Санкт-Петербургское государственное бюджетное профессиональное образовательное учреждение
«Академия управления городской средой, градостроительства и печати»

УТВЕРЖДАЮ
Заместитель директора
по учебно-методической работе
_____О.В. Фомичёва
«____» _____ 20_____г.

Методические рекомендации по организации и проведению практических занятий

ПМ.02 «АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ»

МДК.02.02 Технология разработки и защиты баз данных

для специальности

специальности 09.02.13 Интеграция решений с применением технологий искусственного интеллекта

Форма обучение –очная

Санкт-Петербург
2025

Разработчик: Ипатова С.В./Оболенская Е.Г., методисты СПб ГБПОУ АУГСГиП

Одобрены на заседании цикловой комиссии

Общетехнических дисциплин и компьютерных технологий

Протокол № 4

09.12.2025 г.

Председатель цикловой комиссии:

Шурухина И.Е.

В результате изучения профессионального модуля обучающихся должен освоить основной вид деятельности ВД2. «Администрирование баз данных» и соответствующие ему общие компетенции и профессиональные компетенции:

1.1.1. Перечень общих компетенций

Код	Наименование 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 03	Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях
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 06	Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных российских духовно-нравственных ценностей, в том числе с учетом гармонизации межнациональных и межрелигиозных отношений, применять стандарты антикоррупционного поведения
ОК 07	Содействовать сохранению окружающей среды, ресурсосбережению, применять знания об изменении климата, принципы бережливого производства, эффективно действовать в чрезвычайных ситуациях
ОК 08	Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности
ОК 09	Пользоваться профессиональной документацией на государственном и иностранном языках

1.1.2. Перечень профессиональных компетенций

Код	Наименование видов деятельности и профессиональных компетенций
ВД 2	Администрирование баз данных
ПК 2.1	Выявлять проблемы, возникающие в процессе эксплуатации баз данных.
ПК 2.2	Осуществлять процедуры администрирования баз данных.
ПК 2.3	Проводить аудит систем безопасности баз данных с использованием регламентов по защите информации.
ПК 2.4	Формировать требования хранилищ банка данных для обучения.
ПК 2.5	Подготавливать данные для базы знаний.

1.1.3. В результате освоения профессионального модуля обучающийся должен:

Иметь практический опыт	– Идентификации проблем, связанных с нормальным функционированием базы данных; – Восстановления системы. – Администрирования сервера баз данных; – Участия в администрировании отдельных компонент серверов; – Документирования результатов аудита безопасности информации; – Использования процедуры резервного копирования баз данных; – Использования процедуры восстановления баз данных

	– Подготовки документации по формированию требований хранилищ банка данных – Проектирования, разработки и эксплуатации баз данных
Уметь	– Производить идентификацию проблем, связанных с нормальным функционированием базы данных; – Принимать решения по локализации проблем, связанных с нормальным функционированием базы данных; – Документировать внештатные ситуации связанные с нормальным функционированием базы данных; – 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. – Производить формирование требований к обработке данных и их извлечению; – Добавлять, удалять и изменять данные в базе данных; – Производить операции по импорту и экспорту данных в различных форматах
Знать	– Основные коды ошибок при работе с базой данных; – Методы и средства устранения ошибок, возникающих при работе с базой данных; – Тенденции развития банков данных; – Технология установки и настройки сервера баз данных; Требования к безопасности сервера базы данных; – Протоколы безопасности при работе с базой данных; – Методы и средства защиты информации от несанкционированного доступа; – Уровни угроз безопасности информации – Формы документов, необходимых для формирования, ведения и использования банка данных – Типы данных хранения информации в базе данных

Практические работы

тема	название ПР	часы
МДК.02.02 Технология разработки и защиты баз данных		
Тема 2.1. Основы хранения и обработки данных. Проектирование БД.	**Практическая работа 1** Создание концептуальной модели базы данных с использованием диаграммы "сущность-связь" (ER-диаграмма). **Практическая работа 2** Разработка логической модели базы данных на основе ER-диаграммы. **Практическая работа 3** Нормализация базы данных: приведение таблиц к третьей нормальной форме (3НФ). **Практическая работа 4** Создание базы данных с использованием языка SQL (CREATE DATABASE, CREATE TABLE). **Практическая работа 5** Анализ и оптимизация структуры базы данных на основе требований к производительности.	8
	Практическая работа 6 Разработка ER-диаграммы для базы данных информационной системы (например, библиотечной системы). **Практическая работа 7** Нормализация данных на примере существующей базы (устранение избыточности). **Практическая работа 8** Проектирование структуры таблиц для реляционной базы данных с учётом первичных и внешних ключей. **Практическая работа 9** Определение индексов для оптимизации запросов к базе данных. **Практическая работа 10** Проектирование базы данных для хранения данных IoT (Интернет вещей) с учётом особенностей структуры.	10
Тема 2.2. Разработка и администрирование БД.	**Практическая работа 11** Создание базы данных и таблиц с использованием языка SQL (CREATE DATABASE, CREATE TABLE). **Практическая работа 12** Реализация ограничений целостности (PRIMARY KEY, FOREIGN KEY, UNIQUE) в таблицах базы данных. **Практическая работа 13** Написание и выполнение SQL-запросов для добавления, изменения и удаления данных (INSERT, UPDATE, DELETE). **Практическая работа 14** Настройка индексов для оптимизации производительности запросов (CREATE INDEX). **Практическая работа 15** Реализация хранимых процедур и триггеров для автоматизации работы с базой данных.	8
	Практическая работа 16 Настройка учётных записей пользователей и управление их правами доступа к базе данных. **Практическая работа 17** Оптимизация запросов к базе данных с использованием индексов и анализа плана выполнения запросов. **Практическая работа 18** Создание резервной копии	18

	базы данных и восстановление данных в случае сбоя. **Практическая работа 19** Разработка сценариев миграции данных между двумя базами данных. **Практическая работа 20** Администрирование базы данных: настройка параметров производительности и мониторинг активных запросов.	
Тема 2.3. Организация защиты данных в хранилищах	**Практическая работа 21** Настройка шифрования данных в MySQL с использованием встроенных функций (например, AES_ENCRYPT, AES_DECRYPT). **Практическая работа 22** Реализация ролевой модели безопасности в PostgreSQL (создание ролей и управление их правами). **Практическая работа 23** Настройка аудита действий пользователей в Microsoft SQL Server. **Практическая работа 24** Конфигурация шифрования трафика между клиентом и сервером базы данных (TLS/SSL). **Практическая работа 25** Организация резервного копирования с шифрованием в Oracle Database.	10
	Практическая работа 26 Разработка политики управления доступом к данным на уровне таблиц и столбцов. **Практическая работа 27** Настройка защиты конфиденциальных данных с использованием маскирования данных (Data Masking) в Microsoft SQL Server. **Практическая работа 28** Организация двухфакторной аутентификации для доступа к базам данных. **Практическая работа 29** . Анализ и устранение уязвимостей базы данных с использованием встроенных инструментов безопасности PostgreSQL. **Практическая работа 30** Разработка и реализация стратегии защиты данных от несанкционированного доступа в корпоративной базе данных.	10
Тема 2.4. Векторные базы данных	**Практическая работа 31** Установка и настройка векторной базы данных (например, Milvus, Pinecone или Weaviate). **Практическая работа 32** Создание и управление коллекциями данных в векторной базе (создание индексов и добавление векторов). **Практическая работа 33** Реализация функции поиска ближайших соседей (Nearest Neighbor Search) на примере текстовых или изображений. **Практическая работа 34** Интеграция векторной базы данных с Python для загрузки и обработки векторов. **Практическая работа 35** Проведение кластеризации данных в векторной базе с использованием встроенных функций.	10
	Практическая работа 36 Построение векторов для текстовых данных с использованием моделей преобразования (например, Word2Vec, BERT). **Практическая работа 37** Создание векторного	12

	хранилища для изображений и реализация поиска по сходству. **Практическая работа 38** Оптимизация индексов в векторной базе данных для увеличения скорости поиска. **Практическая работа 39** Обеспечение масштабируемости и высокой доступности векторной базы данных. **Практическая работа 40** Интеграция векторной базы данных в приложение для рекомендаций или кластеризации пользователей.	
		78

Тема 2.1. Основы хранения и обработки данных. Проектирование БД.

Практическая работа 1 Создание концептуальной модели базы данных с использованием диаграммы "сущность-связь" (ER-диаграмма).

Цель работы: освоить методику проектирования концептуальной модели базы данных посредством построения диаграммы «сущность-связь» (ER-диаграммы), научиться выделять ключевые сущности предметной области, их атрибуты и взаимосвязи.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

ER-диаграмма (Entity-Relationship diagram) — графическая модель, описывающая структуру данных предметной области через:

сущности (объекты, о которых хранится информация);

атрибуты (свойства сущностей);

связи (взаимоотношения между сущностями).

Основные элементы нотации (на примере нотации Чена):

Сущность — прямоугольник с именем сущности.

Атрибут — овал, соединённый с сущностью.

Связь — ромбовидный элемент между сущностями.

Ключи (первичный, внешний) — выделяются особым образом (например, подчёркиванием).

Типы связей:

1:1 (один-к-одному);

1:N (один-ко-многим);

M:N (многие-ко-многим).

Порядок выполнения работы

Анализ предметной области

Определите границы моделируемой системы. Выпишите основные объекты (сущности), о которых необходимо хранить информацию.

Выделение сущностей

Для каждой сущности:

задайте уникальное имя (существительное в единственном числе);

перечислите атрибуты (свойства);

выделите первичный ключ (уникальный идентификатор).

Определение атрибутов

Для каждого атрибута укажите:

имя;

тип данных (строка, число, дата и т. п.);

обязательность (может ли быть пустым);

уникальность (если требуется).

Установка связей между сущностями

Определите типы связей (1:1, 1:N, M:N).

Укажите смысловую нагрузку связи (например, «преподаватель ведёт курс»).

Для связей 1:N и M:N определите внешние ключи.

Построение ER-диаграммы

Изобразите сущности, атрибуты и связи в графической форме. Используйте стандартизированную нотацию (Чена, Мартина и др.).

Проверка модели

Убедитесь, что:

все ключевые объекты учтены;

связи корректны и полно описывают предметную область;

отсутствуют избыточные или дублирующие элементы.

Пример выполнения (на основе предметной области «Университет»)

Сущности и их атрибуты:

Студент

ID_студента (первичный ключ);

ФИО;

Дата_рождения;

Группа.

Курс

ID_курса (первичный ключ);

Название;

Количество_часов;

Семестр.

Преподаватель

ID_преподавателя (первичный ключ);

ФИО;

Должность;

Кафедра.

Оценка

ID_оценки (первичный ключ);

ID_студента (внешний ключ);

ID_курса (внешний ключ);

Оценка;

Дата.

Связи:

Студент → изучает → Курс (1:N: один студент может изучать несколько курсов).

Преподаватель → ведёт → Курс (1:N: один преподаватель может вести несколько курсов).

Студент + Курс → получает → Оценка (M:N, реализована через сущность «Оценка»).

Графическое представление (схема):

Студент ——< изучает >—— Курс

Преподаватель ——< ведёт >—— Курс

Студент ——< получает >—— Оценка ——< получает >—— Курс

(В графической ER-диаграмме вместо текста используются стандартные символы: прямоугольники для сущностей, ромбы для связей, овалы для атрибутов.)

Требования к отчёту

Отчёт должен содержать:

Описание предметной области.

Список выделенных сущностей с атрибутами и ключами.

ER-диаграмму (в виде рисунка или скриншота из инструмента).

Описание связей между сущностями (типы, смысловая нагрузка).

Краткие выводы о проделанной работе.

Инструменты для построения ER-диаграмм

Для визуализации можно использовать:

draw.io (бесплатный онлайн-инструмент);

Lucidchart;

Microsoft Visio;
ERDPlus (специализированный сервис для ER-моделей);
MySQL Workbench (если планируется дальнейшая реализация в СУБД).

Критерии оценки

Полнота охвата предметной области.
Корректность выделения сущностей и атрибутов.
Правильность определения ключей и связей.
Читаемость и стандартизированность ER-диаграммы.
Качество оформления отчёта.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.
2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)
Сформулировать выводы по результатам работы.
Сдать и защитить работу.

Практическая работа 2 Разработка логической модели базы данных на основе ER-диаграммы.

Цель работы: освоить методику перехода от концептуальной модели (ER-диаграммы) к логической модели реляционной базы данных, включая:

- выявление сущностей и их атрибутов;
- определение первичных и внешних ключей;
- нормализацию отношений до 3NF;
- построение схемы реляционных таблиц.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.

- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

ER-модель («сущность-связь») — концептуальная модель данных, состоящая из:

- **сущностей** (прямоугольники) — объектов предметной области;
- **атрибутов** — свойств сущностей;
- **связей** (ромбы) — ассоциаций между сущностями.

Логическая модель — набор реляционных отношений (таблиц) с чётко определёнными атрибутами, ключами и ограничениями.

Основные правила преобразования ER → реляционная модель:

1. Каждая сущность становится таблицей.
2. Атрибуты сущности становятся столбцами таблицы.
3. Первичный ключ сущности становится PRIMARY KEY таблицы.
4. Связи «1:1» и «1:N» реализуются через внешние ключи (FOREIGN KEY).
5. Связь «M:N» преобразуется в отдельную таблицу с составным первичным ключом.

Порядок выполнения работы

Шаг 1. Анализ ER-диаграммы

- Выпишите все сущности и их атрибуты.
- Определите типы данных для каждого атрибута (INT, VARCHAR, DATE и т.д.).
- Выделите первичные ключи (PK) для каждой сущности.
- Проанализируйте типы связей между сущностями (1:1, 1:N, M:N).

Шаг 2. Преобразование сущностей в таблицы

Для каждой сущности создайте таблицу со структурой:

```
Название_таблицы (
   PK_атрибут TYPE,
   атрибут_1 TYPE,
   атрибут_2 TYPE,
   ...
   PRIMARY KEY (PK_атрибут)
);
```

Шаг 3. Реализация связей

1:1 — добавьте внешний ключ в одну из таблиц (обычно в ту, где связь необязательна).

