



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ  
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Институт  
Кафедра

компьютерных наук  
автоматизированных систем управления

Курсовая работа  
по дисциплине “Базы данных”

Студент АС-21-1

\_\_\_\_\_  
(подпись, дата)

Станиславчук С. М.

Руководитель  
Доцент

\_\_\_\_\_  
(подпись, дата)

Алексеев В. А.

Липецк 2024

## Содержание

### 1. Техническое задание

1.1. Характеристика предметной области

1.2. Цель и задачи системы (зачем нужна эта система в данной предметной области,  
какие задачи она будет решать)

1.3. Автоматизируемые бизнес-процессы (какие процессы будут автоматизироваться  
в информационной системе)

1.4. Характеристика пользователей

1.5. Модель вариантов использования (use-case диаграммы UML,  
обязательно учесть

генерацию отчетов, привести требуемые формы отчетов)

1.6. Модели бизнес-процессов в нотации BPMN (привести детальную  
модель

минимум одного бизнес-процесса)

1.7. Требования к информационному обеспечению

1.8. Требования к программному обеспечению

### 2. Технический проект

2.1. Модели локальных представлений

2.1.1. Локальные ER-диаграммы (по категориям пользователей в  
нотации Чена)

2.1.2. Основные запросы (нетривиальные запросы по категориям  
пользователей –

формулировка на естественном языке)

2.1.3. Общая ER-диаграмма (в нотации Чена)

2.2. Концептуальная модель данных

2.2.1. ER-диаграмма (PowerDesigner, в нотации Craw's Foot)

2.2.2. Спецификация сущностей (из отчета PowerDesigner)

### 2.2.3. Спецификация связей (из отчета PowerDesigner)

## 2.3. Логическая модель данных

### 2.3.1. Диаграмма логической модели (PowerDesigner, в нотации Craw's Foot)

### 2.3.2. Спецификация реляционных отношений (из отчета PowerDesigner)

## 2.4. Физическая модель данных

### 2.4.1. Обоснование выбора СУБД

### 2.4.2. Диаграмма физической модели (PowerDesigner)

### 2.4.3. Спецификация таблиц (из отчета PowerDesigner)

### 2.4.4. Проектирование вторичных индексов (с обоснованием)

## 2.5. Основные запросы к базе данных

### 2.5.1. Запрос №1

### 2.5.2. Запрос №2

### 2.5.3. Запрос №3

### 2.5.4. Запрос №4

### 2.5.5. Запрос №5

## 2.6. Пользовательские представления

### 2.6.1. Представление №1

### 2.6.2. Представление №2

## 2.7. Архитектура информационной системы

### 2.7.1. Диаграмма компонентов (в нотации UML, показывает, из каких компонентов состоит ИС и как эти компоненты связаны)

### 2.7.2. Спецификация компонентов (на какой платформе работает каждый компонент, какие библиотеки и модули включает и т.д.)

### 2.7.3. Распределение бизнес-логики между компонентами (за какой конкретно функционал отвечает каждый компонент: пользовательский интерфейс, бизнес-логика, хранение данных и т.п.)

### 2.7.4. Интерфейсы взаимодействия компонентов (с помощью к каких интерфейсов, протоколов взаимодействуют компоненты)

## 2.8. Хранимые процедуры и триггеры (не менее 3-х)

2.8.1. Хранимая процедура №1

2.8.2. Триггер №1

2.8.3. Триггер №2

2.8.3. Триггер №3

## 3. Рабочий проект

3.1. SQL-скрипт создания структуры БД (ссылка на приложение №1)

3.2. SQL-скрипт триггеров и хранимых процедур (ссылка на приложение №2)

3.3. Текст программы (только основные модули, ссылка на приложение №3)

3.4. Руководство пользователя (ссылка на приложение №4)

## Основная часть

### 1 Техническое задание

#### 1.1 Характеристика предметной области

Выбранная АИС: “Система управления учебным процессом в высшем учебном заведении” (Личный кабинет)

Система разработана с учетом трех типов пользователей: студент, преподаватель и администратор.

Процесс работы происходит следующим образом: администратор системы (superuser) до начала учебного семестра генерирует расписание предметов на семестр; преподаватель просматривает расписание предметов, выставляет оценки студентам, также может предоставить отчет успеваемости студентов, у которых он ведет занятия; студент также имеет доступ к просмотру расписания предметов своей группы, а также может просматривать полученные оценки. Таким образом, система позволяет контролировать учебный процесс с обеих сторон – и для студента, и для преподавателя.

#### Характеристика пользователей ИС

##### 1. Отчет об успеваемости студентов:

Цель отчета: позволить студентам просмотреть свои баллы по предметам, по которым он обучается

##### Требования:

1. Отображать все оценки студента, полученные в период семестра
2. У каждой выводимой оценки должен быть преподаватель, который еёставил, и средний балл студента по этому предмету.

##### 2. Отчет о расписании и аудиториях:

##### Цель отчета:

Предоставить студентам и преподавателям информацию о расписании занятий и занятых аудиториях.

##### Требования:

1. Персонализированный доступ для студентов и преподавателей.
2. Подробная информация о каждом занятии, включая название предмета,

преподавателя и аудиторию.

### 3. Отчет об успеваемости и учебной активности студентов:

Цель отчета:

Предоставить преподавателям информацию о посещаемости студентов и их активности в учебном процессе.

Требования:

1. Возможность выбора периода (например, текущий семестр или учебный год).
2. Подробная информация о оценках, включая дату, название предмета и преподавателя.

## 1.2 Цель и задачи системы

Целью АИС является автоматизация и улучшение управления учебным процессом в высшем учебном заведении с целью повышения его эффективности и обеспечения более качественного образования для студентов. АИС предназначена для сбора, хранения и предоставления информации, необходимой для всех участников учебного процесса, включая студентов, преподавателей, а также администраторов данной системы; для автоматизации административных задач, связанных с учебным процессом.

Перечень решаемых задач:

- Учет студентов: АИС должна позволять учреждению вести учет всех студентов, включая личные данные, контактную информацию, учетные записи и другие сведения.
- Учет преподавателей: Система должна поддерживать информацию о преподавателях, их квалификации, учебных нагрузках и контактных данных.
- Расписание занятий: АИС должна автоматизировать процесс создания и управления расписанием практик, лекций и других учебных мероприятий.
- Учет успеваемости и оценок: Система должна позволять вводить и отслеживать оценки, успеваемость студентов и предоставлять студентам и преподавателям доступ к этой информации.
- Учет учебных предметов и программ: АИС должна содержать информацию о предметах, учебных программах и учебных планах.

## 1.3 Автоматизируемые бизнес-процессы

### 1) Управление расписанием:

Этап 1: Создание расписания

Клиенты организации: Преподаватели, администраторы

Этап 2: Публикация расписания

Клиенты организации: Студенты, преподаватели

### 2) Учет успеваемости и оценок:

Этап 1: Ввод оценок

Клиенты организации: Преподаватели

Этап 2: Просмотр успеваемости

Клиенты организации: Студенты

Этап 3: Анализ успеваемости

Клиенты организации: Администраторы (деканат)

### 3) Учет учебных предметов и программ:

Этап 1: Добавление учебных предметов и программ

Клиенты организации: Администраторы

Этап 2: Редактирование информации о предметах

Клиенты организации: Администраторы

## 1.4 Характеристика пользователей

1) Студенты:

Решаемые задачи:

1. Просмотр расписания занятий.
2. Отслеживание успеваемости и оценок.

2) Преподаватели:

Решаемые задачи:

1. Ввод и редактирование оценок и успеваемости студентов.
2. Просмотр расписаний.

3) Администраторы системы:

Решаемые задачи:

1. Управление доступом и безопасностью системы.
2. Решение технических проблем

## 1.5 Модель вариантов использования

В соответствии с приведенным в п. 1.3 описанием бизнес-процессов была разработана следующая диаграмма вариантов использования ИС (рисунок 1)



Рисунок 1 – Use-case диаграмма АИС «Система управления учебным процессом в высшем учебном заведении»

## 1.6 Модель бизнес-процесса в нотации BPMN

Модель бизнес-процесса “Учет успеваемости и оценок” в нотации BPMN представлена на рисунке 2.

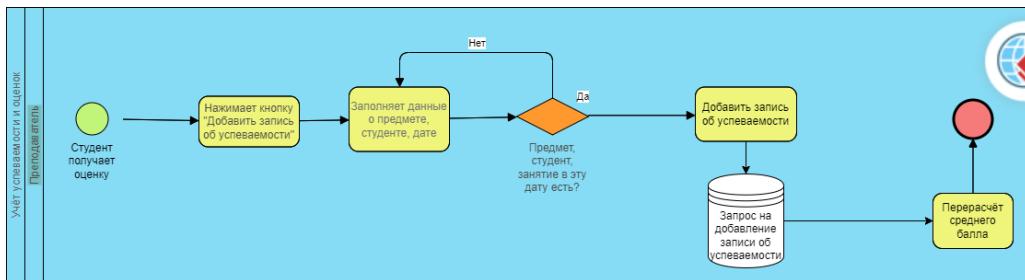


Рисунок 2 – BPMN модель “Учет успеваемости и оценок”

Модель бизнес-процесса “Управление расписанием” представлен на рисунке 3

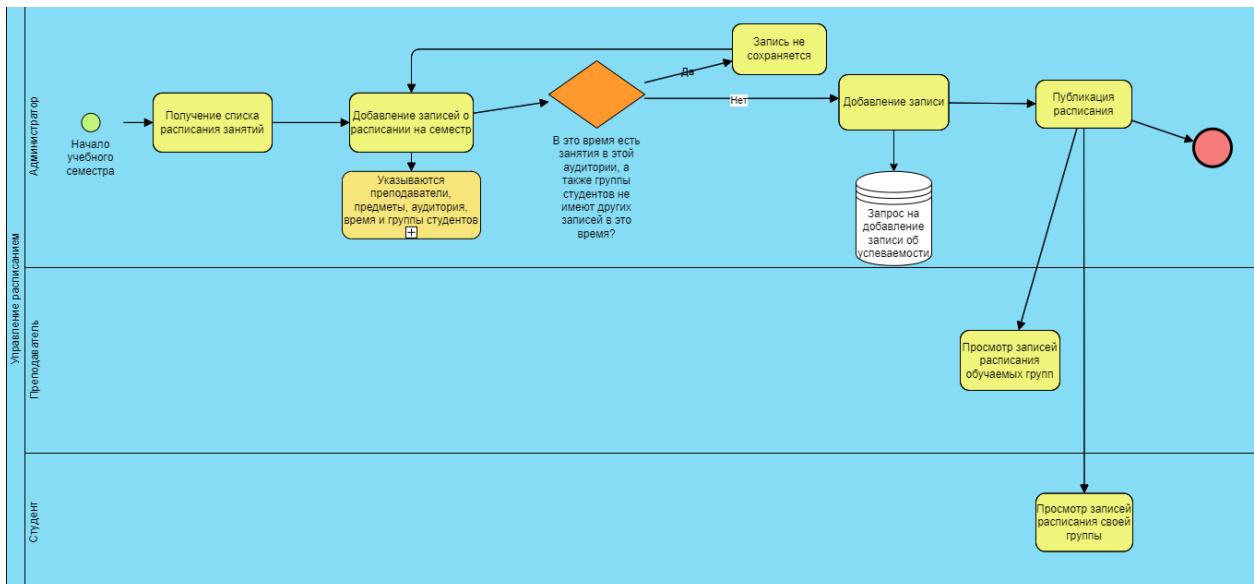


Рисунок 3 – BPMN модель “Управление расписанием”

## 1.7 Требования к информационному обеспечению

1. Концептуальная модель данных должна содержать не менее 5 сущностей.
2. СУБД – PostgreSQL
3. Физическая модель должна предусматривать реализацию индексов и пользовательского представления.

Информационное обеспечение системы должно быть достаточным для выполнения всех функций системы, обеспечивать информационную

совместимость подсистем со смежными.

Совокупность информационных массивов должна входить в базу данных всей системы и размещаться на сервере. Это связано со значительными объемами информации и в целях увеличения быстродействия работы, а также с длительными сроками хранения.

В системе должны быть предусмотрены меры по контролю и обновлению данных в информационных массивах и восстановлению массивов при сбоях технических устройств.

В системе должен быть предусмотрен удобный и быстрый доступ к необходимой информации.

Формы выходных документов как печатных, так и экранных должны отличаться наглядностью с целью облегчения восприятия информации персоналом.

