | | |
|  | | | | | | | | | |
| 2. Срок сдачи студентом законченной работы | | | | | |  | | | |
| 3. Исходные данные к проекту | | | | Тип операционной системы – ОС Windows; | | | | | |
| Языки программирования – SQL, PL/SQL, Python; Библиотеки: pandas | | | | | | | | | |
|  | | | | | | | | | |
| Цель проекта: реализация архитектуры БД, которая может быть использована для анализа | | | | | | | | | |
| спортивных данных. | | | | | | | | | |
|  | | | | | | | | | |
| 4. Содержание пояснительной записки (перечень подлежащих разработке вопросов) | | | | | | | | | |
| Введение | | | | | | | | | |
| 1 Формирование требований к проекту | | | | | | | | | |
| 2 Основная идея реализации требований | | | | | | | | | |
| 3 Проектирование проекта | | | | | | | | | |
| 4 Программная реализация | | | | | | | | | |
| 5 Тестирование проекта | | | | | | | | | |
| Заключение | | | | | | | | | |
| Список использованных источников | | | | | | | | | |
| Приложение А - Текст программы | | | | | | | | | |
| 5. Перечень графического материала (с точным указанием наименования) и обозначения | | | | | | | | |
| вида и типа материала) | | |  | | | | | |
| Общая схема архитектуры проекта | | | | | | | | |
| Схема устройства кэша базы данных | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |

**КАЛЕНДАРНЫЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапов дипломного проекта (работы) | Объём этапа в % | Срок выполнения этапа | Примечание |
| Анализ предметной |  |  |  |
| области, разработка технического задания | 15-20 | 11.09–15.10 |  |
| Разработка функциональных требований, |  |  |  |
| проектирование архитектуры программы | 20-15 | 16.10–15.11 |  |
| Разработка схемы программы, алгоритмов, |  |  |  |
| схемы данных | 20-15 | 16.11–15.12 |  |
| Разработка программного средства | 15-20 | 16.12– (в зависимости от курса) |  |
| Тестирование и отладка | 10 | 5 дней до сдачи |  |
| Оформление пояснительной записки |  |  |  |
| и графического материала | 20 | 2 дня до сдачи |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Дата выдачи задания |  | | Руководитель | |  | А.В. Давыдчик |
| Задание принял к исполнению | |  | | Ю.П. Плахотникова | |

**ВВЕДЕНИЕ**

В наше время огромное количество фирм используют персональные компьютеры для сохранения и обработки любого вида информации. Эта информация содержится в базах данных. Базы данных играют важную роль в развивающемся мире технологий. Всё, с чем мы каждый день взаимодействуем в жизни, по всей видимости, зафиксировано в какой-нибудь базе. Работа с базами данных является важнейшим навыком в работе с компьютером, а специалисты данной области становятся всё более востребованными. Главные идеи нынешней информационной методики базируются на представлении, в соответствии чему информация должна быть образована в базы данных с задачей отображения динамически изменяющегося мира и удовлетворения всех потребностей в информации у пользователей. Базы данных формируются и работают под управлением специальных программных средств, называемых системами управления базами данных.

База данных, которая представлена в объективной форме, это совокупность таких материалов: статей, счетов, нормативных актов, судебных решений или иных подобных материалов, собранные вместе таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины.

База данных — это организованная структура, которая предназначена для хранения информации. В то время, когда происходило развитие термина баз данных, в них сохранялись исключительно информация, однако уже в наши дни многие системы управления базами данных позволяет размещать в своих структурах и данные, и программный код, с помощью которого совершается связь с пользователями или с другими программно-аппаратными комплексами. При этом данные должны не противоречить друг другу, целостны и не избыточными. База данных создается для сохранения и непосредственного доступа к информации, содержащей сведения об искомой предметной области. Степень конкретизации данных обуславливается группой факторов. Прежде всего, целью использования информации из баз данных и сложностью информационных процессов, существующих в пределах предметной области в конкретных условиях.

Система управления базами данных — это программный механизм, предназначенный для записи, поиска, обработки и печати информации, содержащейся в базе данных.

В компьютере данные базы данных представляется в виде таблицы, схожей на электронную таблицу. Названия столбцов, представляющих заголовки таблицы, называют именами полей, а сами столбцы - полями. Данные, которые находятся в полях, называют значениями полей.