1:N — внешний ключ добавляется в таблицу «N» стороны.

M:N — создайте промежуточную таблицу с двумя внешними ключами и составным PK:

```
Связь_Таблица (
```

```
    PK_сущности_1 TYPE,
    PK_сущности_2 TYPE,
    дополнительные_атрибуты,
    PRIMARY KEY (PK_сущности_1, PK_сущности_2),
    FOREIGN KEY (PK_сущности_1) REFERENCES Таблица_1(PK),
    FOREIGN KEY (PK_сущности_2) REFERENCES Таблица_2(PK)
);
```

Шаг 4. Нормализация до 3NF

Проверьте каждую таблицу на соответствие нормальным формам:

1. **1NF**: все атрибуты атомарны, нет повторяющихся групп.
2. **2NF**: таблица в 1NF + все неключевые атрибуты полностью зависят от первичного ключа.
3. **3NF**: таблица в 2NF + нет транзитивных зависимостей (неключевые атрибуты не зависят друг от друга).

При обнаружении нарушений выполните декомпозицию таблиц.

Шаг 5. Формирование итоговой схемы

Составьте полный список таблиц с:

названиями;

списком столбцов и их типами;

первичными ключами (PK);

внешними ключами (FK) и ссылками на родительские таблицы;

ограничениями (NOT NULL, UNIQUE и т.д., если требуются).

Шаг 6. Проверка целостности

Убедитесь, что:

каждый FK ссылается на существующий PK;

все связи корректно отражены в структуре таблиц;

отсутствуют избыточные данные (соблюдена нормализация).

Пример выполнения (упрощённый)

ER-диаграмма:

Сущность «Студент» (ID, ФИО, Группа).

Сущность «Предмет» (ID, Название, Часы).

Связь «Изучает» (M:N) с атрибутом «Оценка».

Логическая модель:

```
sql
Студент (
    ID_студента INT PRIMARY KEY,
    ФИО VARCHAR(100) NOT NULL,
    Группа VARCHAR(10)
);

Предмет (
    ID_предмета INT PRIMARY KEY,
    Название VARCHAR(100) NOT NULL,
    Часы INT
);
```

```
Изучает (
    ID_студента INT,
    ID_предмета INT,
    Оценка DECIMAL(2,1),
    PRIMARY KEY (ID_студента, ID_предмета),
    FOREIGN KEY (ID_студента) REFERENCES Студент(ID_студента),
    FOREIGN KEY (ID_предмета) REFERENCES Предмет(ID_предмета)
);
```

Требования к отчёту

1. Исходная ER-диаграмма (изображение или описание).
2. Список сущностей и их атрибутов с типами данных.
3. Описание всех связей и их типов.
4. Итоговая схема реляционных таблиц (SQL-код или таблица с колонками: название, столбцы, PK, FK).
5. Краткое обоснование нормализации (какие таблицы в каких NF, были ли декомпозиции).
6. Выводы по работе.

Критерии оценки

Корректность преобразования ER → реляционная модель.

Соблюдение правил нормализации (до 3NF).

Правильность определения PK и FK.

Полнота и ясность отчётной документации.

Отсутствие логических ошибок в структуре таблиц и связях.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 3 Нормализация базы данных: приведение таблиц к третьей нормальной форме (3НФ).

Цель работы: освоить методику приведения реляционных таблиц к третьей нормальной форме (3НФ), устранить транзитивные зависимости и избыточность данных.

Задачи:

- Систематизировать подходы к изучению предмета.

• Научить правильному плану работы ИИ

• Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

• Для выполнения данного задания необходимо изучить теоретический материал по данной теме.

- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

• Предъявить преподавателю результаты работы.

• Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).

• Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

Нормализация — метод проектирования базы данных, направленный на устранение избыточности и предотвращение аномалий при модификации данных.

Третья нормальная форма (3НФ) требует:

Таблица уже находится во второй нормальной форме (2НФ).

В таблице отсутствует **транзитивная зависимость**: неключевые столбцы не должны зависеть от других неключевых столбцов.

Каждый неключевой атрибут должен **непосредственно зависеть от первичного ключа**.

Транзитивная зависимость — ситуация, когда:

$A \rightarrow B$ и $B \rightarrow C$, следовательно, $A \rightarrow C$

где A — первичный ключ, B и C — неключевые атрибуты.

Порядок выполнения работы

Анализ исходной таблицы

Определить первичный ключ.

Выявить все функциональные зависимости между атрибутами.

Проверить наличие транзитивных зависимостей.

Выявление транзитивных зависимостей

Для каждого неключевого атрибута проверить:

Зависит ли он непосредственно от первичного ключа?

Или его значение определяется через другой неключевой атрибут?

Декомпозиция таблицы

При обнаружении транзитивной зависимости:

Создать новую таблицу для зависимых атрибутов.

Включить в новую таблицу атрибут-определитель как внешний ключ.

Удалить избыточные данные из исходной таблицы.

Проверка результата

Убедиться, что:

Все таблицы находятся в 2НФ.

В каждой таблице нет транзитивных зависимостей.

Каждый неключевой атрибут зависит только от первичного ключа.

Пример приведения к 3НФ

Исходная таблица (2НФ, но не 3НФ):

Табельный_номер (ПК)	ФИО	Должность	Подразделение	Описание_подразделе
1	Иванов И.И.	Программист	Отдел разработки	Разработка и сопровождение приложений
2	Сергеев С.С.	Бухгалтер	Бухгалтерия	Ведение бухгалтерского учёта

Проблема:

Атрибут `Описание_подразделения` зависит от `Подразделение`, а не напрямую от первичного ключа `Табельный_номер`. Это транзитивная зависимость:

Табельный_номер→Подразделение и Подразделение→Описание_подразделения

Решение: декомпозиция на две таблицы

Таблица «Сотрудники» (3НФ):

Табельный_номер (ПК)	ФИО	Должность	ID_подразделения (ВК)
1	Иванов И.И.	Программист	1
2	Сергеев С.С.	Бухгалтер	2

Таблица «Подразделения» (3НФ):

ID_подразделения (ПК)	Подразделение	Описание_подразделения
1	Отдел разработки	Разработка и сопровождение приложений
2	Бухгалтерия	Ведение бухгалтерского учёта

Результат:

Устранена транзитивная зависимость.

Данные не дублируются (описание подразделения хранится один раз).

Обеспечена целостность данных (изменение описания подразделения затрагивает только одну запись).

Задания для самостоятельной работы

Дана таблица «Заказы»:

Номер_заказа (ПК), Дата, Клиент, Адрес_клиента, Товар, Цена, Сумма

Привести к 3НФ, обосновать шаги.

Дана таблица «Студенты»:

ID_студента (ПК), ФИО, Группа, Староста_группы, Телефон_старосты

Привести к 3НФ, показать итоговые таблицы.

Дана таблица «Автомобили»:

VIN (ПК), Модель, Производитель, Страна_производителя, Год_выпуска

Проверить, находится ли в 3НФ. Если нет — нормализовать.

Требования к отчёту

Исходные таблицы с указанием первичного ключа.

Анализ функциональных зависимостей и выявленных транзитивных связей.

Пошаговая декомпозиция с объяснением.

Итоговые таблицы в 3НФ с первичными и внешними ключами.

Вывод о достигнутой нормализации.

Критерии оценки

Корректность выявления транзитивных зависимостей.

Правильность декомпозиции таблиц.

Соблюдение правил 3НФ в итоговых структурах.

Чёткость обоснования шагов нормализации.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 4 Создание базы данных с использованием языка SQL (CREATE DATABASE, CREATE TABLE).

Цель работы: освоить базовые команды SQL для создания базы данных и таблиц: `CREATE DATABASE`, `CREATE TABLE`, а также сопутствующие команды управления (`USE`, `SHOW`, `DROP`).

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретическая часть

Основные команды SQL

1. `CREATE DATABASE` — создаёт новую базу данных.

Синтаксис:

```
sql
CREATE DATABASE имя_базы_данных;
```

Имя БД: до 64 символов, может включать буквы, цифры, `_`, `$`.

Нельзя использовать пробелы (заменяйте на `_`).

Если БД с таким именем уже существует, возникнет ошибка.

2. `USE` — выбирает БД для дальнейшей работы.

Синтаксис:

```
sql
USE имя_базы_данных;
```

3. `CREATE TABLE` — создаёт таблицу в выбранной БД.

Синтаксис:

```
sql
CREATE TABLE имя_таблицы (
   имя_столбца1 тип_данных [ограничения],
   имя_столбца2 тип_данных [ограничения],
```

```
    ...
);
```

Для каждого столбца указывается тип данных (например, `INT`, `VARCHAR`, `DATE`).

Можно задать ограничения: `PRIMARY KEY`, `NOT NULL`, `UNIQUE` и др.

`SHOW` — показывает список объектов.

`SHOW DATABASES;` — список всех БД.

`SHOW TABLES;` — список таблиц в текущей БД.

`DROP` — удаляет объекты.

`DROP DATABASE имя_БД;` — удаляет БД.

`DROP TABLE имя_таблицы;` — удаляет таблицу.

Типичные типы данных

`INT` — целое число.

`VARCHAR(n)` — строка длиной до *n* символов.

`TEXT` — длинная строка.

`DATE` — дата.

`DECIMAL(m,n)` — число с плавающей точкой (m — общее количество цифр, n — после запятой).

`BOOLEAN` — логическое значение (`TRUE`/`FALSE`).

Практическая часть

Шаг 1. Создание базы данных

1. Откройте клиент SQL (например, MySQL Command Line Client, phpMyAdmin, SQL Server Management Studio).
2. Введите команду для создания БД:

```
sql
CREATE DATABASE SchoolDB;
```

3. Проверьте список БД:

```
sql
SHOW DATABASES;
```

В выводе должна появиться `SchoolDB`.

Шаг 2. Выбор базы данных

Переключитесь на созданную БД:

```
sql
USE SchoolDB;
```

Шаг 3. Создание таблиц

Создайте таблицу `Students` (ученики):

```
sql
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
```

```
    LastName VARCHAR(50) NOT NULL,
    BirthDate DATE,
    Grade INT
);
```

`StudentID` — уникальный идентификатор (первичный ключ).

`FirstName`, `LastName` — обязательные поля (`.NOT NULL`).

`BirthDate` — дата рождения.

`Grade` — класс обучения.

Создайте таблицу `Subjects` (предметы):

```
sql
CREATE TABLE Subjects (
    SubjectID INT PRIMARY KEY,
    SubjectName VARCHAR(100) NOT NULL
);
```

Создайте таблицу `Grades` (оценки):

```
sql
CREATE TABLE Grades (
    GradeID INT PRIMARY KEY,
    StudentID INT,
    SubjectID INT,
    Score DECIMAL(3,2),
    Date DATE,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (SubjectID) REFERENCES Subjects(SubjectID)
);
```

- Связывает учеников и предметы через внешние ключи (`FOREIGN KEY`).

Шаг 4. Проверка создания таблиц

Выведите список таблиц в БД:

```
sql
SHOW TABLES;
```

Ожидаемый вывод:

```
Students
Subjects
Grades
```

Шаг 5. Просмотр структуры таблицы

Проверьте структуру таблицы `Students`:

```
sql
DESCRIBE Students;
```

Вывод покажет имена столбцов, типы данных и ограничения.

Шаг 6. Удаление объектов (опционально)

Если нужно удалить таблицу:

```
sql
```

```
DROP TABLE Grades;
```

Если нужно удалить БД:

```
sql
DROP DATABASE SchoolDB;
```

Внимание! Эти команды необратимы — данные будут удалены.

Контрольные вопросы

1. Какова максимальная длина имени базы данных в SQL?
2. Зачем используется команда `USE`?
3. Что такое первичный ключ (`PRIMARY KEY`)?
4. Какие типы данных вы использовали в таблицах?
5. Для чего нужны внешние ключи (`FOREIGN KEY`)?

Требования к отчёту

1. Текст команд SQL, использованных в работе.
2. Скриншоты вывода команд (`SHOW DATABASES`, `SHOW TABLES`, `DESCRIBE`).
3. Ответы на контрольные вопросы.
4. Описание возникших ошибок (если были) и способов их устранения.

Дополнительные задания (по желанию)

1. Создайте таблицу `Teachers` (учителя) с полями: `TeacherID`, `FirstName`, `LastName`, `SubjectID`.
2. Добавьте в таблицу `Students` поле `Email` типа `VARCHAR(100)`.
3. Используйте условие `IF NOT EXISTS` при создании БД, чтобы избежать ошибки, если БД уже существует:

```
sql
CREATE DATABASE IF NOT EXISTS SchoolDB;
```

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 5 Анализ и оптимизация структуры базы данных на основе требований к производительности.

Цель работы: освоить методы анализа и оптимизации структуры базы данных для повышения производительности системы: научиться выявлять «узкие места», применять техники нормализации/денормализации, индексирования, оптимизации запросов и мониторинга.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.
- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Исходные данные

Предположим, имеется реляционная база данных (например, для интернет-магазина) со следующими таблицами:

`Products` (товары);

`Categories` (категории);

`Orders` (заказы);

`OrderItems` (позиции заказа);

`Customers` (клиенты).

Типичные запросы: поиск товара по названию/категории, получение списка заказов клиента, расчёт суммы заказа.

Ход работы

1. Анализ текущей структуры и производительности

1. **Профилирование запросов**
   - Соберите реальные SQL-запросы приложения.

- Используйте `EXPLAIN` (или аналог в вашей СУБД) для анализа планов выполнения:

```
sql
EXPLAIN SELECT * FROM Orders WHERE customer_id = 123;
```

- Выявите запросы с полным сканированием таблиц (`Seq Scan`), длинными соединениями (`JOIN`), отсутствием индексов.

