МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ИС

КУРСОВАЯ РАБОТА

по дисциплине «Управление данными»

Тема: Проектирование БД и разработка клиентского приложения для администратора футбольной команды

Студентка гр. 3373 _____ Касаткина А.Р.

Преподаватель _____ Татарникова Т.М.

Санкт-Петербург

2025

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Касаткина А.Р.

Группа 3373

Тема работы: Проектирование БД и разработка клиентского приложения для администратора футбольной команды

Исходные данные:

Вариант 13. Спроектировать базу данных, построить программу, обеспечивающей взаимодействие с ней в режиме диалога, для администратора футбольной команды. В БД должны храниться сведения о командах, участвующих в первенстве, и об игроках, играющих в данной команде, стадионах, на которых проходят встречи, и цене билета на игры.

Сведения о команде представляют собой название команды, город, где она базируется, ФИО тренера, даты встреч команды, счет встреч, противников команды, стадион, на котором играет команда, место в таблице прошлого сезона.

Сведения об игроках включают в себя ФИО игроков, их номера, результативность данного игрока в данной встрече. В один день команда может играть только в одном матче. Сведения о стадионе содержат: название, город, вместимость. Цена билета на матч зависит от вместимости стадиона и положения команды в прошлом году (наибольшая - при игре тройки призеров, наименьшая - при игре тройки аутсайдеров). Игроки могут переходить из одной команды в другую. Некоторые встречи могут быть перенесены.

Администратору могут потребоваться следующие сведения:

- даты встреч команды, ее противник и счет;
- ФИО и номера игроков, участвовавших во встрече (по названию команды, городу и дате встречи);

- результативность данного игрока в данной встрече (по названию команды, городу, дате встречи и ФИО игрока);
- цена билета на матч указанных команд.

Администратор БД может вносить следующие изменения:

- переход игрока из одной команды в другую;
- отмена встречи;
- назначение нового тренера.

Необходимо предусмотреть возможность выдачи справки об играх на указанном стадионе и отчет о их проведении (количество проведенных встреч, число побед хозяев и гостей, ФИО игроков, забивавших мячи в каждой команде, названия стадионов, где проводились встречи).

Содержание пояснительной записки:

Содержание

Введение

- Анализ предметной области
- Обоснование модели данных
- Обоснование выбора СУБД
- Описание функций групп пользователей
- Описание функций управления данными
- Организация защиты БД

Заключение

Список источников

Приложение А. Руководство пользователя БД

Приложение Б. Листинг кода

Предполагаемый объем пояснительной записки:

Не менее 25 страниц.

Дата выдачи задания: 01.09.2025

Дата сдачи реферата: 20.12.2025

Дата защиты реферата: 20.12.2025

Студентка _____ Касаткина А.Р.

Преподаватель _____ Татарникова Т.М.

АННОТАЦИЯ

В рамках курсовой работы разработана информационная система для управления футбольной командой. При разработке использован системный подход, включающий проектирование структуры хранения данных, реализацию бизнес-логики на языке Python с применением объектно-ориентированного программирования и построение интуитивного консольного интерфейса. Для управления данными применялась реляционная СУБД MySQL с обеспечением целостности данных. Программный модуль предоставляет администратору функционал для просмотра статистики, формирования отчетов, управления кадровыми изменениями и корректировки спортивного календаря. Разработанное решение демонстрирует практическую применимость для автоматизации рабочих процессов в спортивных организациях.

SUMMARY

As part of this coursework, an information system for managing a football team was developed. A systems approach was used during development, including designing a data storage structure, implementing business logic in Python using object-oriented programming, and building an intuitive console interface. MySQL, a relational DBMS, was used for data management, ensuring data integrity. The software module provides the administrator with functionality for viewing statistics, generating reports, managing personnel changes, and adjusting the sports calendar. The developed solution demonstrates its practical applicability for automating workflows in sports organizations.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Работа администратора футбольной команды связана с обработкой большого количества информации: данных об игроках, расписании матчей, результатах игр и ценах на билеты. Хранение этих данных вручную (в таблицах или документах) неудобно, приводит к ошибкам и замедляет работу.

Автоматизация должна решить три ключевые проблемы:

1. Необходимость надёжного хранения данных. Требуется создать единую централизованную среду для хранения всех связанных сущностей: сведений о командах, игроках, стадионах, матчах и билетах, что обеспечит целостность и непротиворечивость информации.
2. Необходимость эффективного управления данными. Пользователю системы требуется полноценный функционал для выполнения основных операций: поиска конкретных записей (например, статистики игрока), добавления новых (о перенесённой встрече), изменения существующих (при смене тренера или переходе игрока).
3. Необходимость быстрого получения отчётов и справок. Для анализа и планирования администратору необходимо по запросу формировать сводные отчёты: статистику выступлений команды, списки бомбардиров, финансовую отчётность по продаже билетов или отчётность по использованию стадионов.

Как следствие возникает задача проектирования и реализации реляционной базы данных (БД) и клиентского приложения для взаимодействия с ней.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Предметная область — это администрирование футбольной команды, участвующей в первенстве. Основными объектами для учета являются: Команда (название, город, тренер, место в прошлом сезоне), Игрок (ФИО, номер, позиция, текущая команда), Стадион (название, город, вместимость) и Матч (дата, статус, команды-участницы, счет, стадион). Особенностями являются возможность перехода игроков между командами, перенос матчей и правило, что команда играет не более одного матча в день. Также учитывается Статистика игрока в каждом матче (голы, передачи, сыгранные минуты) и Билеты, цена которых зависит от прошлых достижений команд и вместимости стадиона.

Основные задачи БД: надежно хранить все эти данные, обеспечивать быстрый поиск и выборку информации (расписание или статистику игрока), поддерживать целостность данных при операциях перевода игрока или отмены матча, автоматически рассчитывать цену билетов и формировать отчеты по запросу.

Алгоритмы решения строятся на SQL-запросах. Поиск и формирование отчетов выполняются с помощью оператора SELECT с соединениями таблиц (JOIN) и условиями (WHERE). Операции изменения (добавление, обновление, удаление) — через INSERT, UPDATE, DELETE. Ключевые бизнес-процессы, такие как перевод игрока, выполняются в рамках транзакций для сохранения целостности данных. Расчет цены билета реализуется запросом с оператором CASE, который анализирует место команд в прошлом сезоне и параметры стадиона.

Пользователи системы делятся на две группы. Основной пользователь — Администратор команды — обладает полными правами на работу с данными: ввод, изменение, удаление информации и формирование отчетов. Технический специалист — Системный администратор БД — отвечает за обеспечение

работоспособности, безопасность и резервное копирование базы данных, не взаимодействуя с её содержимым напрямую.

Система должна генерировать выходные документы и отчеты: календарь матчей команды, протокол конкретной игры с составом и статистикой игроков, сводку по результативности игроков или команды, отчет по использованию стадиона.

2. ОБОСНОВАНИЕ МОДЕЛИ ДАННЫХ

Разработанная модель БД для информационной системы управления футбольной командой является реляционной и основана на анализе предметной области. Её структура отражает все ключевые сущности и бизнес-процессы, описанные в техническом задании.

Ключевые принципы проектирования:

- Нормализация: модель приведена к третьей нормальной форме (3NF). Данные не дублируются. И информация о команде хранится в одной таблице (teams), а не повторяется для каждого её игрока или матча.
- Обеспечение целостности: использование ограничений внешнего ключа гарантирует, что не может существовать матча с несуществующей командой или стадионом, а также статистики игрока для несуществующего матча.

Основные сущности:

1. stadiums (Стадионы) - спортивные сооружения

Атрибуты: название, город, вместимость

2. teams (Команды) - футбольные коллективы

Атрибуты: название, город, тренер, место в прошлом сезоне

3. players (Игроки) - спортсмены

Атрибуты: ФИО, игровой номер, позиция

4. matches (Матчи) - спортивные события

Атрибуты: дата и время, статус (запланирован/перенесен/отменен), счет

5. player_stats (Статистика игроков) - связующая сущность

Атрибуты: голы, передачи, желтые карточки, сыгранные минуты

6. tickets (Билеты) - товары для продажи

Атрибуты: категория, цена, количество доступных мест

Связи между сущностями базы данных

1. stadiums → matches

Тип: 1:N (один ко многим)

Ключ: stadiums.id → matches.stadium_id

2. teams → matches (домашние)

Тип: 1:N (один ко многим)

Ключ: teams.id → matches.home_team_id

3. teams → matches (гостевые)

Тип: 1:N (один ко многим)

Ключ: teams.id → matches.away_team_id

4. matches → tickets

Тип: 1:N (один ко многим)

Ключ: matches.id → tickets.match_id

5. players → player_stats

Тип: 1:N (один ко многим)

Ключ: players.id → player_stats.player_id

6. matches → player_stats

Тип: 1:N (один ко многим)

Ключ: matches.id → player_stats.match_id

7. teams → player_stats

Тип: 1:N (один ко многим)

Ключ: teams.id → player_stats.team_id

Выбранные типы данных (INT для идентификаторов и счетчиков, VARCHAR для текста, DATETIME для даты и времени, DECIMAL для цены, ENUM для статуса) характерны хранимой информации и обеспечивают эффективное использование памяти.

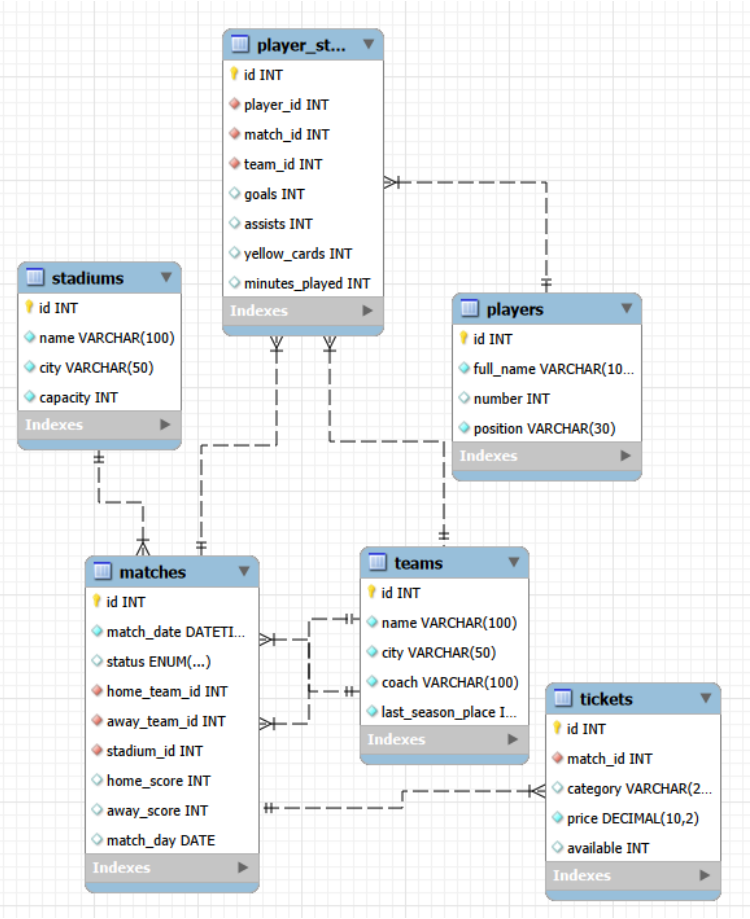

Рис. 1. Диаграмма модели базы данных

3. ОБОСНОВАНИЕ ВЫБОРА СУБД

Была выбрана реляционная СУБД MySQL. Это решение обусловлено несколькими ключевыми факторами. Во-первых, данные имеют четкую структуру и логические связи (например, матч связан с двумя командами и стадионом), что идеально ложится на реляционную модель. MySQL позволяет легко задать эти связи через внешние ключи и гарантирует, что не появится матч с несуществующей командой.

Во-вторых, MySQL обладает достаточной производительностью и эффективно выполняет сложные запросы, необходимые для отчетов, например, поиск всех матчей команды или подсчет голов игрока.

В-третьих, функционал MySQL полностью покрывает нужды: поддержка транзакций (что важно для операций перевода игрока, где нужно обновить несколько таблиц), работа с представлениями для упрощения частых запросов.

В-четвертых, MySQL проста в установке и эксплуатации, имеет понятные инструменты MySQL Workbench для проектирования базы и обширное сообщество для решения возможных проблем.

4. ОПИСАНИЕ ФУНКЦИЙ ГРУПП ПОЛЬЗОВАТЕЛЕЙ

Функции пользователя "Администратор команды":

Администратор команды выполняет все основные операции по управлению футбольным клубом "Краснодар-U18".

Функции просмотра (S):

- Просмотр информации о командах
- Просмотр расписания и результатов матчей
- Просмотр состава игроков и их статистики
- Просмотр информации о стадионах
- Просмотр цен на билеты
- Формирование отчетов (по стадионам, топ бомбардиров)

Функции добавления (I):

- Добавление новых игроков
- Добавление информации о командах-соперниках
- Добавление данных о стадионах
- Добавление записей о матчах
- Добавление статистики игроков
- Добавление информации о билетах

Функции изменения (U):

- Изменение данных тренера
- Изменение информации об игроках
- Изменение расписания (перенос матчей)
- Изменение статистики игроков
- Перевод игроков между командами

Функции удаления (D):

- Удаление данных об игроках
- Отмена встреч (удаление матчей)
- Удаление ошибочных записей

Функции пользователя "Системный администратор БД":

Системный администратор не работает со спортивными данными, а выполняет только технические функции:

- Создание пользователей системы (администратора команды)
- Назначение ролей и прав доступа
- Настройка резервного копирования базы данных
- Восстановление базы данных из резервной копии
- Мониторинг состояния и производительности БД
- Контроль целостности данных

Объект БД	Администратор команды	Системный администратор БД
stadiums	SIU	-
teams	SIUD	-
players	SIUD	-
matches	SIUD	-
player_stats	SIUD	-
tickets	SIUD	-
Представления (VIEWs)	S	-
Пользователи БД	-	SIUD
Резервные копии	-	SIU

5. ОПИСАНИЕ ФУНКЦИЙ УПРАВЛЕНИЯ ДАННЫМИ

Разработанная информационная система реализует комплекс функций управления данными, необходимых для эффективной работы администратора футбольной команды. Эти функции охватывают полный жизненный цикл данных — от их создания и хранения до предоставления конечным пользователям в удобном формате.

