**Оглавление**

[Общая постановка задачи 4](#_Toc102755442)

[Описание предметной области 6](#_Toc102755443)

[Средства и методы решения 10](#_Toc102755444)

[Решение задачи 13](#_Toc102755445)

[Реализация клиента 13](#_Toc102755446)

[Сетевая часть 27](#_Toc102755447)

[Заключение 32](#_Toc102755448)

[Использованные источники 33](#_Toc102755449)

[Приложение 34](#_Toc102755450)

[Распечатка SQL скриптов 34](#_Toc102755451)

# Введение

Ежедневно миллиарды людей тем или иным способом взаимодействуют с продуктами порожденными информационным прогрессом. Социальные сети, электронные дневники, форумы, компьютерные игры, ежедневники, и даже расписание автобусов давно преобразовано в цифровое программное обеспечение.

Согласно сервису по сбору интернет-статистики [internetlivestats](https://www.internetlivestats.com), каждые сутки отправляется более 200 миллиардов сообщений, просматривается около 7 миллиардов запрос и тратится более 10 терабайт трафика. Всё это превращается в бесконечный информационный трафик, в котором отправленное тобой в социальной сети сообщение лишь песчинка в пустыни информации.

Но что же из себя представляет информация? Если обратится к различным источникам, то можно найти несколько определений, например, *информация* — это любые сведения, принимаемые, передаваемые и сохраняемые различными источниками, совокупность знаний, которые могут быть восприняты, живыми организмами, машинами и другими информационными системами. Однако в рамках информатики существует более подходящий тезис, рассматривающий информацию исключительно как меру объема, которая необходима вычислительным системам для построения по ним конкретных данных.

Вся эта гонка за информацией порождает немыслимые объемы данных, которые нужно где-то хранить. Ведь невозможно скачать из интернета картинку, если бы не существовал жесткий диск, на котором она хранилась. Крупные компании по типу google строят огромные дата-центры для хранения информации. Так площадь одного из зданий занимает около 15.000 кв. м.

Но недостаточно только хранить данные, необходимо делать это эффективно. Было бы невозможно найти сообщение конкретного пользователя, если форум записывал все данные в один файл, не подвергая информацию никакому структурированию. На помощь приходят базы данных - организованная структура, которая предназначается для хранения, обработки и изменения большого количества информации.

Базу данных можно определить по следующими признакам:

1. Хранится на машиночитаемых носителях, обрабатываются в информационных системах.
2. Наличие структурированных данных
3. База данных содержит множество данных необходимых для решения задач конкретных пользователей.

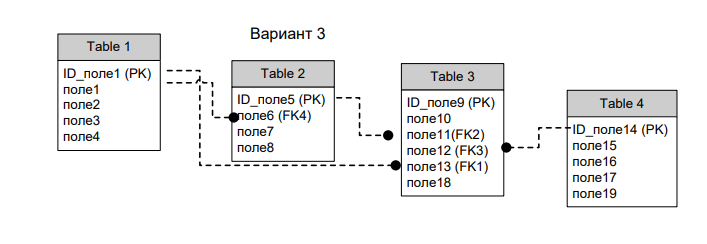
Современные базы данных нужны как для хранения больших объёмов данных, так и почти моментальной их выдачи по запросу от пользователя.

# Общая постановка задачи

Основной целью работы является систематизация знаний и накопление первоначального опыта в области проектирования и разработки баз данных. В рамках работы необходимо разработать клиент – серверное приложение, серверная часть которого будет реализована на PostgreSQL.

Спроектировать и реализовать базу данных в соответствие с индивидуальным вариантом *- Информационная система чемпионата России по футболу, ведение базы данных по составу команд, календарю и результатам игр, анализ результатов выступления команд и игроков.*

В рамках заданной предметной области реализовать заданную (рис 1.) схему отношений, выделить сущности и их атрибуты, так чтобы связи между сущностями соответствовали представленной схеме.

Рис 1.

В рамках работы необходимо реализовать следующие пункты:

* Постоянные таблицы и связи между ними.
* Индексы для увеличения скорости запросов.
* Хранимые процедуры на операции добавления/обновления/удаления данных.
* Распределение прав пользователя.
* Использовать скалярные и векторные функции.
* Триггер на обновление поля в таблице 4.
* Транзакцию, которую может быть зафиксирована или откатана.
* Курсор на обновление отдельных данных.
* Запросы для демонстрации навыков работы.

# Описание предметной области

Предметной областью работы является чемпионат России по футболу. Футбол - командный вид спорта, в котором целью является забить мяч в ворота соперника ногами или другими частями тела (кроме рук) большее количество раз, чем команда соперника. В настоящее время самый популярный и массовый вид спорта в мире.

Постановка задачи: ведение данных по составу команд, календарю и результатам игр, анализ результатов выступления команд и игроков.

В рамках информационной системы необходимо знать:

* Список команд, их состав.
* Расписание проведенных игр, участвующие команды и результаты встречи.
* Игроков чемпионата, их имена, роль, и общее количество забитых голов.
* Список голов, с указанием времени, игрока, забившего гол, и матча, в котором он был произведен.

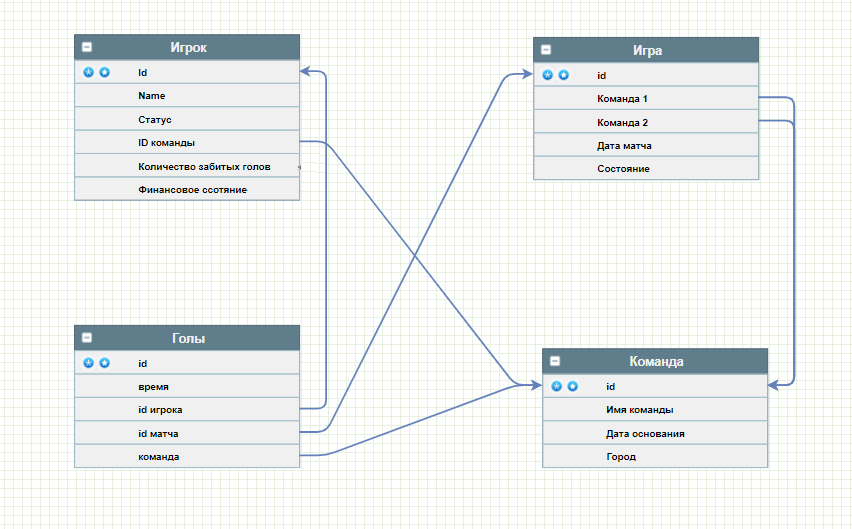


Рис. 2 Структурная модель системы

Информационная модель системы (рис 2.) представляет собой как систему в целом, так и структуру связей между её отдельными компонентами. Сущность команды обособлена от внешних связей, в тоже самое время записи таблицы игроков должны знать к какой команде он принадлежит, матч проходит между двумя командами, а гол забивает конкретной командой. Каждый гол принадлежит игроку, и забивается в определенном матче.

