Приложение 2.ЗЗ.1к ОПОП по специальности
09.02.13 Интеграция решений с применением
технологий искусственного интеллекта

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет им. Г.И. Носова»

Многопрофильный колледж

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ЛАБОРАТОРНЫХ ЗАНЯТИЙ По ПМ. 02 АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ

МДК.02.01 Технология разработки и защиты баз данных

для обучающихся специальности
09.02.13 Интеграция решений с применением технологий искусственного интеллекта

Магнитогорск, 2025

СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ

Важную часть теоретической и профессиональной практической подготовки обучающихся составляют лабораторные занятия.

Состав и содержание лабораторных занятий направлены на реализацию Федерального государственного образовательного стандарта среднего профессионального образования.

Ведущей дидактической целью лабораторных занятий является формирование профессиональных практических умений (умений выполнять определенные действия, операции, необходимые в последующем в профессиональной деятельности).

Ведущей дидактической целью лабораторных занятий является экспериментальное подтверждение и проверка существенных теоретических положений (законов, зависимостей).

В соответствии с рабочей программой профессионального модуля ПМ.02 «Администрирование баз данных» МДК 02.01 «Технология разработки и защиты баз данных» предусмотрено проведение лабораторных занятий.

В результате их выполнения, обучающийся должен:

уметь:

– осуществлять основные функции по администрированию баз данных;

– настраивать политики безопасности при работе с сервером баз данных;

– дать независимую оценку уровня безопасности;

– производить регламентное обновление программного обеспечения;

– разрабатывать перечень рекомендаций по дальнейшей эксплуатации бд с максимальной защитой хранящейся информации;

– добавлять, удалять и изменять данные в базе данных;

– производить операции по импорту и экспорту данных в различных форматах.

Содержание практических и лабораторных занятий ориентировано на освоение вида деятельности программы подготовки специалистов среднего звена по специальности и овладению ***профессиональными компетенциями:***

ПК 2.2 Осуществлять процедуры администрирования баз данных.

ПК 2.3 Проводить аудит систем безопасности баз данных с использованием регламентов по защите информации.

ПК 2.5 Подготавливать данные для базы знаний.

А также формированию общих компетенций:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ОК 04 Эффективно взаимодействовать и работать в коллективе и команде.

ОК 05 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста.

ОК 09 Пользоваться профессиональной документацией на государственном и иностранном языках.

Выполнение обучающихся практических и/или лабораторных работ по профессиональному модулю ПМ.02 «Администрирование баз данных» МДК 02.01 «Технология разработки и защиты баз данных» направлено на:

– обобщение, систематизацию, углубление, закрепление, развитие и детализацию полученных теоретических знаний по конкретным темам учебной дисциплины;

– формирование умений применять полученные знания на практике, реализацию единства интеллектуальной и практической деятельности;

– формирование и развитие умений: наблюдать, сравнивать, сопоставлять, анализировать, делать выводы и обобщения, самостоятельно вести исследования, пользоваться различными приемами измерений, оформлять результаты в виде таблиц, схем, графиков;

– приобретение навыков работы с различными приборами, аппаратурой, установками и другими техническими средствами для проведения опытов;

– развитие интеллектуальных умений у будущих специалистов: аналитических, проектировочных, конструктивных и др.;

– выработку при решении поставленных задач профессионально значимых качеств, таких как самостоятельность, ответственность, точность, творческая инициатива.

Лабораторные занятия проводятся в рамках соответствующей темы, после освоения дидактических единиц, которые обеспечивают наличие знаний, необходимых для ее выполнения.

2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Тема 1.1 Основы хранения и обработки данных. Проектирование базы данных

Лабораторное занятие №1
Создание концептуальной модели базы данных с использованием диаграммы "сущность-связь" (ER-диаграмма)

Цель: освоить принципы проектирования баз данных, разработав концептуальную модель базы данных с использованием диаграммы "сущность-связь" (ER-диаграммы).

Выполнив работу, вы будете уметь:
– создавать концептуальную модель базы данных;
– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:
ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.
ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.
ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:
LibreOffice Draw, Draw io

Задание:
В соответствии со своим вариантом проанализируйте предметную область решаемой задачи и разработайте концептуальную модель соответствующей базы данных с использованием диаграммы "сущность-связь" (ER-диаграммы).

Варианты заданий:

Вариант	Наименование базы данных, описание
1	*БД «Управление запасами (Inventory Management)».* Описание: Система для отслеживания и управления запасами товаров на складе. Включает информацию о товарах, их количестве, местоположении на складе, поставщиках и заказах. Позволяет автоматизировать процессы пополнения запасов и учета движения товаров.
2	*БД «Система управления клиентами (Customer Relationship Management, CRM)».* Описание: База данных для хранения информации о клиентах, их взаимодействии с компанией, истории покупок и предпочтениях. Позволяет анализировать данные для повышения уровня обслуживания клиентов, разработки маркетинговых стратегий и улучшения продаж.
3	*БД «Образовательная платформа (Educational Platform)».* Описание: Система для управления учебным процессом, включающая информацию о курсах, студентах, преподавателях и оценках. Позволяет отслеживать успеваемость студентов, организовывать расписание занятий и управлять материалами курса.
4	*БД «Управление проектами (Project Management)».* Описание: База данных для планирования и отслеживания выполнения проектов. Включает информацию о задачах, сроках, участниках проекта и ресурсах. Позволяет анализировать эффективность работы команды и управлять рисками.

5	*БД «Электронная коммерция (E-commerce)».* Описание: Система для управления онлайн-продажами, включая информацию о товарах, пользователях, заказах и платежах. Обеспечивает функционал для обработки заказов, управления корзиной покупок и анализа продаж.
6	*БД «Система бронирования (Booking System)».* Описание: База данных для управления процессом бронирования услуг (например, отелей, авиабилетов или мероприятий). Включает информацию о доступных ресурсах, клиентах, бронированиях и платежах. Позволяет оптимизировать использование ресурсов и улучшить клиентский сервис.
7	*БД «Здравоохранение (Healthcare)».* Описание: Система для хранения медицинских записей пациентов, истории болезней, назначений и результатов анализов. Обеспечивает доступ к информации для врачей и медицинского персонала, а также позволяет отслеживать лечение и профилактические мероприятия.
8	*БД «Управление персоналом (Human Resource Management)».* Описание: База данных для учета сотрудников компании, их должностей, заработной платы, рабочего времени и других HR-процессов. Позволяет автоматизировать процессы найма, оценки эффективности и обучения сотрудников.
9	*БД «Финансовый учет (Financial Accounting)».* Описание: Система для управления финансовыми данными компании, включая бухгалтерский учет, отчеты о доходах и расходах, а также управление бюджетом. Позволяет контролировать финансовое состояние компании и принимать обоснованные решения.
10	*БД «Социальные сети (Social Networking)».* Описание: База данных для хранения информации о пользователях социальной сети, их профилях, сообщениях, друзьях и взаимодействиях. Позволяет анализировать поведение пользователей и улучшать функционал платформы.

Краткие теоретические сведения:

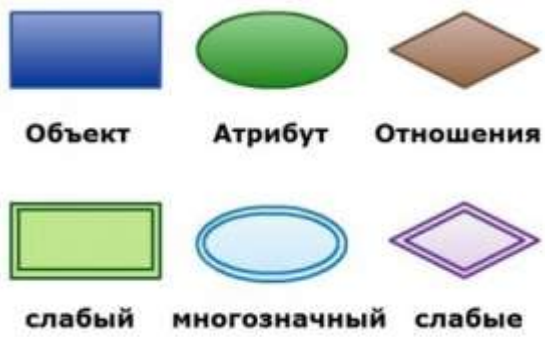
ER-диаграмма – это графическая модель, отображающая сущности предметной области, их атрибуты и взаимосвязи между ними. Концептуальная ER-диаграмма не учитывает особенности конкретной СУБД, а служит для наглядного представления структуры данных и их связей.

Основные элементы ER-диаграммы:

– **Сущность (Entity):** объект или понятие, о котором нужно хранить данные (например, Студент, Курс).

– **Атрибут (Attribute):** характеристика сущности (например, имя, дата рождения).

– **Связь (Relationship):** ассоциация между сущностями (например, студент записан на курс).



Алгоритм проектирования ER-диаграммы

1.**Определить сущности** - выделить основные объекты предметной области.

2.**Определить атрибуты** - для каждой сущности описать ключевые характеристики.

3.**Определить связи** - установить, как сущности взаимодействуют друг с другом (один к одному, один ко многим и многие ко многим).

4.**Назначить ключи** - для каждой сущности выберать первичный ключ, для связей - внешние ключи, если требуется.

5.**Визуализировать модель** - создать ER-диаграмму с помощью Draw.io, разместить сущности, атрибуты и связи.

Пример: ER-диаграмма для учебного процесса

Сущность	Атрибуты	Связи
Студент	ID, ФИО, Дата рождения	Записан на Курс
Курс	ID, Название, Кол-во часов	Ведёт Преподаватель
Преподаватель	ID, ФИО, Должность	Ведёт Курс

– Один студент может быть записан на несколько курсов (1:M).

– Один преподаватель может вести несколько курсов (1:M).

– Один курс могут посещать несколько студентов (M:N).

Порядок выполнения работы:

1. Запуск Draw.io

- Перейдите на сайт [draw.io (diagrams.net)].
- Выберите место для сохранения файла (на устройстве или в облаке).

2. Создание новой диаграммы

- Выберите "Создать новую диаграмму".
- В открывшемся окне выберите шаблон "Entity Relationship" или начните с пустого холста.

3. Добавление сущностей

- На панели слева выберите фигуру "Прямоугольник" (или специальную форму для сущностей).
- Разместите на холсте несколько сущностей (например, "Студент", "Курс", "Преподаватель").

4. Добавление атрибутов

- Для каждой сущности добавьте овалы или текстовые блоки с атрибутами (например, для "Студент": ID, ФИО, Дата рождения).
- Свяжите атрибуты с сущностями линиями или разместите их внутри прямоугольников.

5. Установка связей

- Используйте стрелки или линии для отображения связей между сущностями (например, "Студент" - "записан на" - "Курс").
- Укажите тип связи (1:1, 1:N, M:N), подписав линии.

6. Оформление и сохранение

- Настройте цвета, размеры, шрифты для лучшей читаемости.
- Сохраните диаграмму в нужном формате (PNG, SVG, XML).

Форма представления результата:

Оформленная концептуальная модель базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.1 Основы хранения и обработки данных. Проектирование базы данных

Лабораторное занятие №2
Разработка логической модели базы данных на основе ER-диаграммы

Цель: освоить принципы проектирования баз данных, используя ER-диаграммы для создания логической модели базы данных.

Выполнив работу, вы будете уметь:
– создавать логическую модель базы данных;
– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:
ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.
ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.
ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:
LibreOffice Draw, Draw io

Задание:
В соответствии со своим вариантом проанализируйте предметную область решаемой задачи и разработайте логическую модель на основе ER-диаграммы.

Варианты заданий:

Вариант	Наименование базы данных, описание
1	*БД «Лицензионное программное обеспечение».* Таблица «Лицензии»: номер лицензии; название; код CD-диска; код владельца. Таблица «CD-диски»: код CD-диска; дата выпуска; вид программного обеспечения; общий объем файлов, кбайт; пояснения о назначении и свойствах программного обеспечения. Таблица «Владельцы»: код владельца; владелец; город; адрес; телефон
2	*БД «Студенты МпК».* Таблица «Группы»: отделение; группа; Ф.И.О. кл. руководителя. Таблица «Студенты»: группа; шифр студента; Ф.И.О.; адрес; телефон; хобби. Таблица «Дисциплины»: шифр дисциплины; наименование дисциплины. Таблица «Успеваемость»: дата; шифр дисциплины; шифр студента; оценка; отметка о пропуске занятия
3	*БД «Гостиница».* Таблица «Номерной фонд»: категория номера (люкс, одноместный первой категории, двухместный первой категории и др.); номер помещения; место (А, Б, … – в зависимости от количества мест в номере); стоимость проживания за сутки. Таблица «Проживание»: дата заезда; дата выезда; номер помещения; место; Ф.И.О.; паспортные данные. Таблица «Бронирование»: дата заявки; код брони; категория номера; количество человек; дата заезда; срок пребывания
4	*БД «Товарооборот».* Таблица «Поставщики»: код поставщика; поставщик; адрес; телефон. Таблица «Товары»: код товара; товар; единица измерения; цена. Таблица «Поступление товаров»: дата поступления; код поставщика; код товара; количество. Таблица «Продажа»: дата продажи; код товара; количество
5	*БД «Промышленность региона».* Таблица «Предприятия»: код предприятия; предприятие; форма собственности; адрес; основной вид продукции. Таблица

	«Виды налогов»: код налога; вид налога (на прибыль, на имущество, НДС и др.); ставка, %. Таблица «Налоговые платежи»: код предприятия; код налога; дата платежа; сумма платежа. Таблица «Прибыль»: год; месяц; код предприятия; прибыль
6	*БД «Пассажирские поезда».* Таблица «Поезда»: категория поезда (скорый, пассажирский, пригородный); номер поезда; название поезда (для фирменного поезда). Таблица «Составы вагонов»: номер поезда; код состава; общее количество вагонов; схема формирования (например, 3К + 8П – три купейных и восемь плацкартных вагонов). Таблица «Расписание»: номер поезда; время прибытия; время отправления; режим движения (дни следования); станция назначения. Таблица «Перевозки»: дата отправления; номер поезда; код состава; количество пассажиров; прибыль за поездку
7	*БД «Автопарк».* Таблица «Типы автобусов»: код автобуса; марка автобуса; количество мест. Таблица «Парк»: код автобуса; гаражный номер; государственный номер; год выпуска. Таблица «Водители»: табельный номер водителя; Ф.И.О.; дата рождения; оклад; номер маршрута. Таблица «Перевозки»: дата; код автобуса; номер маршрута; табельный номер водителя; время выхода автобуса на маршрут; время прибытия автобуса с маршрута; причина схода автобуса с маршрута; количество проданных билетов
8	*БД «Агентство недвижимости».* Таблица «Риэлторы»: код риэлтора; Ф.И.О.; телефон. Таблица «Недвижимость»: номер объекта; адрес; тип дома; общая площадь; количество комнат; наличие балкона; наличие телефона; стоимость. Таблица «Сделки»: дата; регистрационный номер договора; код риэлтора; номер объекта
9	*БД «Авиаперевозки».* Таблица «Авиапарк»: код модели самолета; модель самолета; количество мест. Таблица «Рейсы и тарифы»: рейс; аэропорт отправления; аэропорт назначения; код класса; вид класса (бизнес-класс, эконом-класс); стоимость билета. Таблица «Перевозки»: рейс; дата вылета; код модели самолета; количество пассажиров; доход за рейс

Краткие теоретические сведения:

Проектирование базы данных заключается в ее многоступенчатом описании с различной степенью детализации и формализации, в ходе которого производится уточнение и оптимизация структуры базы данных. Проектирование начинается с описания предметной области и задач информационной системы, идет к более абстрактному уровню логического описания данных и далее – к схеме физической (внутренней) модели базы данных. Трем основным уровням моделирования системы – **концептуальному, логическому и физическому** соответствуют три последовательных этапа детализации описания объектов базы данных и их взаимосвязей.

На **концептуальном уровне** проектирования производиться смысловое описание информации предметной области, определяются ее границы, производиться абстрагирование от несущественных деталей. В результате определяются моделируемые объекты, и их свойства, и связи. Выполняется структуризация знаний о предметной области, стандартизируется терминология. Затем строится концептуальная модель, описываемая на естественном языке. Для описания свойств и связей объектов применяют различные диаграммы.

На следующем шаге принимается решение о том, в какой конкретно СУБД будет реализована база данных. **Выбор СУБД** является сложной задачей и должен основывается на потребностях с точки зрения информационной системы и пользователей. Определяющими здесь являются вид программного продукта и категория пользователей (или профессиональные программисты, или конечные пользователи, или то и другое).