Сами базы данных — это хранилища огромного множества систематизированной информации, с которыми производятся следующие операции: изменение, копирование, удаление, добавление. Накопление хранимого объема информации, рост группы пользователей информационных систем служат источником к обширному развитию самых комфортных в интерфейсе и относительно лёгких для понимания табличных систем управления базами данных. Создание доступа к информации базы данных сразу нескольких пользователей одновременного, зачастую находящихся на далеком расстоянии от места хранения баз данных, а также друг от друга, поэтому и созданы многопользовательские сетевые версии баз данных, сформированные на табличной структуре. В них решаются проблемы характерные для параллельных процессов, правильности данных, а также получения не санкционированного входа.

Любая современная организация не может обойтись без базы данных. Это учебные заведения, банки, магазины, заводы, любые предприятия и государственные учреждения. Они используют их для перевода данных в электронный вид и объединения данных, а также оперативного доступа к ним. Это позволяет экономить время и средства на затраты.

Конечно, снижение времени является лишь побочным эффектом автоматизации. Самая главная задача развития информационных технологий в совсем другом - в приобретении той или иной организацией исключительно новых качеств, придающих ей существенную конкурентоспособность. А это дорогого стоит.

**Формирование требований к проекту**

Целью проекта является реализация архитектуры БД которая может быть использована для анализа спортивных данных, которая может быть интегрирована в сервис по прогнозу спортивных данных.

К курсовому проекту предъявляются следующие требования:

1)Пользователь должен иметь возможность просмотреть список спортивных игроков и их матчи.

2)Пользователь должен иметь возможность просмотреть ближайшие матчи.

3)Пользователь должен иметь возможность просмотреть матчи конкретного игрока

4) Матчи и игроки должны поступать в систему через спортивное апи

5)Матчи должны своевременно обновляться в зависимости от текущей даты

**Проектирование проекта**

Ниже приведена общая схема архитектуры проекта

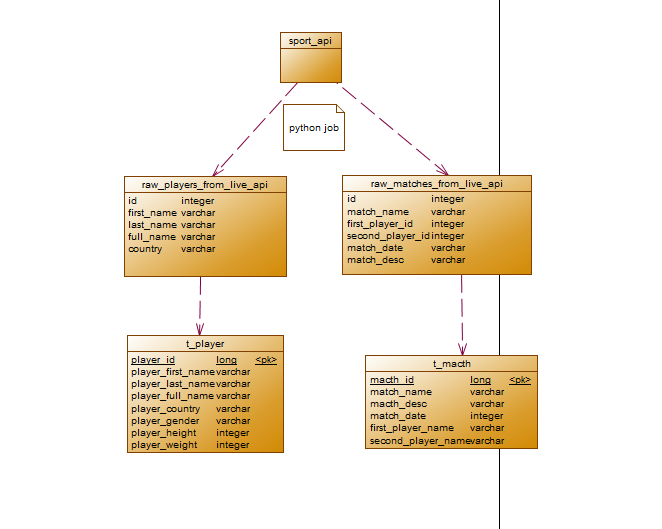


Рис 1. общая схема архитектуры проекта

Дадим пояснение архитектуре проекта: основные сущности — это тип игрок пришедший из апи, игрок в таблице справочнике, матч пришедший из апи, матч в таблице справочнике.

Поля игрока пришедшего из апи:

1) id уникальный номер игрока.

2)first\_name: имя игрока.

3)second\_name: фамилия игрока.

4)full\_name: полное имя игрока.

5)country: страна, за которую играет игрок.

Поля игрока в таблице справочнике:

1)player\_id уникальный номер игрока.

2)player\_first\_name: имя игрока.

3)player\_second\_name: фамилия игрока.

4)player\_full\_name: полное имя игрока.

5)player\_country: страна, за которую играет игрок.

6)player\_gender: пол игрока.

7)player\_weight: вес игрока.

8)player\_height: рост игрока.

Поля матча пришедшего из апи:

1) id: уникальный номер матча.

2) match\_name: название матча.

3) first\_player\_id: номер первого игрока.

4) second\_player\_id: номер второго игрока.

5) match\_date: дата матча.

6) match\_desc: расширенное описание матча.

Поля матча в таблице справочнике:

1) id: уникальный номер матча.

2) match\_name: название матча.

3) first\_player\_name: полное имя первого игрока.

4) second\_player\_name: полное имя второго игрока.

5) match\_date: дата матча.

6) match\_desc: расширенное описание матча.

Будет реализован скрипт по вставке новых игроков и матчей.

Будут реализованы процедуры по выводу матчей, игроков, матчей конкретного игрока.

**Программная реализация**

Для построения архитектуры была выбрана база данных Oracle.

Oracle Database — это объектно-реляционная система управления базами данных (СУБД) от компании Oracle.