2. **Мониторинг показателей**

Отслеживайте:

время ответа на запрос (*t*ответа);
загрузку CPU и памяти СУБД;
количество операций ввода-вывода (`I/O`);
коэффициент попадания в кэш (`cache hit ratio`).

3. **Анализ схемы данных**

Проверьте:

соответствие типов данных (например, `VARCHAR(255)` вместо `TEXT` для коротких строк);
наличие избыточных полей или дублирования данных;
правильность связей (внешние ключи, индексы).

2. Оптимизация структуры данных

1. **Нормализация/денормализация**
   - **Нормализация** (до 3NF): устраните повторяющиеся группы, транзитивные зависимости.

Пример: выделите таблицу `ProductPrices` для истории цен, если она хранится в `Products`.

   - **Денормализация**: добавьте дублирующие поля для частых запросов (например, `total_price` в `Orders`), если это ускорит чтение.
2. **Выбор типов данных**
   - Используйте `INT` вместо `BIGINT` для ID, если диапазон значений позволяет.
   - Для дат — `DATE`/`TIMESTAMP`, а не строки.
3. **Оптимизация связей**
   - Убедитесь, что внешние ключи проиндексированы.
   - Рассмотрите партиционирование больших таблиц (например, по дате заказа).

3. Индексирование

1. **Создание индексов**
   - Для столбцов в `WHERE`, `JOIN`, `ORDER BY`:

```
sql
CREATE INDEX idx_orders_customer ON Orders(customer_id);
CREATE INDEX idx_products_name ON Products(name);
```

   - Составные индексы для частых комбинаций:

```
sql
CREATE INDEX idx_orderitems_product_qty ON OrderItems(product_id, quantity);
```

2. **Ограничения**
   - Избегайте избыточных индексов (ухудшают `INSERT`/`UPDATE`).
   - Удаляйте неиспользуемые индексы (анализируйте через `pg_stat_user_indexes` в PostgreSQL).

4. Оптимизация запросов

1. **Перепись запросов**
   - Замените подзапросы на `JOIN`, если это возможно.

- Используйте `LIMIT` для пагинации:

```sql
SELECT * FROM Orders ORDER BY order_date DESC LIMIT 10;
```

- Избегайте `SELECT *` — указывайте нужные столбцы.

2. **Кэширование результатов**
   - На уровне приложения кэшируйте частые запросы (например, топ-10 товаров).
   - Используйте встроенные механизмы СУБД (например, `QUERY CACHE` в MySQL).
3. **Партиционирование**

Разделите большие таблицы по диапазонам (например, заказы за каждый год в отдельную партицию).

5. Тестирование и мониторинг

1. **Нагрузочное тестирование**
   - Имитируйте реальную нагрузку (например, 100 запросов/сек).
   - Измеряйте:
     - среднее время ответа (tответа);
     - пропускную способность (запросов/сек);
     - ошибки тайм-аута.
2. **Анализ после оптимизации**
   - Сравните показатели до и после (например, снижение времени запроса с 500 мс до 50 мс).
   - Проверьте использование ресурсов (CPU, память, диск).

Пример оптимизации конкретного запроса

Проблема: медленный запрос на получение заказов клиента:

```sql
SELECT * FROM Orders WHERE customer_id = 123;
```

Решение:

1. Создать индекс:

```sql
CREATE INDEX idx_orders_customer ON Orders(customer_id);
```

2. Переписать запрос с явным указанием столбцов:

```sql
SELECT order_id, total_price, order_date
FROM Orders
WHERE customer_id = 123
ORDER BY order_date DESC
LIMIT 10;
```

3. Добавить партиционирование по `customer_id`, если таблица очень большая.

Отчёт по работе

Оформите результаты в виде таблицы:

Этап оптимизации	До	После	Эффект
Индексирование `customer_id`	400 мс	20 мс	×20 быстрее
Добавление `LIMIT`	300 мс	50 мс	×6 быстрее

Выводы

1. Оптимизация требует итеративного подхода: анализ → изменение → тестирование.

2. Ключевые техники: индексирование, нормализация/денормализация, оптимизация запросов.

3. Мониторинг — обязательный этап для подтверждения эффективности изменений.

Контрольные вопросы

1. В чём разница между нормализацией и денормализацией? Когда применять каждую?
2. Как индекс ускоряет запрос? Какие минусы у индексов?
3. Что показывает команда `EXPLAIN`?
4. Почему `SELECT *` может снижать производительность?
5. Какие метрики важно отслеживать при оптимизации?

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 6 Разработка ER-диаграммы для базы данных информационной системы (например, библиотечной системы).

Цель работы: спроектировать концептуальную модель данных (ER-диаграмму) для информационной системы библиотеки, отражающую основные сущности, их атрибуты и взаимосвязи.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.

• Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).

• Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

1. Шаги выполнения

Шаг 1. Определение ключевых сущностей

Для библиотечной системы выделим следующие основные сущности:

- Книга
- Читатель
- ЭкземплярКниги
- Выдача
- Автор
- Жанр
- Издательство

Шаг 2. Определение атрибутов сущностей

- Книга
  - ID_Книги (ключевой атрибут)
  - Название
  - ГодИздания
  - ISBN
  - ID_Издательства (внешний ключ)
  - ID_Жана (внешний ключ)
- Читатель
  - ID_Читателя (ключевой атрибут)
  - ФИО
  - ДатаРегистрации
  - Адрес
  - Телефон
- ЭкземплярКниги
  - ID_Экземпляра (ключевой атрибут)
  - ID_Книги (внешний ключ)
  - Состояние
  - МестоХранения
- Выдача
  - ID_Выдачи (ключевой атрибут)
  - ID_Экземпляра (внешний ключ)
  - ID_Читателя (внешний ключ)

- ДатаВыдачи
- СрокВозврата
- ФактическаяДатаВозврата
- Автор
- ID_Автора (ключевой атрибут)
- ФИО
- Жанр
- ID_Жана (ключевой атрибут)
- НазваниеЖана
- Издательство
- ID_Издательства (ключевой атрибут)
- Название
- Город

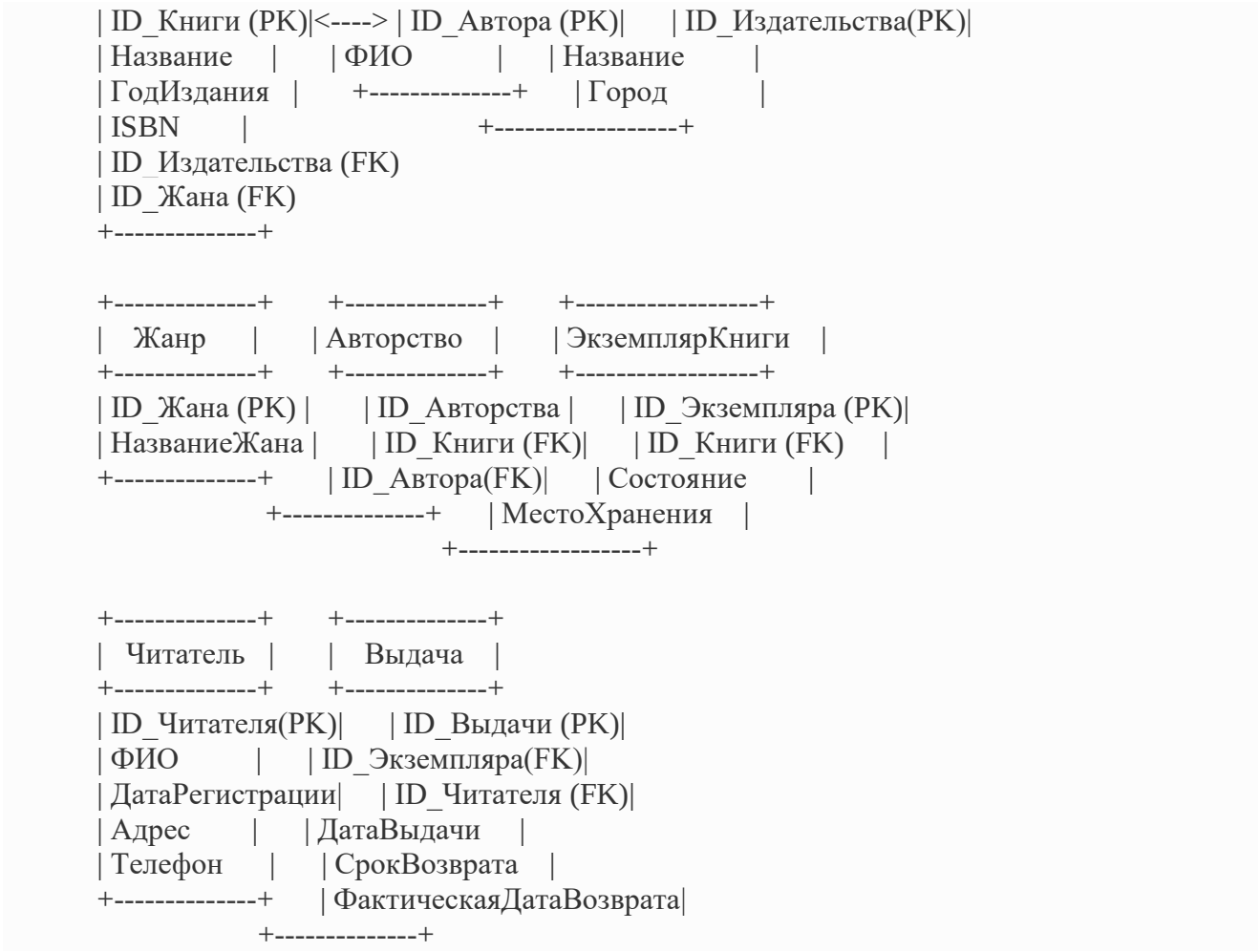
Шаг 3. Определение связей между сущностями

1. Книга ↔ Автор
   - Тип: многие-ко-многим (книга может иметь нескольких авторов, автор может написать несколько книг)
   - Решается через ассоциативную сущность Авторство с атрибутами:
     - ID_Авторства
     - ID_Книги
     - ID_Автора
2. Книга → Издательство
   - Тип: многие-к-одному (много книг может быть издано одним издательством)
   - Внешний ключ: ID_Издательства в сущности Книга
3. Книга → ЭкземплярКниги
   - Тип: один-ко-многим (одна книга может иметь несколько экземпляров)
   - Внешний ключ: ID_Книги в сущности ЭкземплярКниги
4. ЭкземплярКниги → Выдача
   - Тип: один-ко-многим (один экземпляр может быть выдан несколько раз)
   - Внешний ключ: ID_Экземпляра в сущности Выдача
5. Читатель → Выдача
   - Тип: один-ко-многим (читатель может взять несколько книг)
   - Внешний ключ: ID_Читателя в сущности Выдача
6. Книга → Жанр
   - Тип: многие-к-одному (много книг может относиться к одному жанру)
   - Внешний ключ: ID_Жана в сущности Книга

Шаг 4. Построение ER-диаграммы

Визуальное представление (текстовое описание структуры):

```
+--------------+      +--------------+      +------------------+
|   Книга      |      |  Автор       |      |  Издательство    |
+--------------+      +--------------+      +------------------+
```



Обозначения:

- (PK) — первичный ключ
- (FK) — внешний ключ
- <----> — связь многие-ко-многим
- → — связь один-ко-многим

3. Итоговый результат

Разработана ER-диаграмма, включающая:

- 7 основных сущностей;
- ключевые и внешние ключи;
- атрибуты каждой сущности;
- типы связей между сущностями (один-ко-многим, многие-ко-многим).

4. Рекомендации по реализации

1. Для связи «многие-ко-многим» (Книга ↔ Автор) обязательно создать промежуточную таблицу Авторство.
2. Использовать суррогатные ключи (числовые ID) для упрощения связей.
3. Добавить ограничения:
   - NOT NULL для обязательных полей;
   - UNIQUE для ISBN;
   - CHECK для дат (например, СрокВозврата ≥ ДатаВыдачи).
4. Рассмотреть добавление индексов для часто используемых внешних ключей (например, ID_Читателя в Выдача).

5. Инструменты для построения

Для визуализации ER-диаграммы можно использовать:

- **Lucidchart**
- **Draw.io (diagrams.net)**
- **Microsoft Visio**
- **MySQL Workbench** (для прямого проектирования БД)
- **ERD Plus** (онлайн-инструмент)

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 7 Нормализация данных на примере существующей базы (устранение избыточности).

Цель работы: освоить методику нормализации реляционной базы данных: выявить и устранить избыточность, привести структуру к 3NF (третьей нормальной форме), обеспечить целостность данных.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

Нормализация — процесс организации данных в базе, включающий:

- создание таблиц;
- установление связей между ними;
- соблюдение правил, устраняющих избыточность и аномалии.

Основные проблемы избыточности:

- аномалии вставки (невозможно добавить запись без заполнения всех полей);
- аномалии обновления (изменение данных в одном месте не отражается везде);
- аномалии удаления (удаление записи ведёт к потере несвязанных данных).

Нормальные формы (кратко):

1. **1NF** — все атрибуты атомарны, нет повторяющихся групп.
2. **2NF** — выполнена 1NF + все неключевые атрибуты полностью зависят от первичного ключа.
3. **3NF** — выполнена 2NF + нет транзитивных зависимостей (неключевые атрибуты не зависят друг от друга).

Пример исходной базы данных

Предположим, есть таблица Заказы с избыточностью:

ID_заказа	Клиент	Адрес клиента	Товар	Цена	Дата	Менеджер	
1	ООО «Альфа»	ул. Ленина	Стол	5000	01.01	Иванов	
2	ООО «Бета»	пр. Мира	Шкаф	8000	02.01	Петров	
3	ООО «Альфа»	ул. Ленина	Кровать	6000	03.01	Иванов	

Проблемы:

- дублирование данных о клиентах (ООО «Альфа», адрес);
- дублирование данных о менеджерах (Иванов, отдел);
- потенциальная аномалия: изменение адреса клиента требует правки во всех заказах.