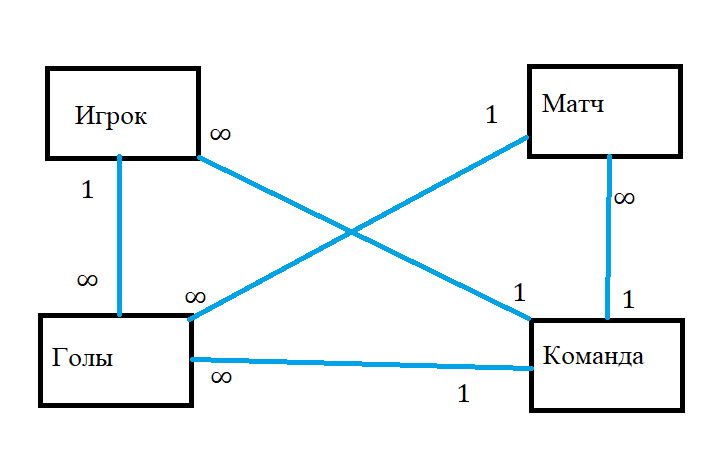


Рис 3. Тип связей между таблицами.

База данных будет состоять их следующих 4 таблиц:

1. Таблица “Игроки”
   1. ID-игрока
   2. Имя
   3. Специализация (вратарь, нападающий)
   4. Внешний ключ – команда к которой игрок принадлежит
   5. Суммарное количество забитых голов
   6. Финансовое состояние футболиста
2. Таблица “Команды”
   1. ID-команды
   2. Название
   3. Год основания
   4. Город
3. Таблица “Матчи”
   1. ID-матча
   2. Первая команда
   3. Вторая команда
   4. Состояние матча (уже состоялся или нет)
   5. Дата матча
4. Таблица “Голы”
   1. ID-гола
   2. Время во время матча
   3. Внешний ключ – игрок, забивший гол
   4. Внешний ключ – матч
   5. Внешний ключ – команда

# Средства и методы решения

Для реализации клиент-серверного приложения необходимо разбить проект на две одноименные подзадачи: создать как клиент, так и сервер, обрабатывающий запросы первого.

Для разработки клиентской части был использован язык программирования C++. Такой выбор был обусловлен безусловными преимуществами языка С:

* Является самым быстрым на сегодняшний день языком программирования.
* Удобен для разработки настольных кроссплатформенных приложений. Такое приложение будет одинаково работать как на Windows, так и на MacOS.
* Наличие обширного количества библиотек для работы с серверной частью приложения.

В качестве соответствующей IDE (Integrated Development Environment -система программных средств, используемая программистами для разработки программного обеспечения) был выбран QT изучаемый мною в рамках дисциплин разработки ПО. Написанный на С++, и предоставляющий мощные расширения для данного языка программирования, Qt часто используется для реализации графических интерфейсов, что и необходимо для реализации клиентской части, именно посредством взаимодействия пользователя с GUI будут происходить запросы к базе данных.

Исходя из требований к приложению, были использованы следующие элементы графического интерфейса:

* LineEdit - представляет собой одиночную строку для записи и считывания информации. Использовался для считывания введённых пользователем данных, для составления запроса к базе данных.
* ComboBox - выпадающий список с возможностью выбора одного элемента. Использовался для поддержания ссылочной целостности, позволяя пользователю выбирать из списка только допустимые значения внешних ключей.
* Table View и Table Widget – контейнеры для отображения прокручиваемых списков, использовались для вывода в них результатов запросов в к базе данных, т.е. табличной информации.

Согласно заданию, серверная часть приложения должна быть реализована посредством баз данных. Те в свою очередь делятся на реляционные (SQL) и не реляционные (noSQL). В рамках дисциплины подразумевается использование SQL – баз данных, имена она и будет использоваться в работе.

Главной особенностью реляционных баз данных является то, что данные в ней хранятся в виде двумерных таблиц. То есть, таблица состоит из столбцов, в которых может указывается, как названия, так и тип данных. Число столбцов в такой модели ограничено, из-за чего структура данных известна заранее в отличие от строк, каждая из которых по сути являются объектов сущности, хранимой в базе данных.

В качестве такой базы был выбран PostgreSQL - объектно-реляционная СУБД (совокупность программный средств для удобной и эффективной организации, контроля и администрирования баз данных). Это проверенная временем, постоянно развивающаяся система, предоставляющая пользователю все необходимые возможности по работе с БД.

По аналогии с IDE для языков программирования, для работы с PostgreSQL использовался pgAdmin 4(рис 4.) — это GUI инструмент для администрирования и управления базой данных PostgreSQL.