Первая функция — хранение данных, то есть создание и поддержание информационных объектов. В основе системы лежит реляционная БД, которая состоит из шести взаимосвязанных таблиц. Каждая таблица представляет собой отдельный класс объектов предметной области: teams для команд, players для игроков, stadiums для стадионов, matches для матчей, player_stats для статистики выступлений и tickets для информации о билетах. Система не просто хранит эти данные в виде отдельных записей, но и обеспечивает их структурную целостность. Связи между таблицами, реализованные через механизм внешних ключей (FOREIGN KEY).

Вторая функция — манипулирование данными, которое включает четыре основные операции: добавление, изменение, удаление и поиск. На практике это означает, что администратор может динамически работать с содержимым базы. Он вносит изменения при кадровых перестановках (например, назначает нового тренера командой UPDATE или переводит игрока, обновляя поле team_id в связанных записях), корректирует расписание, перенося матчи. Эти операции реализованы через SQL-запросы в программном модуле и предоставлены пользователю через консольное меню.

Третья функция — предоставление запрашиваемых данных пользователю в удобной форме, то есть генерация справок, отчетов и итогов. Эта возможность превращает сырые данные в полезную информацию для принятия решений.

6. ОРГАНИЗАЦИЯ ЗАЩИТЫ БАЗЫ ДАННЫХ

1. Ограничения целостности информационных объектов

Для обеспечения логической целостности и непротиворечивости данных в базе применяются следующие ограничения:

- Таблица stadiums (Стадионы):

- PRIMARY KEY (id) — гарантирует уникальность идентификатора каждого стадиона.

- NOT NULL для полей name, city, capacity — обеспечивает обязательное заполнение ключевых атрибутов.

- Таблица teams (Команды):

- PRIMARY KEY (id) — уникальный идентификатор команды.

- NOT NULL для name, city, coach, last_season_place — запрещает пустые значения в основных полях.

- Значение last_season_place должно быть положительным целым числом.

- Таблица players (Игроки):

- PRIMARY KEY (id) — уникальный идентификатор игрока.

- NOT NULL для full_name и position — обязательные для заполнения поля.

- Игровой номер (number) должен быть уникальным в рамках одной команды.

- Таблица matches (Матчи):

- PRIMARY KEY (id) — уникальный идентификатор матча.

- FOREIGN KEY (home_team_id) REFERENCES teams(id) и FOREIGN KEY (away_team_id) REFERENCES teams(id) — обеспечивают ссылочную целостность: матч может быть создан только между существующими командами.

- FOREIGN KEY (stadium_id) REFERENCES stadiums(id) — гарантирует, что матч назначается на существующий стадион.

- ENUM (запланирован, перенесен, отменен) — ограничивает допустимые значения статуса матча.

- UNIQUE (match_day, home_team_id) — бизнес-ограничение: команда может играть не более одного матча в день.

- Счет матча (home_score, away_score) может быть только неотрицательным целым числом.

- Таблица player_stats (Статистика игроков):

- PRIMARY KEY (id) — уникальный идентификатор записи статистики.
- goals, assists, yellow_cards, minutes_played — неотрицательные целые числа.

- Таблица tickets (Билеты):

- PRIMARY KEY (id) — уникальный идентификатор билета.
- FOREIGN KEY (match_id) REFERENCES matches(id) — билет привязан к существующему матчу.
- available INT — неотрицательное количество доступных билетов.

2. Средства физической защиты и резервного копирования

Для обеспечения физической сохранности данных и возможности восстановления после сбоев рекомендуется следующая стратегия резервного копирования:

1. Полное резервное копирование — еженедельно, в выходные дни (например, в воскресенье ночью). Создается полная копия всей БД krasnodar_football. Это основной архив для восстановления.
2. Дифференциальное резервное копирование — ежедневно, в нерабочее время (после 23:00). Сохраняются только изменения, произошедшие с момента последнего полного бэкапа. Ускоряет процесс создания резервных копий в будние дни.

3. Процедура подтверждения подлинности (аутентификации)

Доступ к информационной системе защищен двухуровневой процедурой аутентификации.

Уровень 1: Аутентификация на уровне СУБД

- Каждому пользователю системы создается уникальная учетная запись в MySQL.
- Для доступа необходимо указать логин и пароль.

- Пароли хранятся в хэшированном виде с использованием современного алгоритма для MySQL 8.0+.

Уровень 2: Аутентификация на уровне приложения

- Клиентская программа, написанная на Python, запрашивает у пользователя учетные данные для подключения к БД (хост, имя базы, логин, пароль).
- В текущей учебной версии эти данные зашиты в код для упрощения тестирования.
- Программа не хранит пароль в памяти дольше, чем требуется для установки соединения с БД.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была полностью разработана информационная система для управления футбольной командой. Основным результатом стало создание комплексного решения, включающего спроектированную БД и клиентскую программу с консольным интерфейсом. БД, реализованная на СУБД MySQL, содержит все необходимые таблицы (стадионы, команды, игроки, матчи, статистику и билеты) и корректно отражает связи между сущностями предметной области. Особое внимание было уделено обеспечению целостности данных через внешние ключи и уникальные ограничения, а также реализации ключевых бизнес-правил, таких как перевод игроков, расчёт цены билета и запрет на более одного матча команды в день.

Для проектирования БД применялись стандартные технологии реляционного моделирования и язык SQL. Программный модуль на Python с использованием библиотеки MySQL-connector предоставляет администратору удобный доступ ко всем функциям: от просмотра расписания и статистики до выполнения административных операций (смена тренера, отмена встречи). Созданные SQL-представления упрощают формирование часто запрашиваемых отчётов. Разработанная система является работоспособной, соответствует всем поставленным задачам и может быть использована для автоматизации реальных процессов в спортивной организации, демонстрируя практическую значимость выполненной работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Советов Б.Я., Цехановский В.В., Чертовской В.Д. «Базы данных: теория и практика». — м.: Юрайт, 2021. — 347 с.
2. Документация MySQL 8.0 [электронный ресурс]. — режим доступа: https://dev.mysql.com/doc/ (дата обращения: 10.12.2025).
3. MySQL Workbench Manual (version 8.0) [электронный ресурс]. — режим доступа: https://dev.mysql.com/doc/workbench/en/ (дата обращения: 10.12.2025).
4. Дейт, К. Дж. «Введение в системы баз данных» — 8-е изд. — м.: вильямс, 2019. — 1328 с.
5. MySQL Connector/Python Developer Guide [электронный ресурс]. — режим доступа: https://dev.mysql.com/doc/connector-python/en/ (дата обращения: 10.12.2025).
6. Project Experience. РД 50-34.698-90 Руководство Пользователя (пример оформления) [электронный ресурс]. – режим доступа: https://www.prj-exp.ru/patterns/pattern_user_guide.php (дата обращения: 10.12.2025).

ПРИЛОЖЕНИЕ А
РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ БД

1. Введение

1.1. Область применения

Настоящий документ является руководством по эксплуатации (Руководством пользователя) клиентского приложения для работы с базой данных футбольного клуба "Краснодар-U18". Руководство применяется на этапе опытной и промышленной эксплуатации системы, предназначено для основного пользователя — Администратора команды.

1.2. Краткое описание возможностей

Информационная система предназначена для централизованного управления всеми данными футбольной команды. Система предоставляет следующие возможности:

- Работа с данными: Полноценный цикл операций (просмотр, добавление, изменение, удаление) над сущностями: команды, игроки, стадионы, матчи, билеты, статистика.
- Бизнес-логика: Автоматизация ключевых процессов: перевод игрока между командами, перенос и отмена встреч, динамический расчет цены билета.
- Аналитика и отчетность: Формирование стандартных отчетов (календарь матчей, топ бомбардиров, статистика по стадиону) и выполнение специализированных запросов.

1.3. Уровень подготовки пользователя

Пользователь системы (Администратор команды) должен обладать:

- Базовыми навыками работы с операционной системой Windows или аналогичной.
- Базовыми навыками работы в командной строке (терминале).

- Знаниями в предметной области (футбол, структура команды, организация соревнований).

1.4. Перечень эксплуатационной документации

Для понимания контекста и принципов работы системы рекомендуется ознакомиться с Пояснительной запиской к курсовому проекту, в частности, с разделами "Анализ предметной области" и "Описание функций групп пользователей".

2. Назначение и условия применения системы

Система предназначена для автоматизации ежедневной деятельности администратора футбольной команды, включая ведение базы игроков, формирование календаря матчей, фиксацию результатов, управление билетами и подготовку отчетности.

Условия применения:

- Технические средства: Персональный компьютер или ноутбук.
- Программное обеспечение: Установленные СУБД MySQL (версия 8.0 или выше), интерпретатор Python (версия 3.8 или выше) с библиотекой mysql-connector-python.
- Данные: Предварительно созданная и заполненная БД.
- Пользователь: Наличие прав доступа к базе данных (учетные данные для подключения).

3. Подготовка к работе

3.1. Состав дистрибутива

Для запуска системы необходимы:

- Файл клиентского приложения football_manager.py

- Файл SQL-скрипта schema.sql для создания и заполнения структуры базы данных.

3.2. Порядок загрузки данных и программ

- Убедитесь, что служба MySQL Server запущена.
- Выполните в среде MySQL Workbench скрипт schema.sql для инициализации базы.

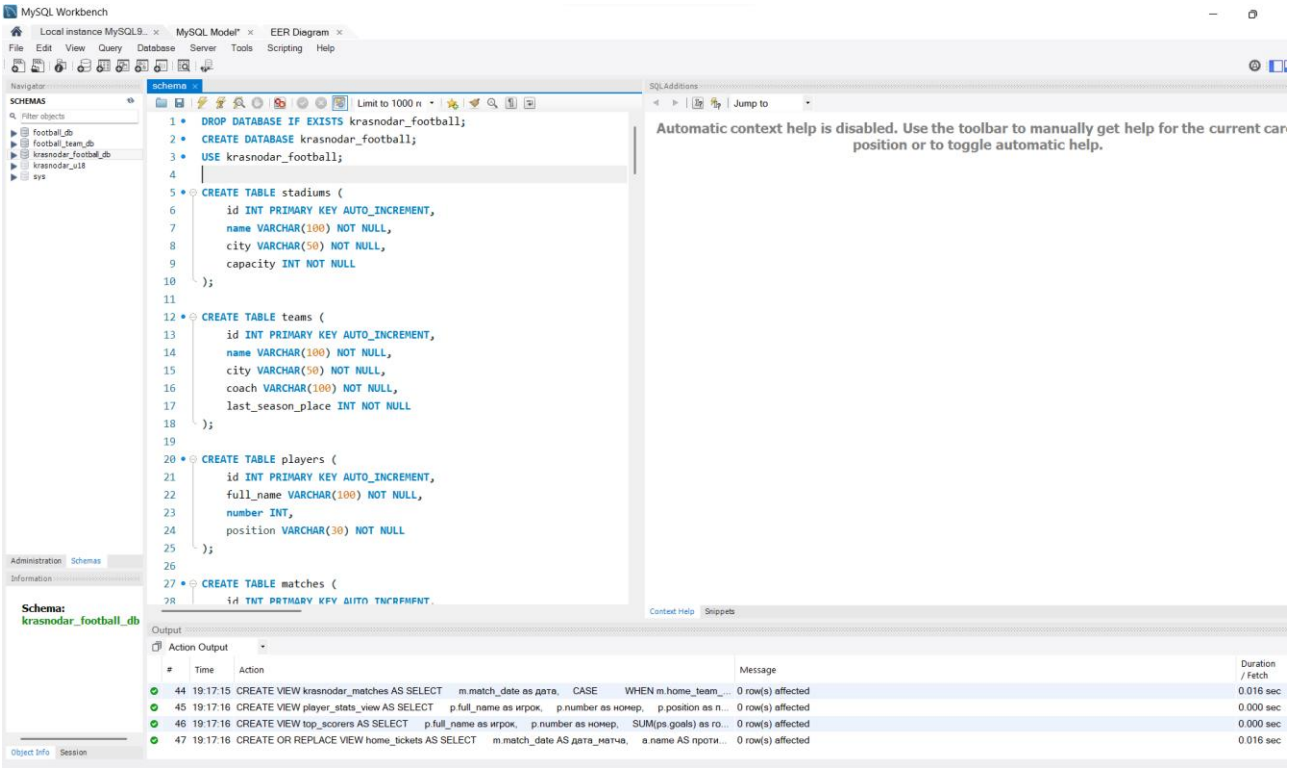

Рис. 2. MySQL Workbench

- Сохраните файл football_manager.py в удобную директорию на вашем компьютере.