## 1.8 Требования к программному обеспечению

1. Проект должен предусматривать реализацию триггеров и хранимых процедур.
2. Платформа разработки прикладного приложения – C#, .Net, Unity.
3. Прикладное приложение должно иметь удобный пользовательский интерфейс, реализующий функции информационной системы, предусмотренные техническим заданием.
4. Прикладное приложение должно скрывать от пользователя технические детали организации данных в БД (искусственные идентификаторы и т.п.).
5. Приложение должно предусматривать генерацию отчетных форм с использованием стандартных средств разработки отчетов (FastReport или т.п.), с возможностью экспорта отчетов в стандартные форматы (PDF/Excel/Word и т.д.).

Программное обеспечение должно быть достаточным для выполнения всех функций системы, реализуемых с применением средств вычислительной техники.

Программное обеспечение должно быть достаточным, и обладать следующими свойствами - надежностью, модульностью построения, удобством эксплуатации.

Программное обеспечение должно быть построено таким образом, чтобы отсутствие отдельных данных не сказывалось на выполнении функций АСУ, при реализации которых, эти данные не используются.

В программном обеспечении АСУ должны быть реализованы меры по защите от ошибок при вводе и обработке информации, обеспечивающие заданное качество выполнения функций АСУ.

Драйверы должны обеспечивать условия связи межмашинного обмена информацией в соответствии принятым размещением комплекса технических средств».

Документация по эксплуатации программного обеспечения системы должна соответствовать стандартам ЕСПД и содержать все сведения, необходимые для персонала системы по его использованию, первичной загрузки в загрузки внутри машинных информационных баз, запуска программ системы.

Программное обеспечение системы должно функционировать как desktop-приложение и использовать СУБД "PostgreSQL".

## 2. Технический проект

### 2.1. Модели локальных представлений

#### 2.1.1 Локальные ER-диаграммы

##### 1) Категория пользователей “Студент”

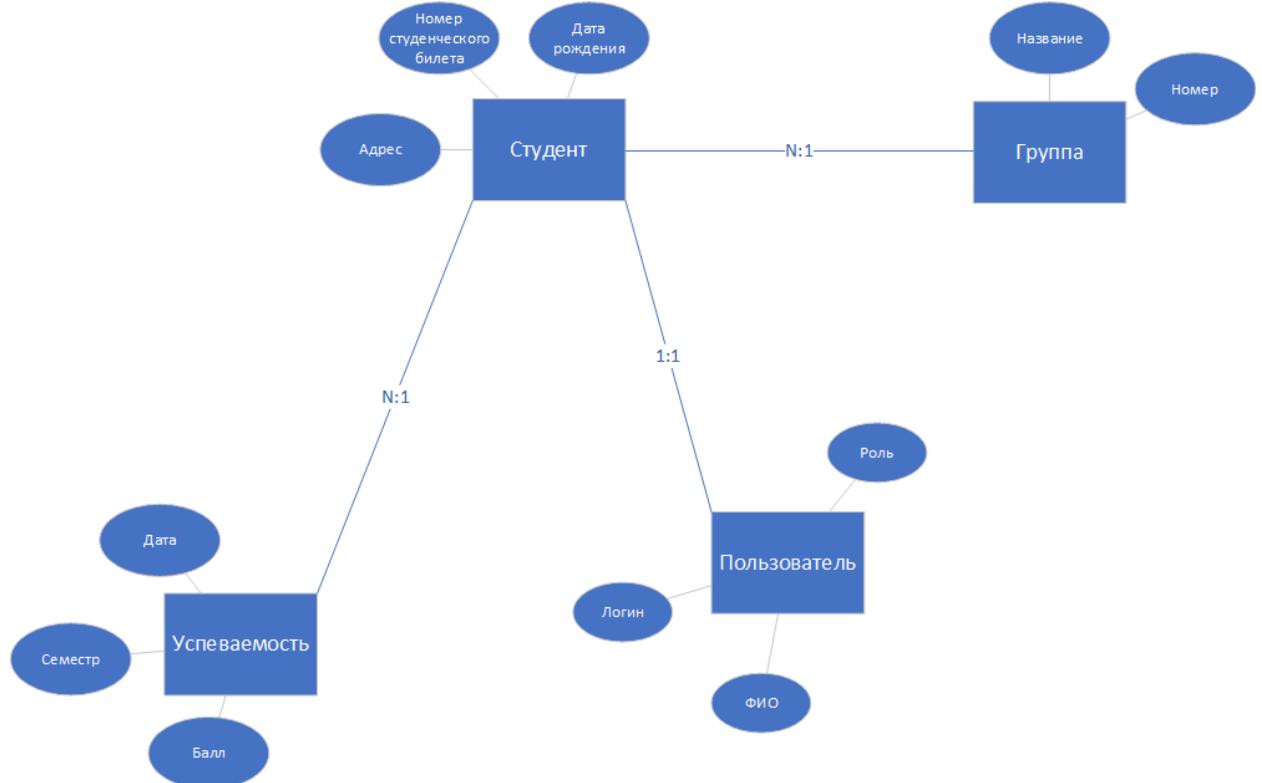


Рисунок 4 — Локальная ER-диаграмма студента

## 2) Категория пользователей «Преподаватель»

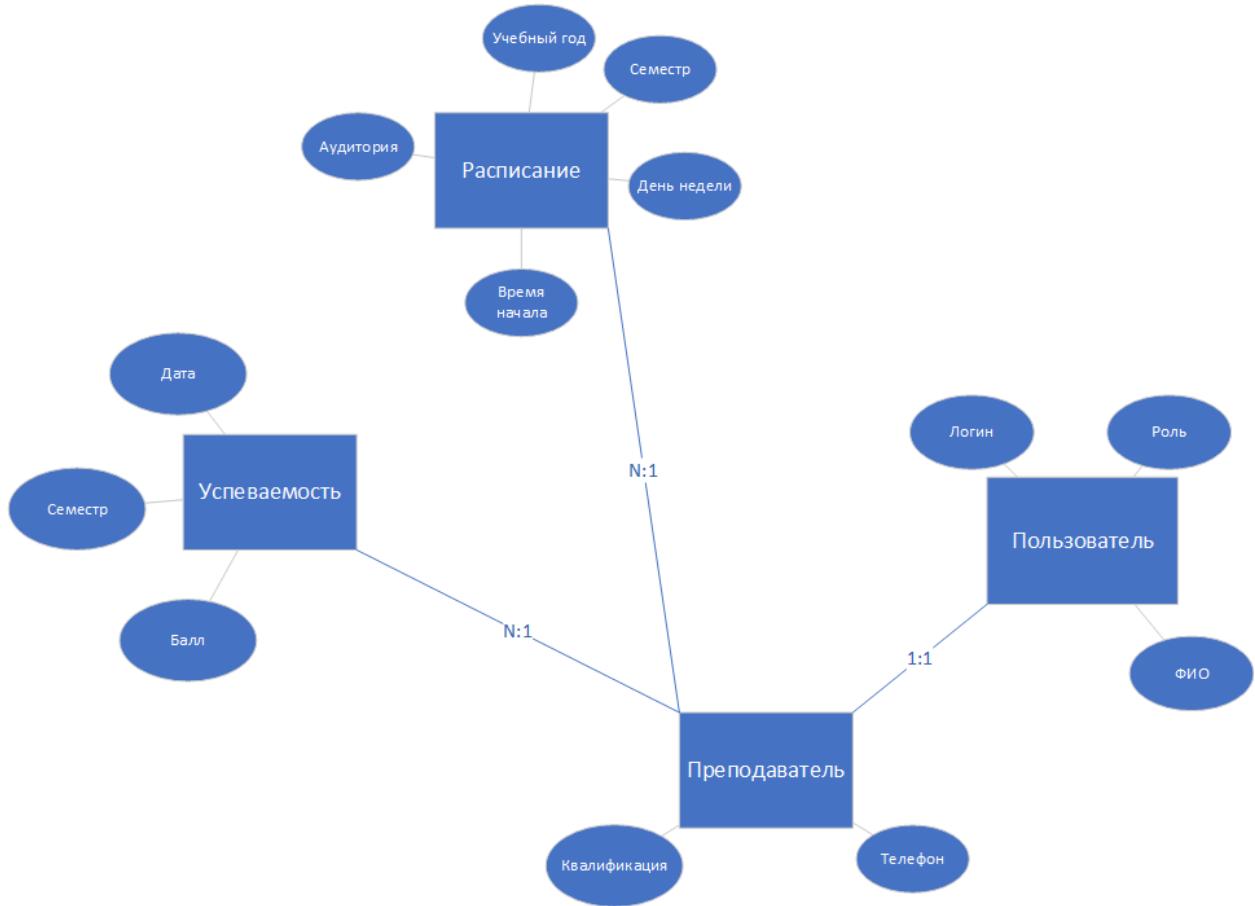


Рисунок 5 — Локальная ER-диаграмма преподавателя

### 2.1.2 Основные запросы

#### 1) Студенты:

1. Просмотр расписания занятий.
2. Отслеживание успеваемости и оценок.
3. Регистрация на курсы и выбор учебных предметов.
4. Подача заявлений и запросов (например, на отпуск).

#### 2) Преподаватели:

1. Создание и управление расписанием занятий.
2. Ввод и редактирование оценок и успеваемости студентов.
3. Просмотр данных о студентах и учебных группах.
4. Обмен информацией с администрацией и студентами.

#### 3) Администраторы:

1. Поддержка пользователей и решение технических проблем.