Другими показателями, влияющими на выбор СУБД, являются:

- Удобство и простота использования;

- Качество средств разработки, защиты и контроля базы данных;
- Уровень коммуникационных средств (в случае применения ее в сетях);
- Фирма-разработчик;
- Стоимость.

Каждая конкретная СУБД работает с определенной моделью данных. Под моделью данных понимается способ их взаимосвязи: в виде иерархического дерева, сложной сетевой структуры или связанных таблиц. В настоящее время большинство СУБД использует табличную модель данных, называемую *реляционной*.

На **логическом уровне** производиться отображение данных концептуальной модели в логическую модель в рамках той структуры данных, которая поддерживается выбранной СУБД. Логическая модель не зависит от конкретной СУБД и может быть реализована на любой СУБД реляционного типа.

На **физическом уровне** производиться выбор рациональной структуры хранения данных и методов доступа к ним, которые обеспечивает выбранная СУБД. На этом уровне решаются вопросы эффективного выполнения запросов к БД, для чего строятся дополнительные структуры, например индексы. В физической модели содержится информация обо всех объектах базы данных (таблицах, индексах, процедурах и др.) и используемых типах данных. Физическая модель *зависит* от конкретной СУБД. Одной и той же логической модели может соответствовать несколько разных физических моделей. Физическое проектирование является начальным этапом реализации базы данных.

Порядок выполнения работы:

1. Выполнить анализ предметной области.
2. Выполнить анализ данных.
3. Определить набор атрибутов для данной предметной области.
4. Определить набор таблиц.

   При проектировании таблиц рекомендуется руководствоваться следующими основными принципами:

   - каждая таблица должна содержать данные только на одну тему;

   - данные не должны дублироваться.

5. Создать словарь имен.
6. Определить состав и типы полей.
7. Создать связи между таблицами.
8. Создать схему базы данных в Draw io.

Форма представления результата:
Оформленная логическая схема базы данных.

Критерии оценки:
Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.1 Основы хранения и обработки данных. Проектирование базы данных

Лабораторное занятие №3
Нормализация базы данных: приведение таблиц к третьей нормальной форме (3НФ)

Цель: освоить нормализацию базы данных.

Выполнив работу, вы будете уметь:
– выполнять нормализацию базы данных (приводить таблицы к третьей нормальной форме);
– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:
ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.
ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.
ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:
MySQL Workbench

Задание:
В соответствии со своим вариантом задания, привести базу данных к 3НФ. Реализовать схему базы данных в среде MySQL Workbench.

Краткие теоретические сведения:

Нормализация баз данных
Нормальные формы – это рекомендации по проектированию баз данных. Рекомендуется нормализовать базу данных в некоторой степени потому, что этот процесс имеет ряд существенных преимуществ с точки зрения эффективности и удобства обращения с базой данных.

- В нормализованной структуре базы данных можно производить сложные выборки данных относительно простыми SQL-запросами.
- Целостность данных. Нормализованная база данных позволяет надежно хранить данные.
- Нормализация предотвращает появление избыточности хранимых данных. Данные всегда хранятся только в одном месте, что делает легким процесс вставки, обновления и удаления данных. Есть исключение из этого правила. Ключи, сами по себе, хранятся в нескольких местах потому, что они копируются как внешние ключи в другие таблицы.
- Масштабируемость – это возможность системы справляться с будущим ростом. Для базы данных это значит, что она должна быть способна работать быстро, когда число пользователей и объемы данных возрастают. Масштабируемость – это очень важная характеристика любой модели базы данных и для РСУБД.

1 Первая нормальная форма (1НФ)
Первая нормальная форма гласит, что таблица базы данных – это представление сущности системы, которая создается. Примеры сущностей: заказы, клиенты, заказ билетов, отель, товар и т.д. Каждая запись в базе данных представляет один экземпляр сущности. Например, в таблице клиентов каждая запись представляет одного клиента.

Первичный ключ
Правило: каждая таблица имеет первичный ключ, состоящий из наименьшего возможного количества полей.

Первичный ключ может состоять из нескольких полей. К примеру, можно выбрать имя и фамилию в качестве первичного ключа (и надеяться, что эта комбинация будет уникальной всегда). Будет намного более хорошим выбором номер социального страхования в качестве первичного ключа, т.к. это единственное поле, которое уникальным образом идентифицирует человека.

Еще лучше, когда нет очевидного кандидата на звание первичного ключа, создается суррогатный первичный ключ в виде числового автоинкрементного поля.

Атомарность

Правило: поля не имеют дубликатов в каждой записи и каждое поле содержит только одно значение.

Например, сайт коллекционеров автомобилей, на котором каждый коллекционер может зарегистрировать его автомобили. Таблица ниже хранит информацию о зарегистрированных автомобилях.

Таблица 1 - Горизонтальное дублирование данных – плохая практика.

id	first_name	last_name	car1	car2	car3	car4	car5
1	paul	johnson	mitsubishi	(NULL)	(NULL)	paul	johnson
2	frank	black	subaru imp	daihatsu	(NULL)	(NULL)	(NULL)
3	robert	smith	Mercedes S	Ferrari f	Maserati	Toyota Pr	Spyker C8
4	john	bonham	Mazda 626	(NULL)	(NULL)	(NULL)	(NULL)

С таким вариантом проектирования можно сохранить только пять автомобилей и если их менее 5, то тратится впустую свободное место в базе данных на хранение пустых ячеек.

Другим примером плохой практики при проектировании является хранение множественных значений в ячейке.

Таблица 2 - Множественные значения в одной ячейке.

id	first_name	last_name	cars
1	john	bonham	Mazda 626
2	robert	smith	Mercedes SL450,Ferrari f40, Maserati...
3	frank	black	subaru impreza, daihatsu cuore
4	paul	johnson	mitsubishi lancer

Верным решением в данном случае будет выделение автомобилей в отдельную таблицу и использование внешнего ключа, который ссылается на эту таблицу.

Порядок записей не должен иметь значение

Правило: порядок записей таблицы не должен иметь значения.

Разработчик может быть склонен использовать порядок записей в таблице клиентов для определения того, какой из клиентов зарегистрировался первым. Для этих целей лучше создать поля даты и времени регистрации клиентов. Порядок записей будет неизбежно меняться, когда клиенты будут удаляться, изменяться или добавляться. Вот почему никогда не следует полагаться на порядок записей в таблице.

2 Вторая нормальная форма

Для того, чтобы база данных была нормализована согласно второй нормальной форме, она должна быть нормализована согласно первой нормальной форме. Вторая нормальная форма связана с избыточностью данных.

Избыточность данных

Правило: поля с не первичным ключом не должны быть зависимы от первичного ключа.

Может звучать немного заумно. А означает это то, что можно хранить в таблице только данные, которые напрямую связаны с ней и не имеют отношения к другой сущности. Следование

второй нормальной форме – это вопрос нахождения данных, которые часто дублируются в записях таблицы и которые могут принадлежать другой сущности.

Таблица 3 - Дублирование данных среди записей в поле store.

car_id	brand	type	color	store	price
1	Maserati	Quattropor	black	Amsterdam South	203000
2	Lada	1118	yellow	Amsterdam South	150
3	Volkswagen	Golf	green	Amsterdam North	14800
4	Volkswagen	Polo	black	Amsterdam West	10200
5	Jaguar	e type	green	The Hague	82399
6	Jaguar	e type	blue	The Hague	22374

Таблица выше может принадлежать компании, которая продает автомобили и имеет несколько магазинов в Нидерландах.

Если посмотреть на эту таблицу, то можно увидеть множественные примеры дублирования данных среди записей. Поле brand могло бы быть выделено в отдельную таблицу. Также, как и поле type (модель), которое также могло бы быть выделено в отдельную таблицу, которая бы имела связь многие-к-одному с таблицей brand потому, что у бренда могут быть разные модели.

Колонка store содержит наименование магазина, в котором в настоящее время находится машина. Store – это очевидный пример избыточности данных и хороший кандидат для отдельной сущности, которая должна быть связана с таблицей автомобилей связью по внешнему ключу.

Ниже пример того, как бы можно смоделировать базу данных для автомобилей, избегая избыточности данных.

car_id	type	color	store	price
1	5	black	1	203000
2	2	yellow	1	150
3	3	green	3	14800
4	4	black	2	10200
5	1	green	4	82399
6	1	blue	4	22374

type_id	brand_id	name
1	3	e type
2	2	1118
3	4	Golf
4	4	Polo
5	1	Quattroporte s

brand_id	name	country_of_origin
1	Maserati	Italy
2	Lada	Russia
3	Jaguar	United Kingdom
4	Volkswagen	Germany

store_id	name	street	hou...	zip...	phone
1	Amsterdam South	Churchil	14	1079HA	020373
2	Amsterdam West	Mercator	27	1056 RT	020838
3	Amsterdam North	Buikslot	76	1031 AB	020387
4	The Hague	Neherstr	82	2491JJ	070387

В примере выше таблица car имеет внешний ключ – ссылку на таблицы type и store. Столбец brand исчез потому, что на бренд есть неявная ссылка через таблицу type. Когда есть ссылка на type, есть ссылка и на brand, т.к. type принадлежит brand.

Избыточность данных была существенным образом устранена из модели базы данных. Но это не окончательный вариант. В поле country_of_origin в таблице brand пока дубликатов нет потому, что есть только четыре бренда из разных стран. Внимательный разработчик базы данных должен выделить названия стран в отдельную таблицу country.

И даже сейчас нельзя удовлетвориться результатом потому, что могжно бы выделить поле color в отдельную таблицу.

Если планируется хранить огромное количество единиц автомобилей в системе, и разработчик хочет иметь возможность производить поиск по цвету (color), то было бы мудрым решением выделить цвета в отдельную таблицу так, чтобы они не дублировались.

Существует другой случай, когда можно захотеть выделить цвета в отдельную таблицу. Если необходимо позволить работникам компании вносить данные о новых автомобилях, чтобы они имели возможно выбирать цвет машины из заранее заданного списка. В этом случае нужно хранить все возможные цвета в базе данных. Даже если еще нет машин с таким цветом, чтобы эти цвета присутствовали в базе данных, и работники могли их выбирать. Это определенно тот случай, когда нужно выделить цвета в отдельную таблицу.

3 Третья нормальная форма

Третья нормальная форма связана с транзитивными зависимостями. Транзитивные зависимости между полями базы данных существуют тогда, когда значения не ключевых полей зависят от значений других не ключевых полей. Чтобы база данных была в третьей нормальной форме, она должна быть во второй нормальной форме.

Транзитивные зависимости

Правило: не может быть транзитивных зависимостей между полями в таблице.

Таблица клиентов (мои клиенты – игроки немецкой и французской футбольной команды) ниже содержит транзитивные зависимости.

client_id	first_name	last_name	province	city	postal_code
23	Khalid	Boulahrouz	Noord-Holland	Alkmaar	1825HH
24	ZinÉdine	Zidane	Noord-Holland	Langedijk	1834DK
25	Ruud	van Nistelrooy	Noord-Holland	Schermer	1844JJ
19	Phillip	Cocu	Noord-Holland	Heilo	1850WI

В этой таблице не все поля зависят исключительно от первичного ключа. Существует отдельная связь между полем postal_code и полями города (city) и провинции (province). В Нидерландах оба значение: город и провинция – определяются почтовым кодом, индексом. Таким образом, нет необходимости хранить город и провинцию в клиентской таблице. Если знать почтовый код, то можно знать город и провинцию.

Такую транзитивную зависимость следует избегать, чтобы модель базы данных была в третьей нормальной форме.

В данном случае устранение транзитивной зависимости из таблицы может быть достигнуто путем удаления полей города и провинции из таблицы и хранение их в отдельной таблице, содержащей почтовый код (первичный ключ), имя провинции и имя города. Получение

комбинации почтовый код-город-провинция для целой страны может быть весьма нетривиальным занятием.

Другим примером для применения третьей нормальной формы может служить (слишком) простой пример таблицы заказов интернет-магазина ниже.

order_number	total_ex_vat	total_inc_vat
23	13,44	16,00
24	34,70	41,30
25	543,44	645,00
19	34,50	41,06

НДС – это процент, который добавляется к цене продукта (19% в данной таблице). Это означает, что значение total_ex_vat может быть вычислено из значения total_inc_vat и vice versa. Вы должны хранить в таблице одно из этих значений, но не оба сразу. Вы должны возложить задачу вычисления total_inc_vat из total_ex_vat или наоборот на программу, которая использует базу данных.

Третья нормальная форма гласит, что нельзя хранить данные в таблице, которые могут быть получены из других (не ключевых) полей таблицы.

Порядок выполнения работы:

Используя метод нормальных форм спроектировать базу данных. Процесс проектирования должен быть представлен в форме отчета, показан процесс формирования таблиц под выделенные сущности и их последовательная нормализация методом нормальных форм

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.1 Основы хранения и обработки данных. Проектирование базы данных

Лабораторное занятие №4
Создание базы данных с использованием языка SQL (CREATE DATABASE, CREATE TABLE)

Цель: освоить операции создания баз данных.

Выполнив работу, вы будете уметь:

– создавать базы данных с использованием языка SQL/

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

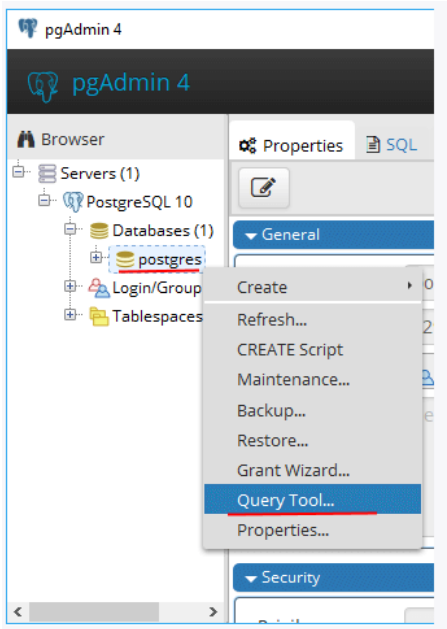
ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:
PostgreSQL pgAdmin, DBeaver

Задание:
Создать базу данных в СУБД PostgreSQL.

Краткие теоретические сведения:

Создание базы данных в среде PostgreSQL



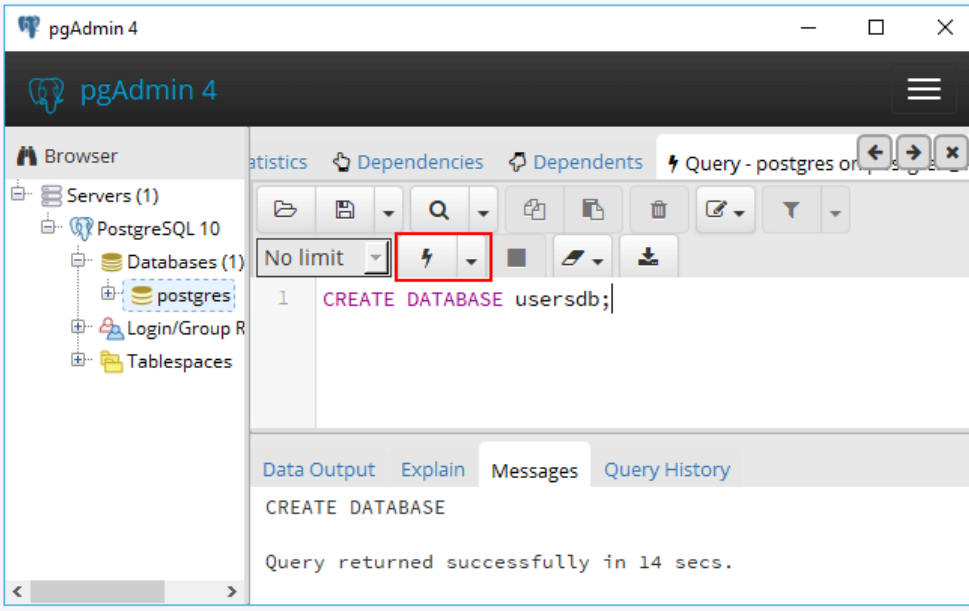
Для создания базы данных используется команда CREATE DATABASE, после которой указывается название базы данных.