3.3. Порядок проверки работоспособности

- Откройте командную строку (терминал) и перейдите в папку с файлом football_manager.py

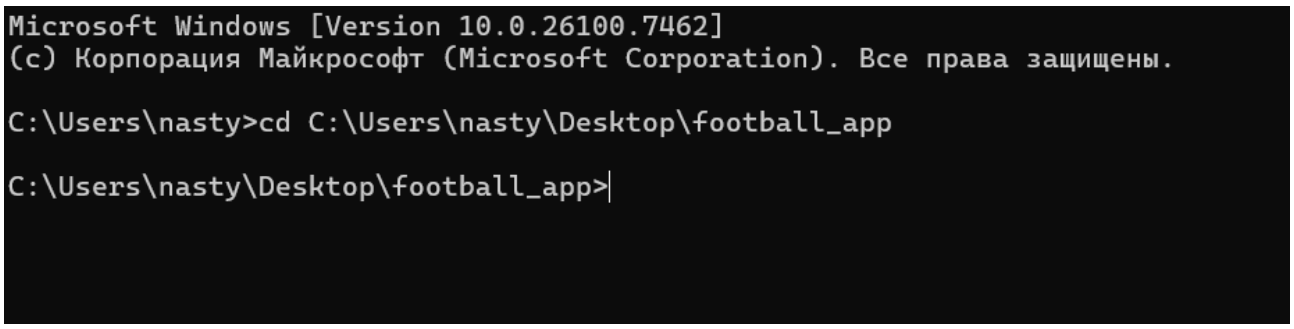

Рис. 3. Терминал

- Выполните команду: python football_manager.py.
- При успешном подключении к базе данных на экране должно появиться главное меню:

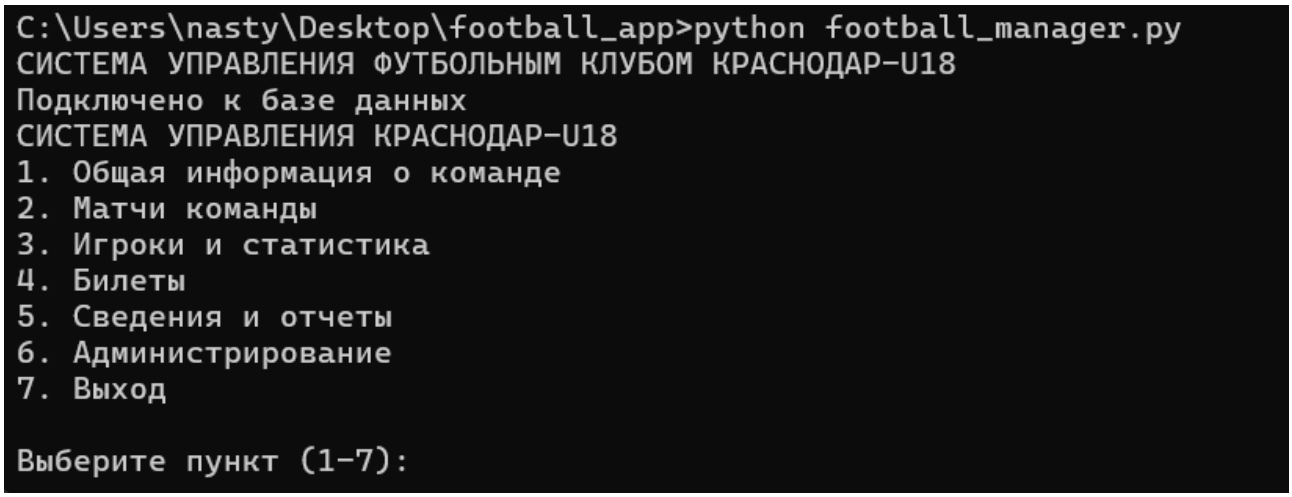

Рис. 4. Главное меню программы

4. Описание операций

4.1. Выполняемые функции и задачи

Введём с клавиатуры 1. Программа выведет на экран полную информацию о команде «Краснодар-U18»:

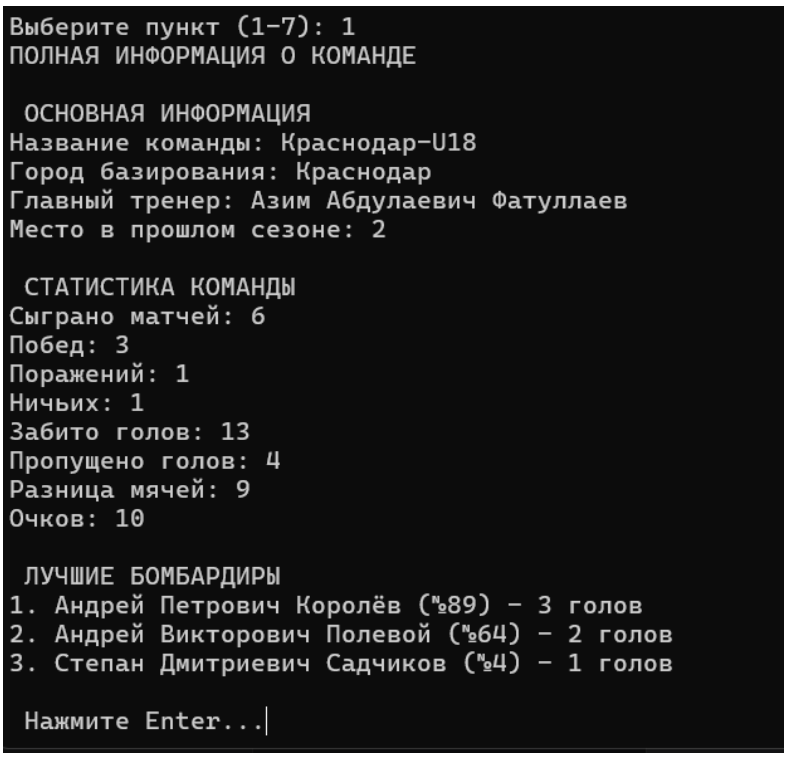

Рис. 5. Вывод полной информации о команде

После нажатия клавиши Enter программа вернётся в главное меню (Рис. 2). Снова в главном меню введём 2. Откроется подменю для работы с матчами.

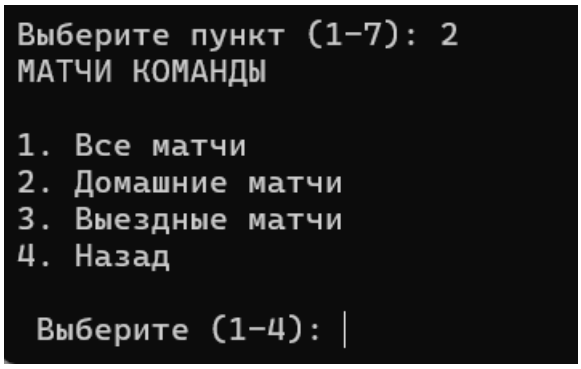

Рис. 6. Меню работы с матчами

Рассмотрим варианты взаимодействия из этого меню. Вариант 2.1. В меню матчей введём 1. Программа отобразит все матчи команды.

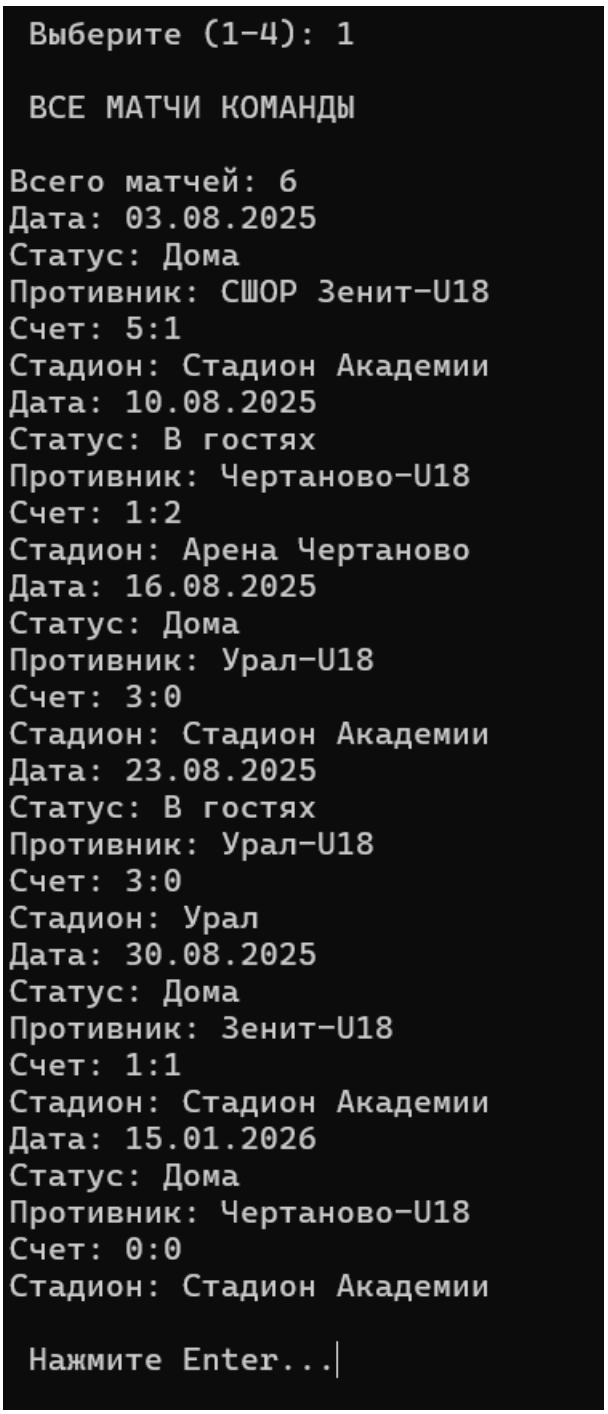

Рис. 7. Вывод всех матчей команды

После нажатия Enter программа вернётся в меню матчей (Рис. 6).

Вариант 2.2. В меню матчей введём 2. Программа выведет только домашние матчи.

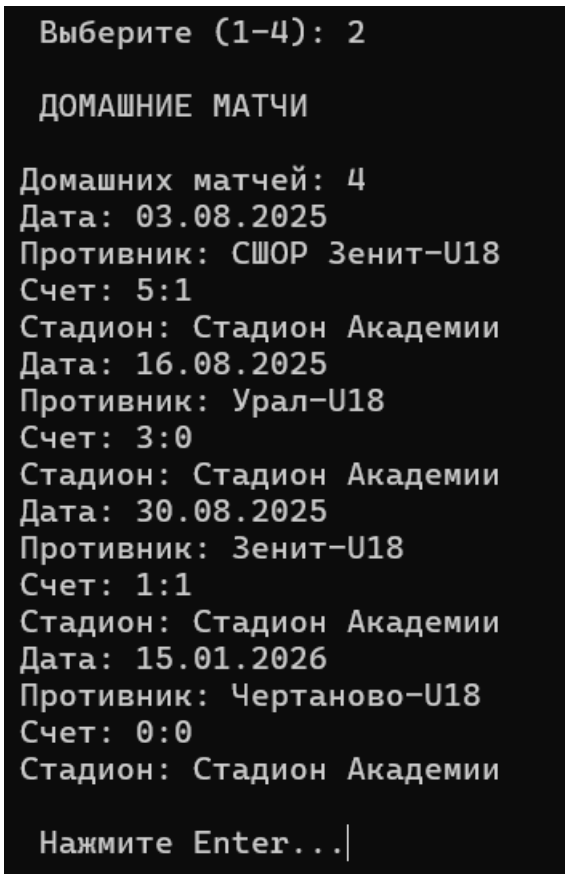

Рис. 8. Вывод домашних матчей команды

После нажатия Enter программа вернётся в меню матчей (Рис. 6).

Вариант 2.3. В меню матчей введём 3. Программа выведет только выездные матчи.

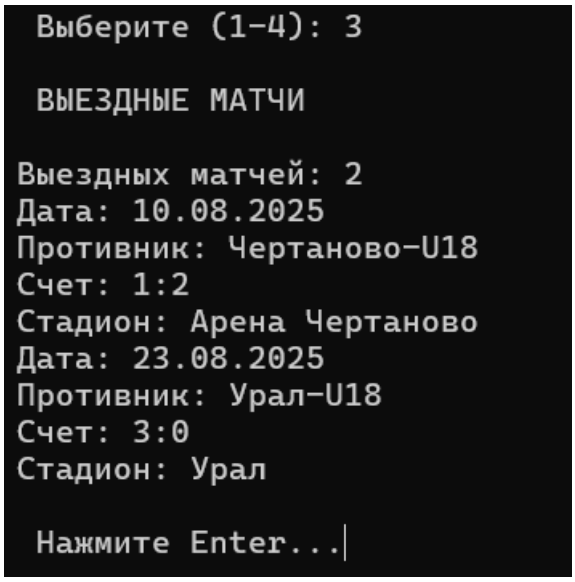

Рис. 9. Вывод выездннных матчей команды

Вариант 2.4. В меню матчей введём 4. Произойдёт возврат в главное меню (Рис. 2).

```
1. Все матчи
2. Домашние матчи
3. Выездные матчи
4. Назад

 Выберите (1-4): 4
СИСТЕМА УПРАВЛЕНИЯ КРАСНОДАР-U18
1. Общая информация о команде
2. Матчи команды
3. Игроки и статистика
4. Билеты
5. Сведения и отчеты
6. Администрирование
7. Выход

Выберите пункт (1-7):
```

Рис. 10. Возврат в главное меню

В главном меню введём 3. Откроется подменю для работы с игроками.