### 2.1.3 Общая ER-диаграмма

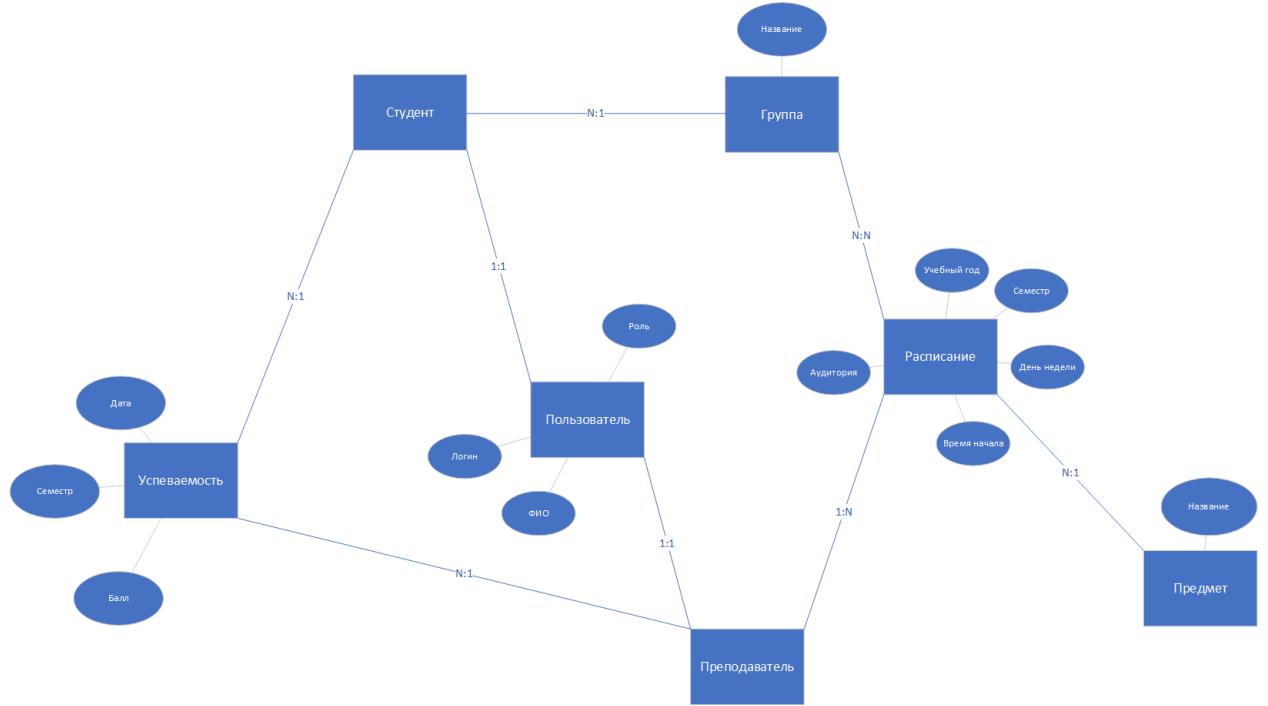


Рисунок 6 — Общая ER-диаграмма

## 2.2 Концептуальная модель данных

### 2.2.1 ER-диаграмма

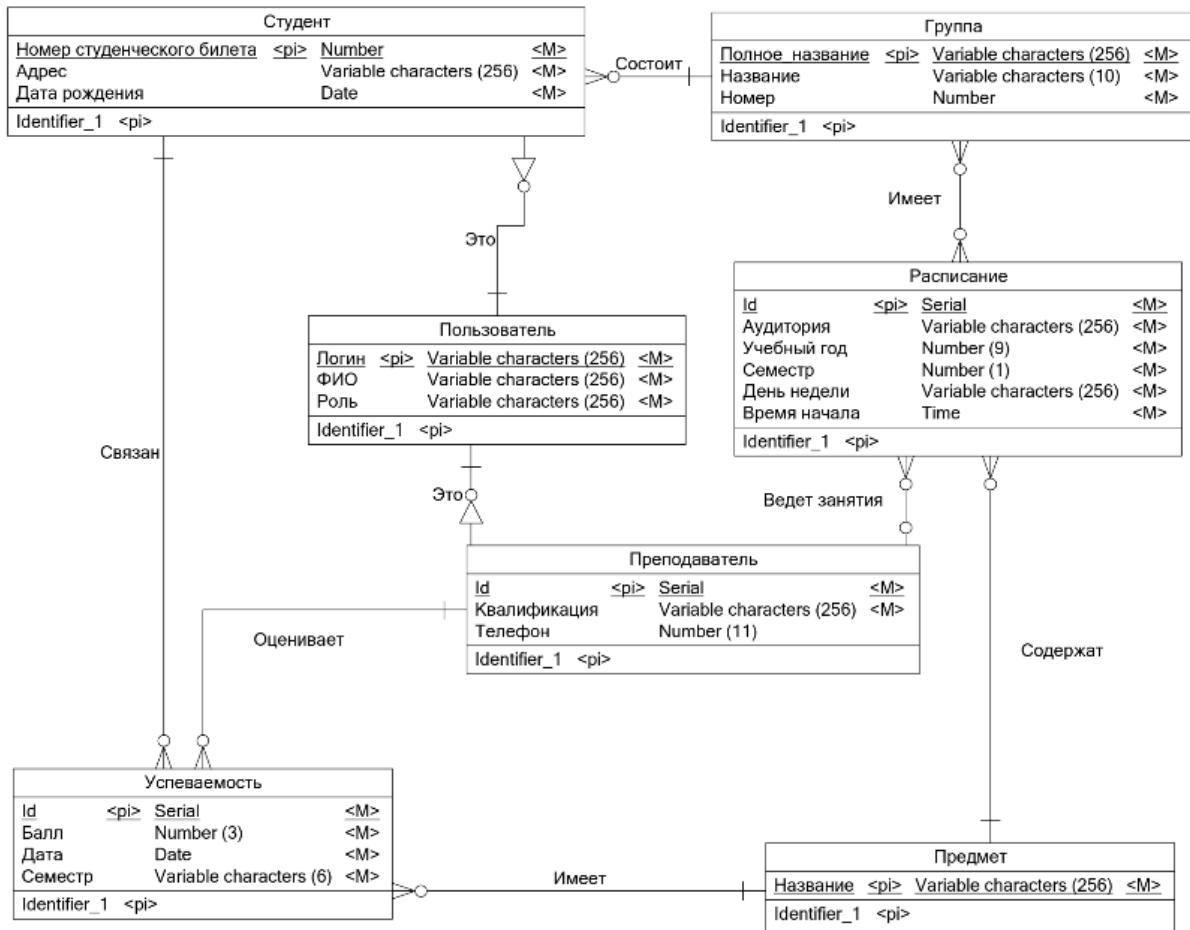


Рисунок 7 - ER-диаграмма концептуальной модели

### 2.2.2 Спецификация сущностей

Таблица 1 — спецификация сущностей

Имя сущности	Код сущности	Имя атрибута	Код атрибута	Обязательность	Первичный ключ	Домен
Студент	Student	Номер студенческого билета	id_number_student	T	T	Number (10)
		Дата рождения	birth_date	T	F	Date

		e_student				
		Адрес	address_student	F	F	VARCHAR
		Полное название	full_name_group	T	T	VARCHAR
Группа	Group	Название	name_group	T	F	VARCHAR (10)
		Номер	number_group	T	F	Number
		Id	Id_teacher	T	T	Serial
Препода вател	Teacher	Квалифи кация	qualification_teacher	T	F	VARCHAR
		Телефон	phone_number_teacher	F	F	NUMBER (11)
		Логин	login_user	T	T	VARCHAR
Пользова тель	User	ФИО	full_name_user	T	F	VARCHAR
		Роль	role_user	T	F	VARCHAR
Расписан ие	Schedule	Id	id_schedule	T	T	Serial
		Аудитор ия	class_schedule	T	F	VARCHAR
		Учебный год	academic_year_schedule	T	F	Number (9)
		Семестр	semestster_schedule	T	F	Number (1)

		День недели	weekday_scheduled	T	F	VARCHAR
		Время начала	start_time_schedule	T	F	Time
Предмет	Subject	Название	name_subject	T	T	VARCHAR
		Id	Id_performance	T	T	Serial
Успеваемость	Performance	Балл	mark_performance	T	F	Number(3)
		Дата	date_performance	T	F	Date
		Семестр	semester_performance	T	F	VARCHAR(6)

### 2.2.3 Спецификация связей

Имя связи	Код связи	Сущность А	Принадлежность	Сущность Б	Принадлежность	Вид связи
Состоит	student_in_group	Студент	Необязательно	Группа	Обязательно	1:m
Имеет	group_has_schedules	Группа	Необязательно	Расписание	Обязательно	1:m
Содержат	schedule_have_subject	Расписание	Обязательно	Предмет	Необязательно	1:m
Имеет	performance_have_subject	Успеваемость	Обязательно	Предмет	Необязательно	1:m
Связан	performance_related_student	Успеваемость	Необязательно	Студент	Обязательно	1:m

Оценивает	teacher_rate_performance	Преподаватель	Обязательно	Успеваемость	Обязательно		1:m
Это	student_is_user	Студент	Необязательно	Пользователь	Обязательно		1:1
Ведет занятия	teacher_has_schedule	Преподаватель	Обязательно	Расписание	Обязательно		1:m
Это	teacher_is_user	Преподаватель	Необязательно	Пользователь	Обязательно		1:1

## 2.3 Логическая модель данных

### 2.3.1 Диаграмма логической модели

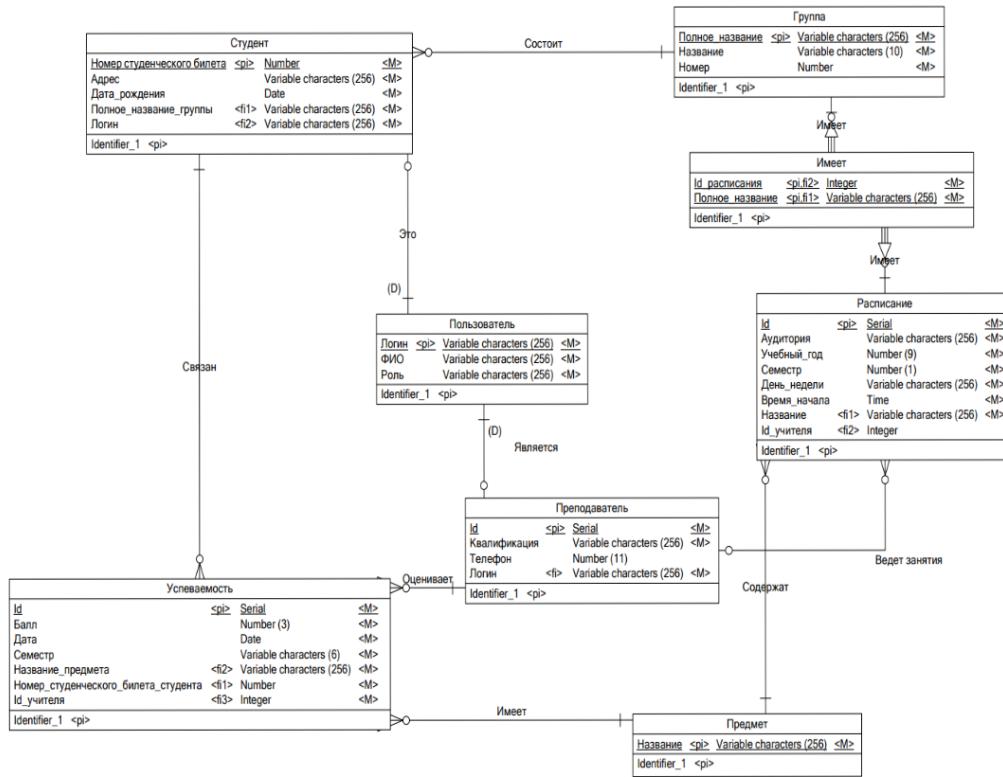


Рисунок 8 — диаграмма логической модели

### 2.3.2 Спецификация реляционных отношений

Спецификация реляционных отношений представлена в таблице 2.

Таблица 2. Спецификация реляционных отношений

Имя сущности и	Код сущности и	Имя атрибута а	Код атрибута а	Внешний ключ	Связь
----------------	----------------	----------------	----------------	--------------	-------

Номер  
студенче  
ского  
билета

Студент Student

Дата  
рождени  
я

Адрес address\_s

F

F

F

			tudent		
		Полное_ названи е_групп ы	full_nam e_group	T	Группа
		Логин_п ользова теля	login_us er	T	Пользов атель
		Полное название	full_nam e_group	F	-
Группа	Group	Название	name_gr oup	F	-
		Номер	number_ group	F	-
		Id	Id_teach er	F	-
Препода ватель	Teacher	Квалифи кация	qualificat ion_teach er	F	-
		Телефон	phone_nu mber_tea cher	F	-
		Логин	login_us er	T	Пользов атель
		Логин	login_use r	F	-
Пользова тель	User	ФИО	full_nam e_user	F	-
		Роль	role_user	F	-
		Id	id_sched ule	F	-
Расписан ие	Schedule	Аудитор ия	class_sch edule	F	-
		Учебный год	academic _year_sc hedule	F	-

		Семестр	semestster _sche dule	F	-
		День недели	weekday _schedul e	F	-
		Время начала	start_tim e_schedu le	F	-
		<b>Название е_предм ета</b>	<b>name_su bject</b>	<b>T</b>	<b>Предмет</b>
		<b>Id_преп одавате ля</b>	<b>Id_teach er</b>	<b>T</b>	<b>Препода ватель</b>
Предмет	Subject	Название	name_su bject	F	-
Успевае мость	Perfoman ce	Id	Id_perfo rmance	F	-
		Балл	mark_per formance	F	-
		Дата	date_perf ormance	F	-
		Семестр	semester _perfo mance	F	-
		<b>Название е_предм ета</b>	<b>name_su bject</b>	<b>T</b>	<b>Предмет</b>
		<b>Номер_с туденчес кого_би лета_сту дента</b>	<b>id_numb er_stude nt</b>	<b>T</b>	<b>Студент</b>
		<b>Id_преп одавате ля</b>	<b>Id_teach er</b>	<b>T</b>	<b>Препода ватель</b>

## 2.4. Физическая модель данных

### 2.4.1. Обоснование выбора СУБД

В качестве СУБД был выбран Postgres ввиду его удобства использования, простоты реализации всех необходимых хранимых процедур, триггеров.