На мировом рынке корпоративных систем управления базами данных (СУБД) доминирующее положение занимает традиционная тройка продуктов: IBM DB2, Microsoft SQL Server и Oracle. Более 80% рынка СУБД в течение долгих лет контролируется тремя компаниями производителями: IBM, Oracle и Microsoft. По статистическим данным на рынке России лидирующее положение занимает Oracle, так как по статистическим данным за 2017 год, данная СУБД занимает более 60% всего рынка, среди других СУБД и около 30% мирового рынка СУБД (рис.2).

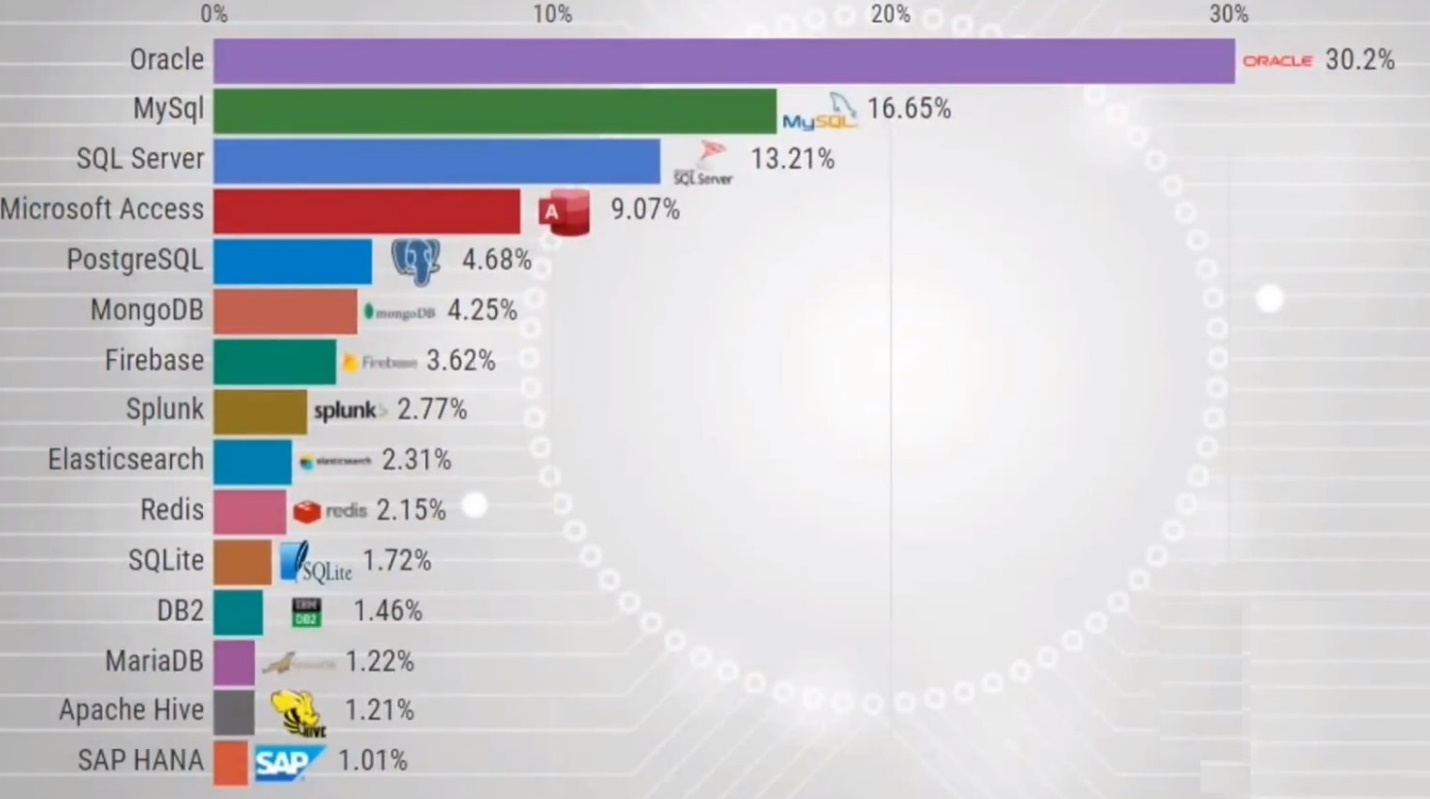


Рис 2. Инфографик популярности СУБД на мировом рынке

На сегодняшний день СУБД Oracle поддерживают свыше 80 вариантов операционной среды в широком диапазоне, включая мэйнфреймы IBM, мини-компьютеры DEC VAX, UNIX, Windows и множество других платформ.

Данная СУБД имеет массу преимуществ.  
  
