МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет безопасности информационных технологий

Направление подготовки: Выберите элемент.
Образовательная программа: Выберите элемент.

Дисциплина:
«Информационная безопасность баз данных**»**

КУРСОВАЯ РАБОТА

на тему *«Создание информационной системы автошколы»*

Выполнил студент(ы):
группа/поток N3350
Мясников Е.С. /_____
ФИО *Подпись*

Проверил:
Ярцева Н.А /_____
ФИО *Подпись*

_____
Отметка о выполнении (один из вариантов: отлично, хорошо, удовлетворительно)

_____
Дата

Санкт-Петербург
2025г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)

Студент Мясников Е. С.

(Фамилия, И., О.)

Факультет ФБИТ

Группа N3350

Направление (специальность) Информационная безопасность

Руководитель Ярцева Н.А, преподаватель

(Фамилия, И.О., должность, ученое звание, степень)

Дисциплина Информационная безопасность баз данных

Наименование темы Создание информационной системы автошколы

Задание

Какая предметная область была выбрана? Основные сущности, связи, ограничения. Какой Комплекс мер защиты БД был разработан? Какой стек технологий использовался?

Краткие методические указания

Содержание пояснительной записки

1-Анализ информационной системы; 2-Реализация БД в рамках СУБД; 3-Защита БД; 4-Тестирование; 5-Аудит безопасности разработанного сервиса.

Рекомендуемая литература

Руководитель

Подпись, дата

Студент

Подпись, дата

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ГРАФИК ВЫПОЛНЕНИЯ КУРСОВОГО ПРОЕКТА (РАБОТЫ)

Студент Мясников Е.С.
(Фамилия, И.О.)

Факультет ФБИТ

Группа N3350

Направление (специальность) Информационная безопасность

Руководитель Ярцева Н.А, преподаватель
(Фамилия, И.О., место работы, должность, ученое звание, степень)

Дисциплина Информационная безопасность баз данных

Наименование темы Создание информационной системы автошколы

№ п/п	Наименование этапа	Дата завершения		Оценка и подпись руководителя
		Планируемая	Фактическая	
1	Выбор ИС	28.09	27.09	
2	Реализация БД	20.10	22.10	
3	Разработка защиты	01.11	01.11	
4	Реализация веб-приложения	20.11	21.11	
…				

Руководитель _____
подпись, дата

Студент _____
подпись, дата

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

АННОТАЦИЯ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)

Студент Мясников Е.С.
(Фамилия, И.О.)

Факультет ФБИТ

Группа N3350

Направление (специальность) Информационная безопасность

Руководитель Ярцева Н.А, преподаватель
(Фамилия, И.О., место работы, должность, ученое звание, степень)

Дисциплина Информационная безопасность баз данных

Наименование темы Создание информационной системы автошколы

ХАРАКТЕРИСТИКА КУРСОВОГО ПРОЕКТА (РАБОТЫ)

1. Цель и задачи работы

☐ Предложены студентом
☐ Сформулированы при участии студента
☐ Определены руководителем

2. Характер работы

☐ Расчет
☐ Конструирование
☐ Моделирование
☐ Другое, _____

4. Содержание работы

5. Выводы

Данные об основных полученных результатах, их новизне, достоверности, применимости для теории и практики.

Студент_____
(подпись)

Руководитель____

(подпись)

«___» ________________20___г.

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИ

БД - база данных

ИС - информационная система

СУБД - система управления базами данных

ТО - техническое обслуживание

1НФ - первая нормальная форма

2НФ - вторая нормальная форма

3НФ - третья нормальная форма

ER-диаграмма (Entity-Relationship Diagram) - диаграмма “сущность-связь”

pgAdmin - графический интерфейс для управления СУБД PostgreSQL

PostgreSQL - реляционная система управления базами данных

НСД - несанкционированный доступ

ВВЕДЕНИЕ

Актуальность разработки определяется необходимостью автоматизации процессов в образовательных организациях, оказывающих услуги подготовки водителей. Современные автошколы обрабатывают значительные объёмы данных. При ручном ведении таких данных возникают риски ошибок, дублирования, утери информации и нарушения целостности расписания, а для обеспечения корректной работы всех механизмов необходима хорошо спроектированная и защищённая база данных, способная гарантировать согласованность, структурированность и доступность информации.

Целью курсовой работы является разработка базы данных информационной системы автошколы и создание многоуровневого веб-сервиса, обеспечивающего безопасный многофункциональный доступ пользователей к данным.

Для достижения были поставлены следующие **задачи**:

- провести анализ предметной области автошколы, выделить сущности, связи и ограничения, определить источники данных и потребителей информации;
- выполнить инфологическое моделирование БД;
- привести структуру к третьей нормальной форме (3НФ), устранить нарушения атомарности и транзитивных зависимостей;
- реализовать физическую модель базы данных в СУБД PostgreSQL;
- разработать комплекс мер защиты БД;
- создать веб-приложение обеспечивающее взаимодействие с информацией;
- провести тестирование сервисов;

1. АНАЛИЗ ИНФОРМАЦИОННОЙ СИСТЕМЫ

1.1. Описание ИС

Мной была выбрана информационная система **автошколы,** предназначенная для автоматизации следующих процессов:

- **Работа с учебными планами и расписанием:** хранение учебных программ, расписаний занятий (теоретических и практических);
- **Управление учебным процессом:** отслеживание статуса обучения студента (зачислен, отчислен, завершил обучение и т.п.), регистрация студентов, формирование учебных групп;
- **Работа с автопарком и инструкторами:** информация про инструкторов (их категория), информация про машины (модель, номер, тех. Состояние и т.п), назначение автомобилей на учебный маршрут;
- **Успеваемость студентов:** результаты экзаменов, допуск к экзамену, количество попыток сдачи, посещаемость;
- **Работа с финансами:** сумма обучения студента, дата оплаты, платежи, задолженности, договоры и т.п.;

Система решает следующие задачи:

- **Централизованное хранение данных** об обучении студентов, ресурсах автошколы и оплате обучения;
- **Контроль ограничения:** один и тот же инструктор и автомобиль не может иметь две брони, аудитория не может вместить больше чем n студентов, студент, имеюший недопуск, не может сдать экзамен;
- **Организация рабочего процесса:** выбор инструктора, машины и прочее.

Источники данных:

- **Приёмная комиссия:** данные доставляются путём заполнения договоров и анкет абитуриентами, их детали договора и персональные данные;
- **Преподаватели:** именно они составляют учебные планы, теорию и расписание;
- **Механики:** они закрывают данные, связанные с авто;
- **Бухгалтеры:** все финансовые вопросы (платежи, задолженности, возвраты и прочее);
- **ГИББД:** результаты экзаменов;
- **Инструкторы по вождению:** расписание практик, посещаемость студента, результаты внутренних экзаменов.

С какой частотой обновляются данные? **Договоры** – в зависимости от событий (изменение договора студентом или при поступлении). **Расписание** – еженедельно, с возможными поправками каждый день**. Посещаемость студента и его успеваемость** – каждый день. **Авто и техническое обслуживание** – доступ к использованию – каждый день, тех. **Обслуживание** – в зависимости от автошколы. **Платежи** – ежедневно. **Результаты сдачи экзамена** – по мере сдачи.

Основные **потребители** информации:

- **Студенты:** просмотр личного расписания теоретических и практических занятий, преподавательский состав, статус допуска к экзаменам;
- **Бухгалтерия:** доступ к всей необходимой информации о договорах со студентами, графики платежей, мониторинг оплаты обучения и прочее;
- **Инструкторы:** маршруты практических поездок, закреплённая машина, её тех состояние, информация о студенте и прочее;
- **Преподаватели:** расписание занятий, занятость аудиторий, информация о группе и прочее.

Ограничения на сущности и связи:

- Инструктор не может проводить две практики одновременно;
- Автомобиль не может быть назначен на два занятия одновременно;
- Аудитория не может быть занята более чем одной группой в одно и то же время;
- Группа не может иметь два теоретических занятия одновременно;
- Группа не может превышать максимальную вместимость аудитории;
- Авто не может эксплуатироваться, если он не в статусе “доступен”;
- Допуск студента возможен если только был пройден блок теории;
- Допуск к экзамену только при прохождении установленного количества часов;
- Допуск к гос. Экзамену доступен только после получения зачёта по внутренним экзаменам;
- Для проведения занятий, инструктор и транспортное средство должно соответствовать категории обучения студента;
- Доступ к учебным занятиям доступен только после оплаты обучения.

С целью обеспечения корректности работы ограничения будут реализованы с помощью следующих комбинаций механизмов бд:

- В **PracticeLesson** вводим составной уникальный индекс (**instructor_id**, **start_time, end_time**) и (**vehicle_id, start_time, end_time**). Данные действия гарантируют, что один и тот же инструктор или автомобиль не смогут быть задействованы одновременно в двух занятиях, пересекающихся по времени;
- В **Classroom** зададим ограничение вместимости, а при назначении урока, **TheoryLesson**, выполняется проверка, что количество студентов в группе <= вместимости аудитории;
- В **Vehicle** зададим флаг status = “available”. В данном случае только такие автомобили могут назначаться на занятия;
- Внешние ключи, **FK**, обеспечат согласованность данных. Например, студент может быть записан только в существующую группу, а платёж - только к существующему договору.
- Для решения более сложных задач могут быть задействованы **триггеры**. Например, допуск к экзамену после прохождения установленного количества часов и другие условия.

1.2. Выделение сущностей и построение ER-диаграммы.

Выделим основные сущности ИС, для которой разрабатывается БД и *примерно* “наметим” структуру каждой:

- Студент (Student) – хранит информацию о конкретном студенте, его ФИО, дату рождения, почту, id. Является центральной сущностью системы, вся деятельность школы направлена на его обучение;
- Учебная программа (Program) – хранит информацию о конкретной программе подготовке (A/B/C), id программы, название, категория. Структуризация процесса подготовки студентов по разным категориям, фиксация содержания обучение;
- Группа (Group) - конкретная группа, идущая по программе. ID программы и группы, индекс группы. Объединение студентов в группы позволяет упростить назначение занятий и планирование расписания;
- Преподаватель (Instructor) – преподаватель теории или практики. Его id, id студента и машины, информация о практике. Без преподавателей невозможно обучение;
- Автомобиль (Vehicle) – учебный автомобиль. Id и тех информация о нём. Аналогично сущности “Преподаватель”;
- Аудитория (Classroom) – аудитория для проведения теоретических занятий. Id, номер здания, номер аудитории. Необходима для организации теоретических занятий;
- Модуль программы (CourseTopic) – отдельная тема внутри программы. Id темы, программы и название темы. Аналогично сущности “Учебная программа”;
- Теоретическое занятие (TheoryLesson) – конкретное занятие в расписании группы. Id занятия, группы, темы, аудитории. Время начала и конца. Фиксация конкретных событий учебного процесса и являются “узлом между преподавателем и студентом”;

- Практическое занятие (PracticeLesson) – практическое занятие по вождению. Id занятия, студента, инструктора, машины. Время начала и конца. Аналогично сущности “ Теоретическое занятие”;
- Договор (Contract) - договор на обучение между студентом и школой. Id программы, студента, контракта, дата оформления и сумма. Прозрачность и контроль оплаты обучения студентом;
- Платёж (Payment) – конкретный платёж по договору. Дата оплаты, сумма, id оплаты и договора. Аналогично сущности “ Договор”;
- Экзамен (Exam) – попытка сдачи экзамена. Id экзамена, студента, результат, дата начала и тип. Завершение процесса обучения. Его результаты определяют успешность подготовки студента.

Дополнительные сущности, без них не будет целостности:

- Зачисление студента (Enrollment) – информация о зачислении студента. Id группы, студента, зачисления, время и статус. Корректное хранение связей многие-ко-многим и контроль за прогрессом студента;
- Посещаемость теории студентом (TheoryAttendance) – посещаемость теории студентом. Id урока, студента, присутствие, оценка. Корректное хранение связей многие-ко-многим и контроль за прогрессом студента.

Связи между сущностями:

- **Program – Group (**1:M) (обяз /обяз): каждая группа может идти по одной программе, одна программа может вестись у многих групп;
- **Group - TheoryLesson** (1:M): у группы много различных теоретических занятий, а каждое занятие относится только к одной группе;
- **Classroom – TheoryLesson** (1:M): одна может вместить много занятий во времени, но каждое занятие проходит только в одной аудитории;
- **CourseTopic - TheoryLesson** (1:M): каждое занятие привязано к одной теме , а одна тема может читаться в нескольких занятиях;
- **Instructor - PracticeLesson** (1:M) (необяз/обяз): преподаватель может не иметь занятий (если например заболел), но у занятия должен быть инструктор;
- **Vehicle - PracticeLesson** (1:M) (необяз/обяз): тс может быть недоступно (если есть проблемы) или не забронировано, но у занятия должно быть обязательно ТС;
- **Student - PracticeLesson** (1:M): у студента может быть много практик, а каждое практическое занятие относится только к одному студенту;
- **Student - Contract** (1:M): у одного студента может быть несколько договоров (в случае, если студент уходил в “академ” или переоформление), договор всегда относится к студенту;
- **Program - Contract** (1:M): договор подписывается на одну конкретную программу, а программа может быть в других договорах;
- **Contract - Payment** (1:M): у договора есть множество платежей, каждый платёж относится к одному договору;
- **Group - Enrollment- Student** (M:M): в один период студент может состоять в одной группе, но он может перевестись. Как раз Enrollment будет и хранить связь;

– **TheoryLesson – TheoryAttendance – Student** (M:M): Каждое занятие посещают многие студенты, а присутствие и полученная оценка фиксируется через TheoryAttendance.