### 2.4.2 Диаграмма физической модели

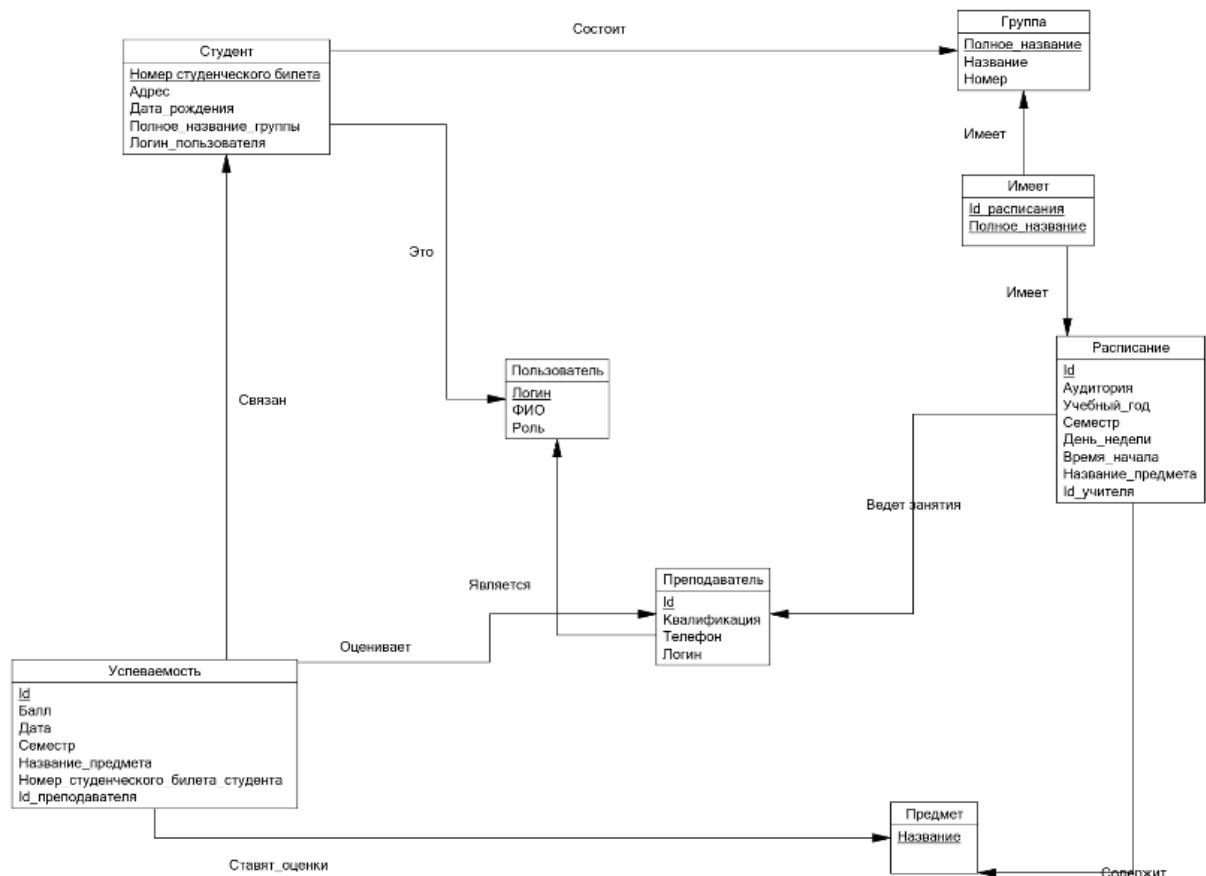


Рисунок 9 - Диаграмма физической модели

### 2.4.3. Спецификация таблиц

Таблица 3 — Спецификация таблиц

Имя таблицы	Код таблицы	Имя атрибута	Код атрибута	Обязательность	Первичный ключ	Внешний ключ	Домен
-------------	-------------	--------------	--------------	----------------	----------------	--------------	-------

		Номер студенче ского билета	id_number_student	T	T	F	Number (10)
Студент	Student	Дата рождени я	birth_date_student	T	F	F	Date
		Адрес	address_student	T	F	F	VARCHAR
		Полное_ названи е_групп ы	full_name_group	T	F	T	VARCHAR (256)
		Логин_п ользова теля	login_user	T	F	T	VARCHAR (256)
		Полное название	full_name_group	T	T	F	VARCHAR
Группа	Group	Название	name_group	T	F	F	VARCHAR (10)
		Номер	number_group	T	F	F	Number
		Id	Id_teacher	T	T	F	Serial
Препода вател	Teacher	Квалифи кация	qualification_teacher	T	F	F	VARCHAR
		Телефон	phone_number_teacher	F	F	F	NUMBER (11)
		Логин	login_user	T	F	T	VARCHAR (256)
Пользова тель	User	Логин	login_user	T	T	F	VARCHAR
		ФИО	full_name_user	T	F	F	VARCHAR

	Роль	role_user	T	F	F	VARCHAR AR
	Id	id_schedule	T	T	F	Serial
	Аудитория	class_schedule	T	F	F	VARCHAR AR
	Учебный год	academic_year_schedule	T	F	F	Number (9)
Расписанье	Schedule	Семестр	semestster		F	
		_schedule	T	F		Number (1)
	День недели	weekday_schedule	T	F	F	VARCHAR AR
	Время начала	start_time_schedule	T	F	F	Time
Предмет	Subject	Название_предмета	name_subject	T	F	T VARCHAR AR (256)
		Id_учителя	Id_teacher	T	F	T INT4
		Название	name_subject	T	T	F VARCHAR AR
		Id	Id_performance	T	T	F Serial
Успеваемость	Performance	Балл	mark_performance	T	F	F Number (3)
		Дата	date_performance	T	F	F Date
		Семестр	semester_perfo rmance	T	F	F VARCHAR AR (6)

	Название предмета	name_subject	T	F	T	VARCHAR(256)
	Номер студенческого_била	Id_number_studen	T	F	T	NUMERIC
	ета_студент	t				C
Имеет	Id_преподавателя	Id_teacher	T	F	T	INT4
	group_has_schedules	Id_расписания	T	T	T	INT4
	Полное_название	full_name_group	T	T	T	VARCHAR(256)

#### 2.4.4 Проектирование вторичных индексов

С учетом специфики обработки и необходимости получения информации из таблиц было принято решение добавить индекс для столбцов, которые часто используются в запросах с условиями и соединениями. Например, добавление индекса `idx_performace_id_number_student` на столбец `id_number_student` в таблице `perfomance` позволит значительно ускорить запросы, связанные с получением оценок студента. Аналогично, индекс `idx_schedule_id_teacher` на столбец `id_teacher` в таблице `schedule` обеспечит более быстрый доступ к расписаниям преподавателя.

SQL запросы для создания вторичных индексов:

```
CREATE INDEX idx_schedule_id_teacher ON schedule(id_teacher);
```

```
CREATE INDEX idx_performace_id_number_student ON
perfomance(id_number_student);
```

2.5. Основные запросы к базе данных:

### 2.5.1. Запрос №1

Формулировка на естественном языке:

отобразить всех студентов и их оценки, полученные первого октября 2023 года.

**Формулировка в виде выражения реляционной алгебры:**

$$\sigma(\text{date\_perfomance} = '2023-10-01')(\pi(\text{mark\_perfomance}, \text{name\_subject}, \text{id\_number\_student})(\text{perfomance}))$$

**Формулировка в виде формулы реляционного исчисления с переменными кортежами:**

$$\{t \mid \exists p (\text{perfomance}(p) \wedge p.\text{date\_perfomance} = '2023-10-01' \wedge t.\text{mark\_perfomance} = p.\text{mark\_perfomance} \wedge t.\text{name\_subject} = p.\text{name\_subject} \wedge t.\text{id\_number\_student} = p.\text{id\_number\_student})\}$$

**Формулировка на языке SQL:**

```
SELECT perfomance.mark_perfomance, perfomance.name_subject,  
perfomance.id_number_student  
  
FROM perfomance  
  
WHERE perfomance.date_perfomance = '2023-10-01'::DATE  
  
ORDER BY perfomance.id_number_student DESC;
```

	mark_perfomance numeric (3)	name_subject character varying (256)	id_number_student numeric
1	4	Математическое программирование	2429998970
2	5	Базы данных	1358211096

Рисунок 10 — Пример выполнения запроса №1

### 2.5.2 Запрос №2

Запрос: какие оценки получили студенты группы ПИ-21-1 с сортировкой по баллу

**Формулировка в виде выражения реляционной алгебры:**

$$\pi(\text{student.id\_number\_student}, \text{student.full\_name\_group}, \text{perfomance.mark\_perfomance}, \text{perfomance.name\_subject})$$

$$(\sigma(\text{student.id\_number\_student} = \text{perfomance.id\_number\_student} \wedge \text{student.full\_name\_group} = 'ПИ-21-1')(\text{student} \bowtie \text{perfomance})) \bowtie \\ \rho(\text{perfomance\_mark\_desc})(\tau(\text{perfomance.mark\_perfomance DESC}))$$

**Формулировка в виде формулы реляционного исчисления с переменными кортежами:**

$$\{t \mid \exists s \exists p (\text{student}(s) \wedge \text{perfomance}(p) \wedge s.\text{id\_number\_student} = p.\text{id\_number\_student} \wedge s.\text{full\_name\_group} = 'ПИ-21-1' \wedge t.\text{id\_number\_student} = s.\text{id\_number\_student} \wedge t.\text{full\_name\_group} = s.\text{full\_name\_group} \wedge t.\text{mark\_perfomance} = p.\text{mark\_perfomance} \wedge t.\text{name\_subject} = p.\text{name\_subject}) \wedge \forall p1 \forall p2 ((\text{perfomance}(p1) \wedge \text{perfomance}(p2) \wedge p1.\text{id\_number\_student} = p2.\text{id\_number\_student}) \rightarrow (p1.\text{mark\_perfomance} \leq p2.\text{mark\_perfomance} \rightarrow p1 \leq p2))\}$$

	<b>id_number_student</b> numeric	<b>full_name_group</b> character varying (256)	<b>mark_performance</b> numeric (3)	<b>name_subject</b> character varying (256)	<b>id_number_student</b> numeric
1	1358211096	ПИ-21-1	5	Базы данных	1358211096
2	2429998970	ПИ-21-1	4	Математическое программирование	2429998970
3	5750846942	ПИ-21-1	4	Операционные системы	5750846942
4	1982246734	ПИ-21-1	4	Архитектура вычислительных систем	1982246734

Рисунок 11 — Пример выполнения запроса №2

### 2.5.3 Запрос №3

Запрос: вывести всех студентов групп, кроме студентов групп ПИ

**Формулировка в виде выражения реляционной алгебры:**

$$\pi(\text{id\_number\_student}, \text{full\_name\_group})(\sigma(\text{full\_name\_group} \notin \pi(\text{full\_name\_group})(\sigma(\text{name\_group} = 'ПИ'))(\text{group\_data})))(\text{student})$$

**Формулировка в виде формулы реляционного исчисления с переменными кортежами:**

$$\{t \mid \text{student}(s) \wedge t.\text{id\_number\_student} = s.\text{id\_number\_student} \wedge t.\text{full\_name\_group} = s.\text{full\_name\_group} \wedge \neg \exists g (\text{group\_data}(g) \wedge g.\text{name\_group} = 'ПИ' \wedge g.\text{full\_name\_group} = s.\text{full\_name\_group})\}$$

**Формулировка на языке SQL:**

```
SELECT id_number_student, full_name_group
FROM student
```

WHERE full\_name\_group NOT IN (SELECT full\_name\_group FROM group\_data WHERE name\_group = 'ПИ');

	id_number_student [PK] numeric	full_name_group character varying (256)
1	6335220554	ИМИТ-21-1
2	6906157794	ИМИТ-21-1
3	5567195580	АИ-21-1
4	9781459292	AC-21-2

Рисунок 12 — Пример выполнения запроса №3

#### 2.5.4 Запрос №4

Запрос: рассчитать сколько всего предметов ведет каждый преподаватель

**Формулировка в виде выражения реляционной алгебры:**

$\pi(\text{Id\_teacher}, \text{qualification\_teacher}, \text{subjects\_count})(\gamma(\text{Id\_teacher}, \text{qualification\_teacher}, \text{COUNT}(\text{Id\_schedule}) \text{ AS subjects\_count})(\text{teacher} \bowtie \text{schedule}))$

**Формулировка в виде формулы реляционного исчисления с переменными кортежами:**

$\{t \mid \exists \text{teacher} (\text{teacher}(t) \wedge \exists q \exists sc (t.\text{Id\_teacher} = q.\text{Id\_teacher} \wedge t.\text{qualification\_teacher} = q.\text{qualification\_teacher} \wedge t.\text{subjects\_count} = sc.\text{subjects\_count} \wedge sc.\text{subjects\_count} = (\text{COUNT}(\text{schedule}(s) \wedge s.\text{Id\_teacher} = q.\text{Id\_teacher}))))\}$

**Формулировка на языке SQL:**

```
SELECT teacher.Id_teacher, teacher.qualification_teacher,  
COUNT(schedule.Id_schedule) AS subjects_count  
FROM teacher  
LEFT JOIN schedule ON teacher.Id_teacher = schedule.Id_teacher  
GROUP BY teacher.Id_teacher, teacher.qualification_teacher;
```

	<b>id_teacher</b> [PK] integer	<b>qualification_teacher</b> character varying (256)	<b>subjects_count</b> bigint
1	4	Доцент	2
2	6	Доцент	0
3	2	Профессор	1
4	3	Старший преподаватель	2
5	1	Доцент	2
6	5	Профессор	3

Рисунок 13 — Пример выполнения запроса №4

#### 2.5.5. Запрос №5

Запрос: рассчитать сколько у каждой квалификации преподавателей ведется предметов и количество предметов, которые ведет каждый преподаватель по отдельности.