Oracle поддерживает самые большие базы данных. Большое количество пользователей для этой системы также не помеха. СУБД способна поддерживать любых пользователей, в любом количестве, которые при этом одновременно выполняют разные задачи. В Oracle не происходит соперничества между разными видами данных.  
  
СУБД Oracle хорошо обрабатывает транзакции. Система сохраняет высокую производительность, в результате чего пользователи не страдают от низкой скорости обработки.  
  
Система обладает высокой степенью готовности. В разных установках продолжительность работы Oracle индивидуальная. Так, например, в некоторых, система способна работать круглосуточно. При этом откат БД или какие-либо сбои системы не приводят к остановке работы базы.

Рассмотрим теперь построение архитектуры БД под нашу задачу.

Построение архитектуры будет включать в себя 3 этапа:

1) Создание таблиц

2) Поиск подходящего апи

3) Реализация скрипта по загрузке данных из апи в таблицы

4) Создание процедур, удовлетворяющих требованиям проекта

Рассмотрим первый этап.

На первом этапе были созданы следующие таблицы:

1) t\_player

2) t\_match

3) raw\_players\_from\_live\_api

4) raw\_matches\_from\_live\_api

Таблица t\_player содержит следующие колонки:

1) player\_id с типом number

2) player\_first\_name с типом varchar2(500)

3) player\_last\_name с типом varchar2(500)

4) player\_full\_name с типом varchar2(500)

5) player\_country с типом varchar2(500)

6) player\_gender с типом varchar2(500)

7) player\_height c типом number

8) player\_weight с типом number

Также на колонку player\_id было наложено ограничение первичного ключа.

Таблица t\_match содержит следующие колонки:

1) match\_id с типом number

2) match\_name с типом varchar2(500)

3) match\_desc с типом varchar2(500)

4) match\_date с типом number

5) first\_player\_full\_name с типом varchar2(500)

6) second\_player\_full\_name с типом varchar2(500)

Также на колонку match\_id было наложено ограничение первичного ключа.

Таблица raw\_players\_from\_live\_api содержит следующие колонки:

1) id с типом number.

2)first\_name с типом varchar2(500)

3)second\_name с типом varchar2(500)

4)full\_name с типом varchar2(500)

5)country с типом varchar2(500)

Таблица raw\_matches\_from\_live\_api содержит следующие колонки:

1) id с типом number

2) match\_name с типом varchar2(500)

3) first\_player\_id с типом number

4) second\_player\_id с типом number

5) match\_date с типом varchar2(500)

6) match\_desc с типом varchar2(500)

Рассмотрим второй этап

Для выбора апи был рассмотрен сервис rapidapi. Это удобный сервис, который предлагает большую выборку разнообразных спортивных апи, подходящих практически под любую задачу.

Для нашего задачи было выбрано следующее апи: “Tennis Live Data API” от пользователя “sportcontentapi”. Это апи позволяет выбирать большое количество игроков и матчей за один запрос. Также это апи дает возможность использовать его бесплатно на постоянной основе.

Ниже приведен список основных запросов, которые можно осуществить

используя это апи:

1) GET Matches By Date – позволяет получить все проводящиеся матчи за выбранную дату

2) GETPlayers – позволяет получить всех игроков выбранного турнира

3) GETMatches By Tournament By Player - позволяет получить все матчи выбранного игрока в выбранном турнире

4) GETTournaments – позволяет получить все турниры по выбранному коду

5) GETMatch - позволяет получить информацию о матче по его id

6) GETPlayer - позволяет получить информацию об игроке по его id

Рассмотрим третий этап

Для загрузки данных в базу напишем скрипт на языке python

Изначально нам понадобится библиотека http.client

С её помощью мы получим данные из апи в виде json строки:



Рис 3. Пример json строки которую возвращает апи

Затем мы используем библиотеку json и с её помощью преобразуем json строку в словарь как объект python:

Изображение выглядит как текст

Автоматически созданное описание

Рис 4. Пример преобразованной json строки в словарь

Грамотно пробегаем словарь, берем нужные поля и преобразуем словарь объектов в массив, где каждый элемент — это неизменяемый массив, соответствующий одной строке значений в базе данных.

Изображение выглядит как текст

Автоматически созданное описание

Рис 5. Пример преобразованной словаря в массив элементов

Затем используя библиотеку pandas мы преобразуем наш массив объектов в dataFrame:

Изображение выглядит как текст

Автоматически созданное описание

Рис 6. Пример pandas dataFrame

Далее используя библиотеку cx\_oracle мы подключаемся к базе, создаем таблицу и вставляем в неё данные, затем делаем выборку в таблице из базы и проверяем, что она заполнилась данными.