Шаги нормализации

Шаг 1. Приведение к 1NF

Таблица уже в 1NF: все поля атомарны, нет повторяющихся групп.

Шаг 2. Приведение к 2NF

Выделяем сущности с частичной зависимостью от ключа:

- **Клиенты** (зависимость: ID_заказа → Клиент, Адрес клиента);
- **Менеджеры** (зависимость: ID_заказа → Менеджер, Отдел менеджера).

Новые таблицы:

`Клиенты`

ID_клиента	Название	Адрес
1	ООО «Альфа»	ул. Ленина
2	ООО «Бета»	пр. Мира

`Менеджеры`

ID_менеджера	ФИО	Отдел
1	Иванов	Продажи
2	Петров	Логистика

`Заказы` (обновлённая)

ID_заказа	ID_клиента	Товар	Цена	Дата	ID_менеджера
1	1	Стол	5000	01.01	1
2	2	Шкаф	8000	02.01	2
3	1	Кровать	6000	03.01	1

Связи:

- `Заказы.ID_клиента` → `Клиенты.ID_клиента`;
- `Заказы.ID_менеджера` → `Менеджеры.ID_менеджера`.

Шаг 3. Приведение к 3NF

Проверяем транзитивные зависимости:

- В `Клиенты`: нет (все атрибуты зависят только от `ID_клиента`).
- В `Менеджеры`: нет (все атрибуты зависят от `ID_менеджера`).
- В `Заказы`: нет (все неключевые атрибуты зависят от `ID_заказа`).

Итог: база приведена к 3NF.

Результаты нормализации

Преимущества:

1. **Устранена избыточность:**
   - адрес клиента хранится один раз;
   - данные о менеджере не дублируются.
2. **Исключены аномалии:**
   - изменение адреса клиента — одна правка в таблице `Клиенты`;
   - удаление заказа не затрагивает данные о клиенте или менеджере.
3. **Улучшена целостность:**
   - связи через внешние ключи гарантируют корректность данных;
   - добавление нового клиента возможно без заказа.

Недостатки:

- увеличилось число таблиц (3 вместо 1);
- для получения полного отчёта требуется JOIN (например, чтобы вывести название клиента вместо ID_клиента).

Контрольные вопросы

1. Что такое избыточность данных и почему она вредна?
2. Перечислите основные нормальные формы и их ключевые правила.
3. Как проверить, что таблица находится в 3NF?
4. Приведите пример аномалии обновления в ненормализованной базе.
5. В каких случаях можно отклониться от 3NF в реальных проектах?

Задание для самостоятельной работы

1. Возьмите реальную таблицу из вашей базы данных (например, учёт товаров, сотрудников, заказов).
2. Выявите избыточные данные и потенциальные аномалии.
3. Приведите таблицу к 3NF, создав необходимые связанные таблицы.
4. Опишите изменения и их преимущества в отчёте.

Формат отчёта:

- Исходная таблица (скриншот/выгрузка).
- Список выявленных проблем.
- Нормализованные таблицы (структура + примеры данных).
- SQL-запросы для создания таблиц и связей.
- Краткий анализ плюсов и минусов нормализации для вашего случая.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 8 Проектирование структуры таблиц для реляционной базы данных с учётом первичных и внешних ключей.

Цель работы: освоить методику проектирования реляционных баз данных: научиться выделять сущности, определять атрибуты, задавать первичные и внешние ключи, устанавливать связи между таблицами.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ

- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

Реляционная база данных — совокупность взаимосвязанных таблиц, каждая из которых соответствует определённой сущности предметной области.

Основные понятия:

- **Таблица (отношение)** — структура для хранения данных об одной сущности.
- **Столбец (поле, атрибут)** — характеристика сущности.
- **Строка (запись, кортеж)** — конкретный экземпляр сущности.
- **Первичный ключ (PK, Primary Key)** — уникальный идентификатор записи в таблице. Гарантирует отсутствие дубликатов.
- **Внешний ключ (FK, Foreign Key)** — столбец (или группа столбцов), ссылающийся на первичный ключ другой таблицы. Обеспечивает связь между таблицами и целостность данных.

Типы связей:

1. **«Один-к-одному»** (1 : 1) — одна запись в таблице A связана с одной записью в таблице B.
2. **«Один-ко-многим»** (1 : M) — одна запись в таблице A связана со множеством записей в таблице B.
3. **«Многие-ко-многим»** (M : N) — множество записей в таблице A связано со множеством записей в таблице B (реализуется через промежуточную таблицу).

Порядок выполнения

1. **Анализ предметной области**

Определите, для какой сферы проектируется БД (например, библиотека, магазин, университет). Выделите основные сущности (объекты), о которых нужно хранить информацию.

2. **Создание списка сущностей и атрибутов**

Для каждой сущности перечислите атрибуты (поля). Пример для университета:

- Студент: ID, ФИО, дата рождения, группа.
- Группа: ID, название, курс.
- Предмет: ID, название, часы.
- Оценка: ID студента, ID предмета, оценка, дата.

3. **Формирование таблиц**

Каждая сущность становится таблицей. Атрибуты — столбцами.

4. **Определение первичных ключей**

Выберите (или создайте) уникальный идентификатор для каждой таблицы:

- Для «Студент» — `student_id` (целое число, автоинкремент).
- Для «Группа» — `group_id`.
- Для «Предмет» — `subject_id`.
- Для «Оценка» — составной ключ из `student_id` и `subject_id`.

5. **Установка внешних ключей**

Свяжите таблицы через FK:

- В таблице «Студент» добавьте `group_id` как FK, ссылающийся на `group_id` в таблице «Группа».
- В таблице «Оценка» добавьте:
  - `student_id` → FK к `student_id` в «Студент».
  - `subject_id` → FK к `subject_id` в «Предмет».

6. **Описание связей**

Укажите типы связей:

- «Студент» : «Группа» = 1 : M (один студент — в одной группе; в группе — много студентов).
- «Студент» : «Оценка» = 1 : M.
- «Предмет» : «Оценка» = 1 : M.

7. **Проверка нормализации**

Убедитесь, что:

- Нет дублирования данных (например, название группы хранится только в таблице «Группа»).
- Каждый атрибут зависит только от первичного ключа.

8. **Оформление схемы БД**

Представьте структуру в виде:

- Текстового описания таблиц с полями, типами данных, PK и FK.
- Графической схемы (ER-диаграммы) с таблицами и связями.

Пример реализации (учебный процесс)

Таблица «Студент»

Поле	Тип данных	Ключ
`student_id`	INT	PK

Поле	Тип данных	Ключ
full_name	VARCHAR(100)	—
birth_date	DATE	—
group_id	INT	FK → «Группа».group_id

Таблица «Группа»

Поле	Тип данных	Ключ
group_id	INT	PK
name	VARCHAR(20)	—
course	INT	—

Таблица «Предмет»

Поле	Тип данных	Ключ
subject_id	INT	PK
title	VARCHAR(50)	—
hours	INT	—

Таблица «Оценка»

Поле	Тип данных	Ключ
student_id	INT	PK, FK → «Студент».student_id
subject_id	INT	PK, FK → «Предмет».subject_id
grade	DECIMAL(2,1)	—
date	DATE	—

Связи:

- student_id в «Оценка» → student_id в «Студент» (1 : M).
- subject_id в «Оценка» → subject_id в «Предмет» (1 : M).
- group_id в «Студент» → group_id в «Группа» (1 : M).

Требования к отчёту

1. Описание предметной области и списка сущностей.
2. Таблицы с полями, типами данных, PK и FK.
3. Схема БД (текстовая или графическая).
4. Перечень установленных связей с указанием типов.

5. Краткий анализ нормализации (отсутствие избыточности, зависимость атрибутов от PK).

Контрольные вопросы

1. Что такое первичный ключ? Приведите пример.
2. Для чего нужен внешний ключ? Как он обеспечивает целостность данных?
3. В чём разница между связями «один-ко-многим» и «многие-ко-многим»?
4. Почему важно избегать дублирования данных в реляционной БД?
5. Как проверить, что таблица находится в третьей нормальной форме?

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 9 Определение индексов для оптимизации запросов к базе данных.

Цель работы: освоить методику выбора и создания индексов для повышения производительности SQL-запросов

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

Индекс — это дополнительная структура данных, создаваемая на основе столбцов таблицы. Он ускоряет поиск, сортировку и фильтрацию за счёт упорядоченной организации данных (чаще всего — на основе B-деревьев).

Основные типы индексов

1. **Первичный индекс**
   - Создаётся автоматически для столбца с `PRIMARY KEY`.
   - Гарантирует уникальность значений и быстрый доступ по идентификатору.
   - Пример:

```
sql
CREATE TABLE orders (
    order_id SERIAL PRIMARY KEY,  -- автоматически индексируется
    customer_id INT,
    order_date DATE
);
```

2. **Уникальный индекс**
   - Предотвращает дублирование значений в столбце.
   - Пример:

```
sql
CREATE UNIQUE INDEX idx_email ON users(email);
```

3. **Составной индекс**
   - Охватывает несколько столбцов. Эффективен для запросов с фильтрацией по всем или первым полям индекса.
   - Пример:

```
sql
CREATE INDEX idx_orders_customer_date ON orders(customer_id, order_date);
```

4. **Полнотекстовый индекс**
   - Оптимизирован для поиска по текстовым данным.
5. **Хеш-индекс**
   - Эффективен для точного поиска, но не поддерживает диапазонные запросы.

Когда индексы полезны

- Столбцы часто используются в `WHERE`, `JOIN`, `ORDER BY`, `GROUP BY`.
- Данные имеют высокую селективность (много уникальных значений).
- Запросы возвращают небольшую часть строк таблицы.

Когда индексы вредны

- Таблица часто обновляется (`INSERT`/`UPDATE`/`DELETE`).
- Столбец имеет низкую селективность (например, пол: «М»/«Ж»).
- Индекс редко используется в запросах.

Методика оптимизации

1. **Анализ проблемных запросов**
   - Используйте `EXPLAIN` для просмотра плана выполнения:

```
sql
EXPLAIN SELECT * FROM orders WHERE order_date > '2024-01-01';
```

- Ищите `Seq Scan` (полное сканирование таблицы) — сигнал к созданию индекса.

2. **Выбор столбцов для индексации**
   - Приоритет: столбцы в `WHERE` и условиях `JOIN`.
   - Для сортировок — столбцы из `ORDER BY`.
   - Для агрегаций — столбцы из `GROUP BY`.
3. **Создание индекса**

```sql
CREATE INDEX idx_name ON table_name(column1, column2);
```

4. **Проверка эффективности**
   - Сравните время выполнения запроса до и после создания индекса.
   - Повторно запустите `EXPLAIN`, чтобы убедиться в использовании индекса.
5. **Обслуживание индексов**
   - Периодически выполняйте `REINDEX` для устранения фрагментации:

```sql
REINDEX TABLE table_name;
```

   - Удаляйте неиспользуемые индексы:

```sql
DROP INDEX idx_unused;
```

Практическое задание

Шаг 1. Подготовка данных

Создайте таблицу `sales` с полями:

- `id` (PRIMARY KEY),
- `product_id` (INT),
- `sale_date` (DATE),
- `amount` (DECIMAL),
- `region` (VARCHAR).

Шаг 2. Анализ запросов

Выполните `EXPLAIN` для следующих запросов:

1. `SELECT * FROM sales WHERE product_id = 123;`
2. `SELECT region, SUM(amount) FROM sales GROUP BY region;`
3. `SELECT * FROM sales WHERE sale_date BETWEEN '2024-01-01' AND '2024-12-31';`

Шаг 3. Создание индексов

На основании анализа `EXPLAIN` создайте индексы для оптимизации:

- Запроса 1: `CREATE INDEX idx_product ON sales(product_id);`
- Запроса 3: `CREATE INDEX idx_sale_date ON sales(sale_date);`

Шаг 4. Проверка результатов

- Повторно выполните `EXPLAIN` для тех же запросов.
- Сравните время выполнения до и после индексации (используйте `EXPLAIN (ANALYZE)`).

Шаг 5. Оптимизация составного запроса

Для запроса:

```sql
SELECT * FROM sales
WHERE product_id = 123
  AND sale_date >= '2024-01-01';
```

Создайте составной индекс:

```sql
CREATE INDEX idx_product_date ON sales(product_id, sale_date);
```

Шаг 6. Анализ избыточности

Проверьте, можно ли удалить отдельные индексы `idx_product` и `idx_sale_date` после создания составного индекса.

Контрольные вопросы

1. Почему индекс на столбце с низкой селективностью может ухудшить производительность?
2. В чём разница между кластерным и некластерным индексом?
3. Как команда `REINDEX` влияет на производительность?
4. Почему важно удалять неиспользуемые индексы?
5. В каких случаях составной индекс эффективнее одиночных?

Требования к отчёту

1. Текст выполненных SQL-запросов (создание таблицы, индексов, `EXPLAIN`).
2. Скриншоты планов выполнения до и после оптимизации.
3. Таблица сравнения времени выполнения запросов.
4. Выводы о эффективности созданных индексов.
5. Рекомендации по дальнейшему улучшению производительности.

Критерии оценки

- Корректность выбора столбцов для индексации.
- Обоснованность создания составных индексов.
- Точность анализа планов выполнения.
- Полнота отчёта и выводов.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 10 Проектирование базы данных для хранения данных IoT (Интернет вещей) с учётом особенностей структуры.

Цель работы изучить, как управлять учётными записями пользователей и контролировать доступ к базам данных

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

1. Анализ предметной области и требований

Особенности IoT-данных:

- **Высокий объём и скорость поступления** — тысячи устройств генерируют данные непрерывно.
- **Разнообразие форматов** — числовые показания, строки, JSON, бинарные данные.
- **Временная привязка** — каждое значение имеет метку времени (timestamp).
- **Неравномерность потоков** — пики активности, пропуски, дубликаты.
- **Масштабируемость** — число устройств может расти экспоненциально.
- **Требования к задержкам** — для критичных сценариев нужна обработка в реальном времени.