В подменю введём 1. На экране появится таблица со всеми игроками и их основными показателями.

```
Выберите пункт (1-7): 3
ИГРОКИ И СТАТИСТИКА

1. Все игроки
2. Поиск игрока
3. Топ-5 бомбардиров
4. Назад

 Выберите (1-4): 1

 ВСЕ ИГРОКИ КРАСНОДАР-U18

 Всего игроков: 23
Игрок                    №    Позиция         М    Г    Мин
Андрей Петрович Королёв  89   Нападающий      3    3    86.7
Андрей Викторович Полевой 64  Полузащитник    3    2    81.0
Андрей Олегович Карпенко 99   Вратарь         2    1    46.5
Степан Дмитриевич Садчиков 4  Защитник        1    1    90.0
Кирилл Сергеевич Леконцев 9   Нападающий      1    1    74.0
Богдан Иванович Кудзаву  3    Защитник        3    0    34.0
Иван Сергеевич Акманов   14   Защитник        3    0    42.7
Сергей Алексеевич Тройчин 21  Защитник        3    0    16.7
Ярослав Олегович Крючков 8    Полузащитник    3    0    60.3
Максим Игоревич Пономаренко 42 Полузащитник   3    0    64.3
Даниэль Артемович Амбарцумов 7 Нападающий      3    0    21.3
Роман Андреевич Сумачёв  11   Нападающий      3    0    87.7
Артём Владимирович Дряхлов 90 Нападающий       3    0    82.3
Иван Александрович Федько 88  Вратарь         2    0    90.0
Виктор Петрович Шипуль   5    Защитник        2    0    15.5
Евгений Михайлович Дедяев 61  Защитник        2    0    90.0
Дмитрий Викторович Ачкинази 91 Вратарь         1    0    90.0
Артём Дмитриевич Кульбаев 96  Полузащитник    1    0    23.0
Марк Сергеевич Федотов   1    Вратарь         0    0    0.0
Вадим Андреевич Матвеенко 6   Защитник        0    0    0.0
Никита Владимирович Закарлюка 50 Защитник      0    0    0.0
Ефим Александрович Буркин 15  Полузащитник    0    0    0.0
Эльдар Раминович Гусейнов 97  Нападающий      0    0    0.0

 Нажмите Enter...
```

Рис. 11. Вывод списка всех игроков

В меню игроков введём 2. Программа запросит имя для поиска и выведет детальную статистику.

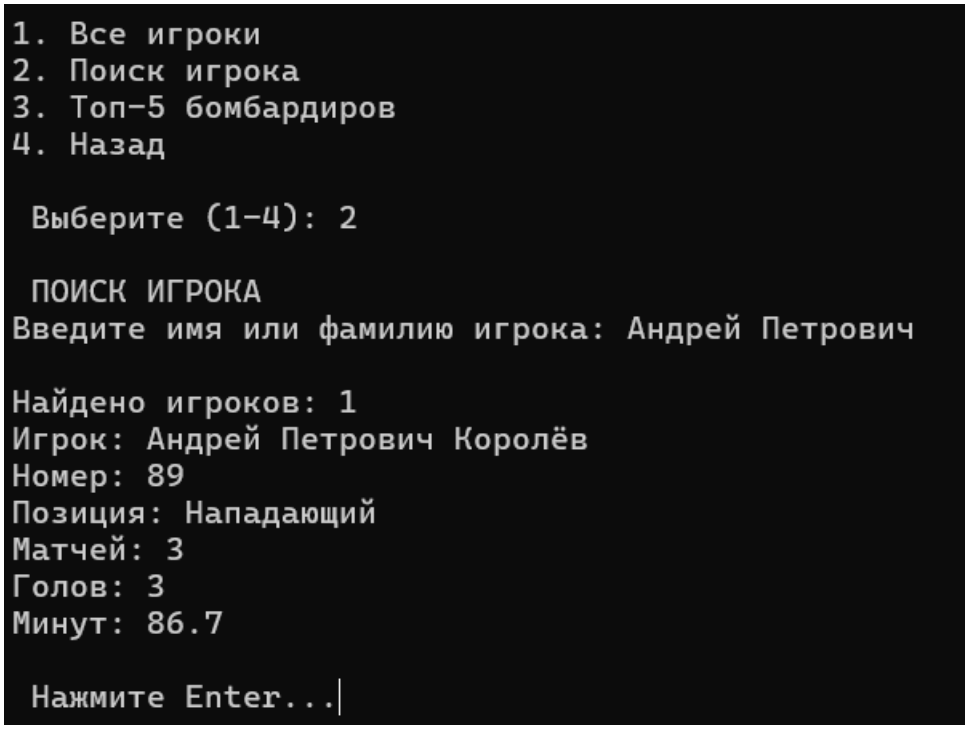

Рис. 12. Поиск игрока

В меню игроков введём 3. Будет выведен рейтинг топ-5 лучших бомбардиров.

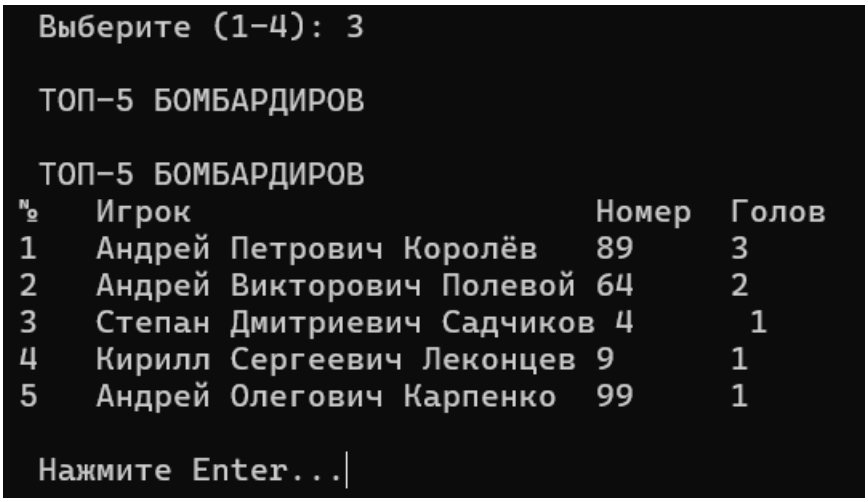

Рис. 13. Топ-5 лучших бомбардиров

В меню игроков введём 4. Произойдёт возврат в главное меню (Рис. 2).

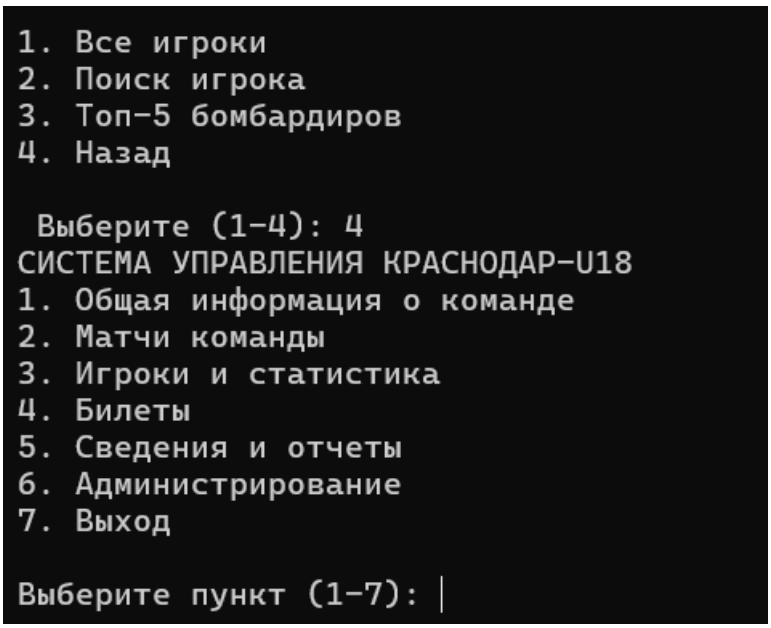

Рис. 14. Возврат в главное меню

В главном меню введём 4. Программа отобразит информацию о билетах на предстоящие домашние матчи с автоматически рассчитанной ценой.

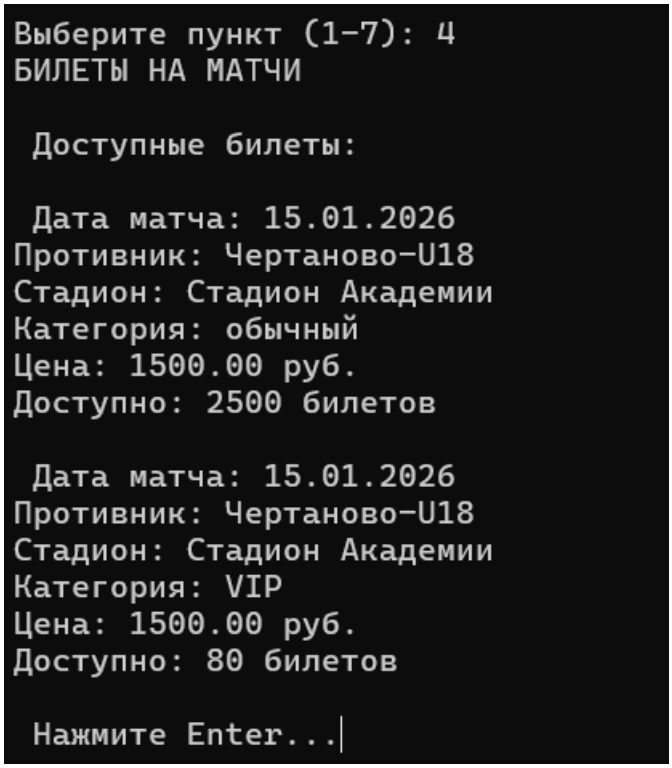

Рис. 15. Вывод информации о билетах

В главном меню введём 5. Откроется меню для получения сводных отчётов.

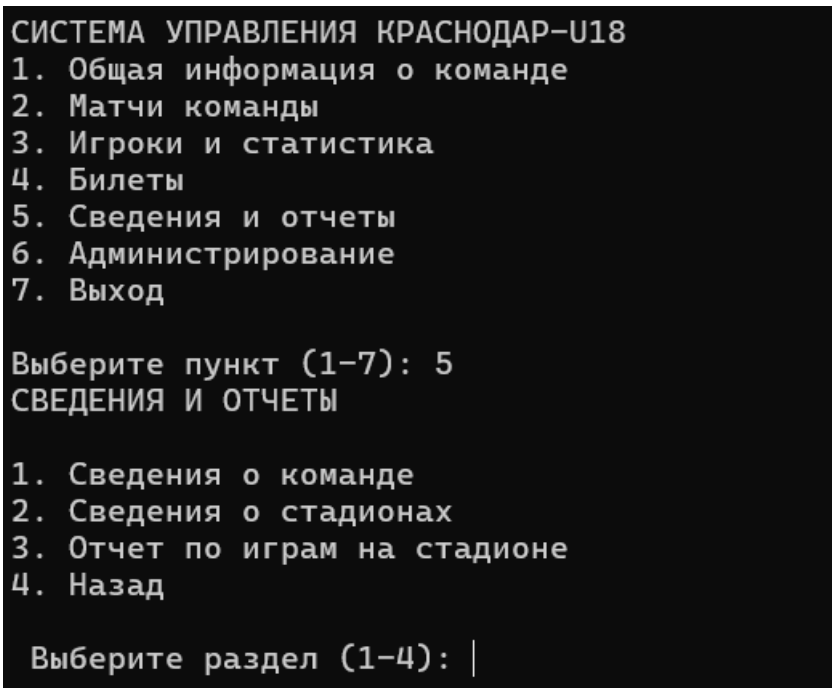

Рис. 16. Меню отчётов

В подменю введем 1. Программа формирует и выводит полные сведения о команде «Краснодар-U18», состоящие из двух логических частей.

Часть 1: Основная информация. Выводится справка по команде, аналогичная пункту 1 главного меню, но в более компактном виде. Часть 2: Детальный список всех матчей. Программа последовательно выводит каждый матч команды в унифицированном формате, включая перенесенные и отмененные встречи.