**Формулировка в виде выражения реляционной алгебры:**

$\gamma(\text{qualification\_teacher}, \text{Id\_teacher}, \text{COUNT}(\text{Id\_schedule}) \text{ AS subjects\_count})$   
 $(\text{teacher} \bowtie \text{schedule})$

**Формулировка в виде формулы реляционного исчисления с переменными кортежами:**

{ t |  $\exists q \exists sc$  ( teacher(t)  $\wedge$  t.qualification\_teacher = q.qualification\_teacher  $\wedge$  t.Id\_teacher = q.Id\_teacher  $\wedge$  t.subjects\_count = sc.subjects\_count  $\wedge$  sc.subjects\_count = (COUNT(schedule(s)  $\wedge$  s.Id\_teacher = q.Id\_teacher)) ) }

**Формулировка на языке SQL:**

```
SELECT teacher.Id_teacher, teacher.qualification_teacher,
       COUNT(schedule.Id_schedule)
AS subjects_count
FROM teacher
LEFT JOIN schedule ON teacher.Id_teacher = schedule.Id_teacher
GROUP BY GROUPING SETS (teacher.qualification_teacher, teacher.Id_teacher)
```

	<b>id_teacher</b> [PK] integer	<b>qualification_teacher</b> character varying (256)	<b>subjects_count</b> bigint
1	[null]	Профессор	4
2	[null]	Доцент	4
3	[null]	Старший преподаватель	2
4	4	[null]	2
5	6	[null]	0
6	2	[null]	1
7	3	[null]	2
8	1	[null]	2
9	5	[null]	3

Рисунок 14 — Пример выполнения запроса №5

## 2.6. Пользовательские представления

### 2.6.1. Представление №1

**Формулировка на естественном языке**

**Формулировка на естественном языке:**

Представление должно отображать информацию о студентах и их оценках за каждый предмет.

**Форма выходных данных:**

Таблица с колонками:

- Идентификатор студента (id\_number\_student)
- Полное имя студента (full\_name\_student)
- Группа студента (full\_name\_group)
- Название предмета (name\_subject)
- Оценка студента (mark\_perfomance)
- Дата получения оценки (date\_perfomance)

**Входные параметры:** Отсутствуют.

### Формулировка запроса на языке SQL:

```
CREATE VIEW student_performance_view AS
SELECT s.id_number_student, s.full_name_student, s.full_name_group,
       p.name_subject, p.mark_perfomance, p.date_perfomance
  FROM student s
 JOIN perfomance p ON s.id_number_student = p.id_number_student;
```

FROM student_performance_view;				
id_number_student	full_name_group	name_subject	mark_perfomance	date_perfomance
1	AC-21-1	Алгебра и геометрия	75	2024-06-04
1	AC-21-1	Информатика	75	2024-06-04
1	AC-21-1	Информатика	96	2024-06-04
1	AC-21-1	Программирование	75	2024-06-04
1	AC-21-1	Микропроцессорные системы	95	2024-06-04
1	AC-21-1	Микропроцессорные системы	75	2024-06-04
1	AC-21-1	Алгебра и геометрия	69	2024-06-04
1	AC-21-1	Алгебра и геометрия	90	2024-06-04
2	ПМ-21-1	Программирование	85	2024-06-04
2	ПМ-21-1	Программирование	90	2024-06-04
2	ПМ-21-1	Теория вероятностей	90	2024-06-04
2	ПМ-21-1	Теория вероятностей	55	2024-06-04
2	ПМ-21-1	Алгебра и геометрия	55	2024-06-04

Рисунок 15 — пример выполнения представления №1

### 2.6.2. Представление №2

#### Формулировка на естественном языке:

Представление для отображения информации о расписании занятий студентов.

Форма выходных данных: Таблица с колонками: id\_schedule, class\_schedule, semester\_schedule, weekday\_schedule, start\_time\_schedule, name\_subject, id\_teacher, full\_name\_group, full\_name\_user.

Входные параметры: Отсутствуют.

#### Формулировка запроса на языке SQL:

```
CREATE VIEW schedule_view AS
```

```
SELECT
```

```
s.id_schedule,
```

```
s.class_schedule,
```

```
s.semester_schedule,
```

```

s.weekday_schedule,
s.start_time_schedule,
s.name_subject,
s.id_teacher,
g.full_name_group,
u.full_name_user

FROM

schedule s

JOIN

group_data g ON s.full_name_group = g.full_name_group

JOIN

teacher t ON s.id_teacher = t.id_teacher

JOIN

user_data u ON t.login_user = u.login_user;

```

1	101			1	Понедельник	09:00:00	Введение в профессию программирование		1	ПИ-20-2	София Лорен
2	202			1	Среда	11:00:00	Алгебра и геометрия		2	АС-21-1	Мэри Джейн
2	202			1	Среда	11:00:00	Алгебра и геометрия		2	ПМ-21-1	Мэри Джейн
3	303			1	Пятница	14:00:00	Инженерная физика		1	АС-21-1	София Лорен
4	201			2	Понедельник	09:00:00	Программирование		1	ПМ-21-1	София Лорен
4	201			2	Понедельник	09:00:00	Программирование		1	АС-21-1	София Лорен
5	203			2	Понедельник	11:00:00	Микропроцессорные системы		2	АС-21-1	Мэри Джейн
6	303			2	Вторник	14:00:00	Программирование		1	АС-21-1	София Лорен
7	102			1	Четверг	13:00:00	Математический анализ		2	ПМ-20-2	Мэри Джейн
8	104			1	Среда	15:00:00	Информатика		1	АС-21-1	София Лорен
9	204			2	Вторник	10:00:00	Теория вероятностей		1	ПМ-21-1	София Лорен
10	101			2	Среда	16:40:00	Алгебра и геометрия		2	АС-21-1	Мэри Джейн
10	101			2	Среда	16:40:00	Алгебра и геометрия		2	ПМ-21-1	Мэри Джейн
11	101			2	Четверг	09:40:00	Микропроцессорные системы		2	АС-21-1	Мэри Джейн
12	224			2	Пятница	09:40:00	Микропроцессорные системы		2	АС-21-1	Мэри Джейн

(15 rows)

Рисунок 16 — пример выполнения представления №2

## 2.7. Архитектура информационной системы

### 2.7.1. Диаграмма компонентов

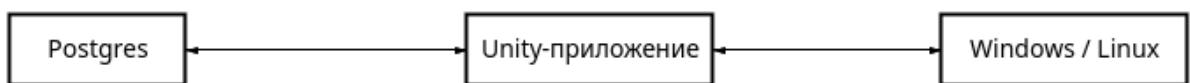


Рисунок 17 - Диаграмма компонентов

### 2.7.2. Спецификация компонентов

Unity-приложение

- Платформы: Windows / Linux

- Используемые библиотеки:

  - NpSql — библиотека для выполнения запросов к БД на языке C#

  - iTextSharp — библиотека для составления PDF-отчетов

- .NET System — библиотека для работы с файлами.

- .NET System.Generics.Collections — библиотека для работы с удобными коллекциями: списками, словарями.

PostgreSQL БД:

- Платформа: Linux

### **2.7.3. Распределение бизнес-логики между компонентами**

Unity-приложение

- Графический пользовательский интерфейс (GUI):

- Отображение экранов авторизации, работа с таблицами и генерация PDF отчетов.

- Отображение сообщений об ошибках.

- Удобная навигация с помощью клавиатуры.

- Бизнес-логика:

- Отправка SQL-запросов на базу данных PostgreSQL для выполнения операций

- Генерация отчетов и сохранение их в pdf

PostgreSQL БД

- Хранение данных:

- Хранение данных пользователей.

- Хранение данных о оценках, расписаниях, предметах.

- Бизнес-логика:

- Обеспечение целостности при помощи триггеров в БД для уменьшения кол-ва ошибок при работе

- Выполнение хранимых процедур и функций

### **2.7.4. Интерфейсы взаимодействия компонентов**

- Интерфейсы взаимодействия между Unity-приложением и PostgreSQL-БД.

Протокол взаимодействия: PostgreSQL использует протокол клиент-сервер для взаимодействия с клиентскими приложениями.

Клиентское приложение на Unity подключается к базе данных PostgreSQL через сеть (TCP/IP).

- Библиотека: Npgsql используется для установления соединения и выполнения SQL-запросов к базе данных PostgreSQL

на устройстве, на котором запущено приложение

- SQL-запросы: Взаимодействие происходит через выполнение SQL-запросов (SELECT, INSERT, UPDATE, DELETE), которые передаются из Flutter-приложения в PostgreSQL и обратно. А также используется вызов хранимых процедур из приложения.

## **2.8. Хранимые процедуры и триггеры**

### **2.8.1. Хранимая процедура №1**

Формулировка транзакции на естественном языке:

Транзакция начинается с получения идентификатора студента по его группе и полному имени, после чего добавляется новая запись об оценке по соответствующему предмету для этого студента, а затем обновляется его средний балл, при условии успешного выполнения всех операций, иначе транзакция отменяется.



Рисунок 18 — Блок-схема хранимой процедуры №1

Код процедуры:

```

CREATE OR REPLACE PROCEDURE update_student_performance(
    IN p_group_name VARCHAR(256),
    IN p_student_full_name VARCHAR(256),
    IN p_subject_name VARCHAR(256),
    IN p_mark_performace NUMERIC,
    IN p_current_year INT,
    IN p_current_semester VARCHAR(6),
    IN p_teacher_login VARCHAR(256)
)
AS $$

DECLARE
    v_id_number_student NUMERIC(10);
    v_id_teacher INT;

```

```

BEGIN

    -- Получаем идентификатор студента по группе и полному имени
    SELECT s.id_number_student
        INTO v_id_number_student
        FROM student s
        JOIN group_data g ON s.full_name_group = g.full_name_group
        JOIN user_data u ON s.login_user = u.login_user
        WHERE g.full_name_group = p_group_name
        AND u.full_name_user = p_student_full_name;

    -- Получаем идентификатор преподавателя по логину
    SELECT t.id_teacher
        INTO v_id_teacher
        FROM teacher t
        WHERE t.login_user = p_teacher_login;

    -- 1. Добавление новой оценки за экзамен для студента
    INSERT INTO perfomance (id_number_student, name_subject, mark_perfomance, date_perfomance, semester_perfomance, id_teacher)
    VALUES (v_id_number_student, p_subject_name, p_mark_perfomance, CURRENT_DATE, p_current_semester, v_id_teacher);

    -- 2. Обновление среднего балла для студента
    UPDATE perfomance
        SET
            student_average_mark =
                SELECT AVG(mark_perfomance)
                FROM perfomance p
                WHERE p.id_number_student = v_id_number_student
                AND p.name_subject = p_subject_name
                AND EXTRACT(YEAR FROM p.date_perfomance) = p_current_year
                AND p.semester_perfomance = p_current_semester
        )
        WHERE id_number_student = v_id_number_student
        AND name_subject = p_subject_name
        AND EXTRACT(YEAR FROM date_perfomance) = p_current_year
        AND semester_perfomance = p_current_semester;

    END;
$$ LANGUAGE plpgsql;

```

## 2.8.2. Триггер №1

Формулировка транзакции на естественном языке:

Триггер, обновляющий средний балл по предмету студента в текущем семестре текущего года.