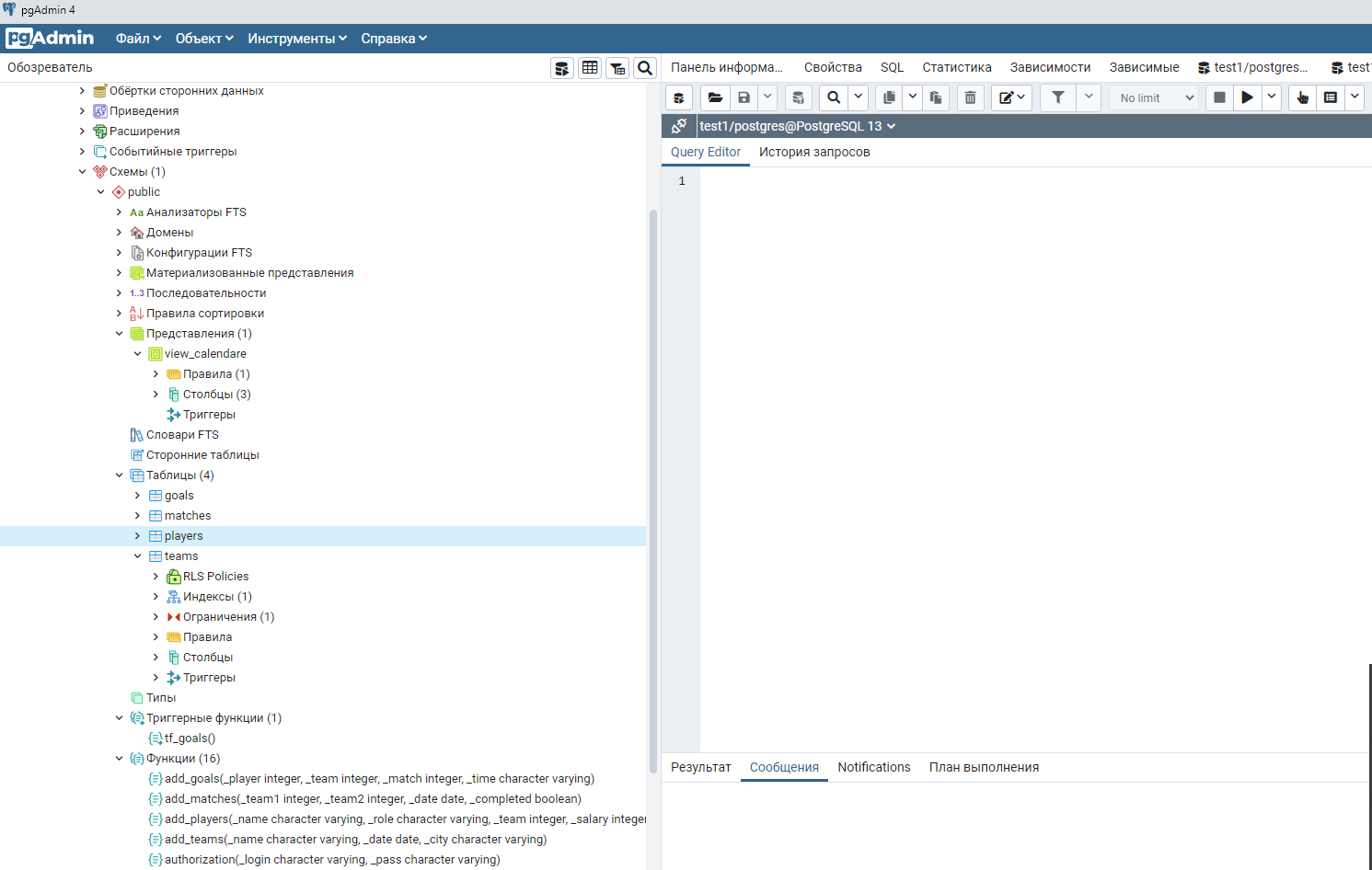


Рис 5. Интерфейс PgAdmin 4

# Решение задачи

## Реализация клиента

Логическая модель клиентской представляет собой иерархическую структуру, в которой каждая форма, представляющая конкретную подзадачу, зависит от родительского окна. Всего было реализовано 4 графические формы, среди них главное окно, ссылающееся на форму демонстрации работы с запросами, и форму для просмотра и редактирования данных, а также отдельная форма для считывания данных для составления транзакции.

На главной форме MainWindow (рис 5.) реализованы кнопки для перехода к двум дочерним окнам, запуска мастер-режима, а также подключения к базе данных.

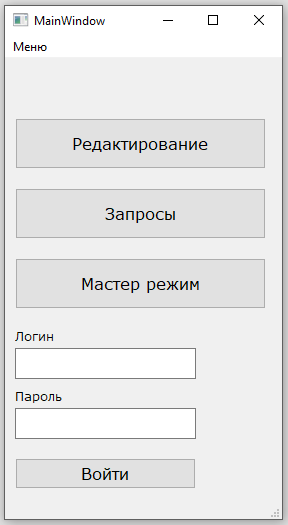
****

Рис 5. Главная форма.

Подключение к базе данных осуществляется посредством вызова конструктора класса Connect.

void MainWindow::**on\_connect\_triggered**()

{

Connect connect;

}

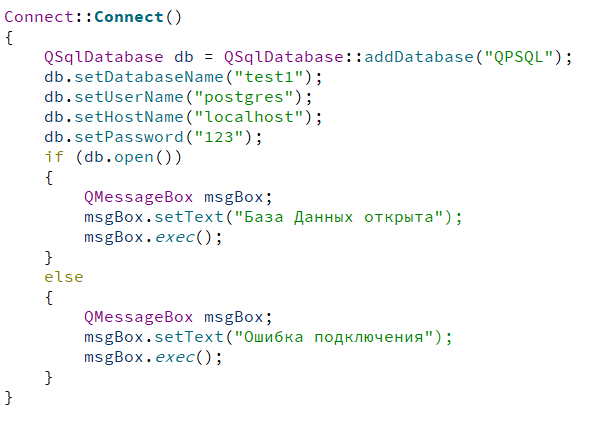


Рис 6. Класс подключения к базе данных

Мастер режим представляет собой реализацию распределения прав пользователей. Невозможно производить удаление из базы данных не в режиме администратора. Метод считывает логин и пароль, введённые пользователем, и сравнивает их с значениями из базы данных.

QSqlQuery query;

query.prepare("select authorization(:log,:pass)");

query.bindValue(":log",ui->lineEdit->text());

query.bindValue(":pass",ui->lineEdit\_2->text());

query.exec();

query.next();

if(query.value(0).toBool())

{

setIsAdmin(true);

QMessageBox msgBox;

msgBox.setText("Вы успешно перешли в режим администратора");

msgBox.*exec*();

}

else

{

QMessageBox msgBox;

msgBox.setText("Неккоректный логин или пароль");

msgBox.*exec*();

}

hide();

Форма выполнения и демонстрации выполнения запросов QueryWindow (рис. 6) реализует необходимые для демонстрации запросы согласно поставленной задаче, и выводит их в Table Widget. Каждая кнопка реализует соответствующий запрос. Поле строчного ввода lineEdit используется для считывания параметра для запросов 1 и 4.