Изображение выглядит как текст

Автоматически созданное описание

Рис.7 Подключение к базе и создание таблицы

Изображение выглядит как текст

Автоматически созданное описание

Рис.8 Подключение к базе и вставка данных в таблицу

Изображение выглядит как текст

Автоматически созданное описание

Рис.9 Выборка данных из таблицы

Рассмотрим четвертый этап.

1) SP\_GET\_PLAYERS - принимает на вход два параметра p\_user\_id – id пользователя и p\_results – курсор, в который она отдаст результат. Возвращает спортивных игроков, которые есть на сервисе. Нужна для того, чтобы пользователь мог увидеть, какие игроки есть на сервисе.

2) SP\_GET\_MATCHES - принимает на вход два параметра p\_user\_id – id пользователя и p\_results – курсор, в который она отдаст результат. Возвращает спортивные матчи за текущую дату. В качестве дополнительной информации эта процедура выводит был ли сделан пользователем прогноз на данный матч или нет, чтобы фильтровать матчи, на который уже был сделал прогноз. Нужна для того, чтобы пользователь мог увидеть, на какие матчи можно сделать прогноз на сервисе.

3) SP\_GET\_MATCHES\_BY\_NAME - принимает на вход два параметра p\_match\_name – имя матча и p\_results – курсор, в который она отдаст результат. Возвращает спортивный матч за текущую дату, который совпадает с переданным параметром p\_match\_name. Нужна для того, чтобы пользователь мог выбрать один матч и затем подтвердить что он хочет спрогнозировать его результат.

4) SP\_GET\_MATCHES\_BY\_PLAYER\_ID - принимает на вход два параметра p\_player\_id – id игрока и p\_results – курсор, в который она отдаст результат. Возвращает спортивные матчи за текущую дату, в которых участвует игрок с id равным p\_player\_id. Нужна для того, чтобы пользователь мог выбрать все матчи пользователя, на которого он кликнул.

**Тестирование проекта**

Проверим работу написанных процедур.

Тест № 1:

Выполним на сервисе вызов процедуры SP\_GET\_PLAYERS

cursor.callproc("SP\_GET\_PLAYERS ", (1, ref\_cursor))

Ожидаем увидеть всех игроков, которые есть на сервисе.

Проверяем результат и видим следующее.

Изображение выглядит как стол

Автоматически созданное описание

Рис 10. Первые строки в результате вызова процедуры SP\_GET\_PLAYERS

Изображение выглядит как текст

Автоматически созданное описание

Рис 11. Количество строк, вернувшихся в результате вызова процедуры SP\_GET\_PLAYERS

Видим, что нам вернулись верные данные и верное число строк.

Тест № 1 пройден.

Тест № 2:

Выполним на сервисе вызов процедуры SP\_GET\_MATCHES

cursor.callproc("SP\_GET\_MATCHES ", (1, ref\_cursor))

Ожидаем увидеть все матчи за текущую дату, которые есть на сервисе.

Проверяем результат и видим следующее.

Изображение выглядит как стол

Автоматически созданное описание

Рис 12. Первые строки в результате вызова процедуры SP\_GET\_MATCHES

Изображение выглядит как текст

Автоматически созданное описание

Рис 13. Количество строк, вернувшихся в результате вызова процедуры SP\_GET\_MATCHES

Видим, что нам вернулись верные данные и верное число строк.

Тест № 2 пройден.

Тест № 3:

Выполним на сервисе вызов процедуры SP\_GET\_MATCHES\_BY\_NAME

cursor.callproc("SP\_GET\_MATCHES\_BY\_NAME ", ‘Molcan A. - Rinderknech A.', ref\_cursor)

Ожидаем увидеть все матчи за текущую дату, которые есть на сервисе.

Проверяем результат и видим следующее.

Изображение выглядит как текст

Автоматически созданное описание

Рис 14. Строка в результате вызова процедуры SP\_GET\_MATCHES

Видим, что нам вернулись верные данные.

Тест № 3 пройден.

Тест № 4:

Выполним на сервисе вызов процедуры SP\_GET\_MATCHES\_BY\_PLAYER\_ID

cursor.callproc("SP\_GET\_MATCHES\_BY\_PLAYER\_ID ", 1207446, ref\_cursor)

Ожидаем увидеть все матчи за текущую дату с заданным именем игрока.

Проверяем результат и видим следующее.

Изображение выглядит как стол

Автоматически созданное описание

Рис 15. Строка в результате вызова процедуры SP\_GET\_MATCHES\_BY\_PLAYER\_ID

Видим, что нам вернулись верные данные.

Тест № 4 пройден.