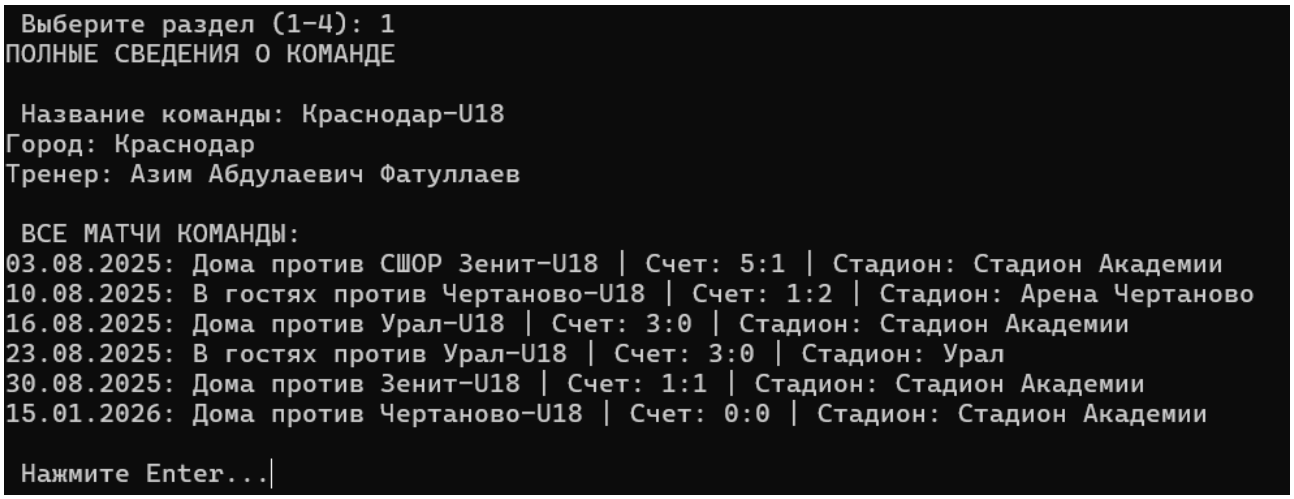

Рис. 17. Вывод полных сведений о команде и её матчах

В подменю введем 2. Программа формирует структурированную таблицу со статистикой использования стадионов.

```
 Выберите раздел (1-4): 2
СВЕДЕНИЯ О СТАДИОНАХ

 ИНФОРМАЦИЯ О СТАДИОНАХ:
Стадион                  Город          Вместимость  Матчей
Стадион Академии         Краснодар      7458         4
Арена Чертаново          Москва         490          1
Урал                     Екатеринбург   35000        1

 Нажмите Enter...
```

Рис. 18. Табличный вывод сведений о стадионах

В подменю введем 3. Это самый комплексный отчет в системе. Его формирование проходит в несколько этапов. Этап 1: Выбор стадиона. Программа запрашивает у пользователя выбор, выводя пронумерованный список всех стадионов из базы данных. Этап 2: Формирование отчета. После выбора стадиона программа выполняет серию запросов к БД и генерирует отчет.

```
 Выберите раздел (1-4): 3

 ОТЧЕТ ПО ИГРАМ НА СТАДИОНЕ

 ВЫБЕРИТЕ СТАДИОН:
1. Арена Чертаново (Москва)
2. Зенит Арена (Санкт-Петербург)
3. Спартак (Москва)
4. Стадион Академии (Краснодар)
5. Урал (Екатеринбург)

 Выберите номер стадиона: 4

 СТАТИСТИКА НА СТАДИОНЕ 'Стадион Академии':
   Всего матчей: 4
   Побед хозяев: 2
   Побед гостей: 0
   Ничьих: 1

 ИГРОКИ, ЗАБИВАВШИЕ МЯЧИ:
    Краснодар-U18:
      • Андрей Петрович Королёв - 3 голов
      • Андрей Викторович Полевой - 2 голов
      • Кирилл Сергеевич Леконцев - 1 голов
      • Степан Дмитриевич Садчиков - 1 голов

 Нажмите Enter...
```

Рис. 19. Пример комплексного отчета по стадиону

В главном меню введём 6. Откроется меню администратора для внесения изменений.

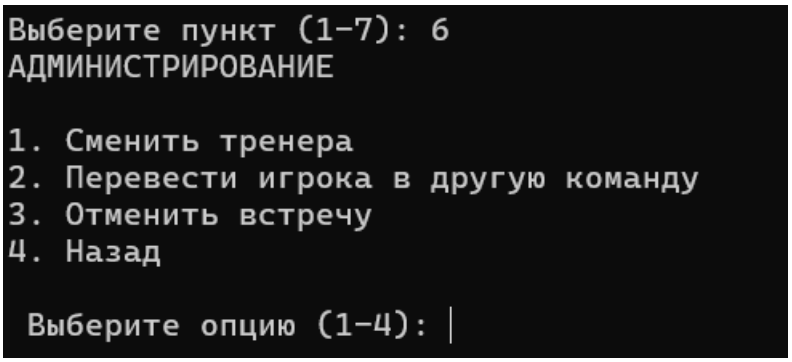

Рис. 20. Меню администратора

При выборе 1 программа покажет текущего тренера, запросит ФИО нового и после подтверждения обновит запись в базе данных.

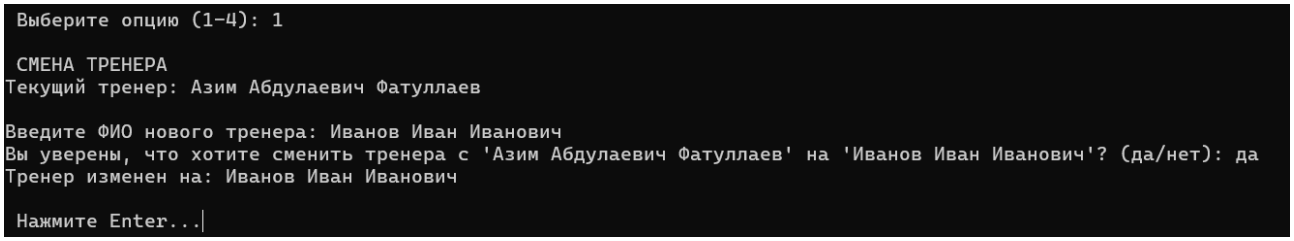

Рис. 21. Смена главного тренера команды

Для проверки повторно выведем всю информацию о команде. Заметим изменения главного тренера.

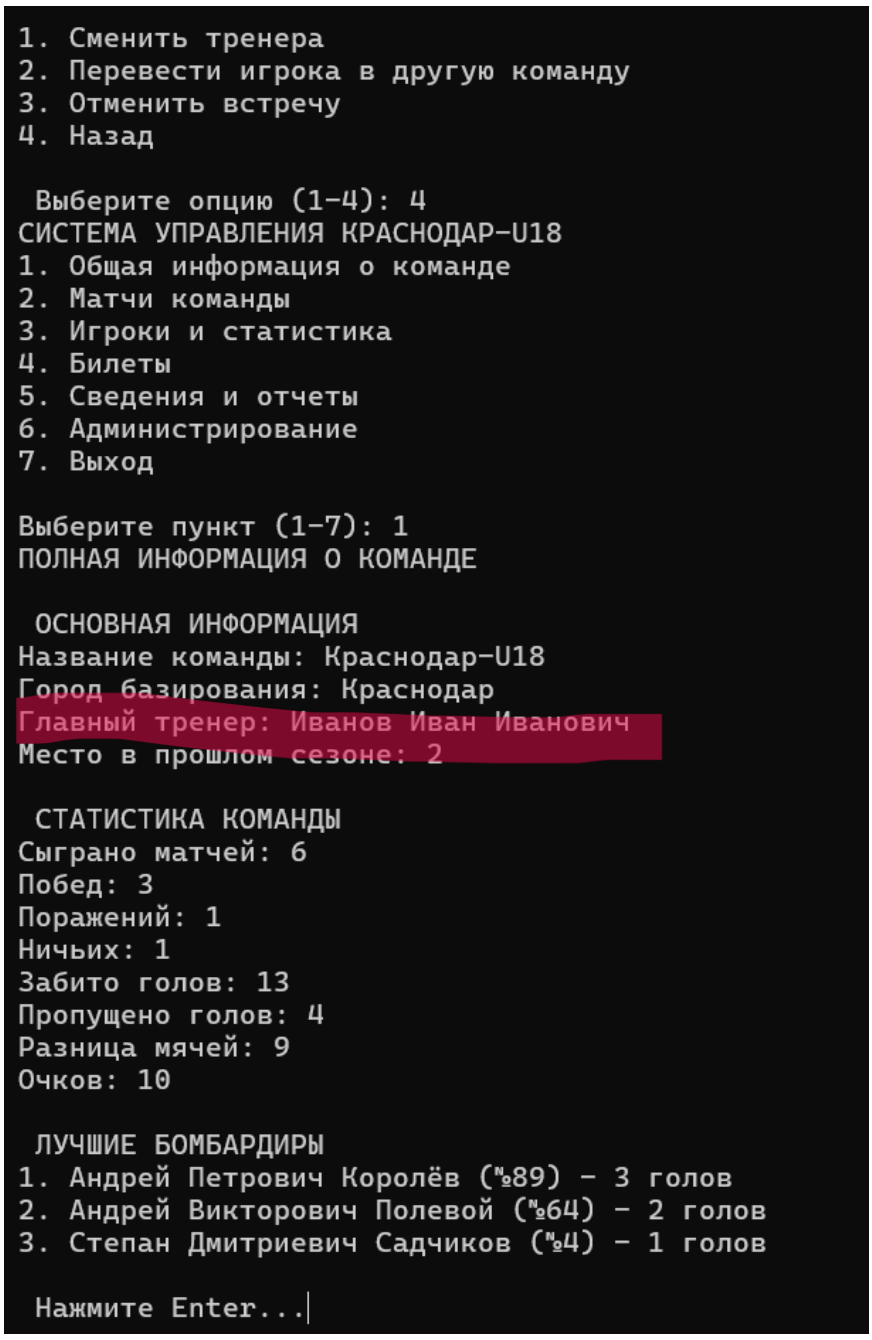

Рис. 22. Измененная информация о команде

При выборе 2 программа проведёт пользователя через процесс перевода: выбор игрока из списка, выбор новой команды, подтверждение операции с обновлением всех связанных данных.

```
 Выберите опцию (1-4): 2

 ПЕРЕВОД ИГРОКА В ДРУГУЮ КОМАНДУ

 ИГРОКИ КРАСНОДАР-U18:
1. Андрей Викторович Полевой (№64, Полузащитник)
2. Андрей Олегович Карпенко (№99, Вратарь)
3. Андрей Петрович Королёв (№89, Нападающий)
4. Артём Владимирович Дряхлов (№90, Нападающий)
5. Артём Дмитриевич Кульбаев (№96, Полузащитник)
6. Богдан Иванович Кудзаву (№3, Защитник)
7. Виктор Петрович Шипуль (№5, Защитник)
8. Даниэль Артемович Амбарцумов (№7, Нападающий)
9. Дмитрий Викторович Ачкинази (№91, Вратарь)
10. Евгений Михайлович Дедяев (№61, Защитник)
11. Иван Александрович Федько (№88, Вратарь)
12. Иван Сергеевич Акманов (№14, Защитник)
13. Кирилл Сергеевич Леконцев (№9, Нападающий)
14. Максим Игоревич Пономаренко (№42, Полузащитник)
15. Роман Андреевич Сумачёв (№11, Нападающий)
16. Сергей Алексеевич Тройчин (№21, Защитник)
17. Степан Дмитриевич Садчиков (№4, Защитник)
18. Ярослав Олегович Крючков (№8, Полузащитник)

 Выберите номер игрока для перевода: 1

 КУДА ПЕРЕВЕСТИ ИГРОКА Андрей Викторович Полевой?
1. Зенит-U18 (Санкт-Петербург)
2. СШОР Зенит-U18 (Санкт-Петербург)
3. Урал-U18 (Екатеринбург)
4. Чертаново-U18 (Москва)

 Выберите номер команды: 1

 ВНИМАНИЕ: Вы собираетесь перевести игрока:
   Игрок: Андрей Викторович Полевой
   ИЗ: Краснодар-U18
   В: Зенит-U18
Вы уверены? (да/нет): да
Игрок Андрей Викторович Полевой успешно переведен в Зенит-U18

 Нажмите Enter...
```

Рис. 23. Процесс перевода игрока в другую команду

Для проверки повторно выведем полный состав команды. Заметим, что игрок Андрей Полевой отсутствует.

```
Выберите (1-4): 1

ВСЕ ИГРОКИ КРАСНОДАР-U18

Всего игроков: 23
Игрок                        №    Позиция          М    Г    Мин
Андрей Петрович Королёв      89   Нападающий       3    3    86.7
Андрей Олегович Карпенко     99   Вратарь          2    1    46.5
Степан Дмитриевич Садчиков   4    Защитник         1    1    90.0
Кирилл Сергеевич Леконцев    9    Нападающий       1    1    74.0
Богдан Иванович Кудзаву      3    Защитник         3    0    34.0
Иван Сергеевич Акманов       14   Защитник         3    0    42.7
Сергей Алексеевич Тройчин    21   Защитник         3    0    16.7
Ярослав Олегович Крючков     8    Полузащитник     3    0    60.3
Максим Игоревич Пономаренко  42   Полузащитник     3    0    64.3
Даниэль Артемович Амбарцумов 7    Нападающий       3    0    21.3
Роман Андреевич Сумачёв      11   Нападающий       3    0    87.7
Артём Владимирович Дряхлов   90   Нападающий       3    0    82.3
Иван Александрович Федько    88   Вратарь          2    0    90.0
Виктор Петрович Шипуль       5    Защитник         2    0    15.5
Евгений Михайлович Дедяев    61   Защитник         2    0    90.0
Дмитрий Викторович Ачкинази  91   Вратарь          1    0    90.0
Артём Дмитриевич Кульбаев    96   Полузащитник     1    0    23.0
Марк Сергеевич Федотов       1    Вратарь          0    0    0.0
Вадим Андреевич Матвеенко    6    Защитник         0    0    0.0
Никита Владимирович Закарлюка 50  Защитник         0    0    0.0
Ефим Александрович Буркин    15   Полузащитник     0    0    0.0
Андрей Викторович Полевой    64   Полузащитник     0    0    0.0
Эльдар Раминович Гусейнов    97   Нападающий       0    0    0.0

Нажмите Enter...
```

Рис. 24. Измененный состав команды

При выборе 3 программа выведет список будущих матчей, позволит выбрать один и после подтверждения полностью удалит его из базы данных вместе со связанной статистикой и информацией о билетах.

```
Выберите пункт (1-7): 6
АДМИНИСТРИРОВАНИЕ

1. Сменить тренера
2. Перевести игрока в другую команду
3. Отменить встречу
4. Назад

 Выберите опцию (1-4): 3

 ОТМЕНА ВСТРЕЧИ

 ВЫБЕРИТЕ ВСТРЕЧУ ДЛЯ ОТМЕНЫ:
1. 15.01.2026: Краснодар-U18 - Чертаново-U18 на Стадион Академии

 Выберите номер встречи: 1

 ВНИМАНИЕ: Вы собираетесь отменить встречу:
   Дата: 15.01.2026
   Матч: Краснодар-U18 - Чертаново-U18
Вы уверены? (да/нет): да
Встреча успешно отменена

 Нажмите Enter...
```

Рис. 25. Процедура отмены запланированной встречи

Для проверки вновь выведем информацию о билетах на предстоящие матчи. Заметим, что данные о билетах отсутствуют.

```
 Выберите опцию (1-4): 4
СИСТЕМА УПРАВЛЕНИЯ КРАСНОДАР-U18
1. Общая информация о команде
2. Матчи команды
3. Игроки и статистика
4. Билеты
5. Сведения и отчеты
6. Администрирование
7. Выход

Выберите пункт (1-7): 4
БИЛЕТЫ НА МАТЧИ
Нет данных о билетах

 Нажмите Enter...
```

Рис. 26. Измененные данные о билетах

В главном меню введём 7. Программа закроет соединение с базой данных, выведет сообщение о завершении работы и завершит своё выполнение.