Блок-схема алгоритма транзакции:

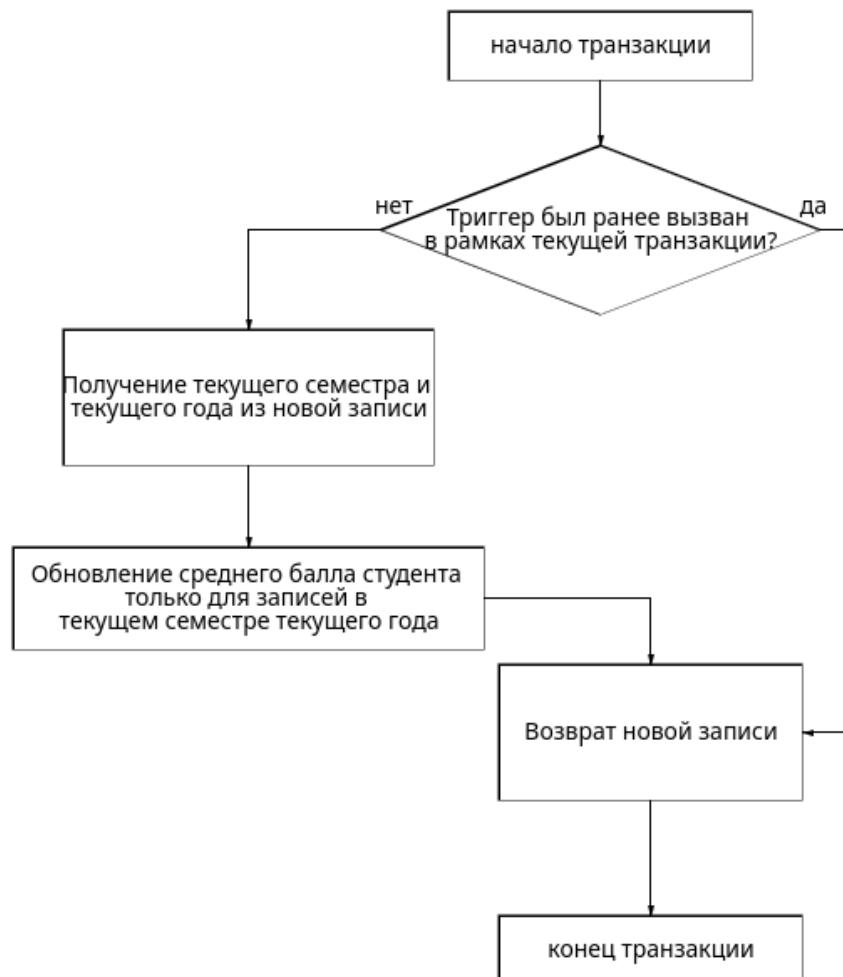


Рисунок 19 — Блок-схема триггера №1

Код триггера:

```
-- Триггер, обновляющий средний балл по предмету студента в текущем семестре текущего года
CREATE OR REPLACE FUNCTION update_student_average_mark() RETURNS TRIGGER AS $$
DECLARE
    is_recursive_update BOOLEAN;
    current_semester VARCHAR(10);
    current_year VARCHAR(4);
BEGIN
    -- Проверяем, был ли триггер вызван ранее в рамках текущей транзакции
    SELECT (pg_trigger_depth() > 1) INTO is_recursive_update;
```

```

-- Если это не рекурсивный вызов триггера, выполняем обновление

IF NOT is_recursive_update THEN

    -- Получаем текущий семестр и текущий год из новой записи
    current_semester := NEW.semester_perfomance::VARCHAR;
    current_year := EXTRACT(YEAR FROM NEW.date_perfomance)::VARCHAR;

    -- Обновляем средний балл студента только для записей в текущем семестре текущего года
    UPDATE perfomance
    SET student_average_mark =
        SELECT AVG(mark_perfomance)
        FROM perfomance p
        WHERE p.id_number_student = NEW.id_number_student
        AND p.semester_perfomance = current_semester
        AND EXTRACT(YEAR FROM p.date_perfomance) = current_year::INT -- Явное преобразование в числовой тип данных
    )
    WHERE perfomance.id_number_student = NEW.id_number_student
    AND perfomance.semester_perfomance = current_semester
    AND EXTRACT(YEAR FROM perfomance.date_perfomance) = current_year::INT; -- Явное преобразование в числовой тип данных

    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

```

CREATE OR REPLACE TRIGGER update_average_mark_trigger
AFTER INSERT OR UPDATE ON perfomance
FOR EACH ROW
EXECUTE FUNCTION update_student_average_mark();

```

### **2.8.3. Триггер №2**

**Формулировка транзакции на естественном языке:**

**Обновить значение семестра в записи успеваемости, в зависимости от даты выставления оценки.**

**Блок-схема алгоритма транзакции:**

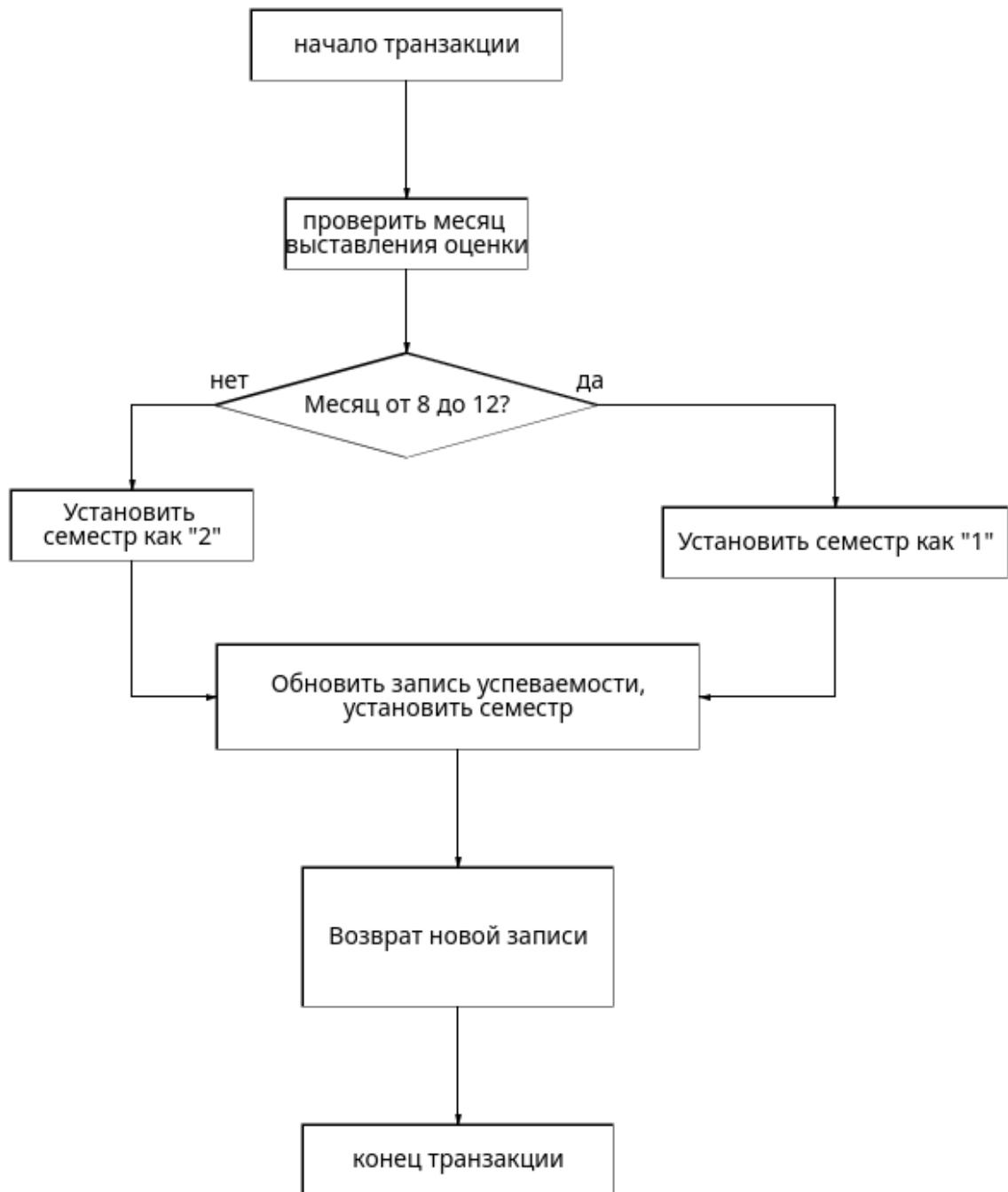


Рисунок 20 — блок-схема триггера №2

### 3. Рабочий проект

#### 3.1. SQL-скрипт создания структуры БД

SQL-скрипт, предназначенный для создания структуры БД рабочего проекта представлен в приложении А.

#### 3.2. SQL-скрипт триггеров и хранимых процедур

SQL-скрипты, предназначенные для создания триггеров, а также для

создания хранимых процедур представлены в приложении Б.

### 3.3. Текст программы

Основные модули программы в текстовом формате представлены в приложении В.

### 3.4. Руководство пользователя

Руководство пользователя представлено в приложении Г.

## Заключение

В ходе выполнения данного курсового проекта была разработана информационная система для управления учебным процессом в ВУЗе. Для решения обозначенных целей и задач ИС было реализовано приложение для ОС Linux с минимальным порогом входа для пользователей.

В разработанном приложении предусмотрен основной функционал

для учета расписаний и оценок, например:

- Добавление оценки для студента преподавателем
- Просмотр расписаний пользователями
- Добавление пользователей и разделение их функциональных возможностей в приложение в соответствие с их должностными обязанностями
- Генерация отчетов по успеваемостям с сохранением их в PDF файле

## Список использованных источников:

1. Postges API [Электронный ресурс] : Режим доступа: URL:

<https://pub.dev/documentation/postgres/latest/> , свободный — 19.05.2024

2. Unity API [Электронный ресурс] : Режим доступа: URL:

<https://docs.unity3d.com/ScriptReference/>

## Приложение А

### (SQL-скрипт создания структуры БД)

```
drop index if exists group_data_pk;
```

```
drop table if exists group_data CASCADE;
```

```
drop index if exists teacher_perfomance_fk;
```

```
drop index if exists student_perfomance_fk;
```

```
drop index if exists subject_perfomance_fk;
```

```
drop index if exists perfomance_pk;
```

```
drop table if exists perfomance CASCADE;
```

```
drop index if exists teacher_schedule_fk;
```

```
drop index if exists schedule_subject_fk;
```

```
drop index if exists schedule_pk;
```

```
drop table if exists schedule CASCADE;
```

```
drop index if exists student_is_user_fk;
```

```
drop index if exists student_group_fk;
```

```
drop index if exists student_pk;
```

```
drop table if exists student CASCADE;
```

```
drop index if exists subject_pk;
```

```
drop table if exists subject CASCADE;
```

```
drop index if exists teacher_is_user_fk;
```

```
drop index if exists teacher_pk;
```

```
drop table if exists teacher CASCADE;
```

```
drop index if exists user_data_pk;
```

```
drop table if exists user_data CASCADE;
```

```
drop index if exists schedule_has_group_fk;
```

```
drop index if exists group_has_schedules_fk;
```

```
drop index if exists group_has_schedules_pk;
```

```
drop table if exists group_has_schedules CASCADE;
```

```
/*=====*/
```

```
/* Table: group_data */  
/*=====*/  
  
create table group_data (  
  
    full_name_group    VARCHAR(256)    not null,  
  
    name_group        VARCHAR(10)     not null,  
  
    number_group      NUMERIC       not null,  
  
    constraint pk_group primary key (full_name_group)  
);
```

```
/*=====*/
```

```
/* Index: group_data_pk */  
/*=====*/  
  
create unique index group_data_pk on group_data (  
  
    full_name_group  
);
```

```
/*=====*/
```

```
/* Table: perfomance */  
/*=====*/  
  
create table perfomance (  
  
    id_perfomance    SERIAL        not null,  
  
    mark_perfomance  NUMERIC(3)    not null,  
  
    date_perfomance  DATE         not null,  
  
    semester_perfomance VARCHAR(6)  not null,  
  
    name_subject     VARCHAR(256)  not null,
```

```

    id_number_student  NUMERIC      not null,
    id_teacher        INT4         not null,
constraint pk_perfomance primary key (id_perfomance)
);

/*=====
/* Index: perfomance_pk
=====
create unique index perfomance_pk on perfomance (
    id_perfomance
);

/*=====
/* Index: subject_perfomance_fk
=====
create index subject_perfomance_fk on perfomance (
    name_subject
);

/*=====
/* Index: student_perfomance_fk
=====
create index student_perfomance_fk on perfomance (
    id_number_student
);

/*=====
/* Index: teacher_perfomance_fk
=====
create index teacher_perfomance_fk on perfomance (
    id_teacher
);

/*=====
/* Table: schedule
=====
create table schedule (
    id_schedule     SERIAL      not null,
    class_schedule  VARCHAR(256) not null,
    academic_year_schedule NUMERIC(9)      not null,

```