**Заключение**

В результате работы над курсовом проектом была создана архитектура базы данных для работы со спортивными данными.

Был произведен анализ бизнес-требований сервиса, бизнес-требования сервиса были преобразованы в конкретный технический замысел.

Были созданы таблицы, подходящие техническим требованиям сервиса.

Был написан скрипт, забирающий данные из апи и переносящий их внутрь базы данных.

Наконец был написан ряд процедур, которые на основе данных, загруженных в таблицы, отдавали необходимый их задумке результат в сервис.

Данная архитектура была вручную протестирована в разделе тестирование проекта.

Результаты всех тестов оказались удовлетворительными.

На основе результатов тестов можно утверждать, что построение архитектуры прошло успешно и полностью удовлетворяет бизнес-требованиям.

В качестве перспектив дальнейшего развития курсового проекта можно отметить следующие.

Во-первых, можно составить расписание, в рамках которого будет автоматически скрипт, который будет забирать актуальные данные из апи и переносить их в базу данных, чтобы не делать этого вручную.

Во-вторых, стоит подумать о расширении функционала путем написания новых процедур на основе текущей архитектуры.

**Список использованных источников**

1. Создание базы данных «Больница» [Электронный ресурс]. – 2013. – Режим доступа: <https://www.skachatreferat.ru/referaty/%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%91%D0%B0%D0%B7%D1%8B-%D0%94%D0%B0%D0%BD%D0%BD%D1%8B%D1%85-%D0%91%D0%BE%D0%BB%D1%8C%D0%BD%D0%B8%D1%86%D0%B0/294423.html> Дата доступа: 29.10.2022

2. Область применения баз данных [Электронный ресурс]. – 2016. – Режим доступа: <https://scienceforum.ru/2016/article/2016020613/> Дата доступа: 29.10.2022

3. Простая транзакция базы данных Oracle [Электронный ресурс]. – 2014. – Режим доступа: <https://oracle-dba.ru/docs/architecture/transactions/simple-transaction/> Дата доступа: 30.10.2022

4. Архитектура СУБД Oracle [Электронный ресурс]. – 2017. – Режим доступа: <https://oracle-patches.com/oracle/begin/%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0-%D1%81%D1%83%D0%B1%D0%B4-oracle> Дата доступа: 31.10.2022

5. Область применения баз данных [Электронный ресурс]. – 2014. – Режим доступа: <http://ora-sql.ru/arxitektura-pamyati-oracle.html> Дата доступа: 28.10.2022

**Приложение А**

**(обязательное)**

**Исходный код**

drop table t\_player;

create table t\_player (

player\_id number,

player\_first\_name varchar2(500),

player\_last\_name varchar2(500),

player\_full\_name varchar2(500),

player\_country varchar2(500),

player\_gender varchar2(500),

player\_height number,

player\_weight number,

constraint pk\_player\_id primary key (player\_id)

);

drop table t\_match

create table t\_match (

match\_id number,

match\_name varchar2(500),

match\_desc varchar2(500),

match\_date number,

first\_player\_full\_name varchar2(500),

second\_player\_full\_name varchar2(500),

constraint pk\_match\_id primary key (match\_id)

);

CREATE OR REPLACE PROCEDURE SP\_GET\_PLAYERS(player\_id number, p\_results OUT SYS\_REFCURSOR)

AS

BEGIN

OPEN p\_results FOR

SELECT

player\_id,

player\_first\_name,

player\_last\_name,

player\_full\_name,

player\_country,

player\_gender,

player\_height,

player\_weight

FROM t\_player

ORDER BY player\_first\_name ASC,

player\_last\_name ASC;

END;

/

CREATE OR REPLACE PROCEDURE SP\_GET\_MATCHES(p\_user\_id number, p\_results OUT SYS\_REFCURSOR)

AS

BEGIN

OPEN p\_results FOR

SELECT

match\_id,

t.match\_name,

match\_desc,

match\_date,

first\_player\_full\_name,

second\_player\_full\_name,

nvl(tp.PREDICTION\_ID, 0) as IS\_PREDICTED

FROM t\_match t

LEFT JOIN t\_prediction tp on tp.user\_id = p\_user\_id and tp.match\_name = t.match\_name

WHERE match\_date = (SELECT MAX(match\_date) FROM t\_match);

END;

/

CREATE OR REPLACE PROCEDURE SP\_GET\_MATCHES\_BY\_NAME(p\_match\_name varchar2, p\_results OUT SYS\_REFCURSOR)

AS

BEGIN

OPEN p\_results FOR

SELECT

match\_id,

match\_name,

match\_desc,

match\_date,

first\_player\_full\_name,

second\_player\_full\_name

FROM t\_match

WHERE match\_name = p\_match\_name;