Для выполнения запросов можно использовать графический клиент pgAdmin, хотя также можно использовать консольный клиент psql.

Чтобы создать новую базу, данных откроем pgAdmin. В левой части программы выберем какую-нибудь базу данных, например, стандартную бд postgres, и нажмем на нее правой кнопкой мыши.

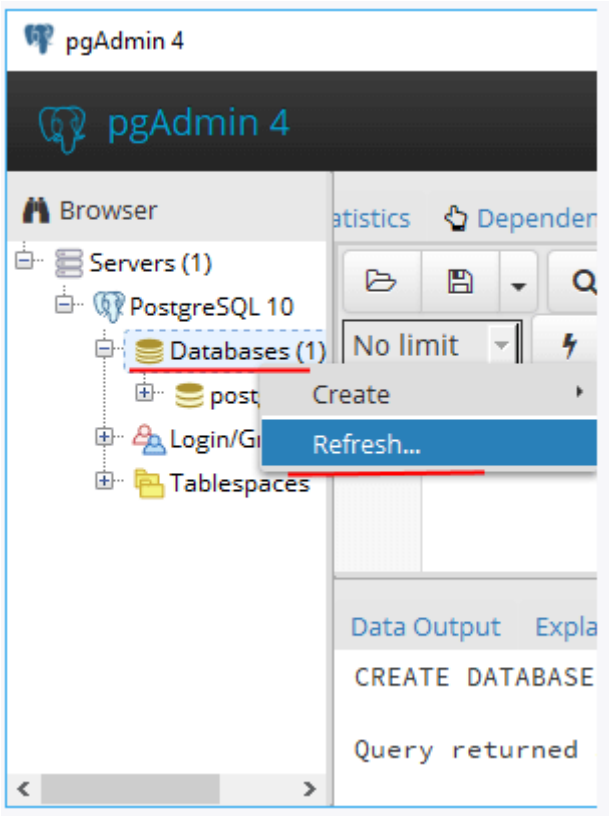
В появившемся меню выберем пункт Query Tool..., и в центральной части программы откроется поле для ввода кода SQL. В это поле введем следующий код:

CREATE DATABASE usersdb;

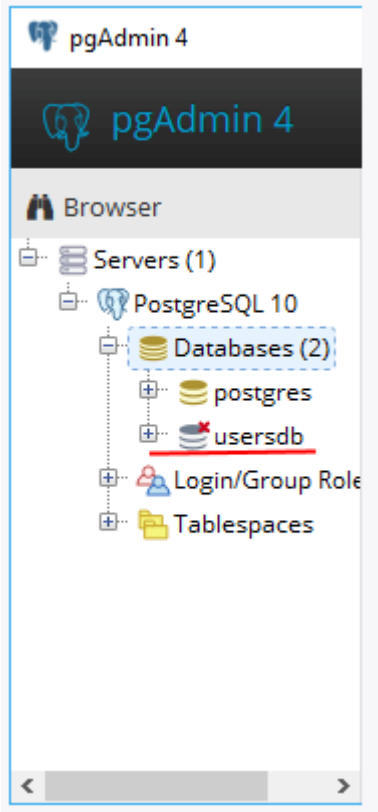


Для выполнения кода нажмем на значок молнии, и после этого будет создана база данных usersdb.

Чтобы увидеть нашу базу данных, нажмем в левой части на узел Databases правой кнопкой мыши и в контекстном меню выберем Refresh...:



Произойдет обновление, и мы увидим созданную базу данных.



По умолчанию база является неактивной, поэтому ее значок имеет серый цвет. Но чтобы к ней подключиться, достаточно нажать на нее и раскрыть ее узел.

Удаление базы данных

Для удаления базы данных применяется команда DROP DATABASE, после которой указывается название базы данных.

Удаляемая база данных должна быть неактивной, то есть подключение к ней должно быть закрыта.

Например, удаление базы данных usersdb:

DROP DATABASE usersdb;

Порядок выполнения работы:

Базу данных создаем одним из способов: с помощью pgAdmin или терминального клиента psql. Проверку создания базы данных и таблиц в ней проводим в терминального клиенте psql.

1. Запустите терминальный клиент для работы с PostgreSQL: **SQL shell (psql)**. Подключитесь к серверу и пройдите авторизацию (пароль - 12345).
2. Создайте новую базу данных с помощью команды: **CREATE DATABASE**. К имени базы данных добавьте аббревиатуру из первых букв своего ФИО.

Например, для Иванова Петра Сидоровича база данных будет иметь имя **storeIPS**:

postgres=# **CREATE DATABASE storeIPS**;

CREATE DATABASE

3. **Проверьте, что ваша база данных создана. Нужно будет сделать скриншоты для каждой таблицы и вставить в отчет.**

1 вариант: выборка из системного каталога **pg_database:**

SELECT datname FROM pg_database;

2 вариант: метакоманда **\l**

3 вариант: ключ **-l** командной строки приложения **psql**

4. **Создайте таблицы БД согласно ER-модели, указывая ОГРАНИЧЕНИЯ НА УРОВНЕ**

Синтаксис команды CREATE TABLE:

```
CREATE TABLE "имя_таблицы"
( имя_поля тип_данных [ограничения_целостности],
 имя_поля тип_данных [ограничения_целостности],
 ...
 имя_поля тип_данных [ограничения_целостности],
 [ограничение_целостности],
 [первичный_ключ],
 [внешний_ключ]
);
```

Для получения справки о синтаксисе SQL-команды:

\h CREATE TABLE

Способы ввода команд

• Способ 1

```
demo=# CREATE TABLE aircrafts (aircraft_code char(3) NOT NULL,
model text NOT NULL, range integer NOT NULL, CHECK (range > 0),
PRIMARY KEY (aircraft_code));
```

• Способ 2

```
demo=# CREATE TABLE aircrafts
demo-# (aircraft_code char(3) NOT NULL,
demo(# model text NOT NULL,
demo(# range integer NOT NULL,
demo(# CHECK (range > 0),
demo(# PRIMARY KEY (aircraft_code)
demo(# );
```

CREATE TABLE

Вместо ввода символа «;» команду можно завершить символами \g:

demo=# CREATE TABLE aircrafts ... \g

Прервать ввод команды можно клавишами Ctrl-C:

demo=# CREATE TABLE aircrafts
(aircraft_code char(3) NOT NULL,
demo(# ^C
demo=#

5. **Получите описание всех созданных таблиц с помощью метакоманды \d.**

\d aircrafts
Таблица "public.aircrafts"
Колонка | Тип | Модификаторы
---+---+---
aircraft_code | character(3) | NOT NULL
model | text | NOT NULL
range | integer | NOT NULL
Индексы:
"aircrafts_pkey" PRIMARY KEY, btree (aircraft_code)
Ограничения-проверки:
"aircrafts_range_check" CHECK (range > 0)
Примечания:
public означает имя так называемой схемы.

• Для реализации первичного ключа (PRIMARY KEY) всегда автоматически создается индекс. В данном случае тип индекса — btree, т. е. B-дерево.

Можно задать свои собственные имена для всех ограничений.

Удаление таблицы

Упрощенный синтаксис:
DROP TABLE имя_таблицы;
Например:
DROP TABLE aircrafts;

Форма представления результата:

Отчет по выполненной лабораторной работе.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.1 Основы хранения и обработки данных. Проектирование базы данных

Лабораторное занятие №5
Определение индексов для оптимизации запросов к базе данных

Цель: освоить принципы определения индексов для оптимизации запросов к базе данных.

Выполнив работу, вы будете уметь:

– оптимизировать запросы к базе данных, использовать индексы;
– анализировать план выполнения запроса.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:

PostgreSQL pgAdmin, DBeaver

Задание:

В созданной базе данных создать индексы для оптимизации запросов к базе.

Краткие теоретические сведения:

Индекс в базе данных — это структура данных, которая улучшает скорость операций выборки (SELECT) на таблицах за счет уменьшения объема данных, которые необходимо просмотреть. Индексы работают аналогично указателям в книге, позволяя быстро находить нужные записи.

Индексы нужны для:

- ускорение запросов: индексы значительно ускоряют выполнение запросов, особенно на больших объемах данных;
- улучшение производительности: они помогают оптимизировать операции поиска, сортировки и фильтрации;
- поддержка уникальности: индексы могут использоваться для обеспечения уникальности значений в столбцах.

Создание индексов

Индексы создаются с помощью команды CREATE INDEX:

```
CREATE INDEX имя_индекса ON имя_таблицы (имя_столбца);
```

Удаление индексов

Индексы можно удалить с помощью команды DROP INDEX:

```
DROP INDEX имя_индекса;
```

Порядок выполнения работы:

1. Подготовка среды

Подключитесь к PostgreSQL через psql:

```
psql -U postgres
```

Создайте новую базу данных:

```
CREATE DATABASE performance_db;
\c performance_db
```

2. Создание тестовой таблицы

Создадим таблицу users с большим количеством записей:

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
```

```
    username VARCHAR(50),
    email VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Заполним её 100 000 строк случайными данными:

```
INSERT INTO users (username, email)
SELECT
    'user_' || i,
    'user' || i || '@example.com'
FROM generate_series(1, 100000) AS i;
```

3. Анализ производительности без индекса

Выполним медленный запрос:

```
EXPLAIN ANALYZE
SELECT * FROM users WHERE username = 'user_99999';
```

Вы увидите, что используется Seq Scan — последовательное сканирование всей таблицы. Это неэффективно!

4. Создание индекса

Создадим индекс по полю username:

```
CREATE INDEX idx_username ON users(username);
```

Теперь повторим тот же запрос:

```
EXPLAIN ANALYZE
SELECT * FROM users WHERE username = 'user_99999';
```

Теперь используется Index Scan — гораздо быстрее!

5. Составной индекс

Если часто фильтруете по двум полям, можно создать составной индекс:

```
CREATE INDEX idx_username_email ON users(username, email);
```

Пример использования:

```
EXPLAIN ANALYZE
SELECT * FROM users WHERE username = 'user_99999' AND email = 'user99999@example.com';
```

6. Уникальный индекс

Чтобы обеспечить уникальность значения (например, email), создайте уникальный индекс:

```
CREATE UNIQUE INDEX idx_unique_email ON users(email);
```

Попробуйте добавить дублирующий email:

```
INSERT INTO users (username, email) VALUES ('test', 'user1@example.com');
```

Получите ошибку: duplicate key value violates unique constraint.

7.: Удаление индекса

Если индекс больше не нужен:

```
DROP INDEX idx_username;
```

8. Анализ существующих индексов

Посмотреть все индексы в текущей базе данных:

```
SELECT
    indexname,
    tablename,
    indexdef
FROM pg_indexes
WHERE tablename = 'users';
```

Задание для самостоятельной работы

1. Создайте таблицу orders со следующими полями:
   - `order_id`,
   - `user_id` (внешний ключ на users),
   - `total_amount`,
   - `order_date`.
2. Добавьте 50 000 записей в таблицу orders.
3. Найдите самые популярные поля для фильтрации (например, `order_date`, `total_amount`) и создайте для них индексы.
4. Сравните время выполнения запросов до и после создания индексов:

```
SELECT * FROM orders WHERE order_date BETWEEN '2025-01-01' AND '2025-03-31';
```

5. Создайте составной индекс по полям `user_id` и `order_date`. Протестируйте его работу.
6. Напишите запрос, который использует этот индекс для группировки и агрегации:

```
SELECT user_id, COUNT(*) AS total_orders, SUM(total_amount) AS total_spent
FROM orders
GROUP BY user_id;
```

Форма представления результата:

Отчет по выполненной лабораторной работе.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.1 Основы хранения и обработки данных. Проектирование базы данных

Лабораторное занятие №6
Проектирование базы данных для хранения данных IoT (Интернет вещей) с учётом особенностей структуры

Цель: научиться проектировать и реализовывать реляционную базу данных в PostgreSQL для хранения данных, поступающих от устройств Интернета Вещей (IoT), учитывая особенности временных рядов, масштабируемости и разнообразия типов датчиков.

Выполнив работу, вы будете уметь:

– анализировать данные IoT и их структуру;
– создавать таблицы с учетом особенностей хранения временных данных.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:

PostgreSQL pgAdmin, DBeaver

Задание:

Спроектировать и реализовать реляционную базу данных в PostgreSQL для хранения данных, поступающих от устройств Интернета Вещей (IoT).

Краткие теоретические сведения:

IoT (Internet of Things) — это сеть физических объектов (устройств), оснащённых сенсорами, ПО и возможностью подключения к интернету. Эти устройства собирают и обмениваются данными.

Особенности данных IoT:

1. Высокая интенсивность записи – миллионы записей в секунду.
2. Временные ряды – данные связаны с метками времени.
3. Разнотипные данные – разные датчики передают разный тип информации.
4. Масштабируемость – требуется эффективное хранение и извлечение больших объёмов данных.
5. Агрегация – часто нужны средние значения, максимумы, минимумы за периоды.

Рекомендации по проектированию:

- Отделить метаданные устройств от временных данных.
- Использовать нормализованную модель для описания устройств и датчиков.
- Для хранения временных рядов можно использовать:
- Нормализованную таблицу sensor_readings.
- Денормализованную таблицу для высокой производительности.
- Расширение PostgreSQL TimescaleDB для автоматической шардинговой оптимизации.

Порядок выполнения работы:

1. Настройка PostgreSQL

Установка расширения TimescaleDB (опционально)

```
# Ubuntu
sudo apt install timescaledb-postgresql-15

# Затем активируйте расширение в нужной БД
```

2. Создание БД и подключение

```
-- Создаем новую базу данных
CREATE DATABASE iot_db;

-- Подключаемся к ней
```

```
\c iot_db
```

3. Создание таблиц

Таблица devices — информация об устройствах
Хранит метаданные о каждом устройстве.

```
CREATE TABLE devices (
    device_id UUID PRIMARY KEY,
    name VARCHAR(100),
    location VARCHAR(255),
    description TEXT,
    registered_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Комментарий: используем UUID, т.к. устройства могут регистрироваться из разных источников. location может быть географическим местоположением или комнатой, где установлено устройство.

Таблица sensors — описание сенсоров
Каждое устройство может иметь несколько сенсоров.

```
CREATE TABLE sensors (
    sensor_id SERIAL PRIMARY KEY,
    device_id UUID REFERENCES devices(device_id),
    type VARCHAR(50), -- например, "temperature", "humidity"
    unit VARCHAR(20), -- единицы измерения
    description TEXT
);
```

Комментарий: здесь мы связываем сенсоры с устройствами через внешний ключ device_id.

Таблица sensor_readings — показания сенсоров
Самый важный элемент — хранение временных рядов.

```
CREATE TABLE sensor_readings (
    reading_id BIGSERIAL PRIMARY KEY,
    sensor_id INT REFERENCES sensors(sensor_id),
    value DOUBLE PRECISION,
    measured_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Комментарий: это нормализованная структура. Каждая запись содержит ссылку на конкретный сенсор и значение, измеренное в момент времени.

4. Использование TimescaleDB (для оптимизации)
Если вы используете TimescaleDB , преобразуйте таблицу sensor_readings в гипертаблицу:

```
-- Активируем расширение
CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;

-- Преобразуем таблицу в гипертаблицу
SELECT create_hypertable('sensor_readings', 'measured_at');
```

Теперь PostgreSQL автоматически будет создавать чанки (chunks) по времени, что ускоряет работу с большими объемами временных данных.