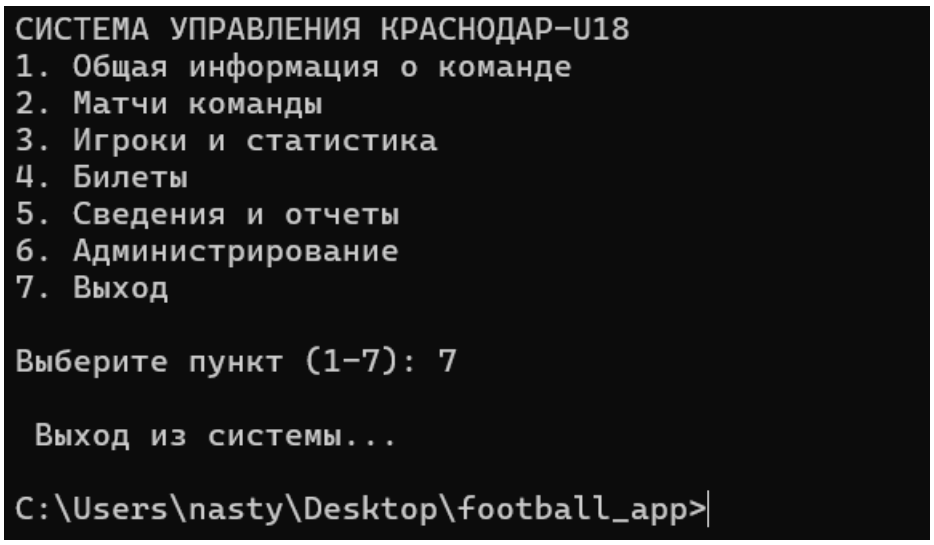

Рис. 27. Завершение работы программы

ПРИЛОЖЕНИЕ Б

ЛИСТИНГ ПРОГРАММНОГО КОДА

SQL-скрипт создания базы данных

```
DROP DATABASE IF EXISTS krasnodar_football;
CREATE DATABASE krasnodar_football;
USE krasnodar_football;

CREATE TABLE stadiums (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    city VARCHAR(50) NOT NULL,
    capacity INT NOT NULL
);

CREATE TABLE teams (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    city VARCHAR(50) NOT NULL,
    coach VARCHAR(100) NOT NULL,
    last_season_place INT NOT NULL
);

CREATE TABLE players (
    id INT PRIMARY KEY AUTO_INCREMENT,
    full_name VARCHAR(100) NOT NULL,
    number INT,
    position VARCHAR(30) NOT NULL
);

CREATE TABLE matches (
```

```
    id INT PRIMARY KEY AUTO_INCREMENT,
    match_date DATETIME NOT NULL,
    status ENUM('запланирован','перенесен','отменен') DEFAULT 'запланирован',
    home_team_id INT NOT NULL,
    away_team_id INT NOT NULL,
    stadium_id INT NOT NULL,
    home_score INT,
    away_score INT,
    FOREIGN KEY (home_team_id) REFERENCES teams(id),
    FOREIGN KEY (away_team_id) REFERENCES teams(id),
    FOREIGN KEY (stadium_id) REFERENCES stadiums(id)
) ENGINE=InnoDB;

ALTER TABLE matches
ADD COLUMN match_day DATE
GENERATED ALWAYS AS (DATE(match_date)) STORED;

ALTER TABLE matches
ADD CONSTRAINT one_match_per_day
UNIQUE (match_day, home_team_id);

CREATE TABLE player_stats (
    id INT PRIMARY KEY AUTO_INCREMENT,
    player_id INT NOT NULL,
    match_id INT NOT NULL,
    team_id INT NOT NULL,
    goals INT DEFAULT 0,
    assists INT DEFAULT 0,
    yellow_cards INT DEFAULT 0,
    minutes_played INT,
```

```
    FOREIGN KEY (player_id) REFERENCES players(id),
    FOREIGN KEY (match_id) REFERENCES matches(id),
    FOREIGN KEY (team_id) REFERENCES teams(id)
);

CREATE TABLE tickets (
    id INT PRIMARY KEY AUTO_INCREMENT,
    match_id INT NOT NULL,
    category VARCHAR(20) DEFAULT 'обычный',
    price DECIMAL(10,2) NOT NULL,
    available INT DEFAULT 100,
    FOREIGN KEY (match_id) REFERENCES matches(id)
);


INSERT INTO stadiums (name, city, capacity) VALUES
('Стадион Академии', 'Краснодар', 7458),
('Арена Чертаново', 'Москва', 490),
('Урал', 'Екатеринбург', 35000),
('Спартак', 'Москва', 45360),
('Зенит Арена', 'Санкт-Петербург', 64468);

INSERT INTO teams (name, city, coach, last_season_place) VALUES
('Краснодар-U18', 'Краснодар', 'Азим Абдулаевич Фатуллаев', 2),
('СШОР Зенит-U18', 'Санкт-Петербург', 'Иван Петрович Сидоров', 5),
('Чертаново-U18', 'Москва', 'Сергей Владимирович Иванов', 3),
('Урал-U18', 'Екатеринбург', 'Александр Николаевич Петров', 9),
('Зенит-U18', 'Санкт-Петербург', 'Дмитрий Сергеевич Смирнов', 1);

INSERT INTO players (full_name, number, position) VALUES
```

('Марк Сергеевич Федотов', 1, 'Вратарь'),
('Иван Александрович Федько', 88, 'Вратарь'),
('Дмитрий Викторович Ачкинази', 91, 'Вратарь'),
('Андрей Олегович Карпенко', 99, 'Вратарь'),

('Богдан Иванович Кудзаву', 3, 'Защитник'),
('Степан Дмитриевич Садчиков', 4, 'Защитник'),
('Виктор Петрович Шипуль', 5, 'Защитник'),
('Вадим Андреевич Матвеенко', 6, 'Защитник'),
('Иван Сергеевич Акманов', 14, 'Защитник'),
('Сергей Алексеевич Тройчин', 21, 'Защитник'),
('Никита Владимирович Закарлюка', 50, 'Защитник'),
('Евгений Михайлович Дедяев', 61, 'Защитник'),

('Ярослав Олегович Крючков', 8, 'Полузащитник'),
('Ефим Александрович Буркин', 15, 'Полузащитник'),
('Максим Игоревич Пономаренко', 42, 'Полузащитник'),
('Андрей Викторович Полевой', 64, 'Полузащитник'),
('Артём Дмитриевич Кульбаев', 96, 'Полузащитник'),

('Даниэль Артемович Амбарцумов', 7, 'Нападающий'),
('Кирилл Сергеевич Леконцев', 9, 'Нападающий'),
('Роман Андреевич Сумачёв', 11, 'Нападающий'),
('Андрей Петрович Королёв', 89, 'Нападающий'),
('Артём Владимирович Дряхлов', 90, 'Нападающий'),
('Эльдар Раминович Гусейнов', 97, 'Нападающий');

INSERT INTO matches (match_date, home_team_id, away_team_id, stadium_id, home_score, away_score) VALUES
('2025-08-03 20:00:00', 1, 2, 1, 5, 1),

('2025-08-10 17:00:00', 3, 1, 2, 2, 1),
('2025-08-16 20:00:00', 1, 4, 1, 3, 0),
('2025-08-23 12:00:00', 4, 1, 3, 0, 3),
('2025-08-30 20:00:00', 1, 5, 1, 1, 1);

INSERT INTO matches (match_date, home_team_id, away_team_id, stadium_id, home_score, away_score) VALUES
('2026-01-15 19:00:00', 1, 3, 1, NULL, NULL);

INSERT INTO player_stats (player_id, match_id, team_id, goals, assists, minutes_played) VALUES
(2, 1, 1, 0, 0, 90),
(6, 1, 1, 1, 0, 90),
(16, 1, 1, 1, 0, 77),
(19, 1, 1, 1, 0, 74),
(21, 1, 1, 2, 0, 90),
(15, 1, 1, 0, 0, 62),
(22, 1, 1, 0, 0, 67),
(20, 1, 1, 0, 0, 88),
(5, 1, 1, 0, 0, 8),
(13, 1, 1, 0, 0, 8),
(10, 1, 1, 0, 0, 13),
(9, 1, 1, 0, 0, 16),
(18, 1, 1, 0, 0, 28),
(17, 1, 1, 0, 0, 23),

(2, 2, 1, 0, 0, 90),
(12, 2, 1, 0, 0, 90),
(13, 2, 1, 0, 0, 83),
(4, 2, 1, 1, 0, 90),

(16, 2, 1, 0, 0, 76),
(15, 2, 1, 0, 0, 64),
(21, 2, 1, 0, 0, 90),
(22, 2, 1, 0, 0, 90),
(20, 2, 1, 0, 0, 90),
(5, 2, 1, 0, 0, 7),
(10, 2, 1, 0, 0, 14),
(7, 2, 1, 0, 0, 26),
(9, 2, 1, 0, 0, 22),
(18, 2, 1, 0, 0, 26),

(3, 3, 1, 0, 0, 90),
(5, 3, 1, 0, 0, 87),
(12, 3, 1, 0, 0, 90),
(13, 3, 1, 0, 0, 90),
(9, 3, 1, 0, 0, 90),
(16, 3, 1, 1, 0, 90),
(15, 3, 1, 0, 0, 67),
(21, 3, 1, 1, 0, 80),
(22, 3, 1, 0, 0, 90),
(20, 3, 1, 0, 0, 85),
(10, 3, 1, 0, 0, 23),
(7, 3, 1, 0, 0, 5),
(4, 3, 1, 0, 0, 3),
(18, 3, 1, 0, 0, 10);

INSERT INTO tickets (match_id, category, price, available) VALUES
(1, 'обычный', 500.00, 3000),
(1, 'VIP', 1500.00, 100),
(3, 'обычный', 500.00, 3000),

```
(3, 'VIP', 1500.00, 100),
(5, 'обычный', 600.00, 3000),
(5, 'VIP', 1800.00, 100);

INSERT INTO tickets (match_id, category, price, available) VALUES
(6, 'обычный', 700.00, 2500),
(6, 'VIP', 2000.00, 80);



CREATE VIEW krasnodar_matches AS
SELECT
   m.match_date as дата,
   CASE
      WHEN m.home_team_id = 1 THEN 'Дома'
      ELSE 'В гостях'
   END as статус,
   CASE
      WHEN m.home_team_id = 1 THEN a.name
      ELSE h.name
   END as противник,
   CASE
      WHEN m.home_team_id = 1 THEN m.home_score
      ELSE m.away_score
   END as голы_краснодара,
   CASE
      WHEN m.home_team_id = 1 THEN m.away_score
      ELSE m.home_score
   END as голы_противника,
   s.name as стадион
FROM matches m
```

```
JOIN teams h ON m.home_team_id = h.id
JOIN teams a ON m.away_team_id = a.id
JOIN stadiums s ON m.stadium_id = s.id
WHERE m.home_team_id = 1 OR m.away_team_id = 1
ORDER BY m.match_date;

CREATE VIEW player_stats_view AS
SELECT
    p.full_name as игрок,
    p.number as номер,
    p.position as позиция,
    COUNT(ps.match_id) as матчей,
    SUM(ps.goals) as голов,
    SUM(ps.assists) as передач,
    SUM(ps.yellow_cards) as жк,
    ROUND(AVG(ps.minutes_played), 1) as средние_минуты
FROM players p
LEFT JOIN player_stats ps ON p.id = ps.player_id AND ps.team_id = 1
GROUP BY p.id, p.full_name, p.number, p.position
ORDER BY голов DESC, матчей DESC;

CREATE VIEW top_scorers AS
SELECT
    p.full_name as игрок,
    p.number as номер,
    SUM(ps.goals) as голов
FROM player_stats ps
JOIN players p ON ps.player_id = p.id
WHERE ps.team_id = 1
GROUP BY p.id, p.full_name, p.number
```

```
HAVING SUM(ps.goals) > 0
ORDER BY голов DESC;

CREATE OR REPLACE VIEW home_tickets AS
SELECT
    m.match_date AS дата_матча,
    a.name AS противник,
    s.name AS стадион,
    tkt.category AS категория,
    CASE
        WHEN tm.last_season_place <= 3 AND s.capacity > 30000 THEN 1800
        WHEN tm.last_season_place <= 3 THEN 1500
        WHEN tm.last_season_place >= 15 THEN 500
        ELSE 800
    END AS цена,
    tkt.available AS доступно
FROM matches m
JOIN teams tm ON m.home_team_id = tm.id
JOIN teams a ON m.away_team_id = a.id
JOIN stadiums s ON m.stadium_id = s.id
JOIN tickets tkt ON m.id = tkt.match_id
WHERE m.match_date > NOW()
  AND m.status = 'запланирован'
ORDER BY m.match_date;
```