Исходя из вышеуказанных сущностей и их связей, построим ER-диаграмму для рассматриваемой предметной области:

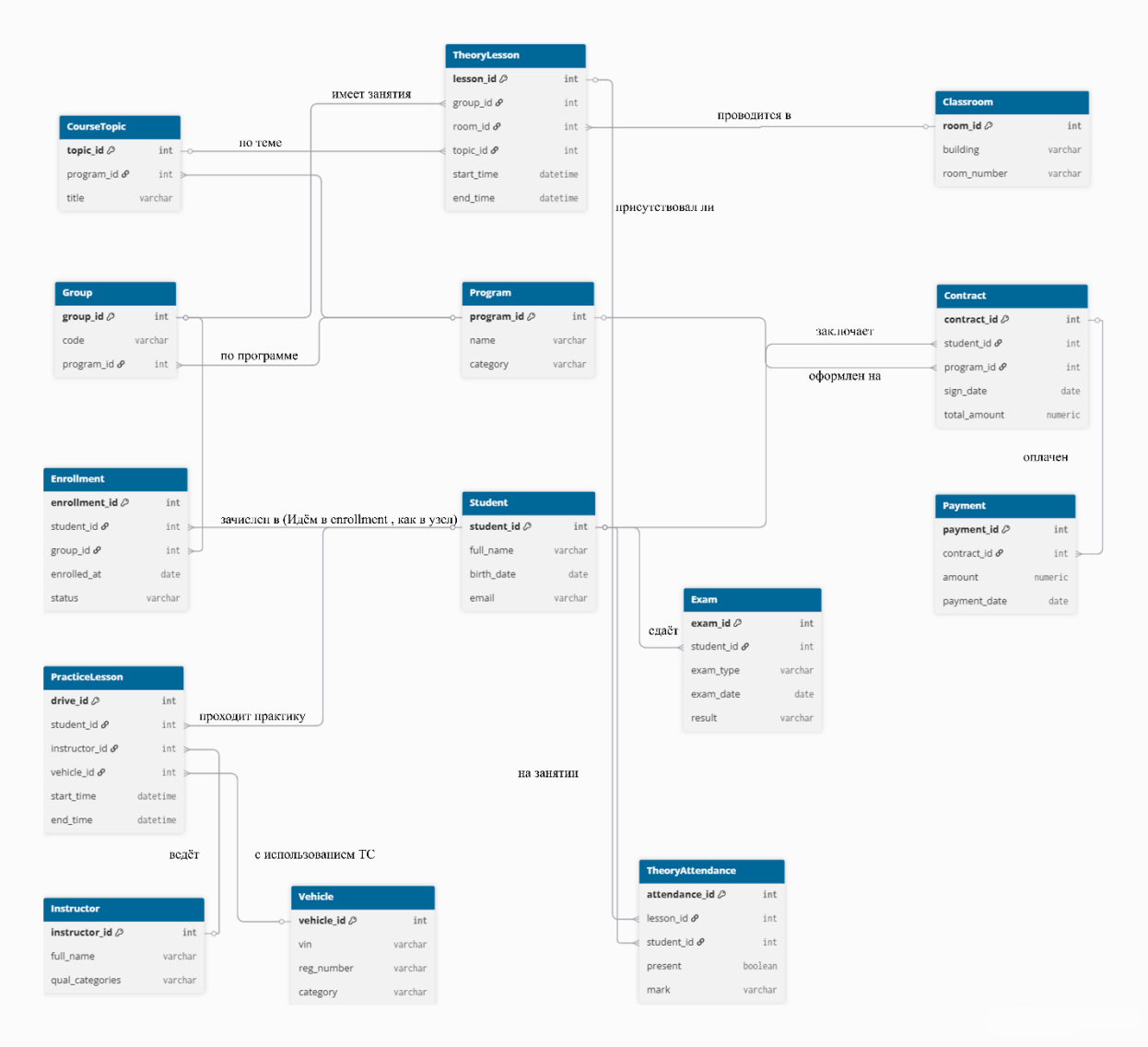


Рисунок 1. ER-диаграмма разрабатываемой БД

В результате была получена ER-диаграмма разрабатываемой БД.

1.3. Преобразование ER-диаграммы в схему отношений с помощью правил формирования предварительных отношений.

После описания сущностей, атрибутов и связей между ними, представим схему отношений:

Студент (Student):

- **Ключ**: student_id;
- **Атрибуты**: full_name, birth_date, email.

Учебная программа (Program):

- **Ключ**: program_id;
- **Атрибуты**: name, category.

Группа (Group):

- **Ключ**: group_id.
- **Атрибуты**: code (уникальный индекс группы), program_id (FK).

Преподаватель (Instructor):

- **Ключ**: instructor_id.
- **Атрибуты**: full_name, qual_categories.

Автомобиль (Vehicle):

- **Ключ**: vehicle_id.
- **Атрибуты**: vin, reg_number, category.

Аудитория (Classroom):

- **Ключ**: room_id.
- **Атрибуты**: building, room_number.

Модуль программы (CourseTopic):

- **Ключ**: topic_id.
- **Атрибуты**: program_id (FK), title.

Теоретическое занятие (TheoryLesson):

- **Ключ**: lesson_id.
- **Атрибуты**: group_id (FK), room_id (FK), topic_id (FK), start_time, end_time.

Практическое занятие (PracticeLesson):

- **Ключ**: drive_id.
- **Атрибуты**: student_id (FK), instructor_id (FK), vehicle_id (FK), start_time, end_time.

Договор (Contract):

- **Ключ**: contract_id.
- **Атрибуты**: student_id (FK), program_id (FK), sign_date, total_amount.

Платёж (Payment):

- **Ключ**: payment_id.
- **Атрибуты**: contract_id (FK), amount, payment_date.

Экзамен (Exam):

- **Ключ**: exam_id.
- **Атрибуты**: student_id (FK), exam_type, exam_date, result.

Зачисление студента (Enrollment):

- **Ключ:** enrollment_id.
- **Атрибуты**: student_id (FK), group_id (FK), enrolled_at, status.

Посещаемость теории студентом (TheoryAttendance):

- **Ключ:** attendance_id.
- **Атрибуты**: lesson_id (FK), student_id (FK), present, mark.

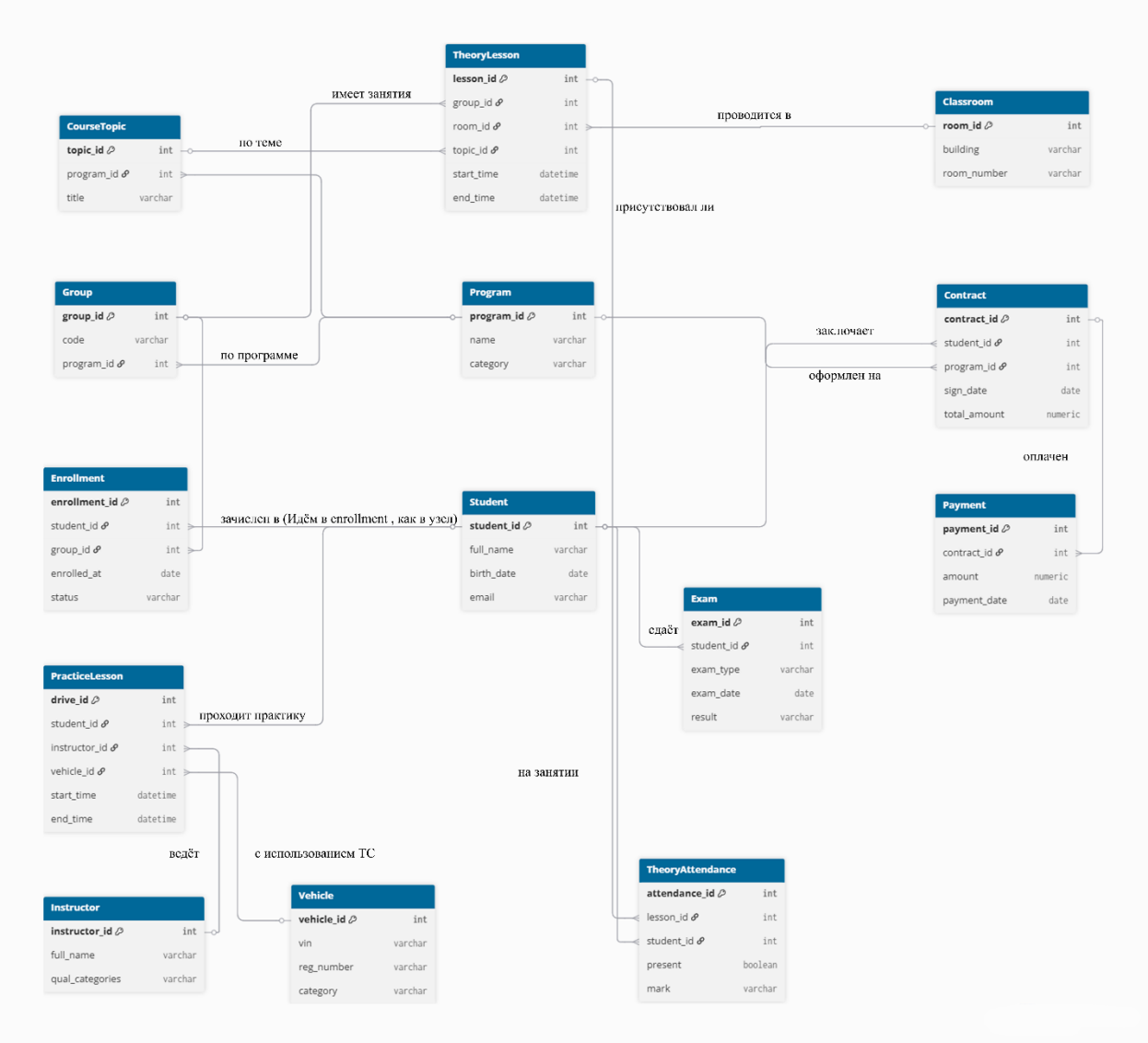


Рисунок 2 – Схема предварительных отношений системы

1.4. Приведение схемы предварительных отношений к 3НФ

Student:

- Атрибуты имеют автомарные поля (1НФ);
- Неключевые атрибуты зависят от соответствующего id (2НФ);
- Транзитивной зависимости не наблюдается (3НФ)
- (email - UNIQUE)

Program:

- Атрибуты имеют автомарные поля (1НФ);
- Неключевые атрибуты зависят от соответствующего id (2НФ);
- Транзитивной зависимости не наблюдается (3НФ)

Group:

- Атрибуты имеют автомарные поля (1НФ);
- Неключевые атрибуты зависят от соответствующего id (2НФ);
- Транзитивной зависимости не наблюдается (3НФ)
- (group - UNIQUE)

Classroom:

- Атрибуты имеют автомарные поля (1НФ);
- Неключевые атрибуты зависят от соответствующего id (2НФ);
- Транзитивной зависимости не наблюдается (3НФ)

CourseTopic:

- Атрибуты имеют автомарные поля (1НФ);
- Неключевые атрибуты зависят от соответствующего id (2НФ);
- Транзитивной зависимости не наблюдается (3НФ)
- (program_id - UNIQUE)

TheoryLesson:

- Атрибуты имеют автомарные поля (1НФ);
- Неключевые атрибуты зависят от соответствующего id (2НФ);
- Транзитивной зависимости не наблюдается (3НФ)

Instructor:

- Qual_categories – список категорий – нарушение. Решение: внести этот атрибут в InstructorCategory(instructor_id, category_code) (M:N) (1НФ);
- Неключевые атрибуты зависят от соответствующего id (2НФ);
- Транзитивной зависимости не наблюдается (3НФ)

Vehicle:

- Атрибуты имеют автомарные поля (1НФ);
- Неключевые атрибуты зависят от соответствующего id (2НФ);
- Транзитивной зависимости не наблюдается (3НФ)
- (vin, reg_number - UNIQUE)

PracticeLesson:

- Атрибуты имеют автомарные поля (1НФ);
- Неключевые атрибуты зависят от соответствующего id (2НФ);
- Транзитивной зависимости не наблюдается (3НФ)

Contract:

- Атрибуты имеют автомарные поля (1НФ);
- Неключевые атрибуты зависят от соответствующего id (2НФ);
- Транзитивной зависимости не наблюдается (3НФ)

Payment:

- Атрибуты имеют автомарные поля (1НФ);
- Неключевые атрибуты зависят от соответствующего id (2НФ);
- Транзитивной зависимости не наблюдается (3НФ)

Exam:

- Exam_type – справочник – нарушение. Решение: создать справочник ExamType (type_code PK, name). То же самое с exam_result, создать справочник ExamResult (result_code PK, name) (1НФ);
- Неключевые атрибуты зависят от соответствующего id (2НФ);
- Транзитивной зависимости не наблюдается (3НФ)

Enrollment:

- Атрибуты имеют автомарные поля (1НФ);
- Неключевые атрибуты зависят от соответствующего id (2НФ);
- Транзитивной зависимости не наблюдается (3НФ)

TheoryAttendance:

- Атрибуты имеют автомарные поля (1НФ);
- Неключевые атрибуты зависят от соответствующего id (2НФ);
- Транзитивной зависимости не наблюдается (3НФ)
- (lesson_id, student_id - UNIQUE)

1.5. Моделирование уровня представлений ИС университета.

Выделим два обязательных потребителя информации из нашей базы данных. Первый потребитель – это студенты. Второй потребитель – бухгалтерия.