Типичные сущности:

- Устройства (датчики, актуаторы, шлюзы).
- Показания датчиков (температура, влажность, вибрация и т. д.).
- Метаданные устройств (модель, местоположение, статус).
- Пользователи и роли.
- События и тревоги.

2. Выбор типа СУБД

Для IoT подходят:

- **Базы данных временных рядов (TSDB)** — оптимизированы для timestamp-данных:
  - InfluxDB,
  - TimescaleDB (расширение PostgreSQL),
  - OpenTSDB.
- **Колоночные СУБД** — эффективны для агрегации и аналитики:
  - ClickHouse,
  - Apache Druid.
- **Документные СУБД** — гибкость схемы для разнородных данных:
  - MongoDB,
  - Couchbase.
- **Гибридные решения** — сочетание реляционной и временной моделей (например, PostgreSQL + TimescaleDB).

Рекомендация: для баланса гибкости и производительности выбрать **TimescaleDB** (на базе PostgreSQL).

3. Логическая модель данных (ER-диаграмма)

Сущности и атрибуты:

1. `device` (устройство):
   - `device_id` (PK, UUID),
   - `name` (строка),
   - `model` (строка),
   - `location` (точка GPS: широта/долгота),
   - `status` (enum: active, offline, maintenance),
   - `last_seen` (timestamp).
2. `sensor` (датчик):
   - `sensor_id` (PK, UUID),
   - `device_id` (FK),
   - `type` (строка: temperature, humidity и т. д.),
   - `unit` (строка: °C, % и т. д.).
3. `measurement` (показание):
   - `measurement_id` (PK, автоинкремент),
   - `sensor_id` (FK),

- `value` (float),
- `timestamp` (timestamp with time zone),
- `quality` (enum: good, suspect, invalid).

4. `event` (событие/тревога):
   - `event_id` (PK, UUID),
   - `device_id` (FK),
   - `type` (строка: low_battery, anomaly и т. д.),
   - `triggered_at` (timestamp),
   - `resolved` (boolean).
5. `user` (пользователь):
   - `user_id` (PK, UUID),
   - `email` (уникальный),
   - `role` (enum: admin, operator, guest).

Связи:

- `device` 1 → N `sensor` (одно устройство имеет много датчиков).
- `sensor` 1 → N `measurement` (один датчик генерирует много показаний).
- `device` 1 → N `event` (одно устройство может генерировать много событий).

4. Физическая модель (SQL-схема для TimescaleDB)

```
sql
-- Создание гипертаблицы для показаний
CREATE TABLE measurement (
    measurement_id BIGSERIAL PRIMARY KEY,
    sensor_id UUID NOT NULL,
    value DOUBLE PRECISION NOT NULL,
    timestamp TIMESTAMP WITH TIME ZONE NOT NULL,
    quality VARCHAR(20) DEFAULT 'good'
);

SELECT create_hypertable('measurement', 'timestamp', chunk_time_interval => INTERVAL
'1 day');



-- Таблицы остальных сущностей
CREATE TABLE device (
    device_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(100) NOT NULL,
    model VARCHAR(50),
    location POINT,  -- (долгота, широта)
    status VARCHAR(20) CHECK (status IN ('active', 'offline', 'maintenance')),
```

```
    last_seen TIMESTAMP WITH TIME ZONE
);

CREATE TABLE sensor (
    sensor_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    device_id UUID REFERENCES device(device_id) ON DELETE CASCADE,
    type VARCHAR(50) NOT NULL,
    unit VARCHAR(20)
);

CREATE TABLE event (
    event_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    device_id UUID REFERENCES device(device_id) ON DELETE CASCADE,
    type VARCHAR(50) NOT NULL,
    triggered_at TIMESTAMP WITH TIME ZONE NOT NULL,
    resolved BOOLEAN DEFAULT FALSE
);

CREATE TABLE user (
    user_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    role VARCHAR(20) CHECK (role IN ('admin', 'operator', 'guest'))
);
```

5. Индексы и оптимизация

Рекомендуемые индексы:

- Для `measurement`:
  - Составной индекс по `(sensor_id, timestamp)` — ускорит запросы по датчику и времени.
  - Индекс по `timestamp` — для временных диапазонов.
- Для `device`: индекс по `last_seen` — мониторинг активности.
- Для `event`: индекс по `triggered_at` и `resolved` — анализ тревог.

Настройки TimescaleDB:

- Интервал чанков: 1 день (оптимально для IoT).
- Политика сохранения данных (data retention): автоматическое удаление записей старше 1 года.
- Компрессия для старых чанков — снижение объёма хранилища.

6. Примеры запросов

1. **Последние показания датчика:**

```
sql
SELECT value, timestamp
FROM measurement
WHERE sensor_id = '...' AND timestamp > NOW() - INTERVAL '1 hour'
ORDER BY timestamp DESC;
```

2. **Средняя температура за сутки:**

```sql
SELECT DATE_TRUNC('day', timestamp) AS day, AVG(value) AS avg_temp
FROM measurement
JOIN sensor ON measurement.sensor_id = sensor.sensor_id
WHERE sensor.type = 'temperature'
  AND timestamp > NOW() - INTERVAL '7 days'
GROUP BY day
ORDER BY day;
```

3. **Активные тревоги:**

```sql
SELECT device_id, type, triggered_at
FROM event
WHERE resolved = FALSE
ORDER BY triggered_at DESC;
```

7. Безопасность и доступ

- **Ролевая модель:**
  - `admin` — полный доступ, управление устройствами.
  - `operator` — просмотр данных, подтверждение тревог.
  - `guest` — только чтение агрегатных данных.
- **Шифрование:**
  - TLS для передачи данных.
  - Шифрование на диске (например, LUKS).
- **Аудит:** логирование действий пользователей.

8. Масштабирование и отказоустойчивость

- **Репликация:** мастер-слейв для чтения.
- **Шардирование:** по `device_id` или временным интервалам.
- **Резервное копирование:** ежедневные снапшоты TimescaleDB.
- **Мониторинг:** метрики PostgreSQL (загрузка CPU, IO, размер БД).

9. Интеграция с IoT-платформой

Потоковая обработка:

- Использование **Apache Kafka** для буферизации данных перед записью в БД.
- **Kafka Connect** — коннектор для TimescaleDB.

Визуализация:

- Подключение к **Grafana** для дашбордов (поддержка TimescaleDB «из коробки»).

10. Заключение

Итоговые рекомендации:

1. Использовать **TimescaleDB** как основу — баланс между SQL-гибкостью и оптимизацией для временных рядов.
2. Применять **гипертаблицы** и **чанки** для управления жизненным циклом данных.
3. Реализовать **индексы** по ключевым полям (`sensor_id`, `timestamp`).
4. Настроить **репликацию и бэкапы** для отказоустойчивости.
5. Интегрировать с **Kafka и Grafana** для полного стека IoT.

Критерии успеха:

- Задержка записи данных < 100 мс.
- Время отклика запросов < 500 мс для последних 24 часов.
- Возможность масштабирования до 100 000 устройств.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Тема 2.2. Разработка и администрирование БД.

Практическая работа 11 Создание базы данных и таблиц с использованием языка SQL (CREATE DATABASE, CREATE TABLE).

Цель работы: освоить базовые команды SQL для создания базы данных и таблиц: CREATE DATABASE и CREATE TABLE. Научиться выбирать активную базу данных и просматривать структуру созданных объектов.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретическая часть

1. Создание базы данных: `CREATE DATABASE`

Синтаксис:

```
sql
CREATE DATABASE имя_базы_данных;
```

Важные правила:

- имя БД может содержать до 64 символов;
- допустимы буквы, цифры, символы `_` и `$`;
- имя может начинаться с цифры, но не может состоять только из цифр;
- каждый SQL-запрос завершается точкой с запятой `;`.

2. Выбор базы данных: `USE`

Перед созданием таблиц нужно указать, в какой базе данных они будут размещены:

```
sql
USE имя_базы_данных;
```

3. Создание таблицы: `CREATE TABLE`

Синтаксис:

```
sql
CREATE TABLE имя_таблицы (
   имя_столбца1 тип_данных [ограничения],
   имя_столбца2 тип_данных [ограничения],
   ...
);
```

Основные типы данных MySQL:

- `INT` — целые числа;
- `VARCHAR(n)` — строки длиной до *n* символов;
- `TEXT` — длинные текстовые данные;
- `DATE` — дата;
- `DATETIME` — дата и время;
- `FLOAT` — числа с плавающей точкой.

Распространённые ограничения (константы):

- `NOT NULL` — значение обязательно;
- `PRIMARY KEY` — первичный ключ (уникальный идентификатор);
- `UNIQUE` — уникальное значение;
- `DEFAULT значение` — значение по умолчанию.

4. Просмотр информации

- `SHOW DATABASES` — список всех БД;
- `SHOW TABLES` — список таблиц в текущей БД;
- `DESCRIBE имя_таблицы` — структура таблицы (столбцы, типы данных).

Практическая часть

Шаг 1. Создание базы данных

1. Откройте клиент командной строки MySQL (или среду разработки, например, MySQL Workbench).

2. Введите команду для создания БД:

```
sql
CREATE DATABASE school;
```

3. Подтвердите создание, выполнив:

```
sql
SHOW DATABASES;
```

В списке должна появиться база данных `school`.

Шаг 2. Выбор базы данных

Активируйте созданную БД:

```
sql
USE school;
```

После выполнения команды вы увидите сообщение: `Database changed`.

Шаг 3. Создание таблиц

Таблица 1: `students` **(студенты)**

```
sql
CREATE TABLE students (
    id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    birth_date DATE,
    grade INT
);
```

Таблица 2: `courses` **(курсы)**

```
sql
CREATE TABLE courses (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(100) NOT NULL,
    credits INT DEFAULT 3
);
```

Таблица 3: `enrollments` **(зачисления)**

```
sql
CREATE TABLE enrollments (
    student_id INT,
    course_id INT,
    enrollment_date DATE,
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES students(id),
    FOREIGN KEY (course_id) REFERENCES courses(id)
);
```

Примечание:

- `AUTO_INCREMENT` — автоматически увеличивает значение поля на 1 при добавлении новой записи;
- `FOREIGN KEY` — устанавливает связь между таблицами (требует поддержки InnoDB).

Шаг 4. Проверка структуры

1. Убедитесь, что таблицы созданы:

```
sql
SHOW TABLES;
```

В выводе должны быть: `students`, `courses`, `enrollments`.

2. Проверьте структуру таблицы `students`:

```
sql
DESCRIBE students;
```

Ожидаемый результат:

```
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int         | NO   | PRI | NULL    | auto_increment |
| first_name | varchar(50) | NO   |     | NULL    |                |
| last_name  | varchar(50) | NO   |     | NULL    |                |
| birth_date | date        | YES  |     | NULL    |                |
| grade      | int         | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
```

Контрольные вопросы

1. Какова максимальная длина имени базы данных в MySQL?
2. Зачем нужна команда `USE`?
3. Что означает ограничение `PRIMARY KEY`?
4. Какой тип данных подойдёт для хранения даты рождения?
5. Как просмотреть список всех таблиц в текущей базе данных?

Дополнительные задания (по желанию)

1. Создайте базу данных `library` и таблицу `books` со столбцами:
   - `id` (первичный ключ),
   - `title` (название, обязат.),
   - `author` (автор),
   - `year` (год издания),
   - `is_available` (доступна, по умолчанию `TRUE`).
2. Добавьте в таблицу `books` запись о книге «Война и мир» Л. Н. Толстого (1869 г.).
3. Выведите структуру таблицы `books` с помощью `DESCRIBE`.

Ответы на контрольные вопросы (кратко)

1. 64 символа.
2. Для выбора активной базы данных перед выполнением операций.
3. Уникальный идентификатор записи, не может быть `NULL`.
4. `DATE`.
5. `SHOW TABLES;`.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.
Сдать и защитить работу.

Практическая работа 12 Реализация ограничений целостности (PRIMARY KEY, FOREIGN KEY, UNIQUE) в таблицах базы данных.

Цель работы: освоить на практике создание и использование ключевых ограничений целостности данных в реляционных базах данных: PRIMARY KEY, FOREIGN KEY и UNIQUE.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

Ограничения целостности — это правила, обеспечивающие корректность и согласованность данных в базе данных. Основные типы:

1. **PRIMARY KEY (первичный ключ)**
   - однозначно идентифицирует каждую запись в таблице;
   - не допускает NULL-значений;
   - должен быть уникальным;
   - в таблице может быть только один PRIMARY KEY.
2. **FOREIGN KEY (внешний ключ)**
   - связывает две таблицы, ссылаясь на PRIMARY KEY или UNIQUE-столбец другой таблицы;

- обеспечивает ссылочную целостность: нельзя добавить запись с несуществующим внешним ключом, удалить родительскую запись при наличии связанных дочерних (без специальных опций).

3. **UNIQUE (уникальность)**
   - гарантирует уникальность значений в столбце или группе столбцов;
   - допускает одно NULL-значение (так как `NULL ≠ NULL`);
   - может быть несколько UNIQUE-ограничений в таблице.

Ход работы

Шаг 1. Создание базовых таблиц

Создадим две связанные таблицы: `Users` (пользователи) и `Orders` (заказы).

```sql
-- Таблица пользователей
CREATE TABLE Users (
    user_id INT PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    registration_date DATE NOT NULL
);

-- Таблица заказов
CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    user_id INT NOT NULL,
    order_date DATE NOT NULL,
    total_amount DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
);
```

Пояснения:

- В `Users`:
  - `user_id` — PRIMARY KEY;
  - `username` и `email` — UNIQUE + NOT NULL (уникальные и обязательные);
  - `registration_date` — NOT NULL (обязательное поле).
- В `Orders`:
  - `order_id` — PRIMARY KEY;
  - `user_id` — FOREIGN KEY, ссылается на `Users.user_id`;
  - опции:
    - `ON DELETE RESTRICT` — запрещает удаление пользователя, если у него есть заказы;

- `ON UPDATE CASCADE` — при обновлении `user_id` в `Users` автоматически обновит `user_id` в `Orders`.