```

semester_schedule  NUMERIC(1)      not null,
weekday_schedule  VARCHAR(256)    not null,
start_time_schedule TIME         not null,
name_subject      VARCHAR(256)    not null,
id_teacher        INT4          null,
constraint pk_schedule primary key (id_schedule)
);

/*=====
/* Index: schedule_pk
*/
/*=====*/
create unique index schedule_pk on schedule (
id_schedule
);

/*=====
/* Index: schedule_subject_fk
*/
/*=====*/
create index schedule_subject_fk on schedule (
name_subject
);

/*=====
/* Index: teacher_schedule_fk
*/
/*=====*/
create index teacher_schedule_fk on schedule (
id_teacher
);

/*=====
/* Table: student
*/
/*=====*/
create table student (
id_number_student  NUMERIC      not null,
address_student    VARCHAR(256)   not null,
birth_date_student DATE        not null,
full_name_group    VARCHAR(256)   not null,
login_user         VARCHAR(256)   not null,
constraint pk_student primary key (id_number_student)
);

```

```

/*=====
/* Index: student_pk
*/
=====*/
create unique index student_pk on student (
    id_number_student
);

/*=====
/* Index: student_group_fk
*/
=====*/
create index student_group_fk on student (
    full_name_group
);

/*=====
/* Index: student_is_user_fk
*/
=====*/
create index student_is_user_fk on student (
    login_user
);

/*=====
/* Table: subject
*/
=====*/
create table subject (
    name_subject      VARCHAR(256)      not null,
    constraint pk_subject primary key (name_subject)
);

/*=====
/* Index: subject_pk
*/
=====*/
create unique index subject_pk on subject (
    name_subject
);

/*=====
/* Table: teacher
*/
=====*/

```

```

create table teacher (
    id_teacher      SERIAL      not null,
    qualification_teacher VARCHAR(256)      not null,
    phone_number_teacher NUMERIC(11)      null,
    login_user      VARCHAR(256)      not null,
    constraint pk_teacher primary key (id_teacher)
);

/*=====
/* Index: teacher_pk
*/
/*=====*/
create unique index teacher_pk on teacher (
    id_teacher
);

/*=====
/* Index: teacher_is_user_fk
*/
/*=====*/
create index teacher_is_user_fk on teacher (
    login_user
);

/*=====
/* Table: user_data
*/
/*=====*/
create table user_data (
    login_user      VARCHAR(256)      not null,
    full_name_user  VARCHAR(256)      not null,
    role_user       VARCHAR(256)      not null,
    constraint pk_user primary key (login_user)
);

/*=====
/* Index: user_data_pk
*/
/*=====*/
create unique index user_data_pk on user_data (
    login_user
);

/*=====

```

```

/* Table: group_has_schedules */



create table group_has_schedules (
    id_schedule      INT4          not null,
    full_name_group  VARCHAR(256)   not null,
    constraint pk_group_has_schedules primary key (id_schedule, full_name_group)
);

/*=====*/
/* Index: group_has_schedules_pk */
/*=====*/

create unique index group_has_schedules_pk on group_has_schedules (
    id_schedule,
    full_name_group
);

/*=====*/
/* Index: group_has_schedules_fk */
/*=====*/

create index group_has_schedules_fk on group_has_schedules (
    full_name_group
);

/*=====*/
/* Index: schedule_has_group_fk */
/*=====*/

create index schedule_has_group_fk on group_has_schedules (
    id_schedule
);

alter table perfomance
    add constraint fk_performan_student_p_student foreign key (id_number_student)
        references student (id_number_student)
        on delete cascade on update cascade;

alter table perfomance
    add constraint fk_performan_subject_p_subject foreign key (name_subject)
        references subject (name_subject)
        on delete cascade on update cascade;

```

```
alter table perfomance
    add constraint fk_perfoman_teacher_p_teacher foreign key (id_teacher)
        references teacher (id_teacher)
    on delete cascade on update cascade;
```

```
alter table schedule
    add constraint fk_schedule_schedule__subject foreign key (name_subject)
        references subject (name_subject)
    on delete set null on update restrict;
```

```
alter table schedule
    add constraint fk_schedule_teacher_s_teacher foreign key (id_teacher)
        references teacher (id_teacher)
    on delete set null on update restrict;
```

```
alter table student
    add constraint fk_student_student_g_group foreign key (full_name_group)
        references group_data (full_name_group)
    on delete restrict on update restrict;
```

```
alter table student
    add constraint fk_student_student_i_user foreign key (login_user)
        references user_data (login_user)
    on delete restrict on update restrict;
```

```
alter table teacher
    add constraint fk_teacher_teacher_i_user foreign key (login_user)
        references user_data (login_user)
    on delete restrict on update restrict;
```

```
alter table group_has_schedules
    add constraint fk_group_ha_group_has_group foreign key (full_name_group)
        references group_data (full_name_group)
    on delete set null on update restrict;
```

```
alter table group_has_schedules
    add constraint fk_group_ha_schedule__schedule foreign key (id_schedule)
        references schedule (id_schedule)
    on delete set null on update restrict;
```

```

-- Insert data into group_data table
INSERT INTO group_data (full_name_group, name_group, number_group) VALUES
('Computer Science 101', 'CS101', 1),
('Mathematics 202', 'MATH202', 2),
('Physics 303', 'PHYS303', 3);

-- Insert data into user_data table (for students and teachers)
INSERT INTO user_data (login_user, full_name_user, role_user) VALUES
('john_doe', 'John Doe', 'student'),
('jane_smith', 'Jane Smith', 'student'),
('peter_parker', 'Peter Parker', 'teacher'),
('mary_jane', 'Mary Jane', 'teacher');

-- Insert data into student table
INSERT INTO student (id_number_student, address_student, birth_date_student, full_name_group, login_user) VALUES
(1, '123 Main St', '2000-05-15', 'Computer Science 101', 'john_doe'),
(2, '456 Oak Ave', '2001-07-20', 'Mathematics 202', 'jane_smith');

-- Insert data into teacher table
INSERT INTO teacher (id_teacher, qualification_teacher, phone_number_teacher, login_user) VALUES
(1, 'PhD in Computer Science', 1234567890, 'peter_parker'),
(2, 'MSc in Mathematics', 9876543210, 'mary_jane');

-- Insert data into subject table
INSERT INTO subject (name_subject) VALUES
('Introduction to Computer Science'),
('Calculus I'),
('Physics for Engineers');

-- Insert data into schedule table
INSERT INTO schedule (id_schedule, class_schedule, academic_year_schedule, semester_schedule, weekday_schedule, start_time_schedule, name_subject, id_teacher) VALUES
(1, 'Room 101', 2024, 1, 'Monday', '09:00:00', 'Introduction to Computer Science', 1),
(2, 'Room 202', 2024, 1, 'Wednesday', '11:00:00', 'Calculus I', 2),
(3, 'Room 303', 2024, 1, 'Friday', '14:00:00', 'Physics for Engineers', 1);

-- Insert data into performance table
INSERT INTO performance (id_performance, mark_performance, date_performance, semester_performance, name_subject, id_number_student, id_teacher) VALUES
(1, 95, '2024-05-10', '2024-2', 'Introduction to Computer Science', 1, 1),
(2, 88, '2024-05-11', '2024-2', 'Calculus I', 2, 2);

```

```
-- Insert data into group_has_schedules table
INSERT INTO group_has_schedules (id_schedule, full_name_group) VALUES
(1, 'Computer Science 101'),
(2, 'Mathematics 202'),
(3, 'Physics 303');
```

## Приложение Б

### (SQL-скрипты триггеров и хранимой процедуры)

-- Триггер, обновляющий средний балл по предмету студента в текущем семестре текущего года

```
CREATE OR REPLACE FUNCTION update_student_average_mark() RETURNS TRIGGER AS $$
```

```
DECLARE
```

```
    is_recursive_update BOOLEAN;
```

```
    current_semester VARCHAR(10);
```

```
    current_year VARCHAR(4);
```

```
BEGIN
```

-- Проверяем, был ли триггер вызван ранее в рамках текущей транзакции

```
SELECT (pg_trigger_depth() > 1) INTO is_recursive_update;
```

-- Если это не рекурсивный вызов триггера, выполняем обновление

```
IF NOT is_recursive_update THEN
```

-- Получаем текущий семестр и текущий год из новой записи

```
    current_semester := NEW.semester_perfomance::VARCHAR;
```

```
    current_year := EXTRACT(YEAR FROM NEW.date_perfomance)::VARCHAR;
```

-- Обновляем средний балл студента только для записей в текущем семестре текущего года

```
UPDATE perfomance
```

```
SET student_average_mark =
```

```
    SELECT AVG(mark_perfomance)
```

```
    FROM perfomance p
```

```
    WHERE p.id_number_student = NEW.id_number_student
```

```
        AND p.semester_perfomance = current_semester
```

```
        AND EXTRACT(YEAR FROM p.date_perfomance) = current_year::INT -- Явное преобразование в числовой тип данных
```

```
)
```

```
WHERE perfomance.id_number_student = NEW.id_number_student
```

```
    AND perfomance.semester_perfomance = current_semester
```

```
    AND EXTRACT(YEAR FROM perfomance.date_perfomance) = current_year::INT; -- Явное преобразование в числовой тип данных
```

```
END IF;
```

```
RETURN NEW;
```

```
END;
```

```

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER update_average_mark_trigger
AFTER INSERT OR UPDATE ON perfomance
FOR EACH ROW
EXECUTE FUNCTION update_student_average_mark();

CREATE OR REPLACE FUNCTION update_semester_perfomance_trigger()
RETURNS TRIGGER AS $$

BEGIN
    IF EXTRACT(MONTH FROM NEW.date_perfomance) BETWEEN 8 AND 12 THEN
        NEW.semester_perfomance := '1';
    ELSE
        NEW.semester_perfomance := '2';
    END IF;
    RETURN NEW;
END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER update_semester_perfomance_trigger
BEFORE INSERT ON perfomance
FOR EACH ROW
EXECUTE FUNCTION update_semester_perfomance_trigger();

CREATE OR REPLACE PROCEDURE update_student_performance(
    IN p_group_name VARCHAR(256),
    IN p_student_full_name VARCHAR(256),
    IN p_subject_name VARCHAR(256),
    IN p_mark_perfomance NUMERIC,
    IN p_current_year INT,
    IN p_current_semester VARCHAR(6),
    IN p_teacher_login VARCHAR(256)
)
AS $$

DECLARE
    v_id_number_student NUMERIC(10);
    v_id_teacher INT;

BEGIN
    -- Получаем идентификатор студента по группе и полному имени
    SELECT s.id_number_student

```

```

INTO v_id_number_student

FROM student s

JOIN group_data g ON s.full_name_group = g.full_name_group

JOIN user_data u ON s.login_user = u.login_user

WHERE g.full_name_group = p_group_name

AND u.full_name_user = p_student_full_name;

-- Получаем идентификатор преподавателя по логину

SELECT t.id_teacher

INTO v_id_teacher

FROM teacher t

WHERE t.login_user = p_teacher_login;

-- 1. Добавление новой оценки для студента

INSERT INTO perfomance (id_number_student, name_subject, mark_perfomance, date_perfomance, semester_perfomance, id_teacher)

VALUES (v_id_number_student, p_subject_name, p_mark_perfomance, CURRENT_DATE, p_current_semester, v_id_teacher);

-- 2. Обновление среднего балла для студента

UPDATE perfomance

SET

student_average_mark = (

SELECT AVG(mark_perfomance)

FROM perfomance p

WHERE p.id_number_student = v_id_number_student

AND p.name_subject = p_subject_name

AND EXTRACT(YEAR FROM p.date_perfomance) = p_current_year

AND p.semester_perfomance = p_current_semester

)

WHERE id_number_student = v_id_number_student

AND name_subject = p_subject_name

AND EXTRACT(YEAR FROM date_perfomance) = p_current_year

AND semester_perfomance = p_current_semester;

END;

$$ LANGUAGE plpgsql;

```

## Приложение В

### (Текст программы)

**Загрузка и подключение к БД:**