Потребитель “Студенты”:

Представление 1. “Личное расписание”

Список атрибутов, отображаемых в рамках представления “Личное расписание”:

- ID студента (отношение “Student”)
- ФИО студента (отношение “Student”)
- Код группы (отношение “Group”)
- ID занятия (отношение “TheoryLesson”)
- Дата и время начала (отношение “TheoryLesson”)
- Дата и время окончания (отношение “TheoryLesson”)
- Тема занятия (отношение “CourseTopic”)
- Корпус (отношение “Classroom”)
- Аудитория (отношение “Classroom”)

Представление 2. “Посещаемость и оценки”

Список атрибутов, отображаемых в рамках представления “Посещаемость и оценки”:

- ID студента (отношение “Student”)
- ФИО студента (отношение “Student”)
- Тема занятия (отношение “CourseTopic”)
- Дата и время начала (отношение “TheoryLesson”)
- Присутствовал (отношение “TheoryAttendance”)
- Оценка/отметка (отношение “TheoryAttendance”)

Потребитель “Бухгалтерия”:

Представление 1. “Реестр договоров и оплат”

Список атрибутов, отображаемых в рамках представления “Реестр договоров и оплат”:

- ID договора (отношение “Contract”)
- Дата подписания (отношение “Contract”)
- Программа обучения (отношение “Program”)
- Категория программы (отношение “Program”)
- Сумма по договору (отношение “Contract”)
- Сумма оплачено (отношение “Payment”)
- Задолженность (отношение “Contract”)
- ID студента (отношение “Student”)
- ФИО студента (отношение “Student”)
- Последняя дата платежа (рассчитывается на основе “Contract” и “Payment”)

Представление 2. “Задолженности студентов”

Список атрибутов, отображаемых в рамках представления “Задолженности студентов”:

- ID студента (отношение “Student”)
- ФИО студента (отношение “Student”)
- Код группы (отношение “Group” через “Enrollment”)
- ID договора (отношение “Contract”)
- Дата подписания (отношение “Contract”)
- Сумма по договору (отношение “Contract”)
- Сумма оплачено (отношение “Payment”)
- Задолженность (рассчитывается на основе “Contract” и “Payment”)

2. РЕАЛИЗАЦИЯ БД В РАМКАХ СУБД

Создадим новую схему **autoschool**:

```
DROP SCHEMA IF EXISTS autoschool CASCADE;
CREATE SCHEMA autoschool;
SET search_path = autoschool, public;
```

После создания схемы необходимо создать **справочники**, которые будут содержать информацию о категории, методе оплаты и т.п.

```
CREATE TABLE Category (
category_code varchar PRIMARY KEY,
name          varchar NOT NULL
);

CREATE TABLE PaymentMethod (
method_code varchar PRIMARY KEY,
name        varchar NOT NULL
);

CREATE TABLE ExamType (
type_code varchar PRIMARY KEY,
name      varchar NOT NULL
);

CREATE TABLE ExamResult (
result_code varchar PRIMARY KEY,
name        varchar NOT NULL
);
```

Далее создадим сущности с индексами + 3 доп. таблицы (**Equipment**, **ClassroomEquipment (M: N)** , **InstructorCategory (M: N)**):

```
CREATE TABLE Student (
  student_id serial PRIMARY KEY,
  full_name  varchar NOT NULL,
  birth_date date NOT NULL,
  email      varchar UNIQUE
);

CREATE TABLE Program (
  program_id    serial PRIMARY KEY,
  name          varchar NOT NULL,
  category_code varchar NOT NULL REFERENCES Category(category_code)
);

CREATE TABLE "Group" (
  group_id   serial PRIMARY KEY,
  code       varchar UNIQUE NOT NULL,
  program_id int NOT NULL REFERENCES Program(program_id)
);

CREATE TABLE Classroom (
  room_id     serial PRIMARY KEY,
  building    varchar NOT NULL,
  room_number varchar NOT NULL
);
```

```
CREATE TABLE Equipment (
  equipment_id serial PRIMARY KEY,
  name         varchar NOT NULL
);

CREATE TABLE ClassroomEquipment (
  room_id      int NOT NULL REFERENCES Classroom(room_id) ON DELETE CASCADE,
  equipment_id int NOT NULL REFERENCES Equipment(equipment_id) ON DELETE CASCADE,
  PRIMARY KEY (room_id, equipment_id)
);

CREATE TABLE CourseTopic (
  topic_id   serial PRIMARY KEY,
  program_id int NOT NULL REFERENCES Program(program_id) ON DELETE CASCADE,
  title      varchar NOT NULL,
  order_no   int NOT NULL,
  CONSTRAINT uq_topic_order UNIQUE (program_id, order_no)
);

CREATE TABLE TheoryLesson (
  lesson_id  serial PRIMARY KEY,
  group_id   int NOT NULL REFERENCES "Group"(group_id) ON DELETE CASCADE,
  room_id    int NOT NULL REFERENCES Classroom(room_id),
  topic_id   int NOT NULL REFERENCES CourseTopic(topic_id),
  start_time timestamp NOT NULL,
  end_time   timestamp NOT NULL,
```

```
  status     varchar NOT NULL CHECK (status IN ('scheduled','done','cancelled')),
  CHECK (end_time > start_time)
);

CREATE TABLE Instructor (
  instructor_id serial PRIMARY KEY,
  full_name     varchar NOT NULL,
  email         varchar,
  phone         varchar,
  hire_date     date,
  status                varchar NOT NULL DEFAULT 'active' CHECK (status IN
('active','inactive','fired'))
);

CREATE TABLE InstructorCategory (
  instructor_id int NOT NULL REFERENCES Instructor(instructor_id) ON DELETE
CASCADE,
  category_code varchar NOT NULL REFERENCES Category(category_code),
  PRIMARY KEY (instructor_id, category_code)
);
CREATE TABLE Vehicle (
  vehicle_id    serial PRIMARY KEY,
  vin           varchar UNIQUE NOT NULL,
  reg_number    varchar UNIQUE NOT NULL,
  category_code varchar NOT NULL REFERENCES Category(category_code),
  make          varchar NOT NULL,
  model         varchar NOT NULL,
  status                varchar NOT NULL DEFAULT 'available' CHECK (status IN
('available','maintenance','unavailable'))
);
```

```
CREATE TABLE PracticeLesson (
  drive_id      serial PRIMARY KEY,
  student_id        int NOT NULL REFERENCES Student(student_id) ON DELETE
CASCADE,
  instructor_id int NOT NULL REFERENCES Instructor(instructor_id),
  vehicle_id    int NOT NULL REFERENCES Vehicle(vehicle_id),
  start_time    timestamp NOT NULL,
  end_time      timestamp NOT NULL,
  route         varchar,
  status                varchar NOT NULL DEFAULT 'scheduled' CHECK (status IN
('scheduled','done','cancelled')),
  CHECK (end_time > start_time)
);

CREATE TABLE Contract (
  contract_id  serial PRIMARY KEY,
  student_id   int NOT NULL REFERENCES Student(student_id),
  program_id   int NOT NULL REFERENCES Program(program_id),
  sign_date    date NOT NULL,
  agreed_price numeric(12,2) NOT NULL CHECK (agreed_price >= 0),
  status                varchar NOT NULL DEFAULT 'active' CHECK (status IN
('active','closed','cancelled'))
);
```

```
CREATE TABLE Payment (
  payment_id   serial PRIMARY KEY,
  contract_id    int NOT NULL REFERENCES Contract(contract_id) ON DELETE CASCADE,
  amount       numeric(12,2) NOT NULL CHECK (amount > 0),
  payment_date date NOT NULL,
  method_code  varchar REFERENCES PaymentMethod(method_code),
  status               varchar NOT NULL DEFAULT 'accepted' CHECK (status IN ('accepted','reverted','pending'))
);

CREATE TABLE Exam (
  exam_id     serial PRIMARY KEY,
  student_id    int NOT NULL REFERENCES Student(student_id) ON DELETE CASCADE,
  type_code   varchar NOT NULL REFERENCES ExamType(type_code),
  exam_date   date NOT NULL,
  result_code varchar NOT NULL REFERENCES ExamResult(result_code),
  inspector   varchar
);
CREATE TABLE Enrollment (
  enrollment_id serial PRIMARY KEY,
  student_id       int NOT NULL REFERENCES Student(student_id) ON DELETE CASCADE,
  group_id          int NOT NULL REFERENCES "Group"(group_id) ON DELETE CASCADE,
  enrolled_at   date NOT NULL,
  status                          varchar NOT NULL CHECK (status IN ('active','transferred','expelled','finished'))
);
```

```
CREATE TABLE TheoryAttendance (
  attendance_id serial PRIMARY KEY,
  lesson_id int NOT NULL REFERENCES TheoryLesson(lesson_id) ON DELETE CASCADE,
  student_id int NOT NULL REFERENCES Student(student_id) ON DELETE CASCADE,
  present boolean NOT NULL,
  mark varchar,
  CONSTRAINT uq_attendance UNIQUE (lesson_id, student_id)
);
```

Создадим ещё **ALTER TABLE** для автоматической проверки чтобы один и тот же инструктор не мог вести две практики в одно и то же время, одно и то же транспортное средство не могло использоваться одновременно в двух практиках.

```
ALTER TABLE PracticeLesson
  ADD COLUMN ts_range tsrange GENERATED ALWAYS AS (tsrange(start_time, end_time, '[)')) STORED;

CREATE INDEX idx_practice_tsrange ON PracticeLesson USING gist (ts_range);
ALTER TABLE PracticeLesson
  ADD CONSTRAINT ex_practice_instructor_overlap
  EXCLUDE USING gist (instructor_id WITH =, ts_range WITH &&);

ALTER TABLE PracticeLesson
  ADD CONSTRAINT ex_practice_vehicle_overlap
  EXCLUDE USING gist (vehicle_id WITH =, ts_range WITH &&);
```

Сделаем индексы на **FK**, так как в PostgreSQL**, PK** и **UNIQUE** индексируются автоматически, а остальные поля нет.

```
CREATE INDEX idx_group_program       ON "Group"(program_id);
CREATE INDEX idx_ct_program          ON CourseTopic(program_id);
CREATE INDEX idx_tl_group            ON TheoryLesson(group_id);
CREATE INDEX idx_tl_room             ON TheoryLesson(room_id);
CREATE INDEX idx_tl_topic            ON TheoryLesson(topic_id);
CREATE INDEX idx_ic_instructor       ON InstructorCategory(instructor_id);
CREATE INDEX idx_ic_category         ON InstructorCategory(category_code);
CREATE INDEX idx_vehicle_category    ON Vehicle(category_code);
CREATE INDEX idx_practice_student    ON PracticeLesson(student_id);
CREATE INDEX idx_practice_instructor ON PracticeLesson(instructor_id);
CREATE INDEX idx_practice_vehicle    ON PracticeLesson(vehicle_id);
CREATE INDEX idx_contract_student    ON Contract(student_id);
CREATE INDEX idx_contract_program    ON Contract(program_id);
CREATE INDEX idx_payment_contract    ON Payment(contract_id);
CREATE INDEX idx_exam_student        ON Exam(student_id);
CREATE INDEX idx_exam_type           ON Exam(type_code);
CREATE INDEX idx_exam_result         ON Exam(result_code);
CREATE INDEX idx_enroll_student      ON Enrollment(student_id);
CREATE INDEX idx_enroll_group        ON Enrollment(group_id);
CREATE INDEX idx_attend_lesson       ON TheoryAttendance(lesson_id);
CREATE INDEX idx_student_email       ON Student(email);
CREATE INDEX idx_group_code          ON "Group"(code);
CREATE INDEX idx_vehicle_reg         ON Vehicle(reg_number);
CREATE INDEX idx_vehicle_vin         ON Vehicle(vin);
```

Для того чтобы контролировать максимальное количество студентов, допустимое на занятии, был добавлен столбец вместимости (**capacity**) в таблицу **Classroom**. А в таблицу **Vehicle** – **year**, для хранения года выпуска транспорта. Таким образом можно модифицировать структуру без потери данных.

```
ALTER TABLE Classroom ADD COLUMN capacity int;
UPDATE Classroom SET capacity = 20;
ALTER TABLE Vehicle ADD COLUMN year int;
```