Шаг 2. Проверка ограничений

Тест 1. PRIMARY KEY

Попытка вставить дубликат `user_id`:

```
sql
INSERT INTO Users (user_id, username, email, registration_date)
VALUES (1, 'alice', 'alice@example.com', '2025-01-01');

INSERT INTO Users (user_id, username, email, registration_date)
VALUES (1, 'bob', 'bob@example.com', '2025-01-02');  -- Ошибка: дубликат PRIMARY KEY
```

Ожидаемый результат: ошибка `Duplicate entry '1' for key 'PRIMARY'`.

Тест 2. UNIQUE

Попытка вставить дубликат `email`:

```
sql
INSERT INTO Users (user_id, username, email, registration_date)
VALUES (2, 'charlie', 'alice@example.com', '2025-01-03');  -- Ошибка: дубликат UNIQUE
```

Ожидаемый результат: ошибка `Duplicate entry 'alice@example.com' for key 'email'`.

Тест 3. FOREIGN KEY

Попытка вставить заказ для несуществующего пользователя:

```
sql
INSERT INTO Orders (order_id, user_id, order_date, total_amount)
VALUES (101, 999, '2025-01-04', 100.00);  -- Ошибка: внешний ключ не существует
```

Ожидаемый результат: ошибка `Cannot add or update a child row: a foreign key constraint fails`.

Шаг 3. Модификация таблиц (добавление ограничений)

Добавим UNIQUE-ограничение на столбец `phone` в таблице `Users`:

```
sql
ALTER TABLE Users
ADD COLUMN phone VARCHAR(20),
ADD CONSTRAINT uq_phone UNIQUE (phone);
```

Проверим:

```
sql
INSERT INTO Users (user_id, username, email, registration_date, phone)
VALUES (3, 'diana', 'diana@example.com', '2025-01-05', '123-456-7890');

INSERT INTO Users (user_id, username, email, registration_date, phone)
VALUES (4, 'eve', 'eve@example.com', '2025-01-06', '123-456-7890');  -- Ошибка: дубликат phone
```

Ожидаемый результат: ошибка `Duplicate entry '123-456-7890' for key 'uq_phone'`.

Шаг 4. Удаление ограничений

Удалим UNIQUE-ограничение на `phone`:

```
sql
ALTER TABLE Users
DROP INDEX uq_phone;  -- Для MySQL/MariaDB
-- Или:
-- ALTER TABLE Users DROP CONSTRAINT uq_phone;  -- Для PostgreSQL/SQL Server
```

Шаг 5. Ссылочная целостность (FOREIGN KEY)

Проверим поведение при удалении пользователя:

```
sql
-- Вставим заказ для пользователя с user_id = 1
INSERT INTO Orders (order_id, user_id, order_date, total_amount)
VALUES (102, 1, '2025-01-07', 200.00);

-- Попытка удалить пользователя с заказами
DELETE FROM Users WHERE user_id = 1;  -- Ошибка: ON DELETE RESTRICT
```

Ожидаемый результат: ошибка `Cannot delete or update a parent row: a foreign key constraint fails`.

Изменим поведение FOREIGN KEY на `ON DELETE CASCADE`:

```
sql
ALTER TABLE Orders
DROP FOREIGN KEY fk_user;  -- Удаляем старое ограничение (имя может отличаться)

ALTER TABLE Orders
ADD CONSTRAINT fk_user
FOREIGN KEY (user_id) REFERENCES Users(user_id)
   ON DELETE CASCADE
   ON UPDATE CASCADE;
```

Теперь удаление пользователя автоматически удалит его заказы:

```
sql
DELETE FROM Users WHERE user_id = 1;  -- Успешно: заказы тоже удалены
```

Контрольные вопросы

1. Чем отличается PRIMARY KEY от UNIQUE?
2. Что произойдёт при попытке вставить запись с NULL в столбец с PRIMARY KEY?
3. Какие опции можно указать для FOREIGN KEY при удалении родительской записи?
4. Можно ли иметь несколько UNIQUE-ограничений в одной таблице?
5. Как добавить FOREIGN KEY в уже существующую таблицу?

Требования к отчёту

Отчёт должен содержать:

1. Цель работы.
2. SQL-код создания таблиц с комментариями.

3. Примеры тестов для каждого типа ограничений (PRIMARY KEY, UNIQUE, FOREIGN KEY).

4. Результаты выполнения тестов (ошибки/успехи).

5. Ответы на контрольные вопросы.

6. Выводы по работе.

Возможные ошибки и их устранение

- **Ошибка дубликата ключа:** проверьте, не вставляете ли вы уже существующее значение в PRIMARY KEY/UNIQUE-столбец.
- **Ошибка FOREIGN KEY:** убедитесь, что значение внешнего ключа существует в родительской таблице.
- **Ошибка синтаксиса:** проверьте правильность написания команд (например, `FOREIGN KEY` вместо `FOREIGN KEY`).

Вывод

В ходе работы освоены:

- создание таблиц с ограничениями PRIMARY KEY, UNIQUE и FOREIGN KEY;
- проверка ограничений на практике (вставка дубликатов, нарушение ссылочной целостности);
- модификация таблиц (добавление/удаление ограничений);
- управление поведением FOREIGN KEY (`ON DELETE`, `ON UPDATE`).

Эти навыки необходимы для обеспечения целостности и согласованности данных в реальных базах данных.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 13 Написание и выполнение SQL-запросов для добавления, изменения и удаления данных (INSERT, UPDATE, DELETE).

Цель работы: освоить базовые операции манипуляции данными в SQL: добавление (`INSERT`), обновление (`UPDATE`) и удаление (`DELETE`) записей в таблицах базы данных.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

• Для выполнения данного задания необходимо изучить теоретический материал по данной теме.

- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

• Предъявить преподавателю результаты работы.

• Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).

• Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретическая часть

Основные операторы DML (Data Manipulation Language) для изменения данных:

1. `INSERT` — добавление новых записей в таблицу.
2. `UPDATE` — изменение существующих записей.
3. `DELETE` — удаление записей из таблицы.

Важные правила:

- Всегда указывайте условие `WHERE` в `UPDATE` и `DELETE`, если не хотите изменить/удалить все строки.
- Без `WHERE` `UPDATE` изменит все строки таблицы, а `DELETE` удалит все записи.
- Для `INSERT` обязательно соответствие количества и типов данных в `VALUES` и перечисляемых столбцах.

Практическая часть

1. Подготовка

Предположим, есть таблица `employees` со структурой:

```
sql
CREATE TABLE employees (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    position VARCHAR(50),
    salary DECIMAL(10, 2),
    hire_date DATE
);
```

2. Добавление данных (INSERT)

Пример 1. Вставка одной строки:

```
sql
INSERT INTO employees (id, name, position, salary, hire_date)
VALUES (1, 'Иван Иванов', 'Менеджер', 50000.00, '2023-01-15');
```

Пример 2. Вставка нескольких строк сразу:

```sql
INSERT INTO employees (id, name, position, salary, hire_date)
VALUES
  (2, 'Мария Петрова', 'Аналитик', 45000.00, '2023-02-10'),
  (3, 'Алексей Сидоров', 'Разработчик', 70000.00, '2023-03-05');
```

Пример 3. Вставка с частичным указанием столбцов (остальные получат `NULL` или значения по умолчанию):

```sql
INSERT INTO employees (id, name, position)
VALUES (4, 'Анна Козлова', 'Тестировщик');
```

3. Изменение данных (UPDATE)

Пример 1. Обновление зарплаты одного сотрудника:

```sql
UPDATE employees
SET salary = 55000.00
WHERE id = 1;
```

Пример 2. Обновление должности и зарплаты для нескольких сотрудников:

```sql
UPDATE employees
SET position = 'Старший аналитик', salary = 50000.00
WHERE position = 'Аналитик';
```

Пример 3. Увеличение зарплаты на 10% для всех разработчиков:

```sql
UPDATE employees
SET salary = salary * 1.10
WHERE position = 'Разработчик';
```

Пример 4. Обновление даты приёма на работу:

```sql
UPDATE employees
SET hire_date = '2023-04-01'
WHERE id = 3;
```

4. Удаление данных (DELETE)

Пример 1. Удаление сотрудника по ID:

```sql
DELETE FROM employees
WHERE id = 4;
```

Пример 2. Удаление всех сотрудников на определённой должности:

```sql
DELETE FROM employees
WHERE position = 'Тестировщик';
```

Пример 3. Удаление сотрудников с зарплатой ниже определённого порога:

```sql
DELETE FROM employees
WHERE salary < 40000.00;
```

Пример 4. Удаление всех записей (осторожно!):

```sql
DELETE FROM employees;
```

Внимание! Без условия `WHERE` команда `DELETE` удалит ВСЕ строки из таблицы. Всегда проверяйте условие перед выполнением.

5. Проверка результатов

После каждого изменения полезно проверять данные:

```sql
SELECT * FROM employees;
```

Задания для самостоятельной работы

1. Создайте таблицу `students` со столбцами:
   - `id` (целое, первичный ключ),
   - `full_name` (строка),
   - `group_number` (строка),
   - `gpa` (число с плавающей точкой).
2. Добавьте 5 записей о студентах с помощью `INSERT`.
3. Обновите GPA одного студента с помощью `UPDATE`.
4. Увеличьте GPA всех студентов группы «ИТ-11» на 0,2.
5. Удалите студента с минимальным GPA.
6. Выведите все оставшиеся записи с помощью `SELECT`.

Контрольные вопросы

1. В чём разница между `INSERT INTO table VALUES (...)` и `INSERT INTO table (col1, col2) VALUES (...)`?
2. Что произойдёт, если выполнить `UPDATE` без условия `WHERE`?
3. Можно ли отменить действие `DELETE`? Как?
4. Какие типы данных подходят для столбца `salary`? Почему?
5. Как добавить запись, если некоторые поля могут быть `NULL`?

Требования к отчёту

Отчёт должен содержать:

- Текст всех выполненных SQL-запросов.
- Скриншоты результатов выполнения (например, вывод `SELECT * FROM ...`).
- Ответы на контрольные вопросы.
- Выводы по работе (что узнали, с какими сложностями столкнулись).

Критерии оценки

- Правильность написания SQL-запросов.
- Корректность результатов выполнения.
- Полнота отчёта.
- Грамотность ответов на контрольные вопросы.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 14 Настройка индексов для оптимизации производительности запросов (CREATE INDEX).

Цель работы: освоить создание и применение индексов в SQL для ускорения выполнения запросов, изучить типы индексов и сценарии их эффективного использования.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

Индекс — это дополнительная структура данных, создаваемая на основе столбцов таблицы. Он ускоряет поиск строк, заменяя полное сканирование таблицы (*Full Table Scan*) на быстрый доступ через упорядоченную структуру (чаще всего — B-дерево).

Принцип работы:

- Без индекса: СУБД последовательно проверяет каждую строку таблицы.
- С индексом: СУБД использует индекс как «указатель» к нужным строкам.

Основные типы индексов:

1. **Первичный индекс** (`PRIMARY KEY`) — создаётся автоматически, гарантирует уникальность и ускоряет поиск по ключевому полю.
2. **Уникальный индекс** (`UNIQUE`) — исключает дублирование значений в столбце.
3. **Составной индекс** — строится по нескольким столбцам, ускоряет запросы с фильтрацией по этим полям.
4. **Частичный индекс** — применяется к подмножеству строк (например, только активным записям).

5. **Индекс по выражению** — индексирует результат функции (например, `lower(column)`).

Ограничения индексов:

- Занимают дополнительное место на диске.
- Замедляют операции `INSERT`, `UPDATE`, `DELETE` (так как индекс нужно обновлять).
- Эффективны только для селективных запросов (малая доля строк удовлетворяет условию).

Практические задания

Задание 1. Создание простого индекса

Задача: ускорить запрос на поиск сотрудников по отделу.

Шаг 1. Создайте таблицу:

```
sql
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    department VARCHAR(50),
    salary DECIMAL(10, 2)
);
```

Шаг 2. Добавьте индекс на столбец `department`:

```
sql
CREATE INDEX idx_department ON employees(department);
```

Шаг 3. Проверьте ускорение запроса:

```
sql
SELECT * FROM employees WHERE department = 'HR';
```

До индекса — полное сканирование таблицы. После — поиск по индексу.

Задание 2. Уникальный индекс

Задача: гарантировать уникальность email-адресов в таблице пользователей.

Шаг 1. Создайте таблицу:

```
sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(100)
);
```

Шаг 2. Добавьте уникальный индекс:

```
sql
CREATE UNIQUE INDEX idx_email ON users(email);
```

Шаг 3. Попробуйте вставить дубликат:

```
sql
INSERT INTO users (email) VALUES ('user@example.com');
INSERT INTO users (email) VALUES ('user@example.com'); -- Ошибка!
```

Задание 3. Составной индекс

Задача: оптимизировать запрос с фильтрацией по двум полям.

Шаг 1. Создайте таблицу заказов:

```sql
CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    total DECIMAL(10, 2)
);
```

Шаг 2. Создайте составной индекс:

```sql
CREATE INDEX idx_orders_customer_date ON orders(customer_id, order_date);
```

Шаг 3. Проверьте запрос:

```sql
SELECT * FROM orders
WHERE customer_id = 101
  AND order_date >= '2024-01-01';
```

Индекс ускорит фильтрацию по обоим полям.

Задание 4. Частичный индекс

Задача: ускорить запросы к активным продуктам.

Шаг 1. Создайте таблицу:

```sql
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    price DECIMAL(10, 2),
    is_active BOOLEAN
);
```

Шаг 2. Создайте частичный индекс:

```sql
CREATE INDEX idx_active_products ON products(price)
WHERE is_active = true;
```

Шаг 3. Проверьте запрос:

```sql
SELECT * FROM products
WHERE is_active = true AND price > 1000;
```

Индекс будет использоваться только для активных записей.