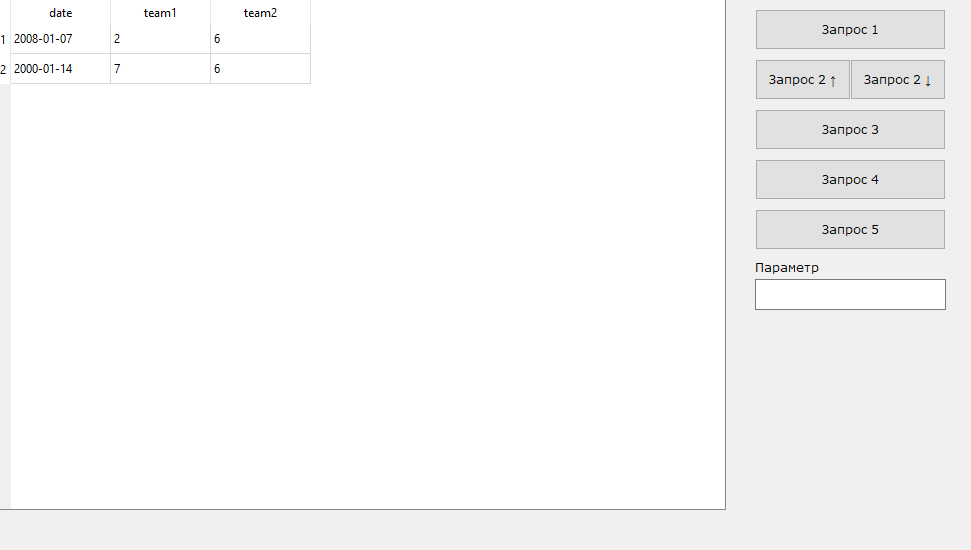


Рис 6. Окно выполнения запросов. Выполнение запроса 2.

В каждом запросе виджет Table адаптируем под новые входные данные, искусственно устанавливая количество и названия столбцов. Пример одного из запроса:

void QueryWindow::**query\_4**()

{

QSqlQuery query;

query.prepare(""

"SELECT team\_name, SUM(salary) as moneys "

"FROM players,teams "

"WHERE team\_id=team "

"GROUP BY team\_name HAVING SUM(salary)>:bindMoney");

query.bindValue(":bindMoney",ui->lineEdit->text());

query.exec();

ui->tableWidget->clear();

ui->tableWidget->setRowCount(query.size());

ui->tableWidget->setColumnCount(2);

QStringList LIST;

LIST << "TeamName" <<"Moneys";

ui->tableWidget->setHorizontalHeaderLabels(LIST);

fillTable(query);

}

Единый для всех запросов метод fillTable считывает данные из запроса и заносит их в таблицу.

void QueryWindow::**fillTable**(QSqlQuery query)

{

for(int row=0; row!=ui->tableWidget->rowCount(); ++row)

{

query.next();

for(int column=0; column!=ui->tableWidget->columnCount(); ++column)

{

QTableWidgetItem \*newItem = new QTableWidgetItem(query.value(column).toString());

ui->tableWidget->setItem(row, column, *newItem*);

}

}

}

Окно просмотра и редактирования (рис 7) позволяет просматривать содержимое каждой из 4 таблиц, совершить транзакцию, а также производить процесс добавления, удаления и обновления данных.

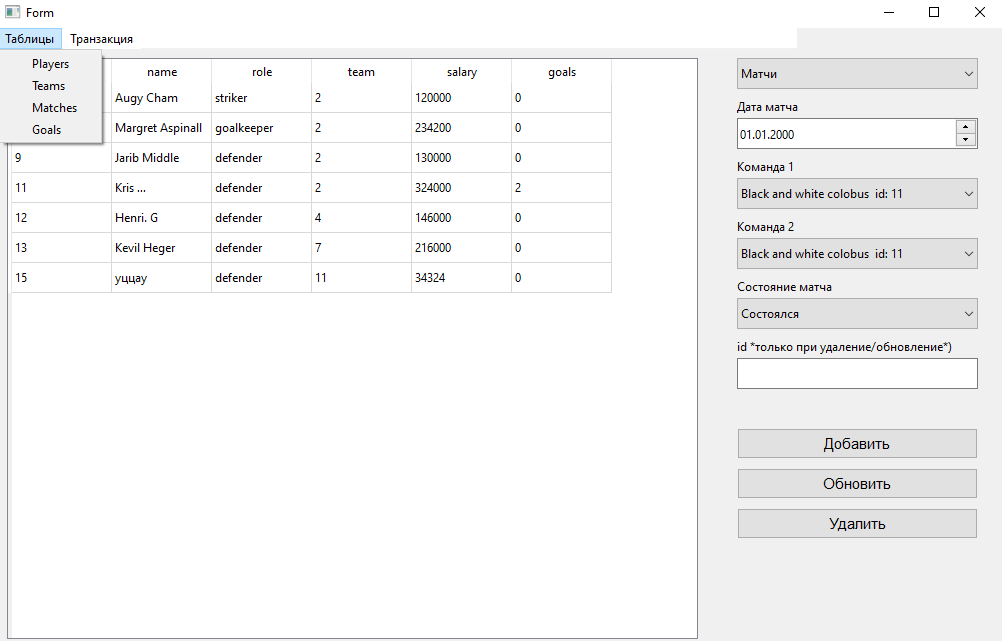


Рис 7. Окно редактирования данных

Правый верхний ComboBox позволяет выбирать таблицу для последующего редактирования. В зависимости от выбранного значения, на форме появляются поля ввода для заполнения соответствующих столбцов. Среди них как текстовая информация, так и данные типа дата.

void ViewWindow::**comboBox\_triggered**()

{

QSqlQuery query;

hideAll();

clearAll();

int value = ui->comboBoxTeams->currentIndex();

switch (value) Действия в зависимости от значения, выбранного в ComboBox индекса.

{

case 0://игрок

{

QString team;

ui->lineEdit->show();

ui->comboBox\_2->show();

ui->comboBox\_3->show();

ui->lineEdit\_4->show();

ui->label->setText("Имя игрока");

ui->label\_2->setText("Роль");

ui->label\_3->setText("Команда игрока");

ui->label\_4->setText("Зарплата");

ui->label\_5->setText("id \*только при удаление/обновление\*)");

QStringList roles = {"defender", "striker", "goalkeeper","midfielder"};

Значения внешних ключей выбираются пользователем из выпадающих списков, для обеспечения ссылочной целостности. Сами списки заполняюстя путем запроса на получение доступных значений из базы данных

query.exec("SELECT team\_name,team\_id FROM teams ORDER BY team\_name ASC");

ui->comboBox\_2->addItems(roles);

while(query.next())

{

team = query.value(0).toString() + " id: " + query.value(1).toString();

ui->comboBox\_3->addItem(team);

}

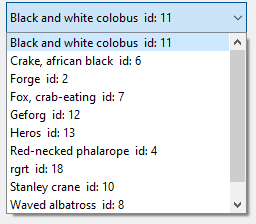


Рис 8. Список доступные внешних ключей на таблицу Команды.