5. Заполнение тестовыми данными
Добавим тестовое устройство, сенсор и несколько показаний.

```
-- Вставляем устройство
```

```
    INSERT INTO devices (device_id, name, location, description)
    VALUES ('a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11', 'Weather Station
1', 'Moscow, Red Square', 'Outdoor weather station');

    -- Вставляем сенсор
    INSERT INTO sensors (device_id, type, unit, description)
    VALUES   ('a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11',   'temperature',
'°C', 'Ambient temperature sensor');

    -- Вставляем показания
    INSERT INTO sensor_readings (sensor_id, value, measured_at)
    VALUES
    (1, 22.5, NOW() - INTERVAL '5 minutes'),
    (1, 22.7, NOW() - INTERVAL '4 minutes'),
    (1, 23.0, NOW() - INTERVAL '3 minutes'),
    (1, 22.8, NOW() - INTERVAL '2 minutes'),
    (1, 23.1, NOW() - INTERVAL '1 minute');
```

6. Запросы и анализ

Последние 5 значений температуры:

```
SELECT s.type, sr.value, sr.measured_at
FROM sensor_readings sr
JOIN sensors s ON sr.sensor_id = s.sensor_id
WHERE s.type = 'temperature'
ORDER BY sr.measured_at DESC
LIMIT 5;
```

Среднее значение температуры за последние 10 минут:

```
SELECT AVG(value) AS avg_temp
FROM sensor_readings sr
JOIN sensors s ON sr.sensor_id = s.sensor_id
WHERE s.type = 'temperature'
AND sr.measured_at > NOW() - INTERVAL '10 minutes';
```

Группировка по 5-минутным интервалам:

```
SELECT
    time_bucket('5 minutes', sr.measured_at) AS five_min,
    ROUND(AVG(sr.value), 2) AS avg_value
FROM sensor_readings sr
JOIN sensors s ON sr.sensor_id = s.sensor_id
WHERE s.type = 'temperature'
GROUP BY five_min
ORDER BY five_min;
```

Если используется TimescaleDB, функция time_bucket() позволяет удобно агрегировать временные данные.

7. Оптимизация

1. Индексы :

```
CREATE INDEX idx_sensor_id ON sensor_readings(sensor_id);
CREATE INDEX idx_measured_at ON sensor_readings(measured_at);
```

2. Денормализация (для повышения скорости): можно создать представление или материализованное представление с предварительно подготовленными данными.

3. Партиционирование : PostgreSQL поддерживает партиционирование по времени или диапазонам, что полезно при работе с очень большими объемами данных.

4. JSONB поля : ИНОГДА IoT-устройства передают сложные структуры данных. PostgreSQL позволяет хранить такие данные в JSONB-полях.

Задание для самостоятельной работы

1. Реализуйте аналогичную систему для нескольких типов сенсоров: температура, влажность, давление.
2. Добавьте возможность хранения GPS-координат устройств.

Форма представления результата:

Отчет по выполненной лабораторной работе.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.2 Разработка и администрирование базы данных

Лабораторное занятие №7
Создание таблиц с использованием языка SQL (CREATE DATABASE, CREATE TABLE)

Цель: научиться создавать базы данных и таблицы в PostgreSQL с помощью команд SQL CREATE DATABASE и CREATE TABLE, учитывая типы данных, ограничения и логическую структуру базы данных.

Выполнив работу, вы будете уметь:

– проектировать и создавать таблицы с заданными полями;

– применять типы данных и ограничения (constraints) при создании таблиц.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.2 Осуществлять процедуры администрирования баз данных.

ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:

PostgreSQL pgAdmin, DBeaver

Задание:

Создать базу данных и таблицы в PostgreSQL с помощью команд SQL CREATE DATABASE и CREATE TABLE, учитывая типы данных, ограничения и логическую структуру базы данных.

Краткие теоретические сведения:

SQL (Structured Query Language) — это язык запросов, используемый для взаимодействия с реляционными базами данных. Он позволяет создавать, изменять, запрашивать и управлять данными в БД.

Основные команды DDL (Data Definition Language):
- CREATE DATABASE — создание новой базы данных.
- CREATE TABLE — создание новой таблицы.
- ALTER TABLE — изменение существующей таблицы.
- DROP TABLE / DATABASE — удаление таблицы или базы данных.

Команда CREATE DATABASE

Используется для создания новой базы данных:

CREATE DATABASE имя_базы;

Команда CREATE TABLE

Синтаксис:

```
CREATE TABLE имя_таблицы (
   столбец1 тип_данных ограничения,
   столбец2 тип_данных ограничения,
   ...
);
```

Типы данных в PostgreSQL:
- SERIAL — автоматически увеличивающееся целое число (ID).
- INT — целое число.
- VARCHAR(n) — строка переменной длины до n символов.
- TEXT — текст неограниченной длины.
- DATE — дата.
- TIMESTAMP — дата и время.
- BOOLEAN — логическое значение (TRUE/FALSE).
- DECIMAL(p, s) — десятичное число с точностью p и масштабом s.
- JSONB — бинарный JSON.

Ограничения (Constraints):
- PRIMARY KEY — уникальный идентификатор строки.
- FOREIGN KEY — ссылка на первичный ключ другой таблицы.
- NOT NULL — поле не может быть пустым.
- UNIQUE — все значения в столбце должны быть уникальными.
- CHECK(условие) — проверка условия.
- DEFAULT — значение по умолчанию.

Порядок выполнения работы:

1. Подключение к PostgreSQL

Откройте psql или клиент, например, pgAdmin .

---

2. Создание новой базы данных

Подключитесь через psql:

```
psql -U postgres
```

Создайте новую базу данных:

```
-- Создаем базу данных
CREATE DATABASE school_db;

-- Подключаемся к ней
\c school_db
```

Теперь все дальнейшие команды будут выполняться в контексте этой БД.

---

3. Создание таблиц

Пример 1: Таблица students — студенты

```
CREATE TABLE students (
```

```
    student_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    birth_date DATE,
    email VARCHAR(100) UNIQUE,
    is_active BOOLEAN DEFAULT TRUE
);
```

Комментарий:

- student_id — уникальный идентификатор студента.
- first_name, last_name — обязательные поля.
- birth_date — необязательное поле.
- email — должен быть уникальным.
- is_active — значение по умолчанию TRUE.

Пример 2: Таблица courses — курсы

```
CREATE TABLE courses (
    course_id SERIAL PRIMARY KEY,
    course_name VARCHAR(100) NOT NULL,
    description TEXT,
    duration_weeks INT CHECK (duration_weeks > 0),
    start_date DATE
);
```

Комментарий:

- Поле duration_weeks должно быть положительным числом благодаря CHECK.

Пример 3: Таблица enrollments — зачисления студентов на курсы

```
CREATE TABLE enrollments (
    enrollment_id SERIAL PRIMARY KEY,
    student_id INT REFERENCES students(student_id),
    course_id INT REFERENCES courses(course_id),
    enrollment_date DATE DEFAULT CURRENT_DATE,
    grade DECIMAL(4, 2) CHECK (grade BETWEEN 0 AND 5)
);
```

Комментарий:

- Здесь используются внешние ключи (REFERENCES) для связи между таблицами.
- Оценка (grade) должна быть от 0 до 5.

---

4. Вставка тестовых данных

Добавим несколько записей:

```
-- Добавляем студентов
INSERT INTO students (first_name, last_name, birth_date, email)
VALUES ('Иван', 'Иванов', '2000-05-10', 'ivan@example.com'),
       ('Мария', 'Петрова', '2001-08-22', 'maria@example.com');

-- Добавляем курсы
INSERT INTO courses (course_name, description, duration_weeks,
start_date)
VALUES ('Введение в программирование', 'Основы программирования
на Python', 12, '2025-02-01'),
       ('Английский для IT', 'Развитие технической грамотности на
английском', 16, '2025-02-05');
```

```
-- Регистрируем зачисления
INSERT INTO enrollments (student_id, course_id)
VALUES (1, 1), (1, 2), (2, 1);
```

---

5. Выполнение простых запросов

Посмотреть всех студентов:

```
SELECT * FROM students;
```

Посмотреть активных студентов:

```
SELECT first_name, last_name
FROM students
WHERE is_active = TRUE;
```

Посмотреть оценки студентов по курсам:

```
SELECT s.first_name, s.last_name, c.course_name, e.grade
FROM enrollments e
JOIN students s ON e.student_id = s.student_id
JOIN courses c ON e.course_id = c.course_id;
```

Задание для самостоятельной работы

1. Создайте новую таблицу teachers со следующими полями:
- teacher_id — уникальный ID (первичный ключ),
- full_name — ФИО преподавателя,
- subject — предмет,
- hire_date — дата найма.

2. Добавьте связь между courses и teachers (например, один преподаватель может вести один курс).

3. Создайте таблицу grades_log, которая будет хранить историю изменения оценок студентов:
- log_id,
- enrollment_id,
- old_grade,
- new_grade,
- changed_at TIMESTAMP DEFAULT NOW().

4. Напишите триггер, который будет добавлять запись в grades_log при обновлении оценки в enrollments

Форма представления результата:

Отчет по выполненной лабораторной работе.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.2 Разработка и администрирование базы данных

Лабораторное занятие №8
Реализация ограничений целостности (PRIMARY KEY, FOREIGN KEY, UNIQUE) в таблицах базы данных

Цель: научиться реализовывать и использовать основные механизмы обеспечения целостности данных в PostgreSQL.

Выполнив работу, вы будете уметь:

– применять ограничения при проектировании структуры базы данных;

– обнаруживать и исправлять ошибки, вызванные нарушением ограничений.

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.2 Осуществлять процедуры администрирования баз данных.

ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:

PostgreSQL pgAdmin, DBeaver

Задание:

Реализовать и использовать основные механизмы обеспечения целостности данных в PostgreSQL.

Краткие теоретические сведения:

Целостность данных — это обеспечение точности, полноты и согласованности данных в базе данных. Для этого используются ограничения (constraints).

Типы данных сами по себе ограничивают множество данных, которые можно сохранить в таблице. Однако для многих приложений такие ограничения слишком грубые. Например, столбец, содержащий цену продукта, должен, вероятно, принимать только положительные значения. Но такого стандартного типа данных нет. Возможно, необходимо ограничить данные столбца по отношению к другим столбцам или строкам. Например, в таблице с информацией о товаре должна быть только одна строка с определённым кодом товара.

Для решения подобных задач SQL позволяет определять ограничения для столбцов и таблиц. Ограничения дают возможность управлять данными в таблицах так, как нужно. Если пользователь попытается сохранить в столбце значение, нарушающее ограничения, возникнет ошибка. Ограничения будут действовать, даже если это значение по умолчанию.

Ограничения столбцов и таблиц

При определении таблиц и их столбцов в SQL мы можем использовать ряд атрибутов, которые накладывают определенные ограничения. Рассмотрим эти атрибуты.

PRIMARY KEY

С помощью выражения **PRIMARY KEY** столбец можно сделать первичным ключом.

```
CREATE TABLE Customers
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(30),
    LastName CHARACTER VARYING(30),
    Email CHARACTER VARYING(30),
    Age INTEGER
)
```

Первичный ключ уникально идентифицирует строку в таблице. В качестве первичного ключа необязательно должны выступать столбцы с типом `SERIAL`, они могут представлять любой другой тип.

Установка первичного ключа на уровне таблицы:

```
CREATE TABLE Customers
(
    Id SERIAL,
    FirstName CHARACTER VARYING(30),
    LastName CHARACTER VARYING(30),
    Email CHARACTER VARYING(30),
    Age INTEGER,
    PRIMARY KEY(Id)
);
```

Первичный ключ может быть составным (compound key). Такой ключ может потребоваться, если у нас сразу два столбца должны уникально идентифицировать строку в таблице. Например:

```
CREATE TABLE OrderLines
(
    OrderId INTEGER,
    ProductId INTEGER,
    Quantity INTEGER,
    Price MONEY,
    PRIMARY KEY(OrderId, ProductId)
);
```

Здесь поля OrderId и ProductId вместе выступают как составной первичный ключ. То есть в таблице OrderLines не может быть двух строк, где для обоих из этих полей одновременно были бы одни и те же значения.

UNIQUE

Если мы хотим, чтобы столбец имел только уникальные значения, то для него можно определить атрибут **UNIQUE**.

```
CREATE TABLE Customers
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(20),
    LastName CHARACTER VARYING(20),
    Email CHARACTER VARYING(30) UNIQUE,
    Phone CHARACTER VARYING(30) UNIQUE,
    Age INTEGER
);
```

В данном случае столбцы, которые представляют электронный адрес и телефон, будут иметь уникальные значения. И мы не сможем добавить в таблицу две строки, у которых значения для этих столбцов будет совпадать.

Также мы можем определить этот атрибут на уровне таблицы:

```
CREATE TABLE Customers
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(20),
    LastName CHARACTER VARYING(20),
    Email CHARACTER VARYING(30),
    Phone CHARACTER VARYING(30),
    Age INTEGER,
    UNIQUE(Email, Phone)
);
```

Или так:

```
CREATE TABLE Customers
(
```

```
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(20),
    LastName CHARACTER VARYING(20),
    Email CHARACTER VARYING(30),
    Phone CHARACTER VARYING(30),
    Age INTEGER,
    UNIQUE(Email),
    UNIQUE(Phone)
);
```

NULL и NOT NULL

Чтобы указать, может ли столбец принимать значение NULL, при определении столбца ему можно задать атрибут **NULL** или **NOT NULL**. Если этот атрибут явным образом не будет использован, то по умолчанию столбец будет допускать значение NULL. Исключением является тот случай, когда столбец выступает в роли первичного ключа - в этом случае по умолчанию столбец имеет значение NOT NULL.

```
CREATE TABLE Customers
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(20) NOT NULL,
    LastName CHARACTER VARYING(20) NOT NULL,
    Age INTEGER
);
```

DEFAULT

Атрибут **DEFAULT** определяет значение по умолчанию для столбца. Если при добавлении данных для столбца не будет предусмотрено значение, то для него будет использоваться значение по умолчанию.

```
CREATE TABLE Customers
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(20),
    LastName CHARACTER VARYING(20),
    Age INTEGER DEFAULT 18
);
```

Здесь для столбца Age предусмотрено значение по умолчанию 18.

CHECK

Ключевое слово **CHECK** задает ограничение для диапазона значений, которые могут храниться в столбце. Для этого после слова CHECK указывается в скобках условие, которому должен соответствовать столбец или несколько столбцов. Например, возраст клиентов не может быть меньше 0 или больше 100:

```
CREATE TABLE Customers
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(20),
    LastName CHARACTER VARYING(20),
    Age INTEGER DEFAULT 18 CHECK(Age >0 AND Age < 100),
    Email CHARACTER VARYING(30) UNIQUE CHECK(Email !=''),
    Phone CHARACTER VARYING(20) UNIQUE CHECK(Phone !='')
);
```

Здесь также указывается, что столбцы Email и Phone не могут иметь пустую строку в качестве значения (пустая строка **не** эквивалентна значению NULL).

Для соединения условий используется ключевое слово **AND**. Условия можно задать в виде операций сравнения больше (>), меньше (<), не равно (!=).

Также с помощью CHECK можно создать ограничение в целом для таблицы:

```
CREATE TABLE Customers
(
    Id SERIAL PRIMARY KEY,
    Age INTEGER DEFAULT 18,
    FirstName CHARACTER VARYING(20),
    LastName CHARACTER VARYING(20),
    Email CHARACTER VARYING(30) UNIQUE,
    Phone CHARACTER VARYING(20) UNIQUE,
    CHECK((Age >0 AND Age<100) AND (Email !='') AND (Phone !=''))
);
```

Оператор CONSTRAINT. Установка имени ограничений.

С помощью ключевого слова **CONSTRAINT** можно задать имя для ограничений. В качестве ограничений могут использоваться PRIMARY KEY, UNIQUE, CHECK.