Далее, заполним сущности и справочники данными:

```
INSERT INTO Category(category_code, name) VALUES
 ('A','Мото'),
 ('B','Легковые'),
 ('C','Грузовые');

INSERT INTO PaymentMethod(method_code, name) VALUES
 ('cash','Наличные'),
 ('card','Банковская карта'),
 ('transfer','Безналичный перевод');

INSERT INTO ExamType(type_code, name) VALUES
 ('theory','Теория'),
 ('driving','Вождение');

INSERT INTO ExamResult(result_code, name) VALUES
 ('pass','Сдал'),
 ('fail','Не сдал');
```

```
INSERT INTO Student(full_name,birth_date,email) VALUES
 ('Иванов Иван Иванович','2003-01-12','ivanov1@example.com'),
 ('Петров Пётр Петрович','2002-05-23','petrov2@example.com'),
 ('Сидорова Анна Сергеевна','2004-03-02','sidorova3@example.com'),
 ('Михайлов Дмитрий Олегович','2001-11-15','mikhailov4@example.com'),
 ('Кузнецова Елена Викторовна','2003-07-07','kuznecova5@example.com'),
 ('Орлова Мария Андреевна','2005-09-19','orlova6@example.com'),
 ('Соколов Артём Денисович','2002-12-30','sokolov7@example.com'),
 ('Фёдоров Никита Ильич','2003-04-18','fedorov8@example.com');

INSERT INTO Program(name, category_code) VALUES
 ('Базовый курс B','B'),
 ('Интенсив B','B'),
 ('Курс A','A'),
 ('Курс C','C');

INSERT INTO "Group"(code, program_id) VALUES
 ('B-101', 1),
 ('B-102', 1),
 ('B-201', 2),
 ('A-101', 3),
 ('C-101', 4),
 ('B-301', 2),
 ('B-401', 2),
 ('A-201', 3);
```

```
INSERT INTO Classroom(building, room_number, capacity) VALUES
 ('A','101',24),
 ('A','102',20),
 ('B','201',18),
 ('B','202',30),
 ('C','301',16),
 ('C','302',22),
 ('C','303',20),
 ('D','401',25);

INSERT INTO Equipment(name) VALUES
 ('Проектор'),
 ('Доска'),
 ('Компьютеры'),
 ('Звуковая система'),
 ('Кондиционер');

INSERT INTO ClassroomEquipment(room_id,equipment_id) VALUES
 (1,1),(1,2),(1,5),
 (2,1),(2,2),
 (3,2),(3,5),
 (4,1),(4,2),(4,4),
 (5,2),
 (6,1),(6,2),(6,3),
 (7,2),
 (8,1),(8,4);
```

```
INSERT INTO CourseTopic(program_id, title, order_no) VALUES
(1,'ПДД: основы',1),(1,'Знаки и разметка',2),(1,'Перекрёстки',3),(1,'Безопасность',4),
(2,'ПДД: основы+',1),(2,'Манёвры',2),(2,'Парковка',3),(2,'Нестандартные ситуации',4),
(3,'Мото: теория',1),(3,'Снаряжение',2),(3,'Манёвры',3),(3,'Безопасность',4),
(4,'Грузовые: ПДД',1),(4,'Габариты и груз',2),(4,'Манёвры',3),(4,'Техника безопасности',4);

INSERT INTO Instructor(full_name,email,phone,hire_date,status) VALUES
('Алексеев Роман Сергеевич','alexeev@example.com','+7-900-111-11-11','2020-02-01','active'),
('Борисова Ирина Павловна','borisova@example.com','+7-900-222-22-22','2019-06-15','active'),
('Воробьёв Павел Николаевич','vorobev@example.com','+7-900-333-33-33','2021-09-10','active'),
('Громов Олег Петрович','gromov@example.com','+7-900-444-44-44','2018-04-20','active'),
('Демидова Татьяна Игоревна','demidova@example.com','+7-900-555-55-55','2022-01-12','active'),
('Егоров Максим Андреевич','egorov@example.com','+7-900-666-66-66','2017-12-05','inactive'),
('Жуков Степан Романович','zhukov@example.com','+7-900-777-77-77','2023-03-08','active'),
('Зайцева Алёна Сергеевна','zaitseva@example.com','+7-900-888-88-88','2020-10-30','active');
```

```
INSERT INTO InstructorCategory(instructor_id, category_code) VALUES
 (1,'B'),(2,'B'),(3,'B'),(4,'C'),(5,'A'),(6,'B'),(7,'B'),(8,'A');

INSERT INTO Vehicle(vin,reg_number,category_code,make,model,status,year)
VALUES
 ('VIN00000000000001','A001AA178','B','Kia','Rio','available',2018),
 ('VIN00000000000002','A002AA178','B','Hyundai','Solaris','available',2019),
 ('VIN00000000000003','A003AA178','B','Volkswagen','Polo','maintenance',2017),
 ('VIN00000000000004','A004AA178','A','Yamaha','YZF-R3','available',2020),
 ('VIN00000000000005','A005AA178','C','MAN','TGL','available',2016),
 ('VIN00000000000006','A006AA178','B','Skoda','Rapid','available',2021),
 ('VIN00000000000007','A007AA178','B','Renault','Logan','available',2015),
 ('VIN00000000000008','A008AA178','A','Honda','CB500F','unavailable',2014);

INSERT INTO Contract(student_id, program_id, sign_date, agreed_price, status)
VALUES
 (1,1,'2025-01-10',45000,'active'),
 (2,1,'2025-01-12',45000,'active'),
 (3,2,'2025-01-15',52000,'active'),
 (4,2,'2025-01-18',52000,'active'),
 (5,3,'2025-01-20',38000,'active'),
 (6,3,'2025-01-25',38000,'active'),
 (7,4,'2025-01-28',60000,'active'),
 (8,1,'2025-02-02',45000,'active');
```

```
INSERT INTO Payment(contract_id,amount,payment_date,method_code,status)
VALUES
 (1,15000,'2025-01-12','card','accepted'),
 (1,15000,'2025-02-01','card','accepted'),
 (1,15000,'2025-02-20','cash','accepted'),
 (2,20000,'2025-01-15','transfer','accepted'),
 (2,25000,'2025-02-10','card','accepted'),
 (3,26000,'2025-01-20','card','accepted'),
 (3,26000,'2025-02-20','card','accepted'),
 (4,52000,'2025-01-22','transfer','accepted'),
 (5,38000,'2025-01-22','card','accepted'),
 (6,18000,'2025-01-27','card','accepted'),
 (6,20000,'2025-02-14','card','accepted'),
 (6,  0.01,'2025-02-16','card','pending'),
 (7,30000,'2025-02-01','transfer','accepted'),
 (7,30000,'2025-02-25','transfer','accepted'),
 (8,45000,'2025-02-05','cash','accepted');

INSERT INTO Enrollment(student_id,group_id,enrolled_at,status) VALUES
 (1,1,'2025-01-11','active'),
 (2,1,'2025-01-13','active'),
 (3,3,'2025-01-16','active'),
 (4,3,'2025-01-19','active'),
 (5,4,'2025-01-22','active'),
 (6,8,'2025-01-28','active'),
 (7,5,'2025-01-29','active'),
 (8,2,'2025-02-04','active');
```

```
INSERT INTO TheoryLesson(group_id,room_id,topic_id,start_time,end_time,status)
VALUES
 (1,1,1,'2025-02-03 10:00','2025-02-03 11:30','done'),
 (1,2,2,'2025-02-05 10:00','2025-02-05 11:30','done'),
 (1,1,3,'2025-02-07 10:00','2025-02-07 11:30','scheduled'),
 (2,3,1,'2025-02-03 12:00','2025-02-03 13:30','done'),
 (3,4,5,'2025-02-04 10:00','2025-02-04 11:30','done'),
 (3,6,6,'2025-02-06 10:00','2025-02-06 11:30','scheduled'),
 (4,5,9,'2025-02-05 14:00','2025-02-05 15:30','done'),
 (5,4,13,'2025-02-06 15:00','2025-02-06 16:30','scheduled');

INSERT INTO TheoryAttendance(lesson_id,student_id,present,mark) VALUES
 (1,1,true,'5'),
 (1,2,true,'4'),
 (2,1,true,'5'),
 (2,2,false,NULL),
 (4,8,true,'5'),
 (5,3,true,'4'),
 (5,4,true,'5'),
 (7,5,true,'5'),
 (7,6,true,'4');
```

```
INSERT INTO PracticeLesson(student_id,instructor_id,vehicle_id,start_time,end_time,route,status)
VALUES
 (1,1,1,'2025-02-10 10:00','2025-02-10 11:00','Маршрут-1','done'),
 (2,1,2,'2025-02-10 11:15','2025-02-10 12:15','Маршрут-2','done'),
 (3,3,6,'2025-02-11 10:00','2025-02-11 11:00','Маршрут-3','scheduled'),
 (4,3,6,'2025-02-11 11:15','2025-02-11 12:15','Маршрут-4','scheduled'),
 (5,5,4,'2025-02-12 09:00','2025-02-12 10:00','Мото-1','scheduled'),
 (6,8,4,'2025-02-12 10:15','2025-02-12 11:00','Мото-2','scheduled'),
 (7,4,5,'2025-02-13 14:00','2025-02-13 15:30','Груз-1','scheduled'),
 (8,2,7,'2025-02-14 16:00','2025-02-14 17:00','Маршрут-5','scheduled');

INSERT INTO Exam(student_id,type_code,exam_date,result_code,inspector)
VALUES
 (1,'theory','2025-02-15','pass','Инспектор А'),
 (2,'theory','2025-02-15','fail','Инспектор А'),
 (2,'theory','2025-02-22','pass','Инспектор А'),
 (3,'theory','2025-02-16','pass','Инспектор Б'),
 (1,'driving','2025-02-20','pass','Инспектор В'),
 (3,'driving','2025-02-24','fail','Инспектор В'),
 (4,'theory','2025-02-18','pass','Инспектор Б'),
 (5,'theory','2025-02-18','pass','Инспектор Б');
```

Реализуем **представления** (**View**), составленные в ЛР№1:

Потребитель “Студенты”:

Представление 1. “Личное расписание”

- Список атрибутов, отображаемых в рамках представления “Личное расписание”:
- ID студента (отношение “Student”)
- ФИО студента (отношение “Student”)
- Код группы (отношение “Group”)
- ID занятия (отношение “TheoryLesson”)
- Дата и время начала (отношение “TheoryLesson”)
- Дата и время окончания (отношение “TheoryLesson”)
- Тема занятия (отношение “CourseTopic”)
- Корпус (отношение “Classroom”)
- Аудитория (отношение “Classroom”)

```
CREATE OR REPLACE VIEW v_student_timetable AS
SELECT
  s.student_id,
  s.full_name,
  g.code        AS group_code,
  tl.lesson_id,
  tl.start_time,
  tl.end_time,
  ct.title      AS topic_title,
  c.building,
  c.room_number
FROM Student s
JOIN Enrollment e    ON e.student_id = s.student_id AND e.status IN ('active','finished')
JOIN "Group" g    ON g.group_id = e.group_id
```

```
JOIN TheoryLesson tl ON tl.group_id = g.group_id
JOIN CourseTopic ct ON ct.topic_id = tl.topic_id
JOIN Classroom c   ON c.room_id = tl.room_id;
```

Представление 2. “Посещаемость и оценки”

Список атрибутов, отображаемых в рамках представления “Посещаемость и оценки”:

- ID студента (отношение “Student”)
- ФИО студента (отношение “Student”)
- Тема занятия (отношение “CourseTopic”)
- Дата и время начала (отношение “TheoryLesson”)
- Присутствовал (отношение “TheoryAttendance”)
- Оценка/отметка (отношение “TheoryAttendance”)

```
CREATE OR REPLACE VIEW v_student_attendance AS
SELECT
  s.student_id,
  s.full_name,
  ct.title       AS topic_title,
  tl.start_time,
  ta.present,
  ta.mark
FROM TheoryAttendance ta
JOIN Student s     ON s.student_id = ta.student_id
JOIN TheoryLesson tl ON tl.lesson_id = ta.lesson_id
JOIN CourseTopic ct ON ct.topic_id = tl.topic_id;
```

Потребитель “Бухгалтерия”:

Представление 1. “Реестр договоров и оплат”

Список атрибутов, отображаемых в рамках представления “Реестр договоров и оплат”:

- ID договора (отношение “Contract”)
- Дата подписания (отношение “Contract”)
- Программа обучения (отношение “Program”)
- Категория программы (отношение “Program”)
- Сумма по договору (отношение “Contract”)
- Сумма оплачено (отношение “Payment”)
- Задолженность (отношение “Contract”)
- ID студента (отношение “Student”)
- ФИО студента (отношение “Student”)
- Последняя дата платежа (рассчитывается на основе “Contract” и “Payment”)

```
CREATE OR REPLACE VIEW v_contracts_payments AS
WITH pay AS (
  SELECT contract_id,
      SUM(CASE WHEN status='accepted' THEN amount ELSE 0 END) AS paid_sum,
      MAX(payment_date) FILTER (WHERE status='accepted') AS last_payment_date
  FROM Payment
  GROUP BY contract_id
)
```

```
SELECT
  c.contract_id,
  c.sign_date,
  p.name AS program_name,
  p.category_code,
  c.agreed_price,
  COALESCE(pay.paid_sum,0)  AS paid_sum,
  (c.agreed_price - COALESCE(pay.paid_sum,0))  AS debt,
  s.student_id,
  s.full_name,
  pay.last_payment_date
FROM Contract c
JOIN Program p ON p.program_id = c.program_id
JOIN Student s ON s.student_id = c.student_id
LEFT  JOIN pay ON pay.contract_id = c.contract_id;
```

Представление 2. “Задолженности студентов”

Список атрибутов, отображаемых в рамках представления “Задолженности студентов”:

- ID студента (отношение “Student”)
- ФИО студента (отношение “Student”)
- Код группы (отношение “Group” через “Enrollment”)
- ID договора (отношение “Contract”)
- Дата подписания (отношение “Contract”)
- Сумма по договору (отношение “Contract”)
- Сумма оплачено (отношение “Payment”)
- Задолженность (рассчитывается на основе “Contract” и “Payment”)

```
CREATE OR REPLACE VIEW v_student_debts AS
WITH pay AS (
  SELECT contract_id,
       SUM(CASE WHEN status='accepted' THEN amount ELSE 0 END) AS paid_sum
  FROM Payment
  GROUP BY contract_id
)
```

```
SELECT
  s.student_id,
  s.full_name,
  g.code AS group_code,
  c.contract_id,
  c.sign_date,
  c.agreed_price,
  COALESCE(pay.paid_sum,0)  AS paid_sum,
  (c.agreed_price - COALESCE(pay.paid_sum,0))  AS debt
FROM Contract c
JOIN Student s ON s.student_id = c.student_id
LEFT  JOIN Enrollment e ON e.student_id = s.student_id AND e.status='active'
LEFT  JOIN "Group" g ON g.group_id = e.group_id
LEFT  JOIN pay ON pay.contract_id = c.contract_id
WHERE  (c.agreed_price - COALESCE(pay.paid_sum,0))  > 0;
```