Задание 5. Индекс по выражению

Задача: ускорить поиск без учёта регистра.

Шаг 1. Создайте таблицу книг:

```sql
CREATE TABLE books (
    id SERIAL PRIMARY KEY,
    title VARCHAR(200),
    author VARCHAR(100)
);
```

Шаг 2. Создайте индекс по результату функции `lower`:

```sql
CREATE INDEX book_lower_author_idx ON books(lower(author));
```

Шаг 3. Проверьте запрос:

```sql
SELECT * FROM books WHERE lower(author) = 'charles bukowski';
```

Индекс ускорит поиск, игнорируя регистр.

Контроль эффективности индексов

1. **Анализ плана запроса** (`EXPLAIN`):

```sql
EXPLAIN SELECT * FROM employees WHERE department = 'HR';
```

Ищите строки с `Index Scan` *вместо* `Seq Scan`.

2. **Проверка использования индексов**:

```sql
SELECT * FROM pg_stat_user_indexes;
```

3. **Удаление неиспользуемых индексов**:

```sql
DROP INDEX idx_unused_index;
```

Рекомендации по оптимизации

1. Создавайте индексы только для столбцов, часто используемых в `WHERE`, `JOIN`, `ORDER BY`.
2. Для сложных фильтров применяйте составные индексы.
3. Избегайте индексов на часто изменяемых столбцах.
4. Периодически анализируйте нагрузку с помощью `EXPLAIN`.
5. Используйте частичные индексы для фильтрации подмножеств данных.

Контрольные вопросы

1. В чём отличие первичного индекса от уникального?
2. Почему составной индекс эффективен только при использовании всех его полей?
3. В каких случаях частичный индекс предпочтительнее полного?
4. Как проверить, используется ли индекс в запросе?
5. Какие минусы у чрезмерного количества индексов?

Итоговый отчёт

Подготовьте отчёт, включающий:

- Скрипты создания таблиц и индексов из заданий.
- Примеры запросов до и после индексации.
- Анализ плана выполнения (`EXPLAIN`) для одного из запросов.
- Выводы о влиянии индексов на производительность.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.
2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)
Сформулировать выводы по результатам работы.
Сдать и защитить работу.

Практическая работа 15 Реализация хранимых процедур и триггеров для автоматизации работы с базой данных.

Цель работы: освоить создание и применение хранимых процедур и триггеров в СУБД для автоматизации операций с данными, обеспечения целостности и реализации бизнес-логики.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

Хранимые процедуры

Хранимая процедура — это предварительно скомпилированный набор SQL-инструкций, сохранённый в базе данных под определённым именем.

Основные преимущества:

- повышение производительности (код компилируется один раз);
- централизация бизнес-логики;

- контроль доступа (разграничение прав через выполнение процедур);
- уменьшение сетевого трафика (клиент отправляет только вызов процедуры).

Синтаксис создания (общий вид):

```
sql
CREATE PROCEDURE имя_процедуры [(@параметр тип_данных [= значение_по_умолчанию])]
AS
   SQL-инструкции
GO
```

Основные команды управления:

- `CREATE PROCEDURE` — создание;
- `ALTER PROCEDURE` — изменение;
- `DROP PROCEDURE` — удаление;
- `EXEC имя_процедуры` — вызов.

Триггеры

Триггер — специальный тип хранимой процедуры, автоматически выполняющийся при возникновении события DML (`INSERT`, `UPDATE`, `DELETE`) на таблице или представлении.

Типы триггеров по времени выполнения:

- `BEFORE` — до операции (проверка/изменение данных);
- `AFTER` — после операции (логирование, обновление связанных таблиц);
- `INSTEAD OF` — вместо операции (для представлений).

Синтаксис создания (общий вид):

```
sql
CREATE TRIGGER имя_триггера
ON имя_таблицы
FOR [INSERT, UPDATE, DELETE]
AS
   SQL-инструкции
GO
```

Ключевые особенности:

- не вызываются явно (активируются событием);
- могут каскадно запускать другие триггеры;
- используются для:
  - поддержания ссылочной целостности;
  - аудита изменений;
  - сложных проверок данных.

Ход выполнения работы

Шаг 1. Подготовка среды

1. Выберите СУБД (например, **Microsoft SQL Server**, **PostgreSQL**, **MySQL**).
2. Создайте базу данных и таблицы для примера (например, `Orders`, `Customers`, `Products`).

3. Настройте права доступа для выполнения операций.

Шаг 2. Создание хранимых процедур

Пример 1. Процедура для добавления заказа:

```
sql
CREATE PROCEDURE AddOrder
    @CustomerID INT,
    @ProductID INT,
    @Quantity INT
AS
BEGIN
    INSERT INTO Orders (CustomerID, ProductID, Quantity, OrderDate)
    VALUES (@CustomerID, @ProductID, @Quantity, GETDATE())
END
GO
```

Вызов:

```
sql
EXEC AddOrder @CustomerID = 1, @ProductID = 5, @Quantity = 3
```

Пример 2. Процедура с выходным параметром (получение суммы заказа):

```
sql
CREATE PROCEDURE GetOrderTotal
    @OrderID INT,
    @Total MONEY OUTPUT
AS
BEGIN
    SELECT @Total = SUM(Quantity * Price)
    FROM OrderDetails
    WHERE OrderID = @OrderID
END
GO
```

Вызов:

```
sql
DECLARE @Sum MONEY
EXEC GetOrderTotal @OrderID = 10, @Total = @Sum OUTPUT
PRINT @Sum
```

Шаг 3. Создание триггеров

Пример 1. Триггер для аудита удалений:

```
sql
CREATE TRIGGER LogDeletedOrders
ON Orders
AFTER DELETE
AS
BEGIN
    INSERT INTO OrderLog (OrderID, Action, Timestamp)
    SELECT OrderID, 'DELETED', GETDATE()
    FROM deleted
```

```
END
GO
```

Пример 2. Триггер для проверки количества товара:

```
sql
CREATE TRIGGER CheckStock
ON OrderDetails
INSTEAD OF INSERT
AS
BEGIN
  IF EXISTS (
    SELECT 1
    FROM inserted i
    JOIN Products p ON i.ProductID = p.ProductID
    WHERE i.Quantity > p.StockQuantity
  )
  BEGIN
    RAISERROR ('Недостаточно товара на складе!', 16, 1)
    RETURN
  END
  ELSE
    INSERT INTO OrderDetails SELECT * FROM inserted
END
GO
```

Шаг 4. Тестирование

1. Выполните процедуры с разными параметрами.
2. Проверьте результаты в таблицах.
3. Протестируйте триггеры:
   - удалите запись из `Orders` (проверьте `OrderLog`);
   - попытайтесь добавить заказ с количеством > остатка (убедитесь в срабатывании проверки).

Шаг 5. Анализ и отладка

1. Используйте `PRINT` или `SELECT` для вывода промежуточных значений.
2. Проверьте план выполнения (`EXPLAIN` в PostgreSQL, `SET SHOWPLAN_ALL ON` в SQL Server).
3. Исправьте ошибки синтаксиса/логики.

Контрольные вопросы

1. В чём отличие хранимой процедуры от триггера?
2. Какие типы триггеров существуют и когда они применяются?
3. Как передать параметры в хранимую процедуру?
4. Почему триггеры не вызываются явно?
5. Приведите пример использования триггера для поддержания целостности данных.

Требования к отчёту

1. Текст задания.

2. Скрипты создания процедур и триггеров.
3. Примеры вызовов и результаты выполнения.
4. Скриншоты интерфейсов СУБД (при наличии).
5. Ответы на контрольные вопросы.
6. Выводы по работе.

Возможные ошибки и их устранение

- **Ошибка синтаксиса:** проверьте соответствие СУБД (синтаксис различается).
- **Нарушение прав доступа:** убедитесь в наличии прав CREATE PROCEDURE / CREATE TRIGGER.
- **Каскадное срабатывание триггеров:** ограничьте глубину вложенных вызовов.
- **Некорректные данные:** добавьте проверки в тела процедур/триггеров.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 16 Настройка учётных записей пользователей и управление их правами доступа к базе данных.

Цель работы: освоить приёмы создания учётных записей пользователей СУБД, назначения и ограничения прав доступа к объектам базы данных (на примере MySQL и PostgreSQL).

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю

результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические сведения

Управление пользователями и правами доступа — ключевая задача администратора БД. Основные операции:

- создание/удаление учётных записей;
- назначение паролей;
- выдача/отзыв привилегий;
- настройка правил подключения (хосты, схемы).

Основные типы прав в СУБД:

- `SELECT` — чтение данных;
- `INSERT` — добавление записей;
- `UPDATE` — изменение данных;
- `DELETE` — удаление записей;
- `CREATE` — создание объектов;
- `DROP` — удаление объектов;
- `ALL PRIVILEGES` — полный доступ.

Ход работы

1. Работа с MySQL

1.1. Создание пользователя

```
sql
CREATE USER 'username'@'localhost' IDENTIFIED BY 'password';
```

Примечание: `'localhost'` ограничивает подключение локальным хостом; `'%'` разрешает подключение с любых хостов.

1.2. Назначение прав

- Полный доступ ко всем БД:

```
sql
GRANT ALL PRIVILEGES ON *.* TO 'username'@'localhost';
```

- Доступ к конкретной БД:

```
sql
GRANT SELECT, INSERT, UPDATE ON db_name.* TO 'username'@'localhost';
```

- Доступ к отдельной таблице:

```
sql
GRANT DELETE ON db_name.table_name TO 'username'@'localhost';
```

1.3. Применение изменений

```
sql
FLUSH PRIVILEGES;
```

1.4. Проверка прав

```
sql
SHOW GRANTS FOR 'username'@'localhost';
```

1.5. Удаление пользователя

```
sql
```

```
DROP USER 'username'@'localhost';
```

2. Работа с PostgreSQL

2.1. Создание пользователя

```
sql
CREATE USER username WITH PASSWORD 'password';
```

2.2. Назначение прав

- Полный доступ к БД:

```
sql
GRANT ALL PRIVILEGES ON DATABASE db_name TO username;
```

- Доступ к таблицам схемы `public`:

```
sql
GRANT SELECT, INSERT ON ALL TABLES IN SCHEMA public TO username;
```

- Права на конкретную таблицу:

```
sql
GRANT UPDATE ON table_name TO username;
```

2.3. Настройка подключения

Изменить файл `pg_hba.conf`:

```
host all username 127.0.0.1/32 md5
```

Перезапустить службу:

```
bash
systemctl restart postgresql
```

2.4. Удаление пользователя

```
sql
DROP USER username;
```

Практические задания

1. **Создать двух пользователей** в MySQL и PostgreSQL:
   - `dev_user` (права на чтение/запись);
   - `report_user` (только чтение).
2. **Настроить доступ** к тестовой БД `test_db`:
   - `dev_user` — полный доступ ко всем таблицам;
   - `report_user` — только `SELECT` для таблицы `sales`.
3. **Проверить права** с помощью команд `SHOW GRANTS` (MySQL) и `\dp` (PostgreSQL).
4. **Отозвать право** `INSERT` у `dev_user` для таблицы `logs`.
5. **Удалить** `report_user` после выполнения заданий.

Контрольные вопросы

1. В чём разница между `'localhost'` и `'%'` в MySQL при создании пользователя?
2. Какая команда обновляет кэш привилегий в MySQL?
3. Какой файл настраивает правила подключения в PostgreSQL?
4. Как предоставить права на все таблицы схемы `public` в PostgreSQL?
5. Почему важно ограничивать права пользователей по принципу «минимальных привилегий»?

Требования к отчёту

Отчёт должен содержать:

1. Цель работы.
2. Последовательное описание выполненных действий (с командами).
3. Скриншоты результатов выполнения команд.

4. Ответы на контрольные вопросы.
5. Выводы по работе.

Критерии оценки

- Корректность выполнения команд — 40 %.
- Полнота отчёта — 30 %.
- Ответы на контрольные вопросы — 30 %.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 17 Оптимизация запросов к базе данных с использованием индексов и анализа плана выполнения запросов.

Цель работы: освоить методы оптимизации SQL-запросов через:

- создание и использование индексов;
- анализ планов выполнения запросов;
- выявление и устранение «узких мест» производительности.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).

• Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

1. Индексы: суть и виды

Индекс — структура данных, ускоряющая поиск строк в таблице за счёт упорядоченного хранения значений ключевых столбцов.

Основные типы индексов:

- **B-tree** — для диапазонных запросов и сортировки (*WHEREcol*>10, *ORDERBY*).
- **Hash** — для точных сравнений (*WHEREcol='value'*).
- **GiST/GIN** — для геоданных, полнотекстового поиска, JSON.
- **Кластерный** — определяет физический порядок строк в таблице.
- **Некластерный** — отдельная структура для поиска без изменения порядка строк.
- **Уникальный** — гарантирует отсутствие дубликатов в столбце.
- **Составной** — индекс по нескольким столбцам (порядок столбцов критичен).

2. План выполнения запроса

Это последовательность операций, которые СУБД выполнит для получения результата. Анализируется через команды:

- `EXPLAIN` — предполагаемый план (без выполнения запроса).
- `EXPLAIN ANALYZE` — реальный план с метриками времени и количества строк.

Ключевые операции в плане:

- **Seq Scan** — полное сканирование таблицы (сигнал к созданию индекса).
- **Index Scan** — чтение данных через индекс.
- **Nested Loop** — соединение таблиц перебором (может быть медленным).
- **Hash Join** — соединение через хеш-таблицу.
- **Merge Join** — соединение отсортированных данных.

3. Показатели плана

- **cost** — оценочная стоимость операции (чем меньше, тем лучше).
- **rows** — предполагаемое количество строк.
- **width** — средний размер строки в байтах.
- **time** — реальное время выполнения (в `ANALYZE`).