Имена ограничений можно задать на уровне столбцов. Они указываются после CONSTRAINT перед атрибутами:

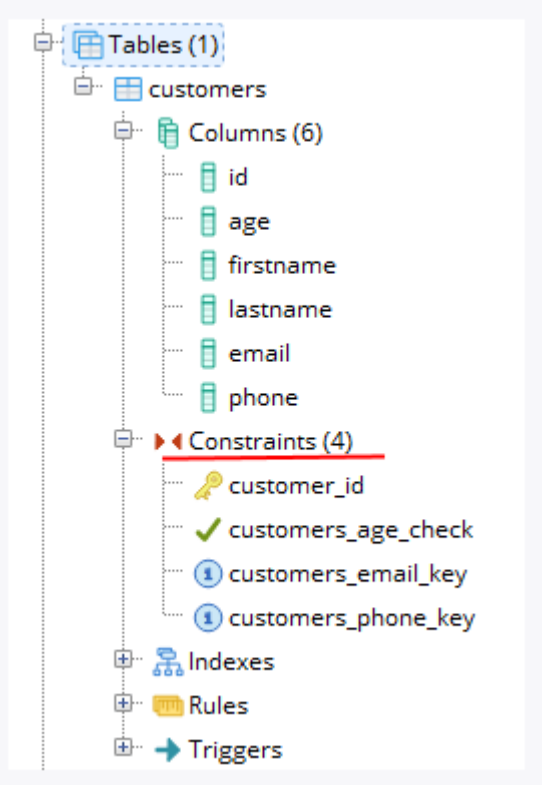
```
CREATE TABLE Customers
(
    Id SERIAL CONSTRAINT customer_Id PRIMARY KEY,
    Age INTEGER CONSTRAINT customers_age_check CHECK(Age >0 AND
Age < 100),
    FirstName CHARACTER VARYING(20) NOT NULL,
    LastName CHARACTER VARYING(20) NOT NULL,
    Email CHARACTER VARYING(30) CONSTRAINT customers_email_key
UNIQUE,
    Phone CHARACTER VARYING(20) CONSTRAINT customers_phone_key
UNIQUE
);
```

В принципе необязательно задавать имена ограничений, при установке соответствующих атрибутов SQL Server автоматически определяет их имена. Но, зная имя ограничения, мы можем к нему обращаться, например, для его удаления.

И также можно задать все имена ограничений через атрибуты таблицы:

```
CREATE TABLE Customers
(
    Id SERIAL,
    Age INTEGER,
    FirstName CHARACTER VARYING(20) NOT NULL,
    LastName CHARACTER VARYING(20) NOT NULL,
    Email CHARACTER VARYING(30),
    Phone CHARACTER VARYING(20),
    CONSTRAINT customer_Id PRIMARY KEY(Id),
    CONSTRAINT customers_age_check CHECK(Age >0 AND Age < 100),
    CONSTRAINT customers_email_key UNIQUE(Email),
    CONSTRAINT customers_phone_key UNIQUE(Phone)
);
```

Вне зависимости от того, используется оператор CONSTRAINT для создания ограничений или нет (в этом случае при установке ограничений PostgreSQL сам дает им имена), мы можем просмотреть все ограничения в pgAdmin в узле базы данных в подузле :



Порядок выполнения работы:

1. Подготовка среды

Подключитесь к PostgreSQL через psql:
psql -U postgres
Создайте новую базу данных:
CREATE DATABASE integrity_db;
\c integrity_db

2. Создание таблиц с ограничениями

Таблица users — пользователи системы

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Комментарий:

- user_id — уникальный идентификатор пользователя.
- username и email — должны быть уникальны и не пустыми.
- created_at — автоматически заполняется датой создания.

Таблица orders — заказы пользователей

```
CREATE TABLE orders (
    order_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(user_id),
    product_name VARCHAR(100) NOT NULL,
    amount NUMERIC(10, 2) CHECK (amount > 0),
    order_date DATE DEFAULT CURRENT_DATE
);
```

Комментарий:

- user_id — внешний ключ, ссылающийся на таблицу users.
- amount — должно быть положительным числом (CHECK(amount > 0)).

Таблица emails — дополнительная проверка UNIQUE

```
CREATE TABLE emails (
    id SERIAL PRIMARY KEY,
    email VARCHAR(100) UNIQUE NOT NULL
);
```

---

3. Попытка нарушить ограничения

После создания таблиц попробуем нарушить каждое из ограничений и посмотрим, как отреагирует система.

Нарушение PRIMARY KEY

```
INSERT INTO users (user_id, username, email)
VALUES (1, 'test_user', 'test@example.com');

-- Попытаемся вставить еще одну запись с тем же user_id
INSERT INTO users (user_id, username, email)
VALUES (1, 'another_user', 'another@example.com');
```

● Ошибка: duplicate key value violates unique constraint.

Нарушение FOREIGN KEY

```
-- Попытаемся создать заказ для несуществующего пользователя
INSERT INTO orders (user_id, product_name, amount)
VALUES (999, 'Laptop', 1200.00);
```

● Ошибка: insert or update on table "orders" violates foreign key constraint.

Нарушение UNIQUE

```
-- Вставляем email
INSERT INTO emails (email) VALUES ('john@example.com');

-- Попытка вставить тот же email
INSERT INTO emails (email) VALUES ('john@example.com');
```

● Ошибка: duplicate key value violates unique constraint.

Нарушение CHECK

```
-- Попытка вставить отрицательную сумму
INSERT INTO orders (user_id, product_name, amount)
VALUES (1, 'Mouse', -50.00);
```

● Ошибка: new row for relation "orders" violates check constraint.

---

4. Работа с существующими таблицами

Добавление ограничения после создания таблицы

Можно добавить ограничение UNIQUE к уже существующей таблице:

```
ALTER TABLE users ADD CONSTRAINT unique_username
UNIQUE(username);
```

Удаление ограничения

```
ALTER TABLE users DROP CONSTRAINT unique_username;
```

Переименование ограничения

```
ALTER TABLE users RENAME CONSTRAINT users_email_key TO
unique_email;
```

---

5. Использование составного PRIMARY KEY

Иногда требуется использовать несколько полей в качестве первичного ключа.

```
CREATE TABLE user_roles (
    user_id INT REFERENCES users(user_id),
    role VARCHAR(50),
    assigned_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (user_id, role)
);
```

Теперь комбинация user_id и role должна быть уникальной.

Задание для самостоятельной работы

1. Создайте таблицу products со следующими полями:
- product_id (первичный ключ),
- name (уникальное, не пустое),
- price (больше нуля),
- category (не пустое).

2. Добавьте внешний ключ в таблицу orders, который связывает её с products.
3. Создайте таблицу employees со следующими полями:
- employee_id,
- full_name,
- position,
- salary (больше 0),
- department_id.

Установите ограничения:
- employee_id — PRIMARY KEY,
- full_name — UNIQUE и NOT NULL,
- salary — CHECK больше 0.

4. Попробуйте вручную нарушить каждое из этих ограничений и зафиксируйте возникающие ошибки.
5. Напишите запросы, которые:
- Выводят всех сотрудников с зарплатой выше средней,
- Выводят количество заказов по каждому продукту.

Форма представления результата:

Отчет по выполненной лабораторной работе.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.2 Разработка и администрирование базы данных

Лабораторное занятие №9
Написание и выполнение SQL-запросов для добавления, изменения и удаления данных (INSERT, UPDATE, DELETE)

Цель: научиться выполнять операции добавления, обновления и удаления записей в таблицах базы данных с помощью команд языка SQL.

Выполнив работу, вы будете уметь:
– писать и выполнять SQL-запросы для манипулирования данными.

Выполнение лабораторной работы способствует формированию:
ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.
ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.
ПК 2.2 Осуществлять процедуры администрирования баз данных.
ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:
PostgreSQL pgAdmin, DBeaver

Задание:
В созданной базе данных выполнить запросы добавления, обновления и удаления данных к базе.

Краткие теоретические сведения:
Основные DML-команды SQL:

Команда	Назначение
INSERT	Добавляет новые строки в таблицу
UPDATE	Обновляет существующие строки
DELETE	Удаляет строки из таблицы

1. Команда INSERT
Добавляет новые данные в таблицу.

Синтаксис:
```
INSERT INTO имя_таблицы (столбец1, столбец2, ...)
VALUES (значение1, значение2, ...);
```

Можно добавить несколько строк за раз:
```
INSERT INTO имя_таблицы (столбец1, столбец2)
VALUES
(значение1, значение2),
(значение3, значение4);
```

2. Команда UPDATE
Изменяет значения существующих строк.

Синтаксис:
```
UPDATE имя_таблицы
SET столбец1 = значение1, столбец2 = значение2
WHERE условие;
```
Важно: всегда используйте WHERE, иначе будут изменены все строки в таблице!

3. Команда DELETE
Удаляет строки из таблицы.

Синтаксис:
```
DELETE FROM имя_таблицы
WHERE условие;
```
Важно: без WHERE будут удалены все данные из таблицы.

Порядок выполнения работы:

1. Подготовка среды
Подключитесь к PostgreSQL через psql или клиент (например, pgAdmin):
```
psql -U postgres
```

Создайте новую базу данных и подключитесь к ней:
```
CREATE DATABASE practice_db;
\c practice_db
```

2. Создание тестовой таблицы
Создадим таблицу students:

```
CREATE TABLE students (
    student_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    age INT CHECK (age >= 16),
    email VARCHAR(100) UNIQUE
);
```

3. Добавление данных — INSERT
Добавим несколько студентов:

```
INSERT INTO students (first_name, last_name, age, email)
VALUES
('Иван', 'Иванов', 20, 'ivan@example.com'),
('Мария', 'Петрова', 21, 'maria@example.com'),
('Алексей', 'Смирнов', 19, 'alex@example.com');
```

Проверим результат:
```
SELECT * FROM students;
```

4. Изменение данных — UPDATE
Обновим возраст одного студента:

```
UPDATE students
SET age = 22
WHERE first_name = 'Иван' AND last_name = 'Иванов';
```

Теперь обновим email:

```
UPDATE students
SET email = 'ivan_new@example.com'
```

```
WHERE student_id = 1;
```

5. Удаление данных — DELETE
Удалим студента по ID:

```
DELETE FROM students
WHERE student_id = 3;
```

Попробуем удалить всех студентов старше 21 года (для примера):

```
DELETE FROM students
WHERE age > 21;
```

6. Работа с несколькими таблицами (связанные данные)
Создадим таблицу courses:

```
CREATE TABLE courses (
    course_id SERIAL PRIMARY KEY,
    course_name VARCHAR(100) NOT NULL
);
```

Создадим связь через внешний ключ:

```
CREATE TABLE enrollments (
    enrollment_id SERIAL PRIMARY KEY,
    student_id INT REFERENCES students(student_id),
    course_id INT REFERENCES courses(course_id),
    enrollment_date DATE DEFAULT CURRENT_DATE
);
```

Добавим курсы:

```
INSERT INTO courses (course_name)
VALUES ('Программирование'), ('Английский язык');
```

Запишем студентов на курсы:

```
INSERT INTO enrollments (student_id, course_id)
VALUES (1, 1), (2, 2);
```

Обновим дату зачисления:

```
UPDATE enrollments
SET enrollment_date = '2025-03-01'
WHERE enrollment_id = 1;
```

Удалим запись о зачислении:

```
DELETE FROM enrollments
WHERE enrollment_id = 2;
```

Задание для самостоятельной работы

1. Создайте таблицу employees со следующими полями:
   - employee_id,
   - full_name,
   - position,
   - salary.

Добавьте туда не менее 5 записей.

2. Обновите должность одного сотрудника и увеличьте его зарплату на 10%.
3. Удалите сотрудника с самой высокой зарплатой.
4. Создайте таблицу `departments` и свяжите её с `employees` через внешний ключ `department_id`.
5. Напишите запросы:
   – Вывести всех сотрудников с зарплатой выше средней.
   – Обновить зарплату всем сотрудникам отдела "IT" на 15%.
   – Удалить всех сотрудников, чья зарплата меньше 30 000.

Форма представления результата:
Отчет по выполненной лабораторной работе.

Критерии оценки:
Оценка «отлично» ставится, если задание выполнено верно.
Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.
Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.
Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.2 Разработка и администрирование базы данных

Лабораторное занятие №10
Настройка индексов для оптимизации производительности запросов (CREATE INDEX)

Цель: познакомиться с механизмами ускорения работы SQL-запросов в PostgreSQL через использование индексов; научиться создавать, анализировать и использовать индексы при работе с большими объемами данных.

Выполнив работу, вы будете уметь:
– настраивать индексы для оптимизации производительности;
– оптимизировать медленные запросы с помощью индексации

Выполнение лабораторной работы способствует формированию:
ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.
ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.
ПК 2.2 Осуществлять процедуры администрирования баз данных.
ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:
PostgreSQL pgAdmin, DBeaver

Задание:
В созданной базе данных создать индексы для оптимизации запросов к базе.

Краткие теоретические сведения:
Типы индексов в PostgreSQL

B-tree: наиболее распространенный тип индекса, используемый по умолчанию. Подходит для большинства случаев. Используется для равенств и диапазонов (=,>,<,BETWEEN).

Hash: используется для быстрого поиска по равенству.

GIN (Generalized Inverted Index): подходит для массивов и полнотекстового поиска.

GiST (Generalized Search Tree): используется для сложных типов данных, таких как геометрические.

BRIN: для больших таблиц с упорядоченными данными.

Когда создавать индекс?

- при частых запросах на выборку по конкретным полям (например, WHERE, JOIN);
- при необходимости обеспечить уникальность (UNIQUE);
- при использовании внешних ключей;
- при группировке (GROUP BY) или сортировке (ORDER BY).

Возможные проблемы:

- индексы занимают место на диске;
- замедляют операции вставки, обновления и удаления;
- не всегда эффективны на маленьких таблицах;
- избыточные индексы могут ухудшить производительность.

Порядок выполнения работы:

Задание 1. Создание индексов по столбцам таблицы

1. Создайте индекс для ускорения поиска по наименованию категории.
2. Создайте индекс для ускорения поиска по клиентам. Индекс будет **составным** и будет включать в себя фамилию, имя и отчество клиента:
   - для фамилии сортировка по умолчанию, NULLS расположены в начале списка;
   - для имени сортировка по убыванию, NULLS расположены в конце списка;
   - для отчества сортировка по убыванию, NULLS расположены в начале списка.
3. Создайте индекс для ускорения поиска по цене товара с **сортировкой по возрастанию, NULLS расположены в начале списка списка**.

Задание 2. Создание индексов, используя выражения

1. Создайте индекс по общей стоимости товаров на складе с **сортировкой по убыванию, NULLS расположены в конце списка** (таблица **products**).
2. Создайте индекс по поиску клиента по полному ФИО (фамилия, имя и отчество разделяются пробелами) с **сортировкой по возрастанию, NULLS расположены в конце списка**. Для создания значения для индекса используйте:
   - конкатенацию строк - функции **concat()** или **concat_ws()** или оператор ||;
   - для обработки возможного значения **NULL** в поле patronymic (отчество) используйте функцию **coalesce()** - возвращает первое ненулевое значение из своих аргументов;
   - для удаления лишних пробелов слева и справа используйте функцию trim**()**.

   Пример:

   (trim(" || last_name || ' ' || first_name || ' ' || coalesce(patronymic, '')))
3. Создайте индекс по поиску клиента по телефону без каких-либо дополнительных символов кроме цифр (например, 89128971815 или 79128971815). Для того чтобы избавиться от “лишних” символов, используйте функцию **replace()**.

Задание 3. Создание уникальных индексов

1. Создайте индекс, который будет проверять уникальность наименований категорий товаров без учета регистра.
2. Создайте индекс, который будет проверять уникальность наименований товаров без учета регистра.

Задание 4. Сбор статистики

Сделав запрос к служебной таблице **pg_indexes**, выведите информацию о всех индексах в вашей базе данных (во всех схемах). Сделайте **сортировку по имени таблицы и названию индекса** (приложите скриншот результата).

Задание 5. Удаление индекса

1. Удалите любой индекс, созданный в задании 2.