Код клиентского приложения

```
import mysql.connector
from mysql.connector import Error
from datetime import datetime
```

```
class FootballDB:
    def __init__(self):
        self.connection = None
        self.connect()

    def connect(self):
        try:
            self.connection = mysql.connector.connect(
                host='localhost',
                database='krasnodar_football',
                user='root',
                password='vino46060'
            )
            if self.connection.is_connected():
                print("Подключено к базе данных")
                return True
        except Error as e:
            print(f"Ошибка подключения: {e}")
            return False

    def execute_query(self, query, params=None, fetch=False):
        try:
            cursor = self.connection.cursor()
            if params:
                cursor.execute(query, params)
            else:
                cursor.execute(query)
            if fetch:
                result = cursor.fetchall()
                cursor.close()
```

```
            return result
        else:
            self.connection.commit()
            cursor.close()
            return True
    except Error as e:
        print(f"Ошибка запроса: {e}")
        return None

def safe_int(self, value):
    if value is None:
        return 0
    try:
        return int(value)
    except:
        return 0

def safe_float(self, value):
    if value is None:
        return 0.0
    try:
        return float(value)
    except:
        return 0.0

def safe_str(self, value):
    if value is None:
        return "Нет данных"
    return str(value)
```

```
def format_date(self, date_value):
    if isinstance(date_value, datetime):
        return date_value.strftime("%d.%m.%Y")
    return self.safe_str(date_value)

def confirm_action(self, message):
    while True:
        response = input(f"{message} (да/нет): ").strip().lower()
        if response in ['да', 'yes', 'y', 'д']:
            return True
        elif response in ['нет', 'no', 'n', 'н']:
            return False
        else:
            print("Пожалуйста, введите 'да' или 'нет'")

def show_menu(self):
    while True:
        print("СИСТЕМА УПРАВЛЕНИЯ КРАСНОДАР-U18")
        print("1. Общая информация о команде")
        print("2. Матчи команды")
        print("3. Игроки и статистика")
        print("4. Билеты")
        print("5. Сведения и отчеты")
        print("6. Администрирование")
        print("7. Выход")
        choice = input("\nВыберите пункт (1-7): ").strip()
        if choice == '1':
            self.show_team_info()
        elif choice == '2':
            self.show_matches_menu()
```

```
        elif choice == '3':
            self.show_players_menu()
        elif choice == '4':
            self.show_tickets()
        elif choice == '5':
            self.show_reports_menu()
        elif choice == '6':
            self.show_admin_menu()
        elif choice == '7':
            print("\n Выход из системы...")
            if self.connection:
                self.connection.close()
            break
        else:
            print("Неверный выбор")

def show_team_info(self):
    print("ПОЛНАЯ ИНФОРМАЦИЯ О КОМАНДЕ")
    print("\n ОСНОВНАЯ ИНФОРМАЦИЯ")
    query = """
    SELECT name, city, coach, last_season_place
    FROM teams
    WHERE name = 'Краснодар-U18'
    """
    result = self.execute_query(query, fetch=True)
    if result:
     print(f"Название команды: {self.safe_str(result[0][0])}")
     print(f"Город базирования: {self.safe_str(result[0][1])}")
     print(f"Главный тренер: {self.safe_str(result[0][2])}")
     print(f"Место в прошлом сезоне: {self.safe_int(result[0][3])}")
```

```
        print("\n СТАТИСТИКА КОМАНДЫ")
        stats_query = """
        SELECT
            COUNT(*) as total_matches,
            SUM(CASE WHEN home_team_id = 1 AND home_score > away_score
THEN 1
                    WHEN away_team_id = 1 AND away_score > home_score THEN 1
ELSE 0 END) as wins,
            SUM(CASE WHEN home_team_id = 1 AND home_score < away_score
THEN 1
                    WHEN away_team_id = 1 AND away_score < home_score THEN 1
ELSE 0 END) as losses,
            SUM(CASE WHEN home_score = away_score THEN 1 ELSE 0 END) as
draws,
            SUM(CASE WHEN home_team_id = 1 THEN home_score ELSE
away_score END) as goals_for,
            SUM(CASE WHEN home_team_id = 1 THEN away_score ELSE
home_score END) as goals_against
        FROM matches
        WHERE home_team_id = 1 OR away_team_id = 1
        """
        stats = self.execute_query(stats_query, fetch=True)
        if stats and stats[0]:
            s = stats[0]
            matches = self.safe_int(s[0])
            wins = self.safe_int(s[1])
            losses = self.safe_int(s[2])
            draws = self.safe_int(s[3])
            goals_for = self.safe_int(s[4])
            goals_against = self.safe_int(s[5])
```

```
    diff = goals_for - goals_against
    points = wins * 3 + draws
    print(f"Сыграно матчей: {matches}")
    print(f"Побед: {wins}")
    print(f"Поражений: {losses}")
    print(f"Ничьих: {draws}")
    print(f"Забито голов: {goals_for}")
    print(f"Пропущено голов: {goals_against}")
    print(f"Разница мячей: {diff}")
    print(f"Очков: {points}")
else:
    print("Нет статистики команды")
print("\n ЛУЧШИЕ БОМБАРДИРЫ")
scorers_query = """
SELECT p.full_name, p.number, SUM(ps.goals) as goals
FROM player_stats ps
JOIN players p ON ps.player_id = p.id
WHERE ps.team_id = 1
GROUP BY p.id
HAVING SUM(ps.goals) > 0
ORDER BY SUM(ps.goals) DESC
LIMIT 3
"""
scorers = self.execute_query(scorers_query, fetch=True)
if scorers:
    for i, s in enumerate(scorers, 1):
        name = self.safe_str(s[0])
        number = self.safe_int(s[1])
        goals = self.safe_int(s[2])
        print(f"{i}. {name} (№{number}) - {goals} голов")
```

```
    input("\n Нажмите Enter...")

def show_matches_menu(self):
    print("МАТЧИ КОМАНДЫ")
    while True:
        print("\n1. Все матчи")
        print("2. Домашние матчи")
        print("3. Выездные матчи")
        print("4. Назад")
        choice = input("\n Выберите (1-4): ").strip()
        if choice == '1':
            self.show_all_matches()
        elif choice == '2':
            self.show_home_matches()
        elif choice == '3':
            self.show_away_matches()
        elif choice == '4':
            break
        else:
            print("Неверный выбор")

def show_all_matches(self):
    print("\n ВСЕ МАТЧИ КОМАНДЫ")
    query = "SELECT * FROM krasnodar_matches"
    result = self.execute_query(query, fetch=True)
    if result:
        print(f"\nВсего матчей: {len(result)}")
        for row in result:
            date_str = self.format_date(row[0])
            status = self.safe_str(row[1])
```

```
            opponent = self.safe_str(row[2])
            our_goals = self.safe_int(row[3])
            their_goals = self.safe_int(row[4])
            stadium = self.safe_str(row[5])
            print(f"Дата: {date_str}")
            print(f"Статус: {status}")
            print(f"Противник: {opponent}")
            print(f"Счет: {our_goals}:{their_goals}")
            print(f"Стадион: {stadium}")
    else:
        print("Нет данных о матчах")
    input("\n Нажмите Enter...")

def show_home_matches(self):
    print("\n ДОМАШНИЕ МАТЧИ")
    query = """
    SELECT m.match_date, a.name, m.home_score, m.away_score, s.name
    FROM matches m
    JOIN teams a ON m.away_team_id = a.id
    JOIN stadiums s ON m.stadium_id = s.id
    WHERE m.home_team_id = 1
    ORDER BY m.match_date
    """
    result = self.execute_query(query, fetch=True)
    if result:
        print(f"\nДомашних матчей: {len(result)}")
        for row in result:
            date_str = self.format_date(row[0])
            opponent = self.safe_str(row[1])
            our_goals = self.safe_int(row[2])
```

```
            their_goals = self.safe_int(row[3])
            stadium = self.safe_str(row[4])
            print(f"Дата: {date_str}")
            print(f"Противник: {opponent}")
            print(f"Счет: {our_goals}:{their_goals}")
            print(f"Стадион: {stadium}")
    input("\n Нажмите Enter...")

def show_away_matches(self):
    print("\n ВЫЕЗДНЫЕ МАТЧИ")
    query = """
    SELECT m.match_date, h.name, m.away_score, m.home_score, s.name
    FROM matches m
    JOIN teams h ON m.home_team_id = h.id
    JOIN stadiums s ON m.stadium_id = s.id
    WHERE m.away_team_id = 1
    ORDER BY m.match_date
    """
    result = self.execute_query(query, fetch=True)
    if result:
        print(f"\nВыездных матчей: {len(result)}")
        for row in result:
            date_str = self.format_date(row[0])
            opponent = self.safe_str(row[1])
            our_goals = self.safe_int(row[2])
            their_goals = self.safe_int(row[3])
            stadium = self.safe_str(row[4])
            print(f"Дата: {date_str}")
            print(f"Противник: {opponent}")
            print(f"Счет: {our_goals}:{their_goals}")
```

```
            print(f"Стадион: {stadium}")
    input("\n Нажмите Enter...")

def show_players_menu(self):
    print("ИГРОКИ И СТАТИСТИКА")
    while True:
        print("\n1. Все игроки")
        print("2. Поиск игрока")
        print("3. Топ-5 бомбардиров")
        print("4. Назад")
        choice = input("\n Выберите (1-4): ").strip()
        if choice == '1':
            self.show_all_players()
        elif choice == '2':
            self.search_player()
        elif choice == '3':
            self.show_top_scorers()
        elif choice == '4':
            break
        else:
            print("Неверный выбор")

def show_all_players(self):
    print("\n ВСЕ ИГРОКИ КРАСНОДАР-U18")
    query = """
    SELECT p.full_name, p.number, p.position,
           COUNT(ps.match_id) as matches,
           SUM(ps.goals) as goals,
           ROUND(AVG(ps.minutes_played), 1) as avg_minutes
    FROM players p
```

```
        LEFT JOIN player_stats ps ON p.id = ps.player_id AND ps.team_id = 1
        GROUP BY p.id, p.full_name, p.number, p.position
        ORDER BY goals DESC, matches DESC
        """
        result = self.execute_query(query, fetch=True)
        if result:
            print(f"\n Всего игроков: {len(result)}")
            print(f"{'Игрок':<30} {'№':<4} {'Позиция':<15} {'М':<4} {'Г':<4}
{'Мин':<6}")
            for row in result:
                player_name = self.safe_str(row[0])
                number = self.safe_int(row[1])
                position = self.safe_str(row[2])
                matches = self.safe_int(row[3])
                goals = self.safe_int(row[4])
                minutes = self.safe_float(row[5])
                if len(player_name) > 29:
                    display_name = player_name[:27] + ".."
                else:
                    display_name = player_name
                print(f"{display_name:<30} {number:<4} {position:<15} {matches:<4}
{goals:<4} {minutes:<6.1f}")
        else:
            print("Нет данных об игроках")
        input("\n Нажмите Enter...")

    def search_player(self):
        print("\n ПОИСК ИГРОКА")
        name = input("Введите имя или фамилию игрока: ").strip()
        if not name:
```

```
        print("Введите имя игрока")
        return
    query = """
    SELECT p.full_name, p.number, p.position,
           COUNT(ps.match_id) as matches,
           SUM(ps.goals) as goals,
           ROUND(AVG(ps.minutes_played), 1) as avg_minutes
    FROM players p
    LEFT JOIN player_stats ps ON p.id = ps.player_id AND ps.team_id = 1
    WHERE p.full_name LIKE %s
    GROUP BY p.id, p.full_name, p.number, p.position
    """
    result = self.execute_query(query, (f"%{name}%",), fetch=True)
    if result:
        print(f"\nНайдено игроков: {len(result)}")
        for row in result:
            print(f"Игрок: {self.safe_str(row[0])}")
            print(f"Номер: {self.safe_int(row[1])}")
            print(f"Позиция: {self.safe_str(row[2])}")
            print(f"Матчей: {self.safe_int(row[3])}")
            print(f"Голов: {self.safe_int(row[4])}")
            print(f"Минут: {self.safe_float(row[5]):.1f}")
    else:
        print("Игрок не найден")
    input("\n Нажмите Enter...")

def show_top_scorers(self):
    print("\n ТОП-5 БОМБАРДИРОВ")
    query = "SELECT * FROM top_scorers"
    result = self.execute_query(query, fetch=True)
```

```
    if result:
        print(f"\n ТОП-{len(result)} БОМБАРДИРОВ")
        print(f"{'№':<3} {'Игрок':<25} {'Номер':<6} {'Голов':<8}")
        for i, row in enumerate(result, 1):
            player_name = self.safe_str(row[0])
            number = self.safe_int(row[1])
            goals = self.safe_int(row[2])
            print(f"{i:<3} {player_name:<25} {number:<6} {goals:<8}")
    else:
        print("Нет данных о бомбардирах")
    input("\n Нажмите Enter...")

def show_tickets(self):
    print("БИЛЕТЫ НА МАТЧИ")
    query = "SELECT * FROM home_tickets"
    result = self.execute_query(query, fetch=True)
    if result:
        print(f"\n Доступные билеты:")
        for row in result:
            date_str = self.format_date(row[0])
            opponent = self.safe_str(row[1])
            stadium = self.safe_str(row[2])
            category = self.safe_str(row[3])
            price = self.safe_float(row[4])
            available = self.safe_int(row[5])
            print(f"\n Дата матча: {date_str}")
            print(f"Противник: {opponent}")
            print(f"Стадион: {stadium}")
            print(f"Категория: {category}")
            print(f"Цена: {price:.2f} руб.")
```

```
            print(f"Доступно: {available} билетов")
        else:
            print("Нет данных о билетах")
        input("\n Нажмите Enter...")