Нажатие на любую из трех кнопок реализует одноименную процедуру по добавлению, удаления, редактированию таблиц. Метод *on\_pushButton\_add*отправляет запрос с вызовом хранимой процедуры *add\_*. Данные для передачи во входные параметры вводится пользователем.

switch (value)

{

case 0:

{

QString team\_id = ui->comboBox\_3->currentText().split("id: ")[1];

query.prepare("SELECT add\_players(:player\_name, :player\_role, :team, :salary)");

query.bindValue(":player\_name",ui->lineEdit->text());

query.bindValue(":player\_role",ui->comboBox\_2->currentText());

query.bindValue(":team",team\_id);

query.bindValue(":salary",ui->lineEdit\_4->text());

query.exec();

Удаление вызывает процедуру remove, в качестве параметра необходим только индекс удаляемой строчки. Из-за разделения прав пользователя, метод не будет срабатывать если пользователь не находится в режиме мастера.

void ViewWindow::**on\_pushButton\_delete\_clicked**()

{

if(!getIsAdmin())

{

QMessageBox msgBox;

msgBox.setText("Удаление данных может происходить в мастер режиме.");

msgBox.*exec*();

return;

}

QSqlQuery query;

int value = ui->comboBoxTeams->currentIndex();

switch (value)

{

case 0:

{

query.prepare("SELECT remove\_players(:id)");

query.bindValue(":id",ui->lineEdit\_5->text());

query.exec();

Обновление использует процедуру update, которая по мимо необходимых для добавления параметров, принимает индекс обновляемой строки.

void ViewWindow::**on\_pushButton\_update\_clicked**()

{

QSqlQuery query;

int value = ui->comboBoxTeams->currentIndex();

switch (value)

{

case 0:

{

query.prepare("SELECT update\_players(:id,:player\_name, :player\_role,:salary)");

query.bindValue(":id",ui->lineEdit\_5->text());

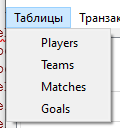
query.bindValue(":player\_name",ui->lineEdit->text());

query.bindValue(":player\_role",ui->comboBox\_2->currentText());

query.bindValue(":salary",ui->lineEdit\_4->text());

query.exec();

Подпункты меню – bar (левый верхний угл) позволяют выбирать таблицы для вывода на экран.



void ViewWindow::**on\_actionPlayers\_triggered**()

{

PlayersViewModel \*model = new PlayersViewModel();

QList<Players> \*list = new QList<Players>;

QSqlQuery query;

query.exec("SELECT \* FROM players ORDER BY player\_id ASC");

while(query.next())

{

Players player = \*new Players(query.value(0).toInt(),query.value(1).toString(),query.value(2).toString(),query.value(3).toInt(),query.value(4).toInt(),query.value(5).toInt());

list->append(player);

}

model->populate(*list*);

this->ui->tableView->*setModel*(*model*);

Согласное поставленной задачи реализована транзакция, представляющая собой перевод средств между игроками, в случае попытки перевода суммы больше, чем имеется в наличие баланс уйдет в минус, и транзакция будет отмена, в противном случае одобрена. Ввод данных производится в отдельном окне (Рис. 9), а сам запрос выполняется в форме редактирования. 

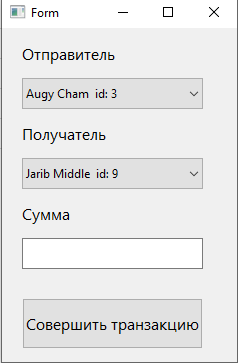
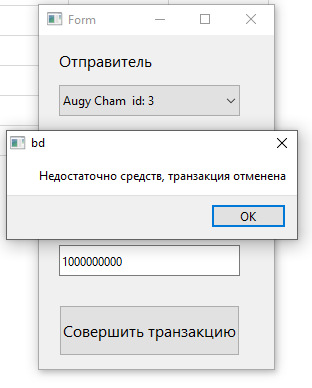
 

Рис 9. Окно совершения транзакции.

void ViewWindow::**transfer**(QString sum,QString sender,QString getter)

{

transquery.exec("BEGIN");

transquery.prepare("select transfer\_money(:sender,:getter,:sum)");

transquery.bindValue(":sender",sender);

transquery.bindValue(":getter",getter);

transquery.bindValue(":sum",sum);

transquery.exec();

transquery.next();

if(transquery.value(0).toBool())

{

transquery.exec("COMMIT");

QMessageBox msgBox;

msgBox.setText("Транзакция одобрена");

msgBox.*exec*();

}

else

{

transquery.exec("ROLLBACK");

QMessageBox msgBox;

msgBox.setText("Недостаточно средств, транзакция отменена");

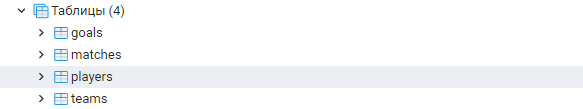
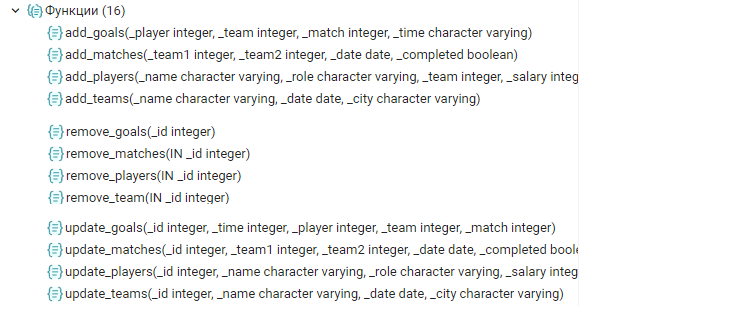
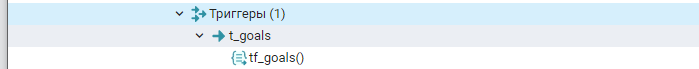
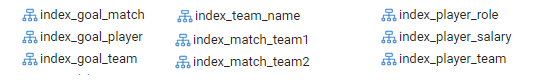
msgBox.*exec*();