Тестовый запрос (SELECT * FROM v_student_timetable ORDER BY student_id, start_time;):

	student_id integer	full_name character varying	group_code character varying	lesson_id integer	start_time timestamp without time zone	end_time timestamp without time zone	topic_title character varying	building character varying	room_number character varying
1	1	Иванов Иван Иванович	B-101	1	2025-02-03 10:00:00	2025-02-03 11:30:00	ПДД: основы	A	101
2	1	Иванов Иван Иванович	B-101	2	2025-02-05 10:00:00	2025-02-05 11:30:00	Знаки и разметка	A	102
3	1	Иванов Иван Иванович	B-101	3	2025-02-07 10:00:00	2025-02-07 11:30:00	Перекрёстки	A	101
4	2	Петров Пётр Петрович	B-101	1	2025-02-03 10:00:00	2025-02-03 11:30:00	ПДД: основы	A	101
5	2	Петров Пётр Петрович	B-101	2	2025-02-05 10:00:00	2025-02-05 11:30:00	Знаки и разметка	A	102
6	2	Петров Пётр Петрович	B-101	3	2025-02-07 10:00:00	2025-02-07 11:30:00	Перекрёстки	A	101
7	3	Сидорова Анна Сергеевна	B-201	5	2025-02-04 10:00:00	2025-02-04 11:30:00	ПДД: основы+	B	202
8	3	Сидорова Анна Сергеевна	B-201	6	2025-02-06 10:00:00	2025-02-06 11:30:00	Манёвры	C	302
9	4	Михайлов Дмитрий Олегов…	B-201	5	2025-02-04 10:00:00	2025-02-04 11:30:00	ПДД: основы+	B	202
10	4	Михайлов Дмитрий Олегов…	B-201	6	2025-02-06 10:00:00	2025-02-06 11:30:00	Манёвры	C	302
11	5	Кузнецова Елена Викторовна	A-101	7	2025-02-05 14:00:00	2025-02-05 15:30:00	Мото: теория	C	301
12	7	Соколов Артём Денисович	C-101	8	2025-02-06 15:00:00	2025-02-06 16:30:00	Грузовые: ПДД	B	202
13	8	Фёдоров Никита Ильич	B-102	4	2025-02-03 12:00:00	2025-02-03 13:30:00	ПДД: основы	B	201

Рис.3 – Тестовый запрос.

3. ЗАЩИТА БАЗЫ ДАННЫХ

Первым делом создадим лог-таблицу для аудита действий, совершенных пользователем:

```
CREATE TABLE autoschool.main_log (
   log_id      BIGSERIAL PRIMARY KEY,
   table_name  TEXT,
   operation   TEXT,
   changed_at  TIMESTAMPTZ DEFAULT now(),
   db_user     TEXT,
   old_data    JSONB,
   new_data    JSONB
);
```

Для того, чтобы автоматизировать нашу бд, необходимы триггеры. При совершении того или иного действия, в нашем случае **“INSERT”, UPDATE“, “DELETE”,** будет произведено логирование:

```
CREATE OR REPLACE FUNCTION autoschool.logging()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
   IF TG_OP = 'INSERT' THEN
      INSERT INTO autoschool.main_log(table_name, operation, db_user, new_data)
      VALUES (TG_TABLE_NAME, 'I', current_user, to_jsonb(NEW));
      RETURN NEW;
   ELSIF TG_OP = 'UPDATE' THEN
      INSERT INTO autoschool.main_log(table_name, operation, db_user, old_data,
new_data)
      VALUES (TG_TABLE_NAME, 'U', current_user, to_jsonb(OLD),
to_jsonb(NEW));
      RETURN NEW;
   ELSIF TG_OP = 'DELETE' THEN
      INSERT INTO autoschool.main_log(table_name, operation, db_user, old_data)
      VALUES (TG_TABLE_NAME, 'D', current_user, to_jsonb(OLD));
      RETURN OLD;
   END IF;
   RETURN NULL;
END;
$$;
```

Теперь подключим нашу функцию к табличке и проверим работоспособность:

```
CREATE TRIGGER student_logging_trg
AFTER INSERT OR UPDATE OR DELETE
ON autoschool.student
FOR EACH ROW
EXECUTE FUNCTION autoschool.logging();
```

Протестируем триггер, добавим тестовые данные в таблицу:

```
INSERT INTO autoschool.student (full_name, birth_date, email)
VALUES ('Тест', '2000-01-01','test@testt.ru');
```

	log_id [PK] bigint	table_name text	operation text	changed_at timestamp with time zone	db_user text	old_data jsonb	new_data jsonb
1	1	student	I	2025-10-31 19:30:09.644573+03	postgres	[null]	{"email": "test@testt.ru", "full_name": "Тест", "birth_date": "2000-01-01", "student_id": 9}

Рис.4 – Таблица с логами.

Аудит работает успешно. Однако необходимо ещё повесить триггеры на следующие таблицы:

```
CREATE TRIGGER contract_logging_trg
AFTER INSERT OR UPDATE OR DELETE
ON autoschool.contract
FOR EACH ROW
EXECUTE FUNCTION autoschool.logging();
CREATE TRIGGER payment_logging_trg
AFTER INSERT OR UPDATE OR DELETE
ON autoschool.payment
FOR EACH ROW
EXECUTE FUNCTION autoschool.logging();
```

```
INSERT INTO autoschool.payment (contract_id, payment_date, amount)
VALUES (1, now(), 5000);
```

Data Output | Messages | Notifications

	log_id [PK] bigint	table_name text	operation text	changed_at timestamp with time zone	db_user text
1	2	payment	I	2025-10-31 19:34:51.060831+03	postgres
2	1	student	I	2025-10-31 19:30:09.644573+03	postgres

Рис.5 – Таблица с логами.

Второе второй этап – **шифрование**.

Подключим расширение **pgcrypto**:

```
CREATE EXTENSION IF NOT EXISTS pgcrypto;
```

Создадим таблицу с “секретными” данными:

```
CREATE TABLE autoschool.secret_data (
  id          BIGSERIAL PRIMARY KEY,
  username    TEXT,
  secret_token  TEXT
);

INSERT INTO autoschool.secret_data (username, secret_token)
VALUES (
  pgp_sym_encrypt('secret_admin_autoschool', digest('stroNgpsw11111', 'sha256')::text),
  pgp_sym_encrypt('secret_token_for_admin', digest('stroNgpsw11111', 'sha256')::text)
);
```

	id [PK] bigint	username text	secret_token text
1	2	\xc30d040703021126c72dda66a1776...	\xc30d040703023888205e0a70950465d24

Рис.6 – Таблица с зашифрованными данными.

В результате получаем таблицу с зашифрованным сообщением при помощи sha256. Однако нам ещё необходимо просматривать данные. Напишем запрос для расшифровки, содержащий пароль, указанный выше.

```
SELECT
  id,
  pgp_sym_decrypt(username::bytea, digest('stroNgpsw11111', 'sha256')::text) AS username_decrypted,
  pgp_sym_decrypt(secret_token::bytea, digest('stroNgpsw11111', 'sha256')::text) AS token_decrypted
FROM autoschool.secret_data;
```

	id [PK] bigint	username_decrypted text	token_decrypted text
1	2	secret_admin_autoschool	secret_token_for_admin

Рис.7 – Расшифрованные данные.

Продемонстрируем вариант с неверным паролем:

```
1  SELECT
2      id,
3      pgp_sym_decrypt(username::bytea, digest('asassaasas', 'sha256')::text) AS username_decrypted
4  FROM autoschool.secret_data;
```

Data Output | Messages | Notifications

```
ERROR:  Wrong key or corrupt data

ОШИБКА:  Wrong key or corrupt data
SQL state: 39000
```

Рис.8 – Вывод при неверном пароле.

Третий и последний этап – **разграничение доступа**. Создадим разграничения под наши представления:

```
CREATE  ROLE student_group_role
   NOLOGIN
   NOSUPERUSER
   NOCREATEDB
   NOCREATEROLE
   INHERIT;

CREATE  ROLE accountant_group_role
   NOLOGIN
   NOSUPERUSER
   NOCREATEDB
   NOCREATEROLE
   INHERIT;

CREATE  ROLE admin_group_role
   NOLOGIN
   NOSUPERUSER
   NOCREATEDB
   NOCREATEROLE
   INHERIT;
```

После создания ролей, привяжем их к представлениям:

- студентам можно смотреть только студенческие представления;
- бухгалтерии можно смотреть только финансовые представления;
- админу можно смотреть всё;

```
GRANT SELECT ON TABLE
    autoschool.v_student_timetable,
    autoschool.v_student_attendance
TO student_group_role;

GRANT SELECT ON TABLE
    autoschool.v_contracts_payments,
    autoschool.v_student_debts
TO accountant_group_role;

GRANT SELECT ON ALL TABLES IN SCHEMA autoschool TO admin_group_role;
GRANT SELECT ON
    autoschool.v_student_timetable,
    autoschool.v_student_attendance,
    autoschool.v_contracts_payments,
    autoschool.v_student_debts
TO admin_group_role;
```

Создадим **пользователей**:

```
CREATE ROLE evg_student WITH
    LOGIN
    PASSWORD 'studpass';

GRANT student_group_role TO evg_student;
```

```
CREATE  ROLE evg_accountant WITH
   LOGIN
   PASSWORD 'accpass';

GRANT  accountant_group_role TO evg_accountant;
CREATE  ROLE evg_admin WITH
   LOGIN
   PASSWORD 'adminpass';
GRANT  admin_group_role TO evg_admin;
```

Проверим на работоспособность:

Студент:

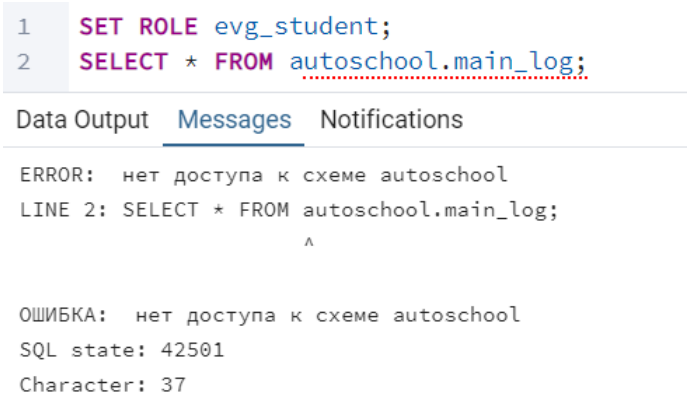
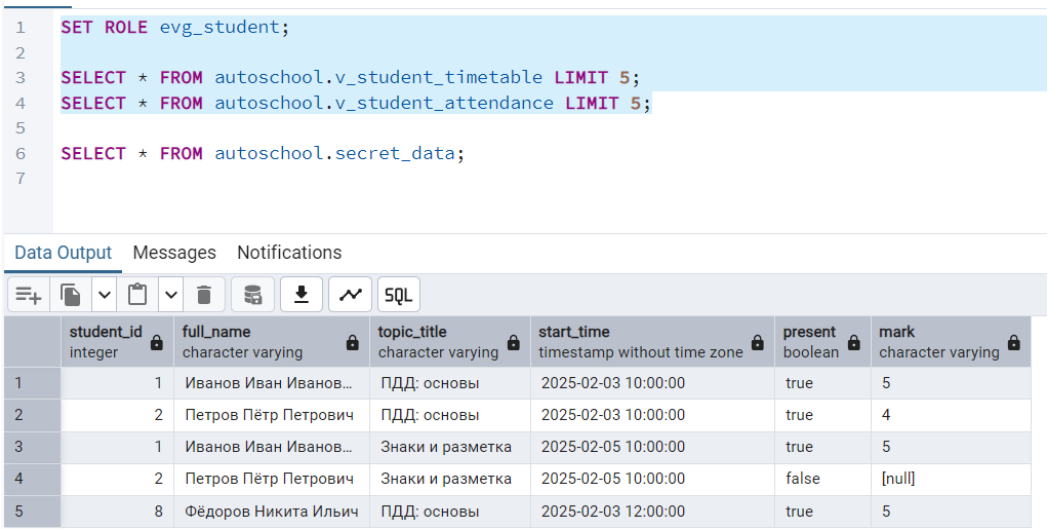


Рис.9 – Тестирование разграничения прав для студента.



	student_id integer	full_name character varying	topic_title character varying	start_time timestamp without time zone	present boolean	mark character varying
1	1	Иванов Иван Иванов...	ПДД: основы	2025-02-03 10:00:00	true	5
2	2	Петров Пётр Петрович	ПДД: основы	2025-02-03 10:00:00	true	4
3	1	Иванов Иван Иванов...	Знаки и разметка	2025-02-05 10:00:00	true	5
4	2	Петров Пётр Петрович	Знаки и разметка	2025-02-05 10:00:00	false	[null]
5	8	Фёдоров Никита Ильич	ПДД: основы	2025-02-03 12:00:00	true	5

Рис.10 – Тестирование разграничения прав для студента.

```
1  SET ROLE evg_student;
2
3  SELECT * FROM autoschool.secret_data;
```

Data Output | Messages | Notifications

```
ERROR:  нет доступа к таблице secret_data

ОШИБКА:  нет доступа к таблице secret_data
SQL state: 42501
```

Рис.11 – Тестирование разграничения прав для студента.