Ход работы

Шаг 1. Подготовка тестовой среды

1. Создайте таблицу с тестовыми данными:

```
sql
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    department_id INT,
    salary DECIMAL(10,2),
    hire_date DATE
```

```
);

-- Заполните таблицу 10 000+ записями (используйте генерацию данных).
```

2. Убедитесь, что индексы отсутствуют:

```
sql
SELECT indexname, tablename
FROM pg_indexes
WHERE tablename = 'employees';
```

Шаг 2. Базовый запрос без индексов

Выполните запрос и зафиксируйте время:

```
sql
EXPLAIN ANALYZE
SELECT * FROM employees
WHERE department_id = 5 AND salary > 50000
ORDER BY hire_date DESC
LIMIT 10;
```

Анализ результата:

- Обратите внимание на `Seq Scan` и высокое значение `cost`.
- Запишите время выполнения (`Execution Time`).

Шаг 3. Создание индексов

1. Простой индекс по `department_id`:

```
sql
CREATE INDEX idx_dept ON employees(department_id);
```

2. Составной индекс для комбинации условий:

```
sql
CREATE INDEX idx_dept_salary_hire ON employees(department_id, salary, hire_date);
```

3. Уникальный индекс (если требуется):

```
sql
CREATE UNIQUE INDEX idx_emp_id ON employees(id);
```

Шаг 4. Повторный анализ плана

Выполните тот же запрос с `EXPLAIN ANALYZE` и сравните:

- Исчез ли `Seq Scan`?
- Появился ли `Index Scan`?
- Как изменилось значение `cost` и время выполнения?

Шаг 5. Оптимизация структуры запроса

Попробуйте переписать запрос, чтобы:

- Избегать `SELECT *` (указывайте нужные столбцы).
- Использовать `JOIN` вместо подзапросов.
- Добавить `LIMIT` для ограничения результатов.

Пример оптимизированного запроса:

```
sql
```

```
EXPLAIN ANALYZE
SELECT id, name, salary, hire_date
FROM employees
WHERE department_id = 5
  AND salary > 50000
ORDER BY hire_date DESC
LIMIT 10;
```

Шаг 6. Мониторинг и профилирование

1. Найдите медленные запросы через `pg_stat_statements`:

```
sql
SELECT query, total_time, calls, total_time/calls AS avg_time
FROM pg_stat_statements
ORDER BY total_time DESC
LIMIT 5;
```

2. Проанализируйте статистику таблиц:

```
sql
ANALYZE employees;
```

Шаг 7. Дополнительные оптимизации

1. **Партиционирование** (для больших таблиц):

```
sql
CREATE TABLE employees_2025 PARTITION OF employees
   FOR VALUES FROM ('2025-01-01') TO ('2026-01-01');
```

2. **Материализованные представления** (для сложных агрегаций):

```
sql
CREATE MATERIALIZED VIEW mv_dept_stats AS
SELECT department_id, AVG(salary), COUNT(*)
FROM employees
GROUP BY department_id;
```

3. **Настройка параметров СУБД** (в `postgresql.conf`):

```
shared_buffers = 4GB
work_mem = 64MB
maintenance_work_mem = 1GB
```

Контрольные вопросы

1. В чём разница между кластерным и некластерным индексом?
2. Почему `EXPLAIN ANALYZE` важнее, чем просто `EXPLAIN`?
3. Когда индекс может замедлить выполнение запроса?
4. Как селективность столбца влияет на эффективность индекса?
5. Какие операции в плане выполнения считаются «дорогими»?

Отчёт по работе

Включите в отчёт:

1. Скрипты создания таблицы и индексов.
2. Результаты `EXPLAIN ANALYZE` до и после оптимизации.
3. График сравнения времени выполнения запросов.
4. Выводы о влиянии индексов на производительность.

5. Рекомендации по дальнейшим оптимизациям.

Полезные команды

- Просмотр существующих индексов:

```
sql
SELECT * FROM pg_indexes WHERE tablename = 'your_table';
```

- Удаление индекса:

```
sql
DROP INDEX idx_name;
```

- Обновление статистики:

```
sql
ANALYZE your_table;
```

- Очистка фрагментации (реиндексация):

```
sql
REINDEX TABLE your_table;
```

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 18 Создание резервной копии базы данных и восстановление данных в случае сбоя.

Цель работы: освоить методы создания резервных копий баз данных (бэкапов) и восстановления данных при сбоях.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

Бэкап (backup) — это резервная копия данных, предназначенная для их восстановления при потере или повреждении оригинала.

Основные типы бэкапов:

- **Полное резервное копирование** — создание полной копии всех данных (наиболее надёжный, но ресурсоёмкий метод).
- **Дифференциальное** — копирование данных, изменённых с момента последнего полного бэкапа.
- **Инкрементное** — копирование только данных, изменённых с момента предыдущего бэкапа любого типа.

Ключевые принципы резервного копирования:

- Регулярность создания копий.
- Хранение бэкапов на отдельном носителе/сервере.
- Проверка работоспособности созданных резервных копий.
- Использование нескольких уровней резервирования (полное + дифференциальное/инкрементное).

Практическая часть

1. Создание резервной копии (на примере SQLite)

Шаг 1. Создание тестовой базы данных:

```
bash
sqlite3 mydatabase.db
```

Шаг 2. Создание таблицы и заполнение данными:

```
sql
CREATE TABLE users (
    id INTEGER PRIMARY KEY,
    name TEXT,
    email TEXT
);

INSERT INTO users VALUES(1, 'Alice', 'alice@example.com');
INSERT INTO users VALUES(2, 'Bob', 'bob@example.com');
```

Шаг 3. Проверка данных:

```
sql
SELECT * FROM users;
```

Шаг 4. Создание резервной копии:

```
bash
sqlite3 mydatabase.db .dump > mydatabase_backup.sql
```

Шаг 5. Проверка содержимого бэкапа:

```
bash
cat mydatabase_backup.sql
```

2. Моделирование сбоя и восстановление данных

Шаг 1. Имитация потери данных (удаление исходной базы):

```
bash
rm mydatabase.db
```

Шаг 2. Восстановление из резервной копии:

```
bash
sqlite3 mydatabase.db < mydatabase_backup.sql
```

Шаг 3. Проверка восстановления:

```
bash
sqlite3 mydatabase.db
SELECT * FROM users;
```

3. Дополнительные методы резервного копирования (на примере MySQL)

Через командную строку:

```
bash
mysqldump -u username -p database_name > backup.sql
```

Восстановление из дампа:

```
bash
mysql -u username -p database_name < backup.sql
```

Через графический интерфейс (например, phpMyAdmin):

1. Выбрать базу данных.
2. Перейти во вкладку «Экспорт».
3. Настроить параметры и сохранить файл.
4. Для восстановления — вкладка «Импорт», выбор файла бэкапа.

Контрольные вопросы

1. Что такое резервное копирование данных?
2. Перечислите основные типы бэкапов и их особенности.
3. Почему важно хранить резервные копии на отдельном носителе?
4. Какие инструменты используются для резервного копирования SQLite и MySQL?
5. Опишите последовательность действий при восстановлении данных из бэкапа.

Требования к отчёту

Отчёт должен содержать:

1. Цель работы.
2. Краткое теоретическое описание.
3. Пошаговое описание выполненных действий с командами и результатами.
4. Ответы на контрольные вопросы.
5. Выводы по работе (что было изучено, какие навыки приобретены).

Критерии оценки

- Правильность выполнения команд.
- Полнота описания процесса.
- Корректность восстановления данных.
- Грамотность оформления отчёта.
- Глубина ответов на контрольные вопросы.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 19 Разработка сценариев миграции данных между двумя базами данных.

Цель работы: отработать навыки проектирования и реализации сценариев переноса данных между разнородными базами данных с учётом требований целостности, безопасности и минимизации простоя системы.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.

• Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).

• Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретическая часть

Миграция данных — процесс переноса данных из одной базы данных (источника) в другую (приёмник) с сохранением структуры, связей и значений.

Основные типы миграции

1. **Полная миграция** — перенос всей базы данных из источника в приёмник (например, из локальной среды в облако).
2. **Версионная миграция** — обновление структуры БД для совместимости с новой версией ПО.

Ключевые этапы миграции

1. **Планирование**:
- определение целей и сроков;
- выбор инструментов и технологий;
- формирование команды;
- разработка плана отката.
2. **Анализ и аудит**:
- изучение структуры и объёма данных;
- выявление потенциальных проблем;
- оценка рисков.
3. **Подготовка**:
- создание резервных копий;
- настройка инфраструктуры приёмника;
- разработка шаблонов загрузки.
4. **Выгрузка**:
- экспорт данных из источника;
- валидация целостности.
5. **Трансформация**:
- преобразование форматов;
- нормализация данных;
- обработка несоответствий.
6. **Загрузка**:
- импорт в приёмник;
- контроль ошибок.
7. **Проверка и тестирование**:
- сверка данных;
- нагрузочное тестирование;
- проверка бизнес-логики.
8. **Активация**:
- переключение приложений на новую БД;
- отключение старой системы.

Практическое задание

Сценарий 1. Полная миграция (PostgreSQL → MySQL)

Условия:

- источник: PostgreSQL 14, схема `public` с таблицами `users`, `orders`, `products`;
- приёмник: MySQL 8.0, пустая база `shop`;
- объём данных: ~100 000 записей;
- требования: сохранить связи, индексы, уникальные ограничения.

Шаги реализации:

1. **Подготовка**:
   - создать дамп PostgreSQL:

```
bash
pg_dump -U user -h localhost -d source_db -F c -b -v -f backup.dump
```

   - настроить MySQL: создать базу `shop`, пользователя с правами.
2. **Трансформация схемы**:
   - конвертировать DDL-скрипты (учесть различия типов данных: `SERIAL` → `AUTO_INCREMENT`, `TIMESTAMP WITH TIMEZONE` → `DATETIME`).
3. **Выгрузка данных**:
   - экспортировать таблицы в CSV:

```
sql
COPY users TO '/tmp/users.csv' WITH (FORMAT CSV, HEADER true);
```

4. **Загрузка в MySQL**:
   - использовать `LOAD DATA INFILE`:

```
sql
LOAD DATA INFILE '/tmp/users.csv'
INTO TABLE shop.users
FIELDS TERMINATED BY ',' ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

5. **Проверка**:
   - сравнить количество записей:

```
sql
SELECT COUNT(*) FROM users; -- в обеих БД
```

   - проверить целостность связей (например, `orders.user_id` должен ссылаться на существующий `users.id`).

Сценарий 2. Версионная миграция (MySQL 5.7 → MySQL 8.0)

Условия:

- текущая версия: MySQL 5.7, схема `app_v1`;
- целевая версия: MySQL 8.0, схема `app_v2`;
- изменения: добавлены поля `created_at TIMESTAMP`, индексы для поиска.

Шаги реализации:

1. **Резервное копирование**:

```
bash
mysqldump -u root -p app_v1 > app_v1_backup.sql
```

2. **Обновление сервера**:
   - установить MySQL 8.0;
   - импортировать дамп:

```
bash
mysql -u root -p app_v2 < app_v1_backup.sql
```

3. **Модификация схемы**:
   - добавить поля и индексы:

```
sql
ALTER TABLE users ADD COLUMN created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP;
CREATE INDEX idx_email ON users(email);
```

4. **Тестирование**:
   - запустить набор тестов приложения;
   - проверить производительность запросов.

Сценарий 3. Горячая миграция (без простоя)

Условия:

- система работает 24/7;
- требуется синхронизация данных в реальном времени.

Решение:

1. Настроить репликацию:
   - источник (PostgreSQL) → промежуточный брокер (Kafka);
   - брокер → приёмник (MongoDB).
2. Реализовать двойную запись:
   - приложение пишет данные в обе БД параллельно;
   - использовать транзакции для согласованности.
3. Переключение:
   - после синхронизации перевести приложение на новую БД;
   - отключить репликацию.

Инструменты для миграции

- **ETL-инструменты**: Apache NiFi, Talend, Pentaho.
- **Утилиты**

СУБД: `pg_dump`/`pg_restore`, `mysqldump`, `mongoexport`/`mongoimport`.

- **Скрипты**: Python (библиотеки `pandas`, `sqlalchemy`), Bash.
- **Репликация**: Debezium, SymmetricDS.

Требования к отчёту

1. Описание исходных и целевых схем БД (в виде DDL-скриптов).
2. План миграции для каждого сценария (таблица с этапами, сроками, ответственными).
3. Логи выполнения команд и результаты проверок.
4. Анализ проблем и пути их решения.
5. Рекомендации по оптимизации процесса.

Контрольные вопросы

1. В чём отличие полной и версионной миграции?
2. Какие риски возникают при миграции и как их минимизировать?
3. Как проверить целостность данных после переноса?
4. В каких случаях применяется горячая миграция?
5. Какие инструменты используют для автоматизации миграции?

Критерии оценки

- корректность проектирования сценариев;
- полнота тестирования;
- обоснованность выбора инструментов;
- качество документации.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 20 Администрирование базы данных: настройка параметров производительности и мониторинг активных запросов.

Цель работы: освоить методы настройки параметров производительности СУБД и освоить инструменты мониторинга активных запросов для выявления и устранения узких мест в работе базы данных.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

• Предъявить преподавателю результаты работы.
• Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
• Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Задачи

1. Настроить ключевые параметры производительности СУБД.
2. Осуществить мониторинг активных запросов.
3. Выявить «медленные» запросы и проанализировать их планы выполнения.
4. Применить методы оптимизации для повышения производительности.

Оборудование и ПО

- Сервер с установленной СУБД (например, Microsoft SQL Server, PostgreSQL, Oracle).
- Клиентские инструменты администрирования (SSMS, pgAdmin, SQL Developer).
- Монитор производительности (System Monitor, Performance Monitor).
- Профилировщик запросов (SQL Server Profiler, pgBadger).

Теоретические основы

Базовые параметры (baseline) — усреднённые показатели работы сервера в штатных