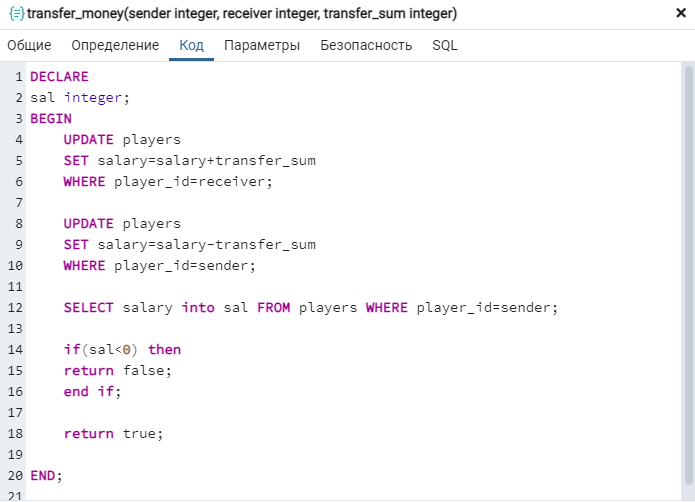
}

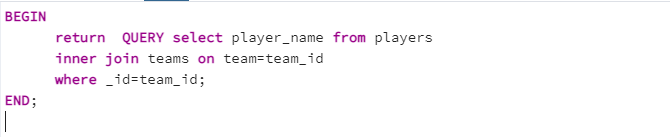
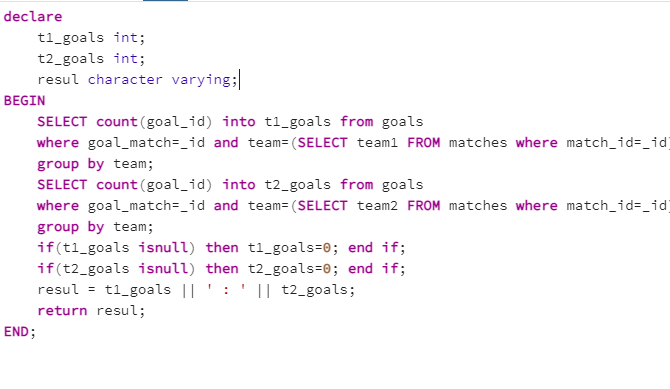
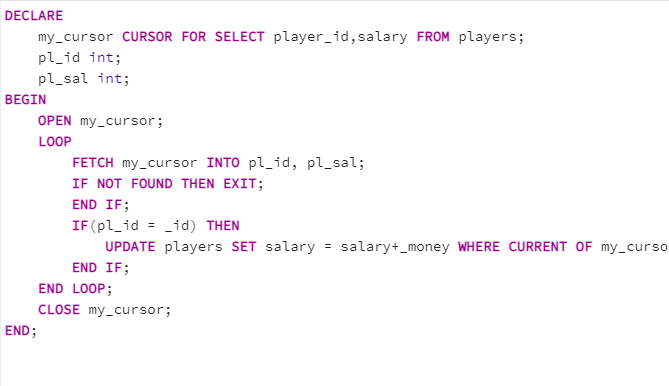
}

## Сетевая часть

В рамках построения сетевой части приложения были реализованы следующие пункты:

1. Созданы таблицы Goals, Players, Teams, Matches представляющие объекты соответствующих сущностей. 
2. Реализованы хранимые процедуры обновления, добавления, удаления для каждой из таблиц. Хранимая процедура – набор инструкций выполнявшиеся как единое целое, и хранящиеся напрямую в БД. 
3. Создан триггер, изменяющий поле total\_goals таблицы Players. Привязанный к таблице голов, триггер работает на обновление, удаление и добавление данных в этой таблицы. Новая добавленная запись гола автоматически увеличит счётчик суммарное количество голов данного игрока. 
4. Для каждой таблицы добавлены индексы - специальные таблицы поиска, которую поисковая система базы данных можно использовать для ускорения поиска данных.
5. Реализована хранящаяся процедура transfer\_money, использующаяся в клиента в рамках транзакции. Суть процедуры в переводе средств со счёта одного футболиста, на другой. Если баланс отправителя уходит в минус, то возвращается значение false, что служит сигналом к откату транзакции.



1. Создана хранимая процедура get\_players, возвращающая таблицу. Возвращаемый список представляет собой перечисление футболистов одной команды, id которой передает в качестве входного параметра. 
2. Реализована пользовательская скалярная функция get\_result. Принимает в качестве параметра id матча, и выводит строку с его итогом счётом между командами. 
3. Курсор на обновление данных. Курсор позволяет извлекать данные из таблицы построчно, а не сразу всей кучей, что позволяет экономить память программы. Функция cursor\_addmoneys использующая курсор позволяет пополнить счёт определенного футболиста на n-ое количество денег. 

# Заключение

В результате проделанной работы было разработано функционирующее клиент – серверное приложение, серверная часть которого реализована посредством баз данных. Была спроектирована и реализована база данных чемпионата по футболу.

В процессе разработки изучены и применены на практике множество инструментов по управлению баз данных, таких как триггеры, индексы, транзакции, курсоры, а также хранимые процедуры. В клиентской части пользователю была предоставлена возможность просмотра, добавление, модификацию и удаление данных по всей предметной области.

В заключение следует отметить что базы данных это мощный инструмент для хранения и управления данными. Благодаря своей функциональности они обеспечивают возможность выполнять молниеносные запросы к огромным количествам данных, производить масштабирование в рамках необходимости, разграничивать правда доступа к данным, а также поддерживать их структурную целостность.