Администратор:

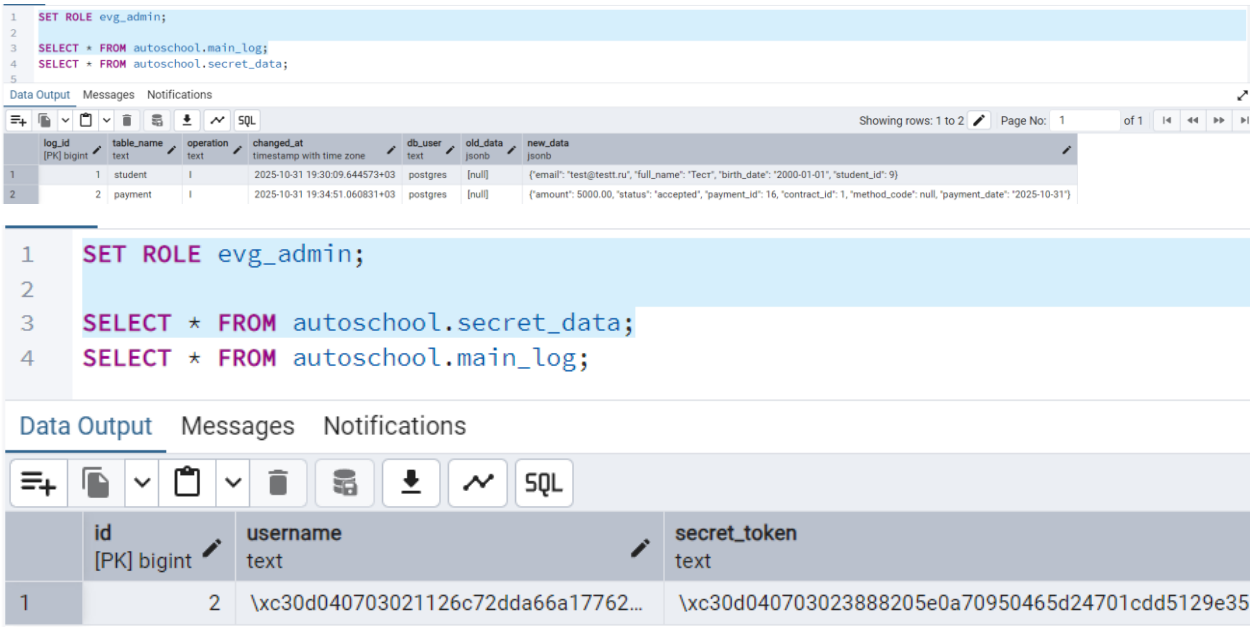


Рис.12 – Тестирование разграничения прав для администратора.

Бухгалтер:

Рис.13 – Тестирование разграничения прав для бухгалтера.

4. РЕАЛИЗАЦИЯ УРОВНЯ ПРИЛОЖЕНИЯ С ВОЗМОЖНОСТЬЮ МНОГОПОЛЬЗОВАТЕЛЬСКОГО ДОСТУПА К СОЗДАННОЙ БД

В качестве языка программирования был выбран **Python**, а в качестве среды разработки - **VSCode**. Взаимодействие с СУБД будет осуществляться при помощи библиотеки **SQLAlchemy** и **psycopg**. Веб-интерфейс реализован с помощью фреймворка Flask. Данный стек технологий позволяет описывать таблицы в БД в виде ORM, питон-классов. Помимо этого, он легко интегрируется с системой разграничения доступа, реализованной в ЛР 3, за счёт подключения к БД от разных ролей.

Для начала установим зависимости:

```
PS E:\УЧЁБА\3 курс\ОСЕНЬ\ИББД\ЛР4\CODE> pip install flask
>> pip install sqlalchemy
>> pip install psycopg2-binary
```

Рисунок 14 – Установка зависимостей.

Далее, наметим структуру проекта. Выделим основные модули:

- **config.py** – параметры подключения к БД и настройкам приложения;
- **models.py** – описание ORM- моделей для таблиц и БД;
- **app.py** – основной модуль, реализующий логику интерфейса;
- **templates/** - HTML-шаблоны веб-интерфейса (форма логина, меню и т.п).

Интерфейс реализован при помощи **config.py,** в котором создана конфигурация подключения к бд. Задаётся имя БД (DB_NAME), в которой находится “autoschool”, параметры подключения к серверу (DB_HOST, DB_PORT), словарь DB_USERS, задающий соответствие логических ролей интерфейса (студент, бухгалтер, администратор), секретный ключ SECRET_KEY, используемый Flask для хранения сессий пользователей.

```python
DB_NAME = "autoschool"

DB_USERS = {
    "student": {
        "user": "evg_student",
        "password": "studpass",
    },
    "accountant": {
        "user": "evg_accountant",
        "password": "accpass",
    },
    "admin": {
        "user": "evg_admin",
        "password": "adminpass",
    },
}

DB_HOST = "localhost"
DB_PORT = 5432

SECRET_KEY = "lab4_autoschool_secret_key"
```

Рисунок 15 – Содержимое config.py.

После, была произведена настройка ORM с использованием SQLAlchemy. В **models.py** был реализован класс Base и и классы модели:

– “Student” - таблица autochool.students;
– “MainLog” - таблица логирования autochool.main_log;
– “VStidentTimetable”, “VStidentAttendance” - v_stident_timetable, v_stident_attendance представления для студента;
– “VContractsPayments”, “VStidentsDebts” – v_contracts_payments, v_stidents_debts представления для бухгалтера.

```python
from sqlalchemy import (
    Column, Integer, String, Date, DateTime, Boolean,
    Numeric, Text
)
from sqlalchemy.orm import declarative_base

Base = declarative_base()
SCHEMA = "autoschool"

class Student(Base):
    __tablename__ = "student"
    __table_args__ = {"schema": SCHEMA}

    student_id = Column(Integer, primary_key=True)
    full_name = Column(String)
    birth_date = Column(Date)
    email = Column(String)

class MainLog(Base):
    __tablename__ = "main_log"
    __table_args__ = {"schema": SCHEMA}

    log_id = Column(Integer, primary_key=True)
    table_name = Column(Text)
    operation = Column(String(1))
    changed_at = Column(DateTime)
    db_user = Column(Text)
    old_data = Column(Text)
    new_data = Column(Text)

class VStudentTimetable(Base):
    __tablename__ = "v_student_timetable"
    __table_args__ = {"schema": SCHEMA}
    student_id = Column(Integer, primary_key=True)
    full_name = Column(Text)
    group_code = Column(Text)
    lesson_id = Column(Integer, primary_key=True)
    start_time = Column(DateTime)
    end_time = Column(DateTime)
    topic_title = Column(Text)
    building = Column(Text)
    room_number = Column(Text)
```

Рисунок 16 – Содержимое models.py.

```python
class VStudentAttendance(Base):
    __tablename__ = "v_student_attendance"
    __table_args__ = {"schema": SCHEMA}

    student_id = Column(Integer, primary_key=True)
    full_name = Column(Text)
    topic_title = Column(Text, primary_key=True)
    start_time = Column(DateTime, primary_key=True)
    present = Column(Boolean)
    mark = Column(Integer)


class VStudentDebts(Base):
    __tablename__ = "v_student_debts"
    __table_args__ = {"schema": SCHEMA}


    student_id = Column(Integer, primary_key=True)
    group_code = Column(Text)
    contract_id = Column(Integer)
    sign_date = Column(Date)
    debt = Column(Numeric)
```

Рисунок 17 – Содержимое models.py.

Реализация веб-интерфейса реализована при помощи HTML на основе Flask:

- base.html – общая навигационная панель и вывод сообщения пользователю;
- login.html – форма авторизации;
- menu.html – главное меню приложения;
- table.html – шаблон для отображения табличных данных;
- aggregate.html – вывод агрегированных результатов с использованием GROUP BY;
- student_form.html – форма для добавления нового студента.

```
{% extends "base.html" %}
{% block content %}
<h2>{{ title }}</h2>
<table class="table table-bordered table-sm">
    <thead>
    <tr>
        {% for col in columns %}
            <th>{{ col }}</th>
        {% endfor %}
    </tr>
    </thead>
    <tbody>
    {% for row in rows %}
        <tr>
            {% for col in columns %}
                <td>
                    {# row может быть объектом Row (результат db.execute)
                       или ORM-объектом. Для Row используем индекс,
                       для ORM — фильтр attr (аналог getattr). #}
                    {% if row.__class__.__name__ == 'Row' or row.__class__.__name__ == 'RowMapping' %}
                        {{ row[col] }}
                    {% else %}
                        {{ row|attr(col) }}
                    {% endif %}
                </td>
            {% endfor %}
        </tr>
    {% endfor %}
    </tbody>
</table>
{% endblock %}
```

Рисунок 18 – Содержимое aggregate.html.

```
<!doctype html>
<html lang="ru">
<head>
    <meta charset="utf-8">
    <title>{{ title or "Автошкола" }}</title>
    <link rel="stylesheet"
          href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body class="p-3">
<nav class="mb-3">
    {% if session.role %}
        Вы вошли как <b>{{ session.role }}</b> |
        <a href="{{ url_for('index') }}">Меню</a> |
        <a href="{{ url_for('logout') }}">Выход</a>
    {% endif %}
</nav>

{% with messages = get_flashed_messages(with_categories=true) %}
  {% if messages %}
    {% for category, message in messages %}
      <div class="alert alert-{{ category }}">{{ message }}</div>
    {% endfor %}
  {% endif %}
{% endwith %}

<div class="container">
    {% block content %}{% endblock %}
</div>
</body>
</html>
```

Рисунок 19 – Содержимое base.html.

```
{% extends "base.html" %}
{% block content %}
<h2>Вход</h2>
<form method="post">
    <div class="mb-3">
        <label class="form-label">Роль</label>
        <select name="role" class="form-select">
            <option value="student">Студент</option>
            <option value="accountant">Бухгалтер</option>
            <option value="admin">Администратор</option>
        </select>
    </div>
    <div class="mb-3">
        <label class="form-label">Пароль</label>
        <input class="form-control" type="password" name="password">
    </div>
    <button class="btn btn-primary" type="submit">Войти</button>
</form>
{% endblock %}
```

Рисунок 20 – Содержимое login.html.

```
{% extends "base.html" %}
{% block content %}
<h2>Меню (роль: {{ role }})</h2>

{% if role == "student" %}
<ul>
    <li><a href="{{ url_for('student_timetable') }}">Личное расписание</a></li>
    <li><a href="{{ url_for('student_attendance') }}">Посещаемость и оценки</a></li>
    <li><a href="{{ url_for('student_analytics') }}">Статистика посещаемости</a></li>
</ul>
{% elif role == "accountant" %}
<ul>
    <li><a href="{{ url_for('accountant_contracts') }}">Реестр договоров и оплат</a></li>
    <li><a href="{{ url_for('accountant_debts') }}">Задолженности студентов</a></li>
    <li><a href="{{ url_for('accountant_analytics') }}">Финансовая статистика</a></li>
</ul>
{% elif role == "admin" %}
<ul>
    <li><a href="{{ url_for('student_timetable') }}">Просмотр расписания</a></li>
    <li><a href="{{ url_for('accountant_contracts') }}">Просмотр финансовых представлений</a></li>
    <li><a href="{{ url_for('admin_students') }}">Управление студентами</a></li>
    <li><a href="{{ url_for('admin_log') }}">Журнал изменений</a></li>
</ul>
{% endif %}
{% endblock %}
```

Рисунок 21 – Содержимое menu.html.

```
{% extends "base.html" %}
{% block content %}
<h2>{{ title }}</h2>
<form method="post">
    <div class="mb-3">
        <label class="form-label">ФИО</label>
        <input class="form-control" type="text" name="full_name" required>
    </div>
    <div class="mb-3">
        <label class="form-label">Дата рождения (YYYY-MM-DD)</label>
        <input class="form-control" type="text" name="birth_date" required>
    </div>
    <div class="mb-3">
        <label class="form-label">Email</label>
        <input class="form-control" type="email" name="email">
    </div>
    <button class="btn btn-primary" type="submit">Сохранить</button>
</form>
{% endblock %}
```

Рисунок 22 – Содержимое student_form.html.

```
{% extends "base.html" %}
{% block content %}
<h2>{{ title }}</h2>
{% if admin_students %}
    <a href="{{ url_for('admin_students_create') }}"
       class="btn btn-success mb-3">
        Добавить студента
    </a>
{% endif %}
<table class="table table-striped table-sm">
    <thead>

    <tr>
        {% for col in columns %}
            <th>{{ col }}</th>
        {% endfor %}
        {% if admin_students %}
            <th>Действия</th>
        {% endif %}
    </tr>
    </thead>
    <tbody>
    {% for row in rows %}
        <tr>
            {% for col in columns %}
                <td>
                    {# Если это результат raw SQL (Row / RowMapping) #}
                    {% if row.__class__.__name__ == 'Row' or row.__class__.__name__ == 'RowMapping' %}
                        {{ row[col] }}
                    {% else %}
                        {{ row|attr(col) }}
                    {% endif %}
                </td>
            {% endfor %}
            {% if admin_students %}
    <td>
        <form method="post"
              action="{{ url_for('admin_students_delete', student_id=row.student_id) }}">
            <button class="btn btn-sm btn-danger" type="submit">Удалить</button>
        </form>
    </td>
{% endif %}

        </tr>
    {% endfor %}
    </tbody>
</table>
{% endblock %}
```

Рисунок 23 – Содержимое table.html.

Модуль app.py реализовывает Flask-приложение, которое обеспечивает взаимодействие пользователя с БД.

Подключение к БД через ORM. Для каждой роли создаётся отдельный engine SQLalchemy с соответствующей учётной записью PostgreSQL.

Авторизация и роль. В /login пользователь выбирает тип учётной записи и вводит пароль. Если проверка успешная – пароль сохраняется в сессии.