END;

/

CREATE OR REPLACE PROCEDURE SP\_GET\_MATCHES\_BY\_PLAYER\_ID(p\_player\_id number, p\_results OUT SYS\_REFCURSOR)

AS

BEGIN

OPEN p\_results FOR

SELECT

match\_id,

match\_name,

match\_desc,

match\_date,

first\_player\_full\_name,

second\_player\_full\_name

FROM t\_match

JOIN t\_player on first\_player\_full\_name = player\_full\_name or second\_player\_full\_name = player\_full\_name

WHERE player\_id = p\_player\_id;

END;

/

import http.client

conn = http.client.HTTPSConnection("tennis-live-data.p.rapidapi.com")

headers = {   'X-RapidAPI-Key': "57fcc65541msh4841dc21fef332dp1f0486jsn7e8c59408325",

    'X-RapidAPI-Host': "tennis-live-data.p.rapidapi.com"

    }

conn.request("GET", "/players/ATP", headers=headers)

res = conn.getresponse()

data = res.read()

print(data.decode("utf-8"))

import json

dictData = json.loads(data.decode("utf-8"))

print(dictData)

DataList = []

for i in dictData['results']['players']:

    DataList.append((i['id'], i['first\_name'], i['last\_name'], i['full\_name'], i['country']))

print(DataList)

import pandas as pd

pd.\_\_version\_\_

import pandas as pd

column\_names=["id","first\_name","last\_name", "full\_name", "country"]

df=pd.DataFrame(DataList,columns=column\_names)

print(df.head)

import cx\_Oracle

from cx\_Oracle import DatabaseError

connection = cx\_Oracle.connect(user="system", password="ADMIN",

                               dsn="192.168.56.1:1521/xepdb1",

                               encoding="UTF-8")

if conn:

    print("cx\_Oracle version:", cx\_Oracle.version)

    print("Database version:", cx\_Oracle.version)

    print("Client version:", cx\_Oracle.clientversion())

pip install cx\_Oracle

import cx\_Oracle as orcCon

from cx\_Oracle import DatabaseError

try:

    conn = orcCon.connect(user="system", password="ADMIN",

                               dsn="192.168.56.1:1521/xe",

                               encoding="UTF-8")

    if conn:

        print("cx\_Oracle version:", orcCon.version)

        print("Database version:", conn.version)

        print("Client version:", orcCon.clientversion())

        # Now execute the sqlquery

        cursor = conn.cursor()

        print("You're connected.................")

        # Drop table if exists

        print('Droping raw\_players\_from\_live\_api table if exists............')

        cursor.execute("BEGIN EXECUTE IMMEDIATE 'DROP TABLE raw\_players\_from\_live\_api'; EXCEPTION WHEN OTHERS THEN NULL; END;")

        print('Creating table raw\_players\_from\_live\_api............')

        cursor.execute("CREATE TABLE raw\_players\_from\_live\_api (id number, first\_name varchar(500), last\_name varchar(500), full\_name varchar(500),country varchar(500))")

        print("raw\_players\_from\_live\_api table is created..............")

except DatabaseError as e:

    err, = e.args

    print("Oracle-Error-Code:", err.code)

    print("Oracle-Error-Message:", err.message)

finally:

    cursor.close()

    conn.close()

import cx\_Oracle as orcCon

from cx\_Oracle import DatabaseError

try:

    #orcCon.connect('username/password@localhost')

    conn = orcCon.connect(user="system", password="ADMIN",

                               dsn="192.168.56.1:1521/xe",

                               encoding="UTF-8")

    if conn:

        print("cx\_Oracle version:", orcCon.version)

        print("Database version:", conn.version)

        print("Client version:", orcCon.clientversion())

        cursor = conn.cursor()

        print("You're connected: ")

        print('Inserting data into table....')

        for i,row in df.iterrows():

            sql = "INSERT INTO raw\_players\_from\_live\_api(id,first\_name,last\_name,full\_name,country) VALUES(:1,:2,:3,:4,:5)"

            cursor.execute(sql, tuple(row))

        # the connection is not autocommitted by default, so we must commit to save our changes

        conn.commit()

        print("Record inserted succesfully")

except DatabaseError as e:

    err, = e.args

    print("Oracle-Error-Code:", err.code)

    print("Oracle-Error-Message:", err.message)

finally:

    cursor.close()

    conn.close()

conn = orcCon.connect(user="system", password="ADMIN",

                               dsn="192.168.56.1:1521/xe",

                               encoding="UTF-8")

cursor = conn.cursor()