Приведите пример запроса, который теперь будет выполняться без использования индекса.

Форма представления результата:
Отчет по выполненной лабораторной работе.

Критерии оценки:
Оценка «отлично» ставится, если задание выполнено верно.
Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.
Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.
Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.2 Разработка и администрирование базы данных

Лабораторное занятие №11
Оптимизация запросов к базе данных с использованием индексов и анализа плана выполнения запросов

Цель: научиться анализировать производительность SQL-запросов в PostgreSQL с помощью EXPLAIN ANALYZE, выявлять узкие места и оптимизировать их с помощью создания индексов.

Выполнив работу, вы будете уметь:
– использовать команду EXPLAIN ANALYZE для анализа плана выполнения запроса;
– создавать индексы для ускорения выборки, группировки и соединений таблиц

Выполнение лабораторной работы способствует формированию:
ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.
ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.
ПК 2.2 Осуществлять процедуры администрирования баз данных.
ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:
PostgreSQL pgAdmin, DBeaver

Задание:

В созданной базе данных создать индексы для оптимизации запросов к базе.

Краткие теоретические сведения:

Оптимизация запросов к базе данных — это процесс улучшения производительности базы данных путем повышения эффективности выполнения SQL-запросов и минимизации нагрузки на систему. Цель оптимизации состоит в сокращении времени отклика на запросы, уменьшении потребления ресурсов сервера баз данных и повышении общей пропускной способности системы.

Основные направления оптимизации запросов:

1. Индексация

Индексы позволяют быстро находить нужные данные в таблице, значительно ускоряя выполнение запросов SELECT, UPDATE и DELETE. Типичные типы индексов включают B-tree, хеш-индексы и полнотекстовые индексы.

Пример: создать индекс по полю name таблицы users.

```
CREATE INDEX idx_users_name ON users(name);
```

2. Нормализация таблиц

Нормализация позволяет избежать избыточности данных и улучшает целостность данных. Однако чрезмерная нормализация может привести к увеличению количества JOIN'ов, замедляя производительность сложных запросов.

Пример нормализации: разделение одной большой таблицы на две меньшие, содержащие взаимосвязанные данные.

3. Выбор правильного типа данных

Использование наиболее подходящего типа данных для каждого поля помогает уменьшить объем хранимых данных и ускорить операции чтения/записи.

Пример: использовать тип INT вместо VARCHAR для численных значений.

4. Использование EXPLAIN ANALYZE

Команда EXPLAIN ANALYZE показывает, как именно выполняется запрос и сколько времени занимает каждая операция. Это полезно для выявления узких мест и неэффективных участков.

Пример: проверка плана выполнения запроса.

```
EXPLAIN ANALYZE SELECT * FROM users WHERE name = 'Иван';
```

Вывод может содержать такие элементы:

Seq Scan – последовательное сканирование таблицы;
Index Scan – использование индекса;
Rows – оценочное количество строк;
Width – размер строки в байтах;
Actual time – время выполнения (в мс).

5. Кэширование результатов

Кэширование часто используемых результатов запросов снижает нагрузку на базу данных и ускоряет доступ к данным.

Пример: настройка кэширования на уровне приложения или использование встроенных механизмов кэша базы данных (например, Redis).

6. Ограничение объема выборки

Ограничивайте количество возвращаемых записей с помощью LIMIT и OFFSET. Если приложение отображает данные постранично, используйте соответствующие конструкции для ограничения числа строк.

Пример: выбрать первые 10 пользователей.

```
SELECT * FROM users ORDER BY id LIMIT 10;
```

7. Избегайте операций с большими объёмами данных

Избегайте сортировки больших объемов данных на стороне базы данных. Если возможно, производите предварительную фильтрацию и агрегацию данных перед передачей клиенту.

Пример: минимизировать передачу данных через сеть.

```
SELECT COUNT(*) FROM users WHERE active = true;
```

8. Регулярное обновление статистики

Для эффективной работы индекса и оптимального выбора планов выполнения запросов важно регулярно обновлять статистику распределения данных в таблицах.

Пример: обновление статистики PostgreSQL.

```
ANALYZE users;
```

Порядок выполнения работы:

1. Подготовка среды

Подключитесь к PostgreSQL через psql:

```
psql -U postgres
```

Создайте новую базу данных:

```
CREATE DATABASE optimization_db;
\c optimization_db
```

2. Создание тестовой таблицы

Создадим таблицу users с большим количеством записей:

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(50),
    email VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Заполним её 100 000 случайных записей:

```
INSERT INTO users (username, email)
SELECT
    'user_' || i,
    'user' || i || '@example.com'
FROM generate_series(1, 100000) AS i;
```

3. Анализ медленного запроса

Выполним простой запрос и посмотрим его план:

```
EXPLAIN ANALYZE
SELECT * FROM users WHERE username = 'user_99999';
```

Вы увидите, что используется Seq Scan — последовательное сканирование всей таблицы.

4. Создание индекса

Добавим индекс по полю username:

```
CREATE INDEX idx_username ON users(username);
```

Теперь повторим тот же запрос:

```
EXPLAIN ANALYZE
SELECT * FROM users WHERE username = 'user_99999';
```

Теперь вы должны увидеть Index Scan и значительное снижение времени выполнения.

5. Составной индекс

Если часто фильтруете по нескольким полям, создайте составной индекс:

```
CREATE INDEX idx_username_email ON users(username, email);
```

Пример использования:

```
EXPLAIN ANALYZE
SELECT * FROM users WHERE username = 'user_99999' AND email = 'user99999@example.com';
```

6. Оптимизация JOIN-запросов

Создадим ещё одну таблицу orders:

```
CREATE TABLE orders (
    order_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(user_id),
    amount NUMERIC(10,2),
    order_date DATE
);
```

Заполняем 50 000 заказов

```
INSERT INTO orders (user_id, amount, order_date)
SELECT
    (random() * 100000)::INT + 1,
    (random() * 1000 + 100)::NUMERIC(10,2),
    CURRENT_DATE - (random() * 365)::INT
FROM generate_series(1, 50000);
```

Выполним JOIN-запрос без индекса:

```
EXPLAIN ANALYZE
SELECT u.username, SUM(o.amount) AS total_spent
FROM users u
JOIN orders o ON u.user_id = o.user_id
GROUP BY u.username
ORDER BY total_spent DESC
LIMIT 10;
```

Создадим индекс по внешнему ключу:

```
CREATE INDEX idx_orders_user_id ON orders(user_id);
```

Повторим запрос и сравните время выполнения.

7. Использование BRIN-индекса для диапазонов

Если часто фильтруете по дате:

```
CREATE INDEX idx_orders_order_date ON orders USING BRIN(order_date);
```

Пример использования:

```
EXPLAIN ANALYZE
SELECT * FROM orders WHERE order_date BETWEEN '2025-01-01' AND '2025-03-31';
```

8. Просмотр существующих индексов

```
SELECT indexname, tablename, indexdef
FROM pg_indexes
WHERE tablename IN ('users', 'orders');
```

Задание для самостоятельной работы

1. Создайте таблицу `products` и свяжите её с `orders`.
2. Добавьте 20 000 записей в `products`.
3. Напишите запрос, который выбирает все заказы с указанием имени пользователя, названия продукта и суммы заказа.
4. Проанализируйте план выполнения запроса и добавьте необходимые индексы для ускорения.
5. Сравните время выполнения до и после создания индексов.
6. Протестируйте работу с различными типами индексов (B-tree, BRIN, HASH).
7. Напишите отчет о проделанной работе, в котором объясните:
   - как изменилась производительность после добавления индексов;
   - какие типы индексов оказались наиболее эффективными;
   - почему некоторые индексы могут быть бесполезны или даже вредны.

Форма представления результата:

Отчет по выполненной лабораторной работе.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.2 Разработка и администрирование базы данных

Лабораторное занятие №12
Реализация хранимых процедур и триггеров для автоматизации работы с базой данных

Цель: научиться создавать и использовать хранимые процедуры (функции) и триггеры в PostgreSQL для автоматизации бизнес-логики, проверки целостности данных и упрощения повседневных операций над базой данных.

Выполнив работу, вы будете уметь:

– создавать функции на языке PL/pgSQL;
– вызывать пользовательские функции;
– создавать триггеры, реагирующие на события в таблицах

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.2 Осуществлять процедуры администрирования баз данных.

ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:

PostgreSQL pgAdmin, DBeaver

Задание:

В созданной базе данных создать хранимые процедуры, функции и триггеры.

Краткие теоретические сведения:

Хранимая процедура (или функция) — это блок SQL-кода, который хранится в самой базе данных и может быть вызван по имени. В PostgreSQL вместо термина "процедура" чаще используется термин "функция".

Преимущества:

– повторное использование кода;
– автоматизация логики;
– уменьшение сетевого трафика между приложением и базой данных.
– повышение производительности за счет предварительной компиляции.

Синтаксис

```
CREATE [ OR REPLACE ] PROCEDURE
    имя ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента [ { DEFAULT | = } выражение_по_умолчанию ] [, ...] ] )
  { LANGUAGE имя_языка
    | TRANSFORM { FOR TYPE имя_типа } [, ... ]
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | SET параметр_конфигурации { TO значение | = значение | FROM CURRENT }
    | AS 'определение'
    | AS 'объектный_файл', 'объектный_символ'
    | тело_sql
  } ...
```

Описание

Команда CREATE PROCEDURE определяет новую процедуру. CREATE OR REPLACE PROCEDURE создаёт новую процедуру либо заменяет определение уже существующей. Чтобы определить процедуру, необходимо иметь право USAGE для соответствующего языка.

Если указано имя схемы, процедура создаётся в заданной схеме, в противном случае — в текущей. Имя новой процедуры должно отличаться от имён существующих процедур и функций с такими же типами аргументов в этой схеме. Однако процедуры и функции с аргументами разных типов могут иметь одно имя (это называется перегрузкой).

Команда CREATE OR REPLACE PROCEDURE предназначена для изменения текущего определения существующей процедуры. С её помощью нельзя изменить имя или типы аргументов (если попытаться сделать это, будет создана новая отдельная процедура).

Когда команда CREATE OR REPLACE PROCEDURE заменяет существующую процедуру, владелец и права доступа к этой процедуре не меняются. Все другие свойства процедуры получают значения, задаваемые командой явно или по умолчанию. Чтобы заменить процедуру, необходимо быть её владельцем (или быть членом роли-владельца).

Параметры

имя

Имя создаваемой процедуры (возможно, дополненное схемой).

режим_аргумента

Режим аргумента: IN, OUT, INOUT или VARIADIC. По умолчанию подразумевается IN.

имя_аргумента

Имя аргумента.

тип_аргумента

Тип данных аргумента процедуры (возможно, дополненный схемой), при наличии аргументов.

Ссылка на тип столбца записывается в виде имя_таблицы.имя_столбца%TYPE. Иногда такое указание бывает полезно, так как позволяет создать процедуру, независящую от изменений в определении таблицы.

выражение_по_умолчанию

Выражение, используемое для вычисления значения по умолчанию, если параметр не задан явно. Результат выражения должен сводиться к типу соответствующего параметра. Для всех входных параметров, следующих за параметром с определённым значением по умолчанию, также должны быть определены значения по умолчанию.

имя_языка

Имя языка, на котором реализована процедура. Это может быть sql, c, internal либо имя процедурного языка, определённого пользователем, например, plpgsql. Если присутствует тело_sql, подразумевается язык sql.

параметр_конфигурации

значение

Предложение SET определяет, что при вызове процедуры указанный параметр конфигурации должен принять заданное значение, а затем восстановить своё предыдущее значение при завершении процедуры. Предложение SET FROM CURRENT сохраняет в качестве значения, которое будет применено при входе в процедуру, значение, действующее в момент выполнения CREATE PROCEDURE.

определение

Строковая константа, определяющая реализацию процедуры; её значение зависит от языка. Это может быть имя внутренней процедуры, путь к объектному файлу, команда SQL или код на процедурном языке.

объектный_файл, объектный_символ

Эта форма предложения AS применяется для динамически загружаемых процедур на языке C, когда имя процедуры в коде C не совпадает с именем процедуры в SQL. Строка объектный_файл задаёт имя файла, содержащего скомпилированную процедуру на C (данная команда воспринимает эту строку так же, как и LOAD). Строка объектный_символ задаёт символ

скомпонованной процедуры, то есть имя процедуры в исходном коде на языке C. Если объектный символ не указан, предполагается, что он совпадает с именем определяемой SQL-процедуры.

Если повторные вызовы CREATE PROCEDURE ссылаются на один и тот же объектный файл, он загружается в рамках сеанса только один раз. Чтобы выгрузить и загрузить этот файл снова (например, в процессе разработки), начните новый сеанс.

тело_sql
Тело процедуры в стиле LANGUAGE SQL. Это должен быть блок вида

```
BEGIN ATOMIC
  оператор;
  оператор;
  ...
  оператор;
END
```

Оно определяется подобно телу, задаваемому строковой константой (см. определение выше), но есть и некоторые различия. Эта форма работает только с функциями в стиле LANGUAGE SQL, тогда как форма со строковой константой поддерживается для всех языков. Она разбирается во время определения процедуры, тогда как форма со строковой константой — во время выполнения; как следствие, эта форма не поддерживает полиморфные типы аргументов и другие конструкции, которые нельзя обработать во время определения процедуры. С данной формой отслеживаются зависимости процедуры от объектов, используемых в её теле, так что команда DROP ... CASCADE выполнится корректно, тогда как в случае определения тела в строковой константе после такого удаления могут остаться неполноценные процедуры. Наконец, данная форма в большей степени соответствует стандарту SQL и совместима с другими реализациями SQL.

Чтобы выполнить процедуру, воспользуйтесь командой CALL.

Примеры

```
CREATE PROCEDURE insert_data(a integer, b integer)
LANGUAGE SQL
AS $$
INSERT INTO tbl VALUES (a);
INSERT INTO tbl VALUES (b);
$$;
```

или

```
CREATE PROCEDURE insert_data(a integer, b integer)
LANGUAGE SQL
BEGIN ATOMIC
  INSERT INTO tbl VALUES (a);
  INSERT INTO tbl VALUES (b);
END;
```

и пример вызова:

```
CALL insert_data(1, 2);
```

Триггер — это специальный вид хранимой процедуры, который автоматически выполняется при возникновении определённого события в таблице (например, INSERT, UPDATE, DELETE).

Где применяются:

Логирование изменений.
Контроль целостности данных.
Автоматическое обновление связанных записей.
Предотвращение ошибочных действий.

Порядок выполнения работы:

1. Подготовка среды
Подключитесь к PostgreSQL через psql:
```
psql -U postgres
```

Создайте новую базу данных:
```
CREATE DATABASE automation_db;
\c automation_db
```

2. Создание тестовой таблицы
Создадим таблицу employees:
```
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    full_name VARCHAR(100),
    salary NUMERIC(10, 2),
    department VARCHAR(50)
);
```

Добавим несколько записей:
```
INSERT INTO employees (full_name, salary, department) VALUES
('Иван Иванов', 60000, 'IT'),
('Мария Петрова', 75000, 'HR'),
('Алексей Смирнов', 80000, 'IT');
```

3. Создание простой функции
Напишем функцию, которая увеличивает зарплату сотрудника на заданное количество процентов:

```
CREATE OR REPLACE FUNCTION increase_salary(
    emp_id INT,
    percent NUMERIC
)
RETURNS VOID AS $$
BEGIN
    UPDATE employees
    SET salary = salary * (1 + percent / 100)
    WHERE employee_id = emp_id;
END;
$$ LANGUAGE plpgsql;
```

RETURNS VOID — функция ничего не возвращает.