```
using System.Collections.Generic;
using System.Data;
using UnityEngine;
using UnityNpgsql;

public class DatabaseManager : MonoBehaviour
{
    public static DatabaseManager Instance { get { return _instance; } }

    private static DatabaseManager _instance;

    private const string connectionString = "Host=localhost;Username=postgres;Password=;Database=lk";

    private void Awake()
    {
        if (_instance != null && _instance != this)
        {
            Destroy(this.gameObject);
        }
        else
        {
            _instance = this;
            DontDestroyOnLoad(this.gameObject);
        }
    }

    public DataTable ExecuteQuery(string query)
    {
        using (NpgsqlConnection connection = new NpgsqlConnection(connectionString))
        {
            connection.Open();

            using (NpgsqlCommand command = new NpgsqlCommand(query, connection))

```

```

    {
        using (NpgsqlDataAdapter adapter = new NpgsqlDataAdapter(command))
        {
            DataTable dataTable = new DataTable();
            adapter.Fill(dataTable);
            return dataTable;
        }
    }
}

public List<Dictionary<string, string>> ExecuteQueryList(string query)
{
    List<Dictionary<string, string>> result = new List<Dictionary<string, string>>();

    using (NpgsqlConnection connection = new NpgsqlConnection(connectionString))
    {
        connection.Open();

        using (NpgsqlCommand command = new NpgsqlCommand(query, connection))
        {
            using (NpgsqlDataReader reader = command.ExecuteReader())
            {
                while (reader.Read())
                {
                    Dictionary<string, string> row = new Dictionary<string, string>();

                    for (int i = 0; i < reader.FieldCount; i++)
                    {
                        string columnName = reader.GetName(i);
                        string value = reader[i].ToString();
                        row.Add(columnName, value);
                    }

                    result.Add(row);
                }
            }
        }
    }
}

```

```

        return result;
    }

    public int GetTeacherIdByLogin(string login)
    {
        string query = $"SELECT id_teacher FROM teacher WHERE login_user = '{login}';";
        List<Dictionary<string, string>> result = ExecuteQueryList(query);

        if (result.Count > 0 && result[0].TryGetValue("id_teacher", out string id))
        {
            if (int.TryParse(id, out int teacherId))
            {
                Debug.Log("[GetTeacherIdByLogin] return teacherId");
                return teacherId;
            }
        }

        Debug.Log("[GetTeacherIdByLogin] return -1");
        return -1;
    }

    public string GetUserNameByLogin(string login)
    {
        string query = $"SELECT full_name_user FROM user_data WHERE login_user = '{login}';";
        List<Dictionary<string, string>> result = ExecuteQueryList(query);

        if (result.Count > 0 && result[0].TryGetValue("full_name_user", out string userName))
        {
            Debug.Log($"[GetUserNameByLogin] return {userName}");
            return userName;
        }

        Debug.Log("[GetUserNameByLogin] return null");
        return null;
    }
}

```

### Заполнение данных пользователя:

```

using TMPro;
using UnityEngine;
using System.Collections.Generic;

```

```
public class UserProfileManager : MonoBehaviour
{
    [SerializeField]
    private TMP_Text usernameText;
    [SerializeField]
    private TMP_Text roleText;
    [SerializeField]
    private TMP_Text additionalDataText;
    [SerializeField]
    private TMP_FontAsset fontAsset;

    private void Start()
    {
        Init();
    }

    private void Init()
    {
        Dictionary<string, string> engRu = new()
        {
            {"teacher", "Преподаватель"},
            {"student", "Студент"}
        };

        string login = UserManager.Login;
        string role = UserManager.Role;
        string additionalData = UserManager.AdditionalData;

        usernameText.text = $"Имя пользователя: <u>{DatabaseManager.Instance.GetUserNameByLogin(login)}</u>";
        roleText.text = $"Должность: <u>{engRu[role]}</u>";
        additionalDataText.text = $"{additionalData}";

        usernameText.font = fontAsset;
        roleText.font = fontAsset;
        additionalDataText.font = fontAsset;
    }
}
```

**Метод, позволяющий получить успеваемость студента по выбранному им предмету за выбранный семестр:**

```
private void ViewMyPerformance()
{
    currentSemester = GetSelectedDropdownOptionText(instantiatedSemesterDropdown);
    subjectDropdownSelectedVal = GetSelectedDropdownOptionText(instantiatedSubjectDropdown);

    string[] semesterParts = currentSemester.Split('-');
    string currentYear = semesterParts[0];
    string currentSemesterNumber = semesterParts[1];

    string query =
        "SELECT p.mark_perform AS mark, " +
        "TO_CHAR(p.date_perform, 'DD/MM') as date_perform, " +
        "p.name_subject AS subject_name, " +
        "p.student_average_mark as average_mark, " +
        "t.qualification_teacher AS teacher_qualification, " +
        "u.full_name_user AS teacher_full_name " +
        "FROM Performance p " +
        "JOIN Subject s ON p.name_subject = s.name_subject " +
        "LEFT JOIN Teacher t ON p.id_teacher = t.id_teacher " +
        "LEFT JOIN user_data u ON t.login_user = u.login_user " +
        "WHERE p.id_number_student = " +
        $"(SELECT id_number_student FROM Student WHERE login_user = '{UserManager.Login}') " +
        $"AND EXTRACT(YEAR FROM p.date_perform) = '{currentYear}' " +
        $"AND p.semester_perform = '{currentSemesterNumber}' " +
        $"AND p.name_subject = '{subjectDropdownSelectedVal}';";

    List<Dictionary<string, string>> performanceData = DatabaseManager.Instance.ExecuteQueryList(query);

    if (DatabaseTableViewer != null)
    {
        DatabaseTableViewer.ToggleTable(performanceData);
    }
}
```

# Приложение Г

## (Руководство пользователя)

### **1. Введение**

#### **1.1 Область применения**

Информационная система разработана для использования студентами и преподавателями, а также административным персоналом высшего учебного заведения для управления учебным процессом, включая планирование занятий, назначение преподавателей, учет оценок успеваемости студентов.

#### **1.2 Краткое описание возможностей**

В данной информационной системе администраторы могут создавать и редактировать учебные курсы, назначать преподавателей на занятия, преподаватели могут вести учет оценок и успеваемости студентов, а студенты могут просматривать расписание и следить за своими отметками.

#### **1.3 Уровень подготовки пользователя**

Для работы с данной информационной системой пользователь должен обладать базовыми навыками работы с компьютером и программными интерфейсами, а также иметь представление о структуре учебного процесса в высшем учебном заведении

#### **1.4 Перечень эксплуатационной документации**

Пользователь должен ознакомиться с настоящим руководством пользователя. Рекомендуется также ознакомиться с техническим заданием и техническим проектом разрабатываемой информационной системы.

### **2. Назначение и условия применения**

#### **2.1 Виды деятельности, функции, для автоматизации которых предназначено данное средство автоматизации**

Это информационная система предназначена для управления учебным процессом в высшем учебном заведении, включая планирование курсов, ведение журналов оценок и мониторинг расписаний.

#### **2.2 Условия, при соблюдении которых обеспечивается применение средства автоматизации в соответствии с назначением**

Для использования данной информационной системы необходим доступ файлам проекта, а также базовые навыки работы с компьютером и интернетом.

### **3 Подготовка к работе**

#### **3.1 Состав и содержание дистрибутивного носителя данных**

Разработанное ПО представляет собой приложение для ПК на базе ОС Linux, распространяемое в сети интернет с согласия владельца.

#### **3.2 Порядок загрузки данных и программ**

Для загрузки разработанного приложения необходимо перейти по ссылке, полученной от владельца или другого лица, имеющего согласие на распространение данного продукта в сети интернет. После чего необходимо произвести перенос приложения в папку для хранения

#### **3.3 Порядок проверки работоспособности**

После успешного запуска приложения должно появиться экран авторизации с картинкой. После входа будут доступны все функции, в соответствии с ролями

## **4 Описание операций**

### **4.1 Описание всех выполняемых функций, задач, комплексов задач, процедур**

Таблица 5 - Описание всех выполняемых функций, задач, комплексов задач, процедур

<b>Функция</b>	<b>Задача</b>	<b>Описание</b>
Поставить оценку	Создание новой записи оценки в таблице perfomance в БД	В ходе выполнения данной функции будет создана запись в базе данных
Изменить расписание	Изменение записи расписания в таблице schedule в БД	В ходе выполнения данной функции будет изменена запись в базе данных
Посмотреть расписание	Отобразить все соответствующие записи в таблице расписании пользователя	В ходе выполнения данной функции будут показаны все релевантные записи в базе данных
Посмотреть оценки	Отобразить все соответствующие записи в таблице успеваемости студента	В ходе выполнения данной функции будут показаны все релевантные записи в базе данных
Создание отчета по успеваемости	Создание документа в формате pdf с содержанием виде отчёта по заказанным запчастям по ремонту	В ходе выполнения данной функции программа генерирует файл в формате pdf в корневую папку

### **4.2 Описание операций технологического процесса обработки данных**

#### **4.2.1 Наименование задачи: «Создание новой оценки в базе данных»**

##### **4.2.1.1 Условия выполнения операции:**

- Пользователь прошел авторизацию, его роль: «преподаватель» и он ведет занятия у студента, оценку которому он хочет поставить.

##### **4.2.1.2 Подготовительные действия**

1. Пройти авторизацию

2. Быть преподавателем

3. Выбрать интересующий семестр

4.2.1.3 Последовательность действий

1. Нажать на кнопку 'Поставить оценку'

2. Ввести необходимую информацию

3. Подтвердить добавление новой оценки

4.2.2 Наименование задачи: «Изменить расписание»

4.2.2.1 Условия выполнения операции:

- Пользователь прошел авторизацию

- Роль пользователя не студент

4.2.2.2 Подготовительные действия

- Пройти авторизацию

- Нажать на кнопку 'Управление расписанием'

4.2.2.3 Последовательность действий

1. Выбрать интересующий семестр из списка

2. Нажать на кнопку 'Управление расписанием'

3. Отредактировать любой интересующий элемент в полученной таблице

4. Подтвердить редактирование

4.2.3 Наименование задачи: «Посмотреть расписание»

4.2.3.1 Условия выполнения операции:

- Пользователь прошел авторизацию

4.2.3.2 Подготовительные действия

- Пройти авторизацию

- Нажать на кнопку 'Посмотреть расписание'

4.2.3.3 Последовательность действий

1. Выбрать интересующий семестр из списка

2. Нажать на кнопку 'Посмотреть расписание'

3. Увидеть расписание выбранного семестра.

#### **4.2.4 Наименование задачи: «Посмотреть расписание»**

##### **4.2.4.1 Условия выполнения операции:**

- Пользователь прошел авторизацию

##### **4.2.4.2 Подготовительные действия**

- Пройти авторизацию
- Нажать на кнопку 'Посмотреть расписание'

##### **4.2.4.3 Последовательность действий**

1. Выбрать интересующий семестр из списка
2. Нажать на кнопку 'Посмотреть расписание'
3. Увидеть расписание выбранного семестра.

#### **4.2.4 Наименование задачи: «Посмотреть оценки»**

##### **4.2.4.1 Условия выполнения операции:**

- Пользователь прошел авторизацию

##### **4.2.4.2 Подготовительные действия**

- Пройти авторизацию
- Нажать на кнопку 'Посмотреть оценки'

##### **4.2.4.3 Последовательность действий**

1. Выбрать интересующий семестр из списка
2. Выбрать интересующий предмет из списка
3. Нажать на кнопку 'Посмотреть оценки'
4. Увидеть расписание выбранного семестра и предмета.

#### **4.2.5 Наименование задачи: «Создание отчета по успеваемости»**

##### **4.2.5.1 Условия выполнения операции:**

- Пользователь прошел авторизацию
- Роль пользователя — преподаватель
- Была просмотрена интересующая таблица с оценками студентов

##### **4.2.5.2 Подготовительные действия**

- Пройти авторизацию
- Посмотреть интересующую таблицу по оценкам студентов

#### 4.2.5.3 Последовательность действий

1. Нажать на кнопку 'Сгенерировать отчет'
2. Сгенерированный отчет в формате PDF будет автоматически открыт

### 5. Аварийные ситуации

Таблица 6 - Описание аварийных ситуаций

<b>Ошибка</b>	<b>Описание ошибки</b>	<b>Действия пользователя</b>
Указан неверный логин или пароль	Пользователь неверно указал логин или парол	Проверить корректность введенных данных и повторить попытку
Не получается зайти в систему	Не получается подключиться к БД	Перезапустить приложение, если не поможет, то и компьютер.

### 6. Рекомендации по освоению

Для полноценного освоения программы обязательно необходимо произвести все действия самостоятельно без опорного материала.