Интерфейсы для ролей. Для каждой роли реализованы соответствующие страницы.

```python
from flask import (
    Flask, render_template, request,
    redirect, url_for, session, flash
)
from sqlalchemy import create_engine, text
from sqlalchemy.orm import sessionmaker

from config import DB_NAME, DB_USERS, DB_HOST, DB_PORT, SECRET_KEY
from models import (
    Base, Student, MainLog,
    VStudentTimetable, VStudentAttendance,
    VStudentDebts
)


app = Flask(__name__)
app.secret_key = SECRET_KEY

_engines = {}


def get_engine_for_role(role: str):
    if role not in DB_USERS:
        raise RuntimeError("Неизвестная роль")
    if role in _engines:
        return _engines[role]

    creds = DB_USERS[role]
    url = (
        f"postgresql+psycopg2://{creds['user']}:{creds['password']}"
        f"@{DB_HOST}:{DB_PORT}/{DB_NAME}"
    )
    engine = create_engine(url, echo=False)
    _engines[role] = engine
    return engine


def get_db_session():
    role = session.get("role")
    if role is None:
        return None
    engine = get_engine_for_role(role)
    SessionLocal = sessionmaker(bind=engine)
    return SessionLocal()

@app.route("/")
def index():
    if "role" not in session:
        return redirect(url_for("login"))
    return render_template("menu.html", role=session["role"])
```

Рисунок 24 – Содержимое app.py.

```python
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        role = request.form.get("role")
        password = request.form.get("password")
        creds = DB_USERS.get(role)
        if creds and creds["password"] == password:
            session["role"] = role
            flash("Успешный вход как " + role, "success")
            return redirect(url_for("index"))
        else:
            flash("Неверная роль или пароль", "danger")

    return render_template("login.html")


@app.route("/logout")
def logout():
    session.clear()
    return redirect(url_for("login"))

@app.route("/student/timetable")
def student_timetable():
    if session.get("role") not in ("student", "admin"):
        flash("Нет доступа", "danger")
        return redirect(url_for("index"))

    db = get_db_session()

    full_name = request.args.get("full_name")

    query = db.query(VStudentTimetable)
    if full_name:
        query = query.filter(VStudentTimetable.full_name.ilike(f"%{full_name}%"))

    sort = request.args.get("sort", "start_time")
    sort_column = getattr(VStudentTimetable, sort, VStudentTimetable.start_time)
    query = query.order_by(sort_column)

    rows = query.all()
    return render_template(
        "table.html",
        title="Личное расписание",
        columns=[
            "student_id", "full_name", "group_code",
            "lesson_id", "start_time", "end_time",
            "topic_title", "building", "room_number"
        ],
        rows=rows
    )
```

Рисунок 25 – Содержимое app.py.

```python
@app.route("/student/attendance")
def student_attendance():
    if session.get("role") not in ("student", "admin"):
        flash("Нет доступа", "danger")
        return redirect(url_for("index"))

    db = get_db_session()
    query = db.query(VStudentAttendance)

    topic = request.args.get("topic")
    if topic:
        query = query.filter(VStudentAttendance.topic_title.ilike(f"%{topic}%"))

    sort = request.args.get("sort", "start_time")
    sort_column = getattr(VStudentAttendance, sort, VStudentAttendance.start_time)
    query = query.order_by(sort_column)

    rows = query.all()
    return render_template(
        "table.html",
        title="Посещаемость и оценки",
        columns=[
            "student_id", "full_name", "topic_title",
            "start_time", "present", "mark"
        ],
        rows=rows
    )


@app.route("/student/analytics")
def student_analytics():
    if session.get("role") not in ("student", "admin"):
        flash("Нет доступа", "danger")
        return redirect(url_for("index"))

    db = get_db_session()

    sql = text("""
    SELECT
        full_name,
        topic_title,
        COUNT(*) AS lessons_total,
        SUM(CASE WHEN present THEN 1 ELSE 0 END) AS lessons_present,
        AVG(CAST(mark AS INTEGER)) AS avg_mark
    FROM autoschool.v_student_attendance
    GROUP BY full_name, topic_title
    ORDER BY full_name, topic_title;
""")

    result = db.execute(sql).fetchall()
    return render_template(
        "aggregate.html",
        title="Статистика посещаемости по темам",
        columns=["full_name", "topic_title", "lessons_total", "lessons_present", "avg_mark"],
        rows=result
    )
```

Рисунок 26 – Содержимое app.py.

```python
@app.route("/accountant/contracts")
def accountant_contracts():
    if session.get("role") not in ("accountant", "admin"):
        flash("Нет доступа", "danger")
        return redirect(url_for("index"))

    db = get_db_session()

    result = db.execute(text("SELECT * FROM autoschool.v_contracts_payments ORDER BY 1")).mappings().all()

    if result:
        columns = list(result[0].keys())
    else:
        columns = []

    return render_template(
        "table.html",
        title="Реестр договоров и оплат",
        columns=columns,
        rows=result
    )

@app.route("/accountant/debts")
def accountant_debts():
    if session.get("role") not in ("accountant", "admin"):
        flash("Нет доступа", "danger")
        return redirect(url_for("index"))
    db = get_db_session()
    rows = db.query(VStudentDebts).order_by(
        VStudentDebts.group_code,
        VStudentDebts.student_id
    ).all()

    return render_template(
        "table.html",
        title="Задолженности студентов",
        columns=[
            "student_id", "group_code",
            "contract_id", "sign_date", "debt"
        ],
        rows=rows
    )

@app.route("/accountant/analytics")
def accountant_analytics():
    if session.get("role") not in ("accountant", "admin"):
        flash("Нет доступа", "danger")
        return redirect(url_for("index"))
    db = get_db_session()
    sql = text("""
        SELECT
            group_code,
            COUNT(student_id) AS students_count,
            SUM(debt) AS total_debt
        FROM autoschool.v_student_debts
        GROUP BY group_code
        ORDER BY group_code;
    """)
    rows = db.execute(sql).fetchall()
    return render_template(
        "aggregate.html",
        title="Задолженность по группам",
        columns=["group_code", "students_count", "total_debt"],
        rows=rows
    )
```

Рисунок 27 – Содержимое app.py.

```python
@app.route("/admin/students")
def admin_students():
    if session.get("role") != "admin":
        flash("Нет доступа", "danger")
        return redirect(url_for("index"))

    db = get_db_session()
    students = db.query(Student).order_by(Student.student_id).all()
    return render_template(
        "table.html",
        title="Список студентов",
        columns=["student_id", "full_name", "birth_date", "email"],
        rows=students,
        admin_students=True
    )

@app.route("/admin/students/create", methods=["GET", "POST"])
def admin_students_create():
    if session.get("role") != "admin":
        flash("Нет доступа", "danger")
        return redirect(url_for("index"))

    if request.method == "POST":
        full_name = request.form["full_name"]
        birth_date = request.form["birth_date"]
        email = request.form["email"]

        db = get_db_session()
        student = Student(full_name=full_name, birth_date=birth_date, email=email)
        db.add(student)
        db.commit()
        flash("Студент добавлен", "success")
        return redirect(url_for("admin_students"))
    return render_template("student_form.html", title="Добавить студента")


@app.route("/admin/students/<int:student_id>/delete", methods=["POST"])
def admin_students_delete(student_id):
    if session.get("role") != "admin":
        flash("Нет доступа", "danger")
        return redirect(url_for("index"))

    db = get_db_session()
    student = db.query(Student).get(student_id)
    if student:
        db.delete(student)
        db.commit()
        flash("Студент удалён", "success")
    return redirect(url_for("admin_students"))

@app.route("/admin/log")
def admin_log():
    if session.get("role") != "admin":
        flash("Нет доступа", "danger")
        return redirect(url_for("index"))

    db = get_db_session()
    rows = db.query(MainLog).order_by(MainLog.changed_at.desc()).limit(50).all()
    return render_template(
        "table.html",
        title="Журнал изменений (main_log)",
        columns=["log_id", "table_name", "operation", "changed_at", "db_user", "old_data", "new_data"],
        rows=rows
    )
```

Рисунок 28 – Содержимое app.py.

```python
if __name__ == "__main__":
    app.run(debug=True)
```

Рисунок 29 – Содержимое app.py.

4.1. ТЕСТИРОВАНИЕ

Зайдём под разными ролями и проверим функционал:

Student:

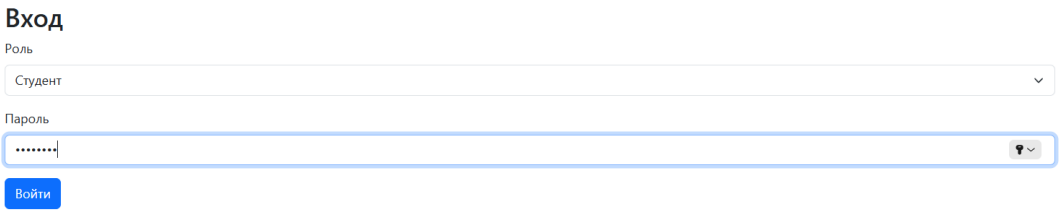


Рисунок 30 – Поле входа.

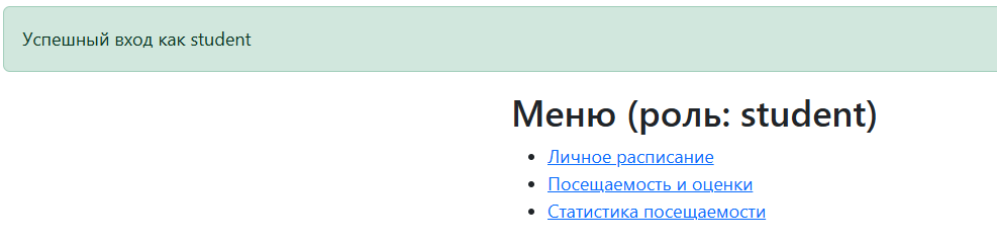


Рисунок 31 – Меню студента.

Личное расписание

student_id	full_name	group_code	lesson_id	start_time	end_time	topic_title	building	room_number
1	Иванов Иван Иванович	B-101	1	2025-02-03 10:00:00	2025-02-03 11:30:00	ПДД: основы	A	101
2	Петров Пётр Петрович	B-101	1	2025-02-03 10:00:00	2025-02-03 11:30:00	ПДД: основы	A	101
8	Фёдоров Никита Ильич	B-102	4	2025-02-03 12:00:00	2025-02-03 13:30:00	ПДД: основы	B	201
3	Сидорова Анна Сергеевна	B-201	5	2025-02-04 10:00:00	2025-02-04 11:30:00	ПДД: основы+	B	202
4	Михайлов Дмитрий Олегович	B-201	5	2025-02-04 10:00:00	2025-02-04 11:30:00	ПДД: основы+	B	202
2	Петров Пётр Петрович	B-101	2	2025-02-05 10:00:00	2025-02-05 11:30:00	Знаки и разметка	A	102
1	Иванов Иван Иванович	B-101	2	2025-02-05 10:00:00	2025-02-05 11:30:00	Знаки и разметка	A	102
5	Кузнецова Елена Викторовна	A-101	7	2025-02-05 14:00:00	2025-02-05 15:30:00	Мото: теория	C	301
3	Сидорова Анна Сергеевна	B-201	6	2025-02-06 10:00:00	2025-02-06 11:30:00	Манёвры	C	302
4	Михайлов Дмитрий Олегович	B-201	6	2025-02-06 10:00:00	2025-02-06 11:30:00	Манёвры	C	302
7	Соколов Артём Денисович	C-101	8	2025-02-06 15:00:00	2025-02-06 16:30:00	Грузовые: ПДД	B	202
1	Иванов Иван Иванович	B-101	3	2025-02-07 10:00:00	2025-02-07 11:30:00	Перекрёстки	A	101
2	Петров Пётр Петрович	B-101	3	2025-02-07 10:00:00	2025-02-07 11:30:00	Перекрёстки	A	101

Рисунок 32 – Личное расписание.

Посещаемость и оценки

student_id	full_name	topic_title	start_time	present	mark
1	Иванов Иван Иванович	ПДД: основы	2025-02-03 10:00:00	True	5
2	Петров Пётр Петрович	ПДД: основы	2025-02-03 10:00:00	True	4
8	Фёдоров Никита Ильич	ПДД: основы	2025-02-03 12:00:00	True	5
3	Сидорова Анна Сергеевна	ПДД: основы+	2025-02-04 10:00:00	True	4
4	Михайлов Дмитрий Олегович	ПДД: основы+	2025-02-04 10:00:00	True	5
2	Петров Пётр Петрович	Знаки и разметка	2025-02-05 10:00:00	False	None
1	Иванов Иван Иванович	Знаки и разметка	2025-02-05 10:00:00	True	5
5	Кузнецова Елена Викторовна	Мото: теория	2025-02-05 14:00:00	True	5
6	Орлова Мария Андреевна	Мото: теория	2025-02-05 14:00:00	True	4

Рисунок 33 – Посещаемость и оценки.

Статистика посещаемости по темам

full_name	topic_title	lessons_total	lessons_present	avg_mark
Иванов Иван Иванович	Знаки и разметка	1	1	5.0000000000000000
Иванов Иван Иванович	ПДД: основы	1	1	5.0000000000000000
Кузнецова Елена Викторовна	Мото: теория	1	1	5.0000000000000000
Михайлов Дмитрий Олегович	ПДД: основы+	1	1	5.0000000000000000
Орлова Мария Андреевна	Мото: теория	1	1	4.0000000000000000
Петров Пётр Петрович	Знаки и разметка	1	0	None
Петров Пётр Петрович	ПДД: основы	1	1	4.0000000000000000
Сидорова Анна Сергеевна	ПДД: основы+	1	1	4.0000000000000000
Фёдоров Никита Ильич	ПДД: основы	1	1	5.0000000000000000

Рисунок 34 – Статистика посещаемости.