Вызов функции:
```
SELECT increase_salary(1, 10); -- увеличиваем зарплату сотрудника с ID=1 на 10%
SELECT * FROM employees;
```

4. Функция с возвратом значения
Создадим функцию, возвращающую среднюю зарплату по отделу:

```
CREATE OR REPLACE FUNCTION avg_salary_by_department(dept VARCHAR)
RETURNS NUMERIC AS $$
DECLARE
    result NUMERIC;
BEGIN
    SELECT AVG(salary) INTO result
    FROM employees
    WHERE department = dept;

    RETURN result;
END;
$$ LANGUAGE plpgsql;
```

Вызов функции:
```
SELECT avg_salary_by_department('IT');
```

5. Создание триггера
Создадим таблицу для хранения истории изменений зарплат:

```
CREATE TABLE salary_log (
    log_id SERIAL PRIMARY KEY,
    employee_id INT,
    old_salary NUMERIC(10,2),
    new_salary NUMERIC(10,2),
    changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Теперь создадим функцию-обработчик для триггера:

```
CREATE OR REPLACE FUNCTION log_salary_change()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD.salary <> NEW.salary THEN
        INSERT INTO salary_log(employee_id, old_salary,
new_salary)
        VALUES (OLD.employee_id, OLD.salary, NEW.salary);
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Теперь привяжем её к таблице employees как триггер:

```
CREATE TRIGGER trigger_salary_change
AFTER UPDATE ON employees
FOR EACH ROW
EXECUTE FUNCTION log_salary_change();
```

Протестируем:

```
-- Обновляем зарплату
UPDATE employees SET salary = 90000 WHERE employee_id = 1;

-- Смотрим лог
SELECT * FROM salary_log;
```

6. Дополнительный пример: запрет обновления имени

Создадим триггер, который запрещает обновление поля full_name :

```
CREATE OR REPLACE FUNCTION prevent_name_update()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD.full_name <> NEW.full_name THEN
        RAISE EXCEPTION 'Изменение имени запрещено!';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_prevent_name_change
BEFORE UPDATE ON employees
FOR EACH ROW
EXECUTE FUNCTION prevent_name_update();
```

Проверяем:

```
UPDATE employees SET full_name = 'Новое имя' WHERE employee_id = 1; -- ошибка
UPDATE employees SET salary = 95000 WHERE employee_id = 1; -- успешно
```

Задание для самостоятельной работы

1. Создайте таблицу users со следующими полями:
   - `user_id`,
   - `username`,
   - `email`,
   - `created_at`.
2. Напишите функцию, которая добавляет нового пользователя, если `email` уникален.
3. Создайте таблицу `user_logs`, куда будут записываться действия (добавление, удаление, обновление).
4. Настройте триггеры:
   - при добавлении пользователя — записывается событие в `user_logs`;
   - при удалении — тоже записывается событие;
   - при обновлении `email` — логируется старый и новый `email`;
   - протестируйте работу всех функций и триггеров.

Форма представления результата:

Отчет по выполненной лабораторной работе.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.2 Разработка и администрирование базы данных

Лабораторное занятие №13
Разработка сценариев миграции данных между двумя базами данных

Цель: научиться разрабатывать и реализовывать сценарии переноса данных из одной базы данных в другую с использованием SQL, утилит PostgreSQL и скриптов на языке программирования Python.

Выполнив работу, вы будете уметь:
– использовать команды COPY, pg_dump и psql для экспорта и импорта данных;
– создавать скрипты автоматизации миграции;
– переносить данные между разными базами данных или серверами PostgreSQL;
– обеспечивать целостность и корректность данных при переносе.

Выполнение лабораторной работы способствует формированию:
ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.
ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.
ПК 2.2 Осуществлять процедуры администрирования баз данных.
ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:
PostgreSQL pgAdmin, DBeaver

Задание:
Выполнить миграции данных между двумя базами данных разными способами

Краткие теоретические сведения:
Миграция данных — это процесс перемещения данных из одной системы хранения информации в другую. Это может быть перенос между разными версиями базы данных, между средами (разработка → тестирование → продакшн), или между различными СУБД.

Основные причины миграции данных:

Причина	Пример
Апгрейд инфраструктуры	Переход с PostgreSQL 12 на PostgreSQL 16
Реорганизация структуры	Изменение схемы таблиц
Миграция на новую платформу	Переход с MySQL на PostgreSQL
Резервное копирование / восстановление	Архивирование данных или восстановление после сбоя
Тестирование / разработка	Перенос данных из production в staging

Инструменты PostgreSQL для миграции:
– pg_dump – экспорт данных и схемы базы данных;
– pg_restore – восстановление из бэкапа;
– COPY - импорт/экспорт в CSV/текстовые файлы;
– dblink/postgres_fdw – подключение к другой базе данных внутри запроса
– SQL-скрипты – ручная миграция через INSERT/SELECT
– Python + psycopg2 – автоматизация миграции через внешний код

Порядок выполнения работы:

1. Подготовка среды
Убедитесь, что установлен PostgreSQL:
```
psql --version
```

Создайте две базы данных: исходную (`source_db`) и целевую (`target_db`):
```
createdb source_db
createdb target_db
```

Подключитесь к исходной базе:
```
psql -d source_db
```

2. Создание структуры и заполнение данных в `source_db`
```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(50),
    email VARCHAR(100)
);

INSERT INTO users (username, email) VALUES
('ivan', 'ivan@example.com'),
('maria', 'maria@example.com'),
('alex', 'alex@example.com');
```

3. Экспорт данных с помощью pg_dump
Выйдите из psql (\q) и выполните в терминале:
```
pg_dump -d source_db -t users -f users_backup.sql
```

`-t users` — указывает, что нужно сохранить только таблицу users.
Теперь у вас есть файл `users_backup.sql`, который можно использовать для импорта.

4. Импорт данных в target_db
Импортируйте данные в целевую базу:
```
psql -d target_db -f users_backup.sql
```

Проверьте результат:
```
psql -d target_db -c "SELECT * FROM users;"
```

5. Использование COPY для экспорта в CSV
Если нужно экспортировать данные в формате CSV:
```
psql -d source_db -c "\COPY users TO 'users.csv' DELIMITER ',' CSV HEADER"
```

Импорт в целевую базу данных:
```
psql -d target_db -c "\COPY users FROM 'users.csv' DELIMITER ',' CSV HEADER"
```

6. Использование `dblink` для прямого переноса
Активируйте расширение `dblink`:
```
CREATE EXTENSION dblink;
```

Выполните перенос из source_db в target_db:

```
INSERT INTO users (user_id, username, email)
SELECT * FROM dblink('host=localhost dbname=source_db
user=postgres password=yourpassword',
                     'SELECT user_id, username, email FROM users')
AS t(user_id INT, username TEXT, email TEXT);
```

Убедитесь, что пароль указан правильно и пользователь имеет доступ.

7. Использование Python для миграции

Пример скрипта на Python с библиотекой psycopg2:

```
import psycopg2

# Подключение к исходной БД
conn_source = psycopg2.connect("dbname=source_db user=postgres
password=yourpassword")
cur_source = conn_source.cursor()

# Подключение к целевой БД
conn_target = psycopg2.connect("dbname=target_db user=postgres
password=yourpassword")
cur_target = conn_target.cursor()

# Выборка данных
cur_source.execute("SELECT user_id, username, email FROM users;")
rows = cur_source.fetchall()

# Вставка данных
for row in rows:
    cur_target.execute(
        "INSERT INTO users (user_id, username, email) VALUES (%s,
%s, %s);",
        row
    )

# Сохраняем изменения
conn_target.commit()

# Закрываем соединения
cur_source.close()
cur_target.close()
conn_source.close()
conn_target.close()
```

Для установки библиотеки:

```
pip install psycopg2-binary
```

Задание для самостоятельной работы

1. Создайте в source_db таблицы orders и products. Добавьте туда не менее 50 записей.
2. Создайте аналогичные таблицы в target_db.
3. Напишите скрипт миграции, который:
   – Экспортирует данные из source_db в CSV-файлы.
   – Импортирует их в target_db.
   – Логирует успешные/ошибочные операции в отдельный файл.
4. Реализуйте миграцию с использованием dblink.
5. Реализуйте миграцию с помощью Python-скрипта.
6. Сравните эффективность каждого метода по времени выполнения и простоте настройки.

Форма представления результата:
Отчет по выполненной лабораторной работе.

Критерии оценки:
Оценка «отлично» ставится, если задание выполнено верно.
Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.
Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.
Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.2 Разработка и администрирование базы данных

Лабораторное занятие №14
Администрирование базы данных: настройка параметров производительности и мониторинг активных запросов

Цель: научиться управлять конфигурацией сервера PostgreSQL для повышения производительности, а также использовать встроенные механизмы для отслеживания и анализа активных SQL-запросов.

Выполнив работу, вы будете уметь:
– настраивать основные параметры, влияющие на производительность;
– использовать системные представления PostgreSQL (pg_stat_statements, pg_stat_activity) для анализа активных и медленных запросов;
– определять "тяжёлые" или "долгие" запросы

Выполнение лабораторной работы способствует формированию:
ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.
ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.
ПК 2.2 Осуществлять процедуры администрирования баз данных.
ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:
PostgreSQL pgAdmin, DBeaver

Задание:

В созданной базе данных выполните несколько тяжелых запросов, проанализируйте их эффективность и настройте логирование.

Краткие теоретические сведения:

Администрирование PostgreSQL – это комплекс задач по настройке, управлению, мониторингу и оптимизации баз данных. Администратор отвечает за стабильную работу СУБД, высокую доступность, безопасность и производительность.

Ключевые аспекты администрирования:

Область	Задачи
Конфигурация	Настройка памяти, подключений, параллелизма
Безопасность	Права пользователей, SSL, брандмауэр
Резервное копирование	pg_dump, pg_basebackup
Мониторинг	Текущие процессы, медленные запросы, логи
Производительность	Индексы, настройки сервера, оптимизация запросов

Важные параметры производительности в postgresql.conf:

– shared_buffers – размер общей памяти для кэширования данных (обычно 25% от RAM);

– work_mem – память на операцию сортировки или хэширования;

– maintenance_work_mem – ПАМЯТЬ для операций обслуживания (VACUUM, CREATE INDEX и др.);

– effective_cache_size – объём кэша операционной системы, доступный для PostgreSQL;

– max_connections – максимальное число одновременных подключений;

– checkpoint_segments, checkpoint_timeout – частота контрольных точек — влияет на нагрузку на диск;

– random_page_cost – стоимость случайного чтения страницы (влияет на выбор плана).

Порядок выполнения работы:

1. Подготовка среды

Убедитесь, что установлен PostgreSQL:

```
psql --version
```

Подключитесь к вашей базе:

```
psql -U postgres
```

2. Путь к конфигурационным файлам

Посмотрите расположение файла postgresql.conf:

```
SHOW config_file;
```

Типичный путь:

```
/etc/postgresql/14/main/postgresql.conf
```

Для редактирования требуются права root:

```
sudo nano /etc/postgresql/14/main/postgresql.conf
```

3. Настройка производительности

Пример изменений в postgresql.conf:

```
shared_buffers = 2GB
work_mem = 64MB
maintenance_work_mem = 512MB
```

```
effective_cache_size = 6GB
max_connections = 100
checkpoint_segments = 32
checkpoint_timeout = 15min
random_page_cost = 1.1
```

После редактирования перезагрузите службу PostgreSQL:
```
sudo systemctl restart postgresql
```

4. Включение расширения pg_stat_statements
Расширение позволяет анализировать статистику выполнения SQL-запросов.

Добавьте в postgresql.conf:
```
shared_preload_libraries = 'pg_stat_statements'
track_functions = all
pg_stat_statements.track = all
```

Инициализируйте расширение:
```
CREATE EXTENSION pg_stat_statements;
```

5. Анализ активных запросов
Посмотрите текущие активные процессы:
```
SELECT pid, usename, application_name, state, query, query_start
FROM pg_stat_activity
WHERE state != 'idle';
```

Это поможет найти долгие или зависшие запросы.

Остановите конкретный запрос (например, с PID=1234):
```
SELECT pg_cancel_backend(1234);
```

6. Использование pg_stat_statements
Посмотрите самые частые и долгие запросы:
```
SELECT query, calls, total_time, mean_time, rows
FROM pg_stat_statements
ORDER BY total_time DESC
LIMIT 10;
```
Полезно для поиска «узких мест» в приложении.

7. Логирование медленных запросов
В postgresql.conf включите логирование:
```
log_min_duration_statement = 500  # миллисекунды
logging_collector = on
log_directory = 'log'
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
```
После этого PostgreSQL начнёт записывать все запросы, которые выполняются дольше 500 мс.

8. Использование инструментов мониторинга
pgAdmin
Интегрированный инструмент для просмотра активных сессий, запросов, графиков нагрузки.

ptop (аналог top для PostgreSQL)

```
sudo apt install ptop
ptop
```

Задание для самостоятельной работы

1. Настройте тестовую среду с PostgreSQL (можно через Docker).
2. Создайте таблицу logs с полями:

– id,
– message TEXT,
– created_at TIMESTAMP.

Добавьте в неё 100 000 записей.

3. Выполните несколько тяжёлых запросов (например, полный скан с группировкой):

```
SELECT COUNT(*) FROM logs GROUP BY created_at::date;
```

4. С помощью `pg_stat_statements` найдите этот запрос и проанализируйте его эффективность.
5. Увеличьте значение `work_mem` и повторите запрос. Зафиксируйте разницу во времени выполнения.
6. Настройте логирование всех запросов дольше 1 секунды. Проверьте содержимое лог-файлов.
7. Реализуйте автоматический скрипт (на Python или bash), который:

– проверяет список активных запросов,
– останавливает те, которые выполняются более 5 минут,
– логирует такие события в файл.

Форма представления результата:

Отчет по выполненной лабораторной работе.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.3. Организация защиты данных в хранилищах

Лабораторное занятие №15
Настройка шифрования данных в MySQL с использованием встроенных функций

Цель: научиться использовать встроенные функции шифрования и расшифровки данных в СУБД MySQL, такие как AES_ENCRYPT и AES_DECRYPT. Понять принципы хранения защищённых данных и особенности реализации шифрования на уровне базы данных.

Выполнив работу, вы будете уметь:

– использовать функции шифрования AES_ENCRYPT() и AES_DECRYPT() в SQL-запросах;
– хранить зашифрованные данные в таблицах MySQL;
– обеспечивать безопасность конфиденциальной информации на уровне базы данных

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.3 Проводить аудит систем безопасности баз данных с использованием регламентов по защите информации.

Материальное обеспечение:

MySQL

Задание:

В созданной базе данных настройте шифрование данных. Реализуйте приложение, которое добавляет новых сотрудников с шифрованием данных и выводит список сотрудников с расшифорвкой.

Краткие теоретические сведения:

Шифрование — это процесс преобразования информации в нечитаемую форму с помощью алгоритма и ключа. Это позволяет защитить данные от несанкционированного доступа даже при утечке.

Важность шифрования данные в базе данных

Причина	Описание
Защита приватности	Конфиденциальные данные (ФИО, телефон, email) нельзя хранить в открытом виде
Соответствие стандартам	GDPR, PCI DSS и другие регулирующие нормы требуют шифрования
Безопасность хранения	Даже при физическом доступе к файлам БД данные останутся защищёнными

В MySQL шифрование данных можно реализовать с использованием встроенных функций AES_ENCRYPT и AES_DECRYPT. Для шифрования используется симметричный алгоритм AES, который имеет различные ключевые длины (128, 256 бит и т.д.).

Алгоритм AES (Advanced Encryption Standard) — один из самых распространённых и безопасных методов симметричного шифрования.

Шаги по настройке шифрования

1. Подготовка:

- создайте таблицу или столбцы, в которых будут храниться данные, подлежащие шифрованию;
- решите, какие данные нужно зашифровать;
- выберете длину ключа aes (обычно 128 или 256 бит) в зависимости от требований безопасности.