    def show_reports_menu(self):
        print("СВЕДЕНИЯ И ОТЧЕТЫ")
        while True:
            print("\n1. Сведения о команде")
            print("2. Сведения о стадионах")
            print("3. Отчет по играм на стадионе")
            print("4. Назад")
            choice = input("\n Выберите раздел (1-4): ").strip()
            if choice == '1':
                self.show_full_team_info()
            elif choice == '2':
                self.show_stadiums_info()
            elif choice == '3':
                self.show_stadium_report()
            elif choice == '4':
                break
            else:
                print("Неверный выбор")

    def show_full_team_info(self):
        print("ПОЛНЫЕ СВЕДЕНИЯ О КОМАНДЕ")
        query = "SELECT name, city, coach FROM teams WHERE name =
'Краснодар-U18'"
        result = self.execute_query(query, fetch=True)
        if result:
```

```
            print(f"\n Название команды: {self.safe_str(result[0][0])}")
            print(f"Город: {self.safe_str(result[0][1])}")
            print(f"Тренер: {self.safe_str(result[0][2])}")
        query = "SELECT * FROM krasnodar_matches"
        matches = self.execute_query(query, fetch=True)
        if matches:
            print(f"\n ВСЕ МАТЧИ КОМАНДЫ:")
            for match in matches:
                date_str = self.format_date(match[0])
                status = self.safe_str(match[1])
                opponent = self.safe_str(match[2])
                our_goals = self.safe_int(match[3])
                their_goals = self.safe_int(match[4])
                stadium = self.safe_str(match[5])

                print(f"{date_str}: {status} против {opponent} | Счет:
{our_goals}:{their_goals} | Стадион: {stadium}")
        input("\n Нажмите Enter...")

    def show_stadiums_info(self):
        print("СВЕДЕНИЯ О СТАДИОНАХ")
        query = """
        SELECT s.name, s.city, s.capacity, COUNT(m.id) as matches_count
        FROM stadiums s
        LEFT JOIN matches m ON s.id = m.stadium_id
        WHERE m.home_team_id = 1 OR m.away_team_id = 1
        GROUP BY s.id, s.name, s.city, s.capacity
        ORDER BY matches_count DESC
        """
        result = self.execute_query(query, fetch=True)
```

```
        if result:
            print("\n ИНФОРМАЦИЯ О СТАДИОНАХ:")
            print(f"{'Стадион':<25} {'Город':<15} {'Вместимость':<12}
{'Матчей':<8}")
            for row in result:
                stadium = self.safe_str(row[0])
                city = self.safe_str(row[1])
                capacity = self.safe_int(row[2])
                matches = self.safe_int(row[3])
                print(f"{stadium:<25} {city:<15} {capacity:<12} {matches:<8}")
        else:
            print("Нет данных о стадионах")
        input("\n Нажмите Enter...")

    def show_stadium_report(self):
        print("\n ОТЧЕТ ПО ИГРАМ НА СТАДИОНЕ")
        query = "SELECT id, name, city FROM stadiums ORDER BY name"
        stadiums = self.execute_query(query, fetch=True)
            if not stadiums:
            print("Нет данных о стадионах")
            input("\n Нажмите Enter...")
            return
        print("\n ВЫБЕРИТЕ СТАДИОН:")
        for i, s in enumerate(stadiums, 1):
            print(f"{i}. {s[1]} ({s[2]})")
        try:
            choice = int(input("\n Выберите номер стадиона: "))
            if 1 <= choice <= len(stadiums):
                stadium_id = stadiums[choice-1][0]
                stadium_name = stadiums[choice-1][1]
```

```
            query = """
            SELECT
                COUNT(*) as total_matches,
                SUM(CASE WHEN home_score > away_score THEN 1 ELSE 0 END)
as home_wins,
                SUM(CASE WHEN away_score > home_score THEN 1 ELSE 0 END)
as away_wins,
                SUM(CASE WHEN home_score = away_score THEN 1 ELSE 0 END)
as draws
            FROM matches
            WHERE stadium_id = %s
            """
            stats = self.execute_query(query, (stadium_id,), fetch=True)
            if stats and stats[0]:
                print(f"\n СТАТИСТИКА НА СТАДИОНЕ '{stadium_name}':")
                print(f"   Всего матчей: {self.safe_int(stats[0][0])}")
                print(f"   Побед хозяев: {self.safe_int(stats[0][1])}")
                print(f"   Побед гостей: {self.safe_int(stats[0][2])}")
                print(f"   Ничьих: {self.safe_int(stats[0][3])}")
            query = """
            SELECT DISTINCT p.full_name, t.name, SUM(ps.goals) as goals
            FROM player_stats ps
            JOIN players p ON ps.player_id = p.id
            JOIN matches m ON ps.match_id = m.id
            JOIN teams t ON ps.team_id = t.id
            WHERE m.stadium_id = %s AND ps.goals > 0
            GROUP BY p.id, t.id
            ORDER BY t.name, SUM(ps.goals) DESC
            """
            scorers = self.execute_query(query, (stadium_id,), fetch=True)
```

```
            if scorers:
                print(f"\n ИГРОКИ, ЗАБИВАВШИЕ МЯЧИ:")
                current_team = ""
                for scorer in scorers:
                    player_name = self.safe_str(scorer[0])
                    team_name = self.safe_str(scorer[1])
                    goals = self.safe_int(scorer[2])
                    if team_name != current_team:
                        print(f"    {team_name}:")
                        current_team = team_name
                    print(f"      • {player_name} - {goals} голов")
            else:
                print(f"\n На этом стадионе никто не забивал мячей")


        else:
            print("Неверный номер стадиона")
    except ValueError:
        print("Введите число")
    input("\n Нажмите Enter...")


def show_admin_menu(self):
    print("АДМИНИСТРИРОВАНИЕ")
    while True:
        print("\n1. Сменить тренера")
        print("2. Перевести игрока в другую команду")
        print("3. Отменить встречу")
        print("4. Назад")
        choice = input("\n Выберите опцию (1-4): ").strip()
        if choice == '1':
            self.change_coach()
```

```
        elif choice == '2':
            self.transfer_player()
        elif choice == '3':
            self.cancel_match()
        elif choice == '4':
            break
        else:
            print("Неверный выбор")

def change_coach(self):
    print("\n СМЕНА ТРЕНЕРА")
    query = "SELECT coach FROM teams WHERE name = 'Краснодар-U18'"
    result = self.execute_query(query, fetch=True)
    if result:
        current_coach = self.safe_str(result[0][0])
        print(f"Текущий тренер: {current_coach}")
    else:
        current_coach = "Неизвестен"
        print("Не удалось получить текущего тренера")
    new_coach = input("\nВведите ФИО нового тренера: ").strip()
    if not new_coach:
        print("Введите ФИО тренера")
        input("\n Нажмите Enter...")
        return
    if new_coach == current_coach:
        print("Новый тренер совпадает с текущим")
        if not self.confirm_action("Продолжить?"):
            print("Изменение отменено")
            input("\n Нажмите Enter...")
            return
```

```
        if self.confirm_action(f"Вы уверены, что хотите сменить тренера с
'{current_coach}' на '{new_coach}'?"):
            update_query = "UPDATE teams SET coach = %s WHERE name =
'Краснодар-U18'"
            if self.execute_query(update_query, (new_coach,)):
                print(f"Тренер изменен на: {new_coach}")
            else:
                print("Ошибка при изменении тренера")
        else:
            print("Изменение отменено")
        input("\n Нажмите Enter...")

    def transfer_player(self):
        print("\n ПЕРЕВОД ИГРОКА В ДРУГУЮ КОМАНДУ")
        query = """
        SELECT p.id, p.full_name, p.number, p.position
        FROM players p
        JOIN player_stats ps ON p.id = ps.player_id
        WHERE ps.team_id = 1
        GROUP BY p.id, p.full_name, p.number, p.position
        ORDER BY p.full_name
        """
        players = self.execute_query(query, fetch=True)
        if not players:
            print("Нет игроков в команде Краснодар-U18")
            input("\n Нажмите Enter...")
            return
        print("\n ИГРОКИ КРАСНОДАР-U18:")
        for i, p in enumerate(players, 1):
```

```
            print(f"{i}. {self.safe_str(p[1])} (№{self.safe_int(p[2])},
{self.safe_str(p[3])})")
        try:
            player_choice = int(input("\n Выберите номер игрока для перевода: "))
            if 1 <= player_choice <= len(players):
                player_id = players[player_choice-1][0]
                player_name = players[player_choice-1][1]
                query = """
                SELECT id, name, city
                FROM teams
                WHERE id != 1
                ORDER BY name
                """
                teams = self.execute_query(query, fetch=True)
                if not teams:
                    print("Нет других команд для перевода")
                    input("\n Нажмите Enter...")
                    return
                print(f"\n КУДА ПЕРЕВЕСТИ ИГРОКА {player_name}?")
                for i, t in enumerate(teams, 1):
                    print(f"{i}. {self.safe_str(t[1])} ({self.safe_str(t[2])})")
                team_choice = int(input("\n Выберите номер команды: "))
                if 1 <= team_choice <= len(teams):
                    new_team_id = teams[team_choice-1][0]
                    new_team_name = teams[team_choice-1][1]
                    print(f"\n ВНИМАНИЕ: Вы собираетесь перевести игрока:")
                    print(f"   Игрок: {player_name}")
                    print(f"   ИЗ: Краснодар-U18")
                    print(f"   В: {new_team_name}")
                    if self.confirm_action("Вы уверены?"):
```

```
                check_query = "SHOW COLUMNS FROM players LIKE 'team_id'"
                has_team_id = self.execute_query(check_query, fetch=True)
                if has_team_id:
                    update_query = "UPDATE players SET team_id = %s WHERE id
= %s"
                else:
                    update_query = None
                update_stats_query = """
                UPDATE player_stats
                SET team_id = %s
                WHERE player_id = %s AND team_id = 1
                """
                success = True
                if update_query:
                    if not self.execute_query(update_query, (new_team_id, player_id)):
                        print("Ошибка при обновлении основной информации
игрока")
                        success = False
                if not self.execute_query(update_stats_query, (new_team_id,
player_id)):
                    print("Ошибка при обновлении статистики игрока")
                    success = False
                if success:
                    print(f"Игрок {player_name} успешно переведен в
{new_team_name}")
                else:
                    print("Ошибка при переводе игрока")
            else:
                print("Перевод отменен")
        else:
```

```
                print("Неверный номер команды")
            else:
                print("Неверный номер игрока")
        except ValueError:
            print("Введите число")
        input("\n Нажмите Enter...")

    def cancel_match(self):
        print("\n ОТМЕНА ВСТРЕЧИ")
        query = """
        SELECT m.id, m.match_date, h.name, a.name, s.name
        FROM matches m
        JOIN teams h ON m.home_team_id = h.id
        JOIN teams a ON m.away_team_id = a.id
        JOIN stadiums s ON m.stadium_id = s.id
        WHERE m.match_date > NOW()
        ORDER BY m.match_date
        """
        matches = self.execute_query(query, fetch=True)
        if not matches:
            print("Нет запланированных встреч")
            input("\n Нажмите Enter...")
            return
        print("\n ВЫБЕРИТЕ ВСТРЕЧУ ДЛЯ ОТМЕНЫ:")
        for i, m in enumerate(matches, 1):
            date_str = self.format_date(m[1])
            home_team = self.safe_str(m[2])
            away_team = self.safe_str(m[3])
            stadium = self.safe_str(m[4])
            print(f"{i}. {date_str}: {home_team} - {away_team} на {stadium}")
```

```
try:
    choice = int(input("\n Выберите номер встречи: "))
    if 1 <= choice <= len(matches):
        match_id = matches[choice-1][0]
        match_date = self.format_date(matches[choice-1][1])
        home_team = self.safe_str(matches[choice-1][2])
        away_team = self.safe_str(matches[choice-1][3])
        print(f"\n ВНИМАНИЕ: Вы собираетесь отменить встречу:")
        print(f"   Дата: {match_date}")
        print(f"   Матч: {home_team} - {away_team}")
        if self.confirm_action("Вы уверены?"):
            # Удаление
            queries = [
                "DELETE FROM tickets WHERE match_id = %s",
                "DELETE FROM player_stats WHERE match_id = %s",
                "DELETE FROM matches WHERE id = %s"
            ]
            success = True
            for q in queries:
                if not self.execute_query(q, (match_id,)):
                    success = False
                    break
            if success:
                print("Встреча успешно отменена")
            else:
                print("Ошибка при отмене встречи")
        else:
            print("Отмена встречи отменена")
    else:
        print("Неверный номер")
```

```
        except ValueError:
            print("Введите число")

        input("\n Нажмите Enter...")

def main():
    print("СИСТЕМА УПРАВЛЕНИЯ ФУТБОЛЬНЫМ КЛУБОМ КРАСНОДАР-U18")
    db = FootballDB()
    if db.connection and db.connection.is_connected():
        db.show_menu()
    else:
        print("Не удалось подключиться к базе данных")
        print("Проверьте настройки подключения")

if __name__ == "__main__":
    main()
```