# Execute query

sql = "SELECT \* FROM raw\_players\_from\_live\_api"

cursor.execute(sql)

# Fetch all the records

result = cursor.fetchall()

for i in result:

    print(i)

cursor.close()

conn.close()

import http.client

conn = http.client.HTTPSConnection("tennis-live-data.p.rapidapi.com")

headers = {  'X-RapidAPI-Key': "57fcc65541msh4841dc21fef332dp1f0486jsn7e8c59408325",

    'X-RapidAPI-Host': "tennis-live-data.p.rapidapi.com"

    }

conn.request("GET", "/matches-by-date/2022-10-26", headers=headers)

res = conn.getresponse()

data\_match = res.read()

print(data\_match.decode("utf-8"))

from datetime import datetime

datetime.today().strftime('%Y-%m-%d')

import http.client

conn = http.client.HTTPSConnection("tennis-live-data.p.rapidapi.com")

headers = {   'X-RapidAPI-Key': "57fcc65541msh4841dc21fef332dp1f0486jsn7e8c59408325",

    'X-RapidAPI-Host': "tennis-live-data.p.rapidapi.com"

req\_str = "/matches-by-date/" + datetime.today().strftime('%Y-%m-%d')

conn.request("GET", req\_str , headers=headers)

res = conn.getresponse()

data\_match\_example = res.read()

print(data\_match\_example.decode("utf-8"))

import json

dictDataMatch = json.loads(data\_match\_example.decode("utf-8"))

print(dictDataMatch)

DataMatchList = []

for j in range(0, len(dictDataMatch['results'])):

    for i in dictDataMatch['results'][j]['matches']:

        DataMatchList.append((i['id'], i['title'], i['home\_id'], i['away\_id'], i['date'], dictDataMatch['results'][j]['tournament']['name']+ ' , ' +i['round\_name']))

print(DataMatchList)

import pandas as pd

pd.\_\_version\_\_

import pandas as pd

column\_names=["id","match\_name","first\_player\_id", "second\_player\_id", "match\_date", "match\_desc"]

df=pd.DataFrame(DataMatchList,columns=column\_names)

print(df.head)

import cx\_Oracle as orcCon

from cx\_Oracle import DatabaseError

try:

    conn = orcCon.connect(user="system", password="ADMIN",

                               dsn="192.168.56.1:1521/xe",

                               encoding="UTF-8")

    if conn:

        print("cx\_Oracle version:", orcCon.version)

        print("Database version:", conn.version)

        print("Client version:", orcCon.clientversion())

        # Now execute the sqlquery

        cursor = conn.cursor()

        print("You're connected.................")

        # Drop table if exists

        print('Droping raw\_players\_from\_live\_api table if exists............')

        cursor.execute("BEGIN EXECUTE IMMEDIATE 'DROP TABLE raw\_matches\_from\_live\_api'; EXCEPTION WHEN OTHERS THEN NULL; END;")

        print('Creating table raw\_players\_from\_live\_api............')

        cursor.execute("CREATE TABLE raw\_matches\_from\_live\_api (id number, match\_name varchar(500), first\_player\_id number, second\_player\_id number, match\_date varchar(500),match\_desc varchar(500))")

        print("raw\_players\_from\_live\_api table is created..............")

except DatabaseError as e:

    err, = e.args

    print("Oracle-Error-Code:", err.code)

    print("Oracle-Error-Message:", err.message)

finally:

    cursor.close()

    conn.close()

import cx\_Oracle as orcCon

from cx\_Oracle import DatabaseError

try:

    #orcCon.connect('username/password@localhost')

    conn = orcCon.connect(user="system", password="ADMIN",

                               dsn="192.168.56.1:1521/xe",

                               encoding="UTF-8")

    if conn:

        print("cx\_Oracle version:", orcCon.version)

        print("Database version:", conn.version)

        print("Client version:", orcCon.clientversion())

        cursor = conn.cursor()

        print("You're connected: ")

        print('Inserting data into table....')

        for i,row in df.iterrows():

            sql = "INSERT INTO raw\_matches\_from\_live\_api(id,match\_name,first\_player\_id,second\_player\_id,match\_date, match\_desc) VALUES(:1,:2,:3,:4,:5, :6)"

            cursor.execute(sql, tuple(row))

        # the connection is not autocommitted by default, so we must commit to save our changes

        conn.commit()

        print("Record inserted succesfully")

except DatabaseError as e:

    err, = e.args

    print("Oracle-Error-Code:", err.code)

    print("Oracle-Error-Message:", err.message)

finally:

    cursor.close()

    conn.close()