# Использованные источники

1. Понятие реляционных и нереляционных баз данных.[Электронный ресурс] Режим доступа:

<https://boodet.online/sqk-nosql>

1. PostgreSQL:Документация[Электронный ресурс] Режим доступа: <https://postgrespro.ru/docs/postgresql/>
2. Моргунов E. PostgreSQL. Основы языка SQL BHV, 2018 г – 336 c.
3. Триггеры в SQL [Электронный ресурс] Режим доступа:<https://codetown.ru/sql/triggery/>
4. Документация QT [Электронный ресурс] Режим доступа: <https://doc.qt.io/>
5. Как использовать курсор в функциях на PL/pgSQL? [Электронный ресурс] Режим доступа: <https://info-comp.ru/obucheniest/254-cursor-in-functions.html>
6. Выполнение хранимых процедур и функций PostgreSQL [Электронный ресурс] Режим доступа: <https://pythonru.com/biblioteki/hranimye-procedury-i-funkcij-postgresql>
7. Связи в реляционных базах данных [Электронный ресурс] Режим доступа: [https://zametkinapolyah.ru](https://zametkinapolyah.ru/zametki-o-mysql/chast-3-2-vidy-svyazej-mezhdu-tablicami-v-baze-dannyx-svyazi-v-relyacionnyx-bazax-dannyx-otnosheniya-kortezhi-atributy.html)
8. Работа с QTableView или с QTableWidget [Электронный ресурс] Режим доступа: <https://www.cyberforum.ru/qt/thread238089.html>
9. SQL – Индексы [Электронный ресурс] Режим доступа: <https://andreyex.ru/bazy-dannyx/uchebnoe-posobie-po-sql/sql-indeksy/>

# Приложение

## Распечатка SQL скриптов

* Создание таблиц вместе с индексами:

Таблица Teams

CREATE TABLE teams

(

team\_id integer NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),

team\_name character varying(25) COLLATE pg\_catalog."default",

foundation\_date date,

city character varying(25) COLLATE pg\_catalog."default",

CONSTRAINT "Team\_pkey" PRIMARY KEY (team\_id)

);

CREATE INDEX index\_team\_name

ON teams (team\_name);

Таблица Players

CREATE TABLE players

(

player\_id integer NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),

player\_name character varying(30) COLLATE pg\_catalog."default" NOT NULL,

player\_role character varying(20) COLLATE pg\_catalog."default" NOT NULL,

team integer,

salary integer,

total\_goals integer,

CONSTRAINT "Player\_pkey" PRIMARY KEY (player\_id),

CONSTRAINT fk\_player\_team FOREIGN KEY (team)

REFERENCES teams (team\_id) MATCH SIMPLE

);

CREATE INDEX index\_player\_role

ON players (player\_role);

CREATE INDEX index\_player\_salary

ON players (salary);

CREATE INDEX index\_player\_team

ON players (team);

Таблица Goals

CREATE TABLE goals

(

goal\_id integer NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),

player integer,

team integer,

goal\_match integer,

goal\_time character varying COLLATE pg\_catalog."default",

CONSTRAINT goals1\_pkey PRIMARY KEY (goal\_id),

CONSTRAINT fk\_goals\_match FOREIGN KEY (goal\_match)

REFERENCES matches (match\_id) MATCH SIMPLE,

CONSTRAINT fk\_goals\_player FOREIGN KEY (player)

REFERENCES players (player\_id) MATCH SIMPLE,

CONSTRAINT fk\_goals\_team FOREIGN KEY (team)

REFERENCES teams (team\_id) MATCH SIMPLE

);

CREATE INDEX index\_goal\_match

ON goals (goal\_match);

CREATE INDEX index\_goal\_player

ON goals(player);

CREATE INDEX index\_goal\_team

ON goals (team);

Таблица Matches

CREATE TABLE matches

(

match\_id integer NOT NULL GENERATED ALWAYS AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),

team1 integer NOT NULL,

team2 integer NOT NULL,

match\_date date,

completed boolean NOT NULL,

CONSTRAINT "Match\_pkey" PRIMARY KEY (match\_id),

CONSTRAINT fk\_match\_team1 FOREIGN KEY (team1)

REFERENCES teams (team\_id) MATCH SIMPLE,

CONSTRAINT fk\_match\_team2 FOREIGN KEY (team2)

REFERENCES teams (team\_id) MATCH SIMPLE

);

CREATE INDEX index\_match\_team1

ON matches (team1);

CREATE INDEX index\_match\_team2

ON matches (team2);

* Курсор на обновление данных

CREATE OR REPLACE FUNCTION cursor\_addmoneys(IN \_id integer,IN \_money integer)

RETURNS void

LANGUAGE 'plpgsql'

VOLATILE

PARALLEL UNSAFE

COST 100

AS $BODY$

DECLARE

my\_cursor CURSOR FOR SELECT player\_id,salary FROM players;

pl\_id int;

pl\_sal int;

BEGIN

OPEN my\_cursor;

LOOP

FETCH my\_cursor INTO pl\_id, pl\_sal;

IF NOT FOUND THEN EXIT;

END IF;

IF(pl\_id = \_id) THEN

UPDATE players SET salary = salary+\_money WHERE CURRENT OF my\_cursor;

END IF;

END LOOP;

CLOSE my\_cursor;

END;

$BODY$;

* Создание триггера и триггерной функции

CREATE FUNCTION public.tf\_goals()

RETURNS trigger

LANGUAGE 'plpgsql'

COST 100

VOLATILE NOT LEAKPROOF

AS $BODY$

BEGIN

IF ( TG\_OP = 'DELETE' ) THEN

UPDATE players

SET total\_goals=total\_goals-1

WHERE player\_id=old.player;

return null;

ELSEIF ( TG\_OP = 'INSERT' ) THEN

UPDATE players

SET total\_goals=total\_goals+1

WHERE player\_id=new.player;

return null;

ELSIF ( TG\_OP = 'UPDATE' ) THEN

UPDATE players

SET total\_goals=total\_goals-1

WHERE player\_id=old.player;

UPDATE players

SET total\_goals=total\_goals+1

WHERE player\_id=new.player;

return null;

END IF ;

END ;

$BODY$;

CREATE TRIGGER t\_goals

AFTER INSERT OR DELETE OR UPDATE

ON public.goals

FOR EACH ROW

EXECUTE PROCEDURE public.tf\_goals();

* Обновляемое табличное представление:

CREATE OR REPLACE VIEW view\_calendare

AS

SELECT matches.match\_date,

matches.team1,

matches.team2

FROM matches;

* Скалярная и векторная функции

CREATE OR REPLACE FUNCTION public.get\_result(IN \_id integer)

RETURNS character varying

LANGUAGE 'plpgsql'

VOLATILE

PARALLEL UNSAFE

COST 100

AS $BODY$

declare

t1\_goals int;

t2\_goals int;

resul character varying;

BEGIN

SELECT count(goal\_id) into t1\_goals from goals

where goal\_match=\_id and team=(SELECT team1 FROM matches where match\_id=\_id)

group by team;

SELECT count(goal\_id) into t2\_goals from goals

where goal\_match=\_id and team=(SELECT team2 FROM matches where match\_id=\_id)

group by team;

if(t1\_goals isnull) then t1\_goals=0; end if;

if(t2\_goals isnull) then t2\_goals=0; end if;

resul = t1\_goals || ' : ' || t2\_goals;

return resul;