Accountant:

Реестр договоров и оплат

contract_id	sign_date	program_name	category_code	agreed_price	paid_sum	debt	student_id	full_name	last_payment_date
1	2025-01-10	Базовый курс B	B	50000.00	50000.00	0.00	1	Иванов Иван Иванович	2025-10-31
2	2025-01-12	Базовый курс B	B	45000.00	45000.00	0.00	2	Петров Пётр Петрович	2025-02-10
3	2025-01-15	Интенсив B	B	52000.00	52000.00	0.00	3	Сидорова Анна Сергеевна	2025-02-20
4	2025-01-18	Интенсив B	B	52000.00	52000.00	0.00	4	Михайлов Дмитрий Олегович	2025-01-22
5	2025-01-20	Курс A	A	38000.00	38000.00	0.00	5	Кузнецова Елена Викторовна	2025-01-22
6	2025-01-25	Курс A	A	38000.00	38000.00	0.00	6	Орлова Мария Андреевна	2025-02-14
7	2025-01-28	Курс C	C	60000.00	60000.00	0.00	7	Соколов Артём Денисович	2025-02-25
8	2025-02-02	Базовый курс B	B	45000.00	45000.00	0.00	8	Фёдоров Никита Ильич	2025-02-05

Рисунок 35 – Реестр договоров и оплат.

Задолженности студентов

student_id	group_code	contract_id	sign_date	debt

Рисунок 36 – Задолженности студентов.

Задолженность по группам

group_code	students_count	total_debt

Рисунок 37 – Финансовая задолженность.

Таблицы с на рисунках 23 и 24 не имеют данных в БД. Поэтому и пользователь не видит информацию.

Admin:

Личное расписание

student_id	full_name	group_code	lesson_id	start_time	end_time	topic_title	building	room_number
1	Иванов Иван Иванович	B-101	1	2025-02-03 10:00:00	2025-02-03 11:30:00	ПДД: основы	A	101
2	Петров Пётр Петрович	B-101	1	2025-02-03 10:00:00	2025-02-03 11:30:00	ПДД: основы	A	101
8	Фёдоров Никита Ильич	B-102	4	2025-02-03 12:00:00	2025-02-03 13:30:00	ПДД: основы	B	201
3	Сидорова Анна Сергеевна	B-201	5	2025-02-04 10:00:00	2025-02-04 11:30:00	ПДД: основы+	B	202
4	Михайлов Дмитрий Олегович	B-201	5	2025-02-04 10:00:00	2025-02-04 11:30:00	ПДД: основы+	B	202
2	Петров Пётр Петрович	B-101	2	2025-02-05 10:00:00	2025-02-05 11:30:00	Знаки и разметка	A	102
1	Иванов Иван Иванович	B-101	2	2025-02-05 10:00:00	2025-02-05 11:30:00	Знаки и разметка	A	102
5	Кузнецова Елена Викторовна	A-101	7	2025-02-05 14:00:00	2025-02-05 15:30:00	Мото: теория	C	301
3	Сидорова Анна Сергеевна	B-201	6	2025-02-06 10:00:00	2025-02-06 11:30:00	Манёвры	C	302
4	Михайлов Дмитрий Олегович	B-201	6	2025-02-06 10:00:00	2025-02-06 11:30:00	Манёвры	C	302
7	Соколов Артём Денисович	C-101	8	2025-02-06 15:00:00	2025-02-06 16:30:00	Грузовые: ПДД	B	202
1	Иванов Иван Иванович	B-101	3	2025-02-07 10:00:00	2025-02-07 11:30:00	Перекрёстки	A	101
2	Петров Пётр Петрович	B-101	3	2025-02-07 10:00:00	2025-02-07 11:30:00	Перекрёстки	A	101

Рисунок 38 – Просмотр расписания.

Реестр договоров и оплат

contract_id	sign_date	program_name	category_code	agreed_price	paid_sum	debt	student_id	full_name	last_payment_date
1	2025-01-10	Базовый курс B	B	50000.00	50000.00	0.00	1	Иванов Иван Иванович	2025-10-31
2	2025-01-12	Базовый курс B	B	45000.00	45000.00	0.00	2	Петров Пётр Петрович	2025-02-10
3	2025-01-15	Интенсив B	B	52000.00	52000.00	0.00	3	Сидорова Анна Сергеевна	2025-02-20
4	2025-01-18	Интенсив B	B	52000.00	52000.00	0.00	4	Михайлов Дмитрий Олегович	2025-01-22
5	2025-01-20	Курс A	A	38000.00	38000.00	0.00	5	Кузнецова Елена Викторовна	2025-01-22
6	2025-01-25	Курс A	A	38000.00	38000.00	0.00	6	Орлова Мария Андреевна	2025-02-14
7	2025-01-28	Курс C	C	60000.00	60000.00	0.00	7	Соколов Артём Денисович	2025-02-25
8	2025-02-02	Базовый курс B	B	45000.00	45000.00	0.00	8	Фёдоров Никита Ильич	2025-02-05

Рисунок 39 – Просмотр финансовых представлений.

Добавим нового студента. Можем наблюдать его появление в таблице. При нажатии кнопки “Удалить” – пользователь удалится.

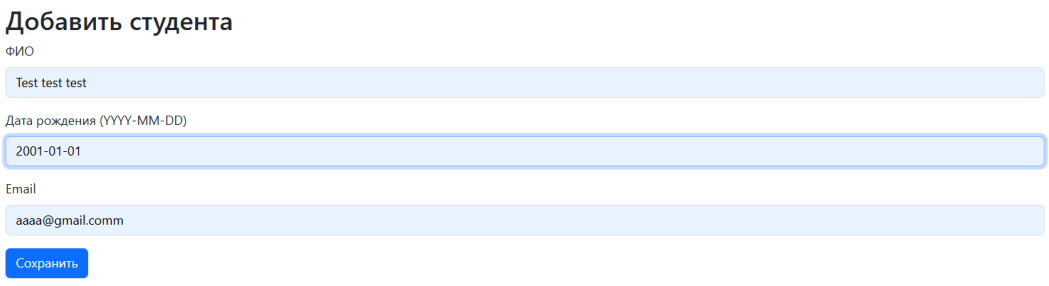


Рисунок 40 – Добавление пользователя.

Список студентов

Добавить студента

student_id	full_name	birth_date	email	Действия
1	Иванов Иван Иванович	2003-01-12	ivanov1@example.com	Удалить
2	Петров Пётр Петрович	2002-05-23	petrov2@example.com	Удалить
3	Сидорова Анна Сергеевна	2004-03-02	sidorova3@example.com	Удалить
4	Михайлов Дмитрий Олегович	2001-11-15	mikhailov4@example.com	Удалить
5	Кузнецова Елена Викторовна	2003-07-07	kuznecova5@example.com	Удалить
6	Орлова Мария Андреевна	2005-09-19	orlova6@example.com	Удалить
7	Соколов Артём Денисович	2002-12-30	sokolov7@example.com	Удалить
8	Фёдоров Никита Ильич	2003-04-18	fedorov8@example.com	Удалить
10	Test test test	2001-01-01	aaaa@gmail.comm	Удалить

Рисунок 41 – Управление студентами.

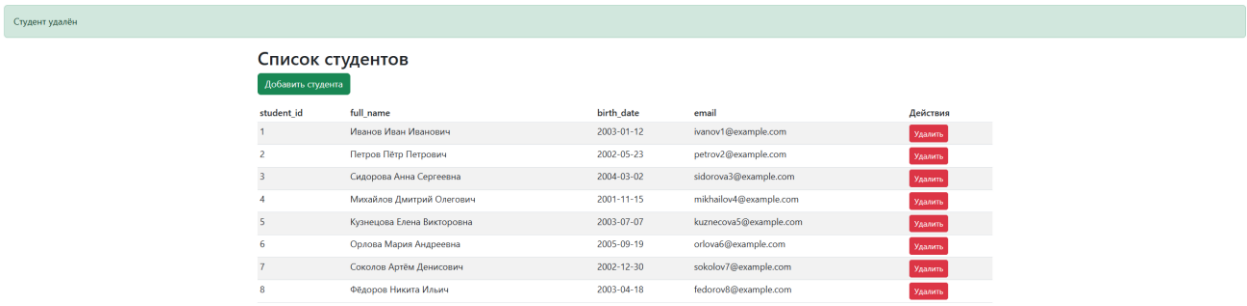


Рисунок 42 – Удаление студента.

	student_id [PK] integer	full_name character varying	birth_date date	email character varying
1	1	Иванов Иван Иванович	2003-01-12	ivanov1@example.com
2	2	Петров Пётр Петрович	2002-05-23	petrov2@example.com
3	3	Сидорова Анна Сергеевна	2004-03-02	sidorova3@example.com
4	4	Михайлов Дмитрий Олегов...	2001-11-15	mikhailov4@example.com
5	5	Кузнецова Елена Викторовна	2003-07-07	kuznecova5@example.com
6	6	Орлова Мария Андреевна	2005-09-19	orlova6@example.com
7	7	Соколов Артём Денисович	2002-12-30	sokolov7@example.com
8	8	Фёдоров Никита Ильич	2003-04-18	fedorov8@example.com

Рисунок 43 – Финальное состояние таблицы.

Журнал изменений (main_log)

log_id	table_name	operation	changed_at	db_user	old_data	new_data
6	student	D	2025-11-22 17:29:26.784268+03:00	evg_admin	{'email': 'aaaa@gmail.comm', 'full_name': 'Test test test', 'birth_date': '2001-01-01', 'student_id': 10}	None
5	student	I	2025-11-22 17:03:58.341563+03:00	evg_admin	None	{'email': 'aaaa@gmail.comm', 'full_name': 'Test test test', 'birth_date': '2001-01-01', 'student_id': 10}
4	student	D	2025-11-22 17:03:34.116294+03:00	evg_admin	{'email': 'test@testt.ru', 'full_name': 'Тест', 'birth_date': '2000-01-01', 'student_id': 9}	None
3	contract	U	2025-11-22 16:52:00.692814+03:00	postgres	{'status': 'active', 'sign_date': '2025-01-10', 'program_id': 1, 'student_id': 1, 'contract_id': 1, 'agreed_price': 45000.0}	{'status': 'active', 'sign_date': '2025-01-10', 'program_id': 1, 'student_id': 1, 'contract_id': 1, 'agreed_price': 50000.0}
2	payment	I	2025-10-31 19:34:51.060831+03:00	postgres	None	{'amount': 5000.0, 'status': 'accepted', 'payment_id': 16, 'contract_id': 1, 'method_code': None, 'payment_date': '2025-10-31'}
1	student	I	2025-10-31 19:30:09.644573+03:00	postgres	None	{'email': 'test@testt.ru', 'full_name': 'Тест', 'birth_date': '2000-01-01', 'student_id': 9}

Рисунок 44 – Журнал изменений.

5. АУДИТ БЕЗОПАСНОСТИ РАЗРАБОТАННОГО СЕРВИСА

5.1. Реализованные меры

Главной **мерой** обеспечения **безопасности** на уровне СУБД стало внедрение системы аудита, контроля доступа и защиты данных. За основу взят принцип минимально необходимых привилегий, ограничивающий доступ пользователей к данным и объектам.

Каждая операция автоматически фиксируется с помощью **триггера**, записывающего информацию в журнал main_log. Создан **набор ролевых групп** (student_group_role, accountant_group_role, admin_group_role), каждая из которых имеет ограниченный доступ только к тем представлениям и данным, которые необходимы в рамках их функциональной деятельности.

Для защиты **конфиденциальных данных** реализовано симметричное шифрование с использованием расширения pgcrypto. Для **предотвращения конфликтов** в расписании практических занятий реализованы EXCLUDE-ограничения с использованием tsrange, исключающие пересечение интервалов времени для одного инструктора или транспортного средства.

На уровне веб-сервиса, реализованного во Flask, предусмотрены механизмы надёжной аутентификации и строгой авторизации. Доступ осуществляется от имени отдельных ролей PostgreSQL. Взаимодействие с данными осуществляется через ORM SQLAlchemy, что снижает риск SQL-инъекций благодаря параметризованным запросам.

5.2. Возможные улучшения

Несмотря на эффективность симметричного шифрования, его использование требует постоянного контроля за безопасностью хранения ключей. Для устранения рисков утечки стоит интегрировать модуль управления ключами.

Для повышения защищённости веб-сервиса необходимо расширить механизм аутентификации. Внедрение двухфакторной авторизации значительно повысит устойчивость учётных записей. Стоит обратить внимание на организация защищённого соединения между клиентом и сервером через протокол HTTPS. Регулярное проведение тестов на уязвимости также является необходимой мерой для обеспечения защиты.

Реализация данных улучшений позволит увеличить уровень безопасности.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была разработана защищённая реляционная база данных для информационной системы автошколы, а также веб-сервис, обеспечивающий многоуровневый доступ различных категорий пользователей. На этапе анализа предметной области выделены ключевые сущности, связи и ограничения. Оптимальная структура обеспечена при помощи построение ER-диаграммы, формирования предварительных отношений и их нормализация до третьей нормальной формы.

В рамках обеспечения безопасности внедрена система аудита, основанная на триггерах, регистрирующих все операции добавления, изменения и удаления данных. Для защиты конфиденциальной информации реализовано симметричное шифрование с использованием расширения pgcrypto. Построена ролевая модель доступа, что существенно снижает риски НСД.Разработанный веб-сервис на Flask поддерживает авторизацию, работу с представлениями согласно роли, а также предоставляет администратору возможность выполнять CRUD-операции, включая добавление и удаление студентов.

Проведённое тестирование подтвердило работоспособность всех реализованных механизмов. Проведённый аудит безопасности показал соответствие системы базовым требованиям. Разработанное решение является полноценным функциональным прототипом, готовым к дальнейшему расширению и внедрению.