2. Шифрование данных при вставке:

Используйте AES_ENCRYPT() для шифрования данных перед их вставкой в таблицу. Например:

```
INSERT INTO users (username, encrypted_password) VALUES ('john_doe', AES_ENCRYPT('password123', 'my_secret_key'));
```

В этом примере AES_ENCRYPT() зашифровывает строку 'password123' с помощью ключа 'my_secret_key' и возвращает зашифрованный двоичный текст, который затем сохраняется в столбце encrypted_password.

3. Извлечение и расшифровка данных:

Используйте AES_DECRYPT() для расшифровки данных при извлечении их из таблицы. Например:

```
SELECT username, AES_DECRYPT(encrypted_password, 'my_secret_key') AS decrypted_password FROM users WHERE username = 'john_doe';
```

В этом примере AES_DECRYPT() расшифровывает данные из столбца encrypted_password с помощью ключа 'my_secret_key' и возвращает расшифрованный текст в поле decrypted_password.

4. Обеспечение безопасности ключа:

Ключ должен быть секретным и храниться в безопасном месте. Нельзя хранить ключ в виде текста в базе данных или скриптах. Используйте разные ключи для разных таблиц или столбцов. Это повышает безопасность и предотвращает возможность взлома, если один ключ будет под угрозой. Не используйте легко угадываемые ключи. Ключ должен быть случайным и сложным.

Пример настройки шифрования для столбца

Создайте таблицу с зашифрованным столбцом:

```
CREATE TABLE sensitive_data (
    id INT AUTO_INCREMENT PRIMARY KEY,
    data_to_encrypt VARCHAR(255),
    encrypted_data BLOB
);
```

Вставьте данные:

```
INSERT INTO sensitive_data (data_to_encrypt, encrypted_data) VALUES ('My secret information', AES_ENCRYPT('My secret information', 'secret_key'));
```

Извлеките и расшифруйте данные:

```
SELECT AES_DECRYPT(encrypted_data, 'secret_key') AS decrypted_data FROM sensitive_data;
```

Важные замечания:

– AES_ENCRYPT() и AES_DECRYPT() возвращают двоичные строки (BLOB). Если вы хотите сохранить расшифрованные данные в текстовом столбце, необходимо преобразовать их в текстовый формат.

– Для обеспечения безопасности, рекомендуется использовать AES_ENCRYPT() и AES_DECRYPT() при выполнении операций с данными, а не хранить зашифрованные данные прямо в столбце. Это поможет защитить данные от несанкционированного доступа, если кто-то получит доступ к базе данных.

– Также можно использовать другие алгоритмы шифрования, такие как MD5(), SHA1(), SHA256(), но они не предназначены для шифрования и используются для хеширования данных.

Порядок выполнения работы:

1. Подготовка среды

Подключитесь к вашему серверу MySQL:

```
mysql -u root -p
```

Создайте тестовую базу данных:

```
CREATE DATABASE encryption_db;
USE encryption_db;
```

2. Простейший пример использования AES_ENCRYPT / AES_DECRYPT

Проверим работу функций шифрования:

```
SELECT AES_ENCRYPT('Тестовое сообщение', 'my_secret_key') AS encrypted;
SELECT AES_DECRYPT(AES_ENCRYPT('Тестовое сообщение', 'my_secret_key'), 'my_secret_key') AS decrypted;
```

Обратите внимание: если ключ неверный или отсутствует, результат будет NULL.

3. Создание таблицы с зашифрованными полями

Создадим таблицу users, где будем хранить логины и пароли в зашифрованном виде:

```
CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50),
    encrypted_password VARBINARY(255)
);
```

VARBINARY используется для хранения бинарных данных (результат шифрования)

4. Вставка зашифрованных данных

Добавим пользователя:

```
INSERT INTO users (username, encrypted_password)
VALUES ('admin', AES_ENCRYPT('StrongPass123!', 'my_secret_key'));
```

5. Получение и расшифровка данных

Выберем пользователя и расшифруем его пароль:

```
SELECT username, AES_DECRYPT(encrypted_password, 'my_secret_key') AS password
FROM users
WHERE username = 'admin';
```

Результат будет в виде строки, если ключ совпадает.

6. Использование шифрования для полей типа TEXT

Если нужно шифровать большие объемы данных, например, персональную информацию:

```
CREATE TABLE personal_data (
    id INT AUTO_INCREMENT PRIMARY KEY,
    full_name VARBINARY(255),
    phone_number VARBINARY(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Вставляем данные:

```
INSERT INTO personal_data (full_name, phone_number)
VALUES
```

```
    (AES_ENCRYPT('Иван            Петров',            'data_key_123'),
AES_ENCRYPT('+79123456789', 'data_key_123'));
```

Получаем обратно:

```
SELECT
    AES_DECRYPT(full_name, 'data_key_123') AS name,
    AES_DECRYPT(phone_number, 'data_key_123') AS phone
FROM personal_data;
```

7. Хранение ключа шифрования безопасно

Важные рекомендации:

- не храните ключ шифрования в самой базе данных;
- используйте переменные окружения или внешние хранилища секретов (Vault, AWS Secrets Manager);
- не передавайте ключ в чистом виде через SQL-запросы.

Использование ключа в приложении (на Python):

```
import mysql.connector

key = "my_secret_key"
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="yourpassword",
    database="encryption_db"
)
cursor = conn.cursor()

cursor.execute(f"""
    SELECT  username,  AES_DECRYPT(encrypted_password,  '{key}')  AS
password
    FROM users
""")
for row in cursor.fetchall():
    print(row)
```

Задание для самостоятельной работы

1. Создайте таблицу employees со следующими полями:
   - employee_id,
   - full_name (зашифрованное),
   - salary (зашифрованное),
   - passport_number (зашифрованное).
2. Добавьте несколько записей сотрудников с реальными данными.
3. Напишите запрос, который выводит расшифрованные данные только для пользователей с зарплатой выше 100 000.
4. Реализуйте простое приложение (на Python), которое:
   - добавляет новых сотрудников в таблицу с шифрованием данных;
   - выводит список сотрудников с расшифровкой.
5. Настройте систему аудита (логирование) всех операций добавления/обновления/удаления зашифрованных данных.

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.3. Организация защиты данных в хранилищах

Лабораторное занятие №16
Настройка защиты конфиденциальных данных с использованием маскирования данных (Data Masking) в Microsoft SQL Server

Цель: освоить принципы проектирования баз данных, используя ER-диаграммы для создания логической модели базы данных.

Выполнив работу, вы будете уметь:

– выполнять нормализацию базы данных (приводить таблицы к третьей нормальной форме);

– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.3 Проводить аудит систем безопасности баз данных с использованием регламентов по защите информации.

Материальное обеспечение:

MySQL Workbench

Задание:

В соответствии со своим вариантом задания, привести базу данных к 3НФ. Реализовать схему базы данных в среде MySQL Workbench.

Краткие теоретические сведения:

Порядок выполнения работы:

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.3. Организация защиты данных в хранилищах

Лабораторное занятие №17
Настройка аудита действий пользователей в Microsoft SQL Server

Цель: освоить принципы проектирования баз данных, используя ER-диаграммы для создания логической модели базы данных.

Выполнив работу, вы будете уметь:

– выполнять нормализацию базы данных (приводить таблицы к третьей нормальной форме);

– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.3 Проводить аудит систем безопасности баз данных с использованием регламентов по защите информации.

Материальное обеспечение:

MySQL Workbench

Задание:

В соответствии со своим вариантом задания, привести базу данных к 3НФ. Реализовать схему базы данных в среде MySQL Workbench.

Краткие теоретические сведения:

Порядок выполнения работы:

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.3. Организация защиты данных в хранилищах

Лабораторное занятие №18
Реализация ролевой модели безопасности в PostgreSQL (создание ролей и управление их правами)

Цель: освоить принципы проектирования баз данных, используя ER-диаграммы для создания логической модели базы данных.

Выполнив работу, вы будете уметь:

– выполнять нормализацию базы данных (приводить таблицы к третьей нормальной форме);

– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.3 Проводить аудит систем безопасности баз данных с использованием регламентов по защите информации.

Материальное обеспечение:

MySQL Workbench

Задание:

В соответствии со своим вариантом задания, привести базу данных к 3НФ. Реализовать схему базы данных в среде MySQL Workbench.

Краткие теоретические сведения:

Порядок выполнения работы:

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.3. Организация защиты данных в хранилищах

Лабораторное занятие №19
Организация двухфакторной аутентификации для доступа к базам данных

Цель: освоить принципы проектирования баз данных, используя ER-диаграммы для создания логической модели базы данных.

Выполнив работу, вы будете уметь:

– выполнять нормализацию базы данных (приводить таблицы к третьей нормальной форме);

– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.3 Проводить аудит систем безопасности баз данных с использованием регламентов по защите информации.

Материальное обеспечение:

MySQL Workbench

Задание:

В соответствии со своим вариантом задания, привести базу данных к 3НФ. Реализовать схему базы данных в среде MySQL Workbench.

Краткие теоретические сведения:

Порядок выполнения работы:

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.3. Организация защиты данных в хранилищах

Лабораторное занятие №20
Конфигурация шифрования трафика между клиентом и сервером базы данных (TLS/SSL)

Цель: освоить принципы проектирования баз данных, используя ER-диаграммы для создания логической модели базы данных.

Выполнив работу, вы будете уметь:

– выполнять нормализацию базы данных (приводить таблицы к третьей нормальной форме);

– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.3 Проводить аудит систем безопасности баз данных с использованием регламентов по защите информации.

Материальное обеспечение:

MySQL Workbench

Задание:

В соответствии со своим вариантом задания, привести базу данных к 3НФ. Реализовать схему базы данных в среде MySQL Workbench.

Краткие теоретические сведения:

Порядок выполнения работы:

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.3. Организация защиты данных в хранилищах

Лабораторное занятие №21
Организация резервного копирования с шифрованием в Oracle Database

Цель: освоить принципы проектирования баз данных, используя ER-диаграммы для создания логической модели базы данных.

Выполнив работу, вы будете уметь:

– выполнять нормализацию базы данных (приводить таблицы к третьей нормальной форме);

– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.3 Проводить аудит систем безопасности баз данных с использованием регламентов по защите информации.

Материальное обеспечение:

MySQL Workbench

Задание:

В соответствии со своим вариантом задания, привести базу данных к 3НФ. Реализовать схему базы данных в среде MySQL Workbench.

Краткие теоретические сведения:

Порядок выполнения работы:

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.1 Основы хранения и обработки данных. Проектирование базы данных

Лабораторное занятие №22
Нормализация базы данных: приведение таблиц к третьей нормальной форме (3НФ)

Цель: освоить принципы проектирования баз данных, используя ER-диаграммы для создания логической модели базы данных.

Выполнив работу, вы будете уметь:

– выполнять нормализацию базы данных (приводить таблицы к третьей нормальной форме);

– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:

MySQL Workbench

Задание:

В соответствии со своим вариантом задания, привести базу данных к 3НФ. Реализовать схему базы данных в среде MySQL Workbench.

Краткие теоретические сведения:

Порядок выполнения работы:

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.1 Основы хранения и обработки данных. Проектирование базы данных

Лабораторное занятие №23
Нормализация базы данных: приведение таблиц к третьей нормальной форме (3НФ)

Цель: освоить принципы проектирования баз данных, используя ER-диаграммы для создания логической модели базы данных.

Выполнив работу, вы будете уметь:

– выполнять нормализацию базы данных (приводить таблицы к третьей нормальной форме);

– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:

MySQL Workbench

Задание:

В соответствии со своим вариантом задания, привести базу данных к 3НФ. Реализовать схему базы данных в среде MySQL Workbench.

Краткие теоретические сведения:

Порядок выполнения работы:

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.1 Основы хранения и обработки данных. Проектирование базы данных

Лабораторное занятие №24
Нормализация базы данных: приведение таблиц к третьей нормальной форме (3НФ)

Цель: освоить принципы проектирования баз данных, используя ER-диаграммы для создания логической модели базы данных.

Выполнив работу, вы будете уметь:

– выполнять нормализацию базы данных (приводить таблицы к третьей нормальной форме);

– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:

MySQL Workbench

Задание:

В соответствии со своим вариантом задания, привести базу данных к 3НФ. Реализовать схему базы данных в среде MySQL Workbench.

Краткие теоретические сведения:

Порядок выполнения работы:

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.3. Организация защиты данных в хранилищах

Лабораторное занятие №25
Нормализация базы данных: приведение таблиц к третьей нормальной форме (3НФ)

Цель: освоить принципы проектирования баз данных, используя ER-диаграммы для создания логической модели базы данных.

Выполнив работу, вы будете уметь:

– выполнять нормализацию базы данных (приводить таблицы к третьей нормальной форме);

– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:

MySQL Workbench

Задание:

В соответствии со своим вариантом задания, привести базу данных к 3НФ. Реализовать схему базы данных в среде MySQL Workbench.

Краткие теоретические сведения:

Порядок выполнения работы:

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.1 Основы хранения и обработки данных. Проектирование базы данных

Лабораторное занятие №26
Нормализация базы данных: приведение таблиц к третьей нормальной форме (3НФ)

Цель: освоить принципы проектирования баз данных, используя ER-диаграммы для создания логической модели базы данных.

Выполнив работу, вы будете уметь:

– выполнять нормализацию базы данных (приводить таблицы к третьей нормальной форме);

– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:

MySQL Workbench

Задание:

В соответствии со своим вариантом задания, привести базу данных к 3НФ. Реализовать схему базы данных в среде MySQL Workbench.

Краткие теоретические сведения:

Порядок выполнения работы:

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.1 Основы хранения и обработки данных. Проектирование базы данных

Лабораторное занятие №27
Нормализация базы данных: приведение таблиц к третьей нормальной форме (3НФ)

Цель: освоить принципы проектирования баз данных, используя ER-диаграммы для создания логической модели базы данных.

Выполнив работу, вы будете уметь:

– выполнять нормализацию базы данных (приводить таблицы к третьей нормальной форме);

– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:

MySQL Workbench

Задание:

В соответствии со своим вариантом задания, привести базу данных к 3НФ. Реализовать схему базы данных в среде MySQL Workbench.

Краткие теоретические сведения:

Порядок выполнения работы:

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.1 Основы хранения и обработки данных. Проектирование базы данных

Лабораторное занятие №28
Нормализация базы данных: приведение таблиц к третьей нормальной форме (3НФ)

Цель: освоить принципы проектирования баз данных, используя ER-диаграммы для создания логической модели базы данных.

Выполнив работу, вы будете уметь:

– выполнять нормализацию базы данных (приводить таблицы к третьей нормальной форме);

– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:

MySQL Workbench

Задание:

В соответствии со своим вариантом задания, привести базу данных к 3НФ. Реализовать схему базы данных в среде MySQL Workbench.

Краткие теоретические сведения:

Порядок выполнения работы:

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.

Тема 1.1 Основы хранения и обработки данных. Проектирование базы данных

Лабораторное занятие №29
Нормализация базы данных: приведение таблиц к третьей нормальной форме (3НФ)

Цель: освоить принципы проектирования баз данных, используя ER-диаграммы для создания логической модели базы данных.

Выполнив работу, вы будете уметь:

– выполнять нормализацию базы данных (приводить таблицы к третьей нормальной форме);

– работать с современными case-средствами проектирования баз данных

Выполнение лабораторной работы способствует формированию:

ОК 01 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам.

ОК 02 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности.

ПК 2.5 Подготавливать данные для базы знаний.

Материальное обеспечение:

MySQL Workbench

Задание:

В соответствии со своим вариантом задания, привести базу данных к 3НФ. Реализовать схему базы данных в среде MySQL Workbench.

Краткие теоретические сведения:

Порядок выполнения работы:

Форма представления результата:

Оформленная схема базы данных.

Критерии оценки:

Оценка «отлично» ставится, если задание выполнено верно.

Оценка «хорошо» ставится, если ход выполнения задания верный, но была допущена одна или две ошибки, приведшие к неправильному результату.

Оценка «удовлетворительно» ставится, если приведено неполное выполнение задания.

Оценка «неудовлетворительно» ставится, если задание не выполнено.