END;

$BODY$;

CREATE OR REPLACE FUNCTION get\_players(

\_id integer)

RETURNS TABLE(playername character varying)

LANGUAGE 'plpgsql'

COST 100

VOLATILE PARALLEL UNSAFE

ROWS 1000

AS $BODY$

BEGIN

return QUERY select player\_name from players

inner join teams on team=team\_id

where \_id=team\_id;

END;

$BODY$;

* Хранимые процедуры:
  + CREATE OR REPLACE FUNCTION add\_goals(

\_player integer,

\_team integer,

\_match integer,

\_time character varying)

RETURNS void

LANGUAGE 'sql'

COST 100

VOLATILE PARALLEL UNSAFE

AS $BODY$

INSERT INTO goals (player,team,goal\_match,goal\_time)

VALUES (\_player, \_team,\_match,\_time);

$BODY$;

* + CREATE OR REPLACE FUNCTION add\_matches(

\_team1 integer,

\_team2 integer,

\_date date,

\_completed boolean DEFAULT false)

RETURNS void

LANGUAGE 'sql'

COST 100

VOLATILE PARALLEL UNSAFE

AS $BODY$

INSERT INTO matches (team1, team2, match\_date,completed)

VALUES (\_team1,\_team2,\_date,\_completed);

$BODY$;

* + CREATE OR REPLACE FUNCTION add\_players(

\_name character varying,

\_role character varying,

\_team integer,

\_salary integer)

RETURNS void

LANGUAGE 'sql'

COST 100

VOLATILE PARALLEL UNSAFE

AS $BODY$

INSERT INTO players (player\_name, player\_role, team, salary,total\_goals)

VALUES (\_name, \_role, \_team,\_salary,0);

$BODY$;

* + CREATE OR REPLACE FUNCTION add\_teams(

\_name character varying,

\_date date,

\_city character varying)

RETURNS void

LANGUAGE 'sql'

COST 100

VOLATILE PARALLEL UNSAFE

AS $BODY$

INSERT INTO teams (team\_name, foundation\_date, city)

VALUES (\_name,\_date,\_city);

$BODY$;

* + CREATE OR REPLACE FUNCTION public."authorization"(

\_login character varying,

\_pass character varying)

RETURNS boolean

LANGUAGE 'plpgsql'

COST 100

VOLATILE PARALLEL UNSAFE

AS $BODY$

BEGIN

if(\_login='admin' and \_pass='admin') then

return true;

end if;

return false;

END;

$BODY$;

* + CREATE OR REPLACE FUNCTION remove\_goals(

\_id integer)

RETURNS void

LANGUAGE 'sql'

COST 100

VOLATILE PARALLEL UNSAFE

AS $BODY$

DELETE FROM goals WHERE goal\_id=\_id;

$BODY$;

* + CREATE OR REPLACE FUNCTION remove\_matches(

\_id integer)

RETURNS void

LANGUAGE 'sql'

COST 100

VOLATILE PARALLEL UNSAFE

AS $BODY$

delete FROM goals WHERE goal\_match=\_id;

DELETE FROM matches WHERE match\_id=\_id;

$BODY$;

* + CREATE OR REPLACE FUNCTION remove\_players(

\_id integer)

RETURNS void

LANGUAGE 'sql'

COST 100

VOLATILE PARALLEL UNSAFE

AS $BODY$

delete FROM goals WHERE player=\_id;

DELETE FROM players WHERE player\_id=\_id;

$BODY$;

* + CREATE OR REPLACE FUNCTION remove\_team(

\_id integer)

RETURNS void

LANGUAGE 'sql'

COST 100

VOLATILE PARALLEL UNSAFE

AS $BODY$

delete from goals where goal\_match in (SELECT match\_id FROM matches where team1=\_id or team2=\_id);

delete from matches where team1=\_id or team2=\_id;

delete from players where team=\_id;

DELETE FROM teams WHERE team\_id=\_id

$BODY$;

* + CREATE OR REPLACE FUNCTION transfer\_money(

sender integer,

receiver integer,

transfer\_sum integer)

RETURNS boolean

LANGUAGE 'plpgsql'

COST 100

VOLATILE PARALLEL UNSAFE

AS $BODY$

DECLARE

sal integer;

BEGIN

UPDATE players

SET salary=salary+transfer\_sum

WHERE player\_id=receiver;

UPDATE players

SET salary=salary-transfer\_sum

WHERE player\_id=sender;

SELECT salary into sal FROM players WHERE player\_id=sender;

if(sal<0) then

return false;

end if;

return true;

END;

$BODY$;

* + CREATE OR REPLACE FUNCTION update\_goals(

\_id integer,

\_time integer,

\_player integer,

\_team integer,

\_match integer)

RETURNS void

LANGUAGE 'sql'

COST 100

VOLATILE PARALLEL UNSAFE

AS $BODY$

UPDATE goals

SET goal\_time=\_time,player=\_player,team=\_team,goal\_match=\_match

WHERE goal\_id=\_id

$BODY$;

* + CREATE OR REPLACE FUNCTION update\_matches(

\_id integer,

\_team1 integer,

\_team2 integer,

\_date date,

\_completed boolean)

RETURNS void

LANGUAGE 'sql'

COST 100

VOLATILE PARALLEL UNSAFE

AS $BODY$

UPDATE matches

SET team1=\_team1,team2=\_team2,match\_date=\_date,completed=\_completed

WHERE match\_id=\_id

$BODY$;

* + CREATE OR REPLACE FUNCTION update\_players(

\_id integer,

\_name character varying,

\_role character varying,

\_salary integer)

RETURNS void

LANGUAGE 'sql'

COST 100

VOLATILE PARALLEL UNSAFE

AS $BODY$

UPDATE players

SET player\_name=\_name,player\_role=\_role,salary=\_salary

WHERE player\_id=\_id

$BODY$;

* + CREATE OR REPLACE FUNCTION update\_teams(

\_id integer,

\_name character varying,

\_date date,

\_city character varying)

RETURNS void

LANGUAGE 'sql'

COST 100

VOLATILE PARALLEL UNSAFE

AS $BODY$

UPDATE teams

SET team\_name=\_name,foundation\_date=\_date,city=\_city

WHERE team\_id=\_id

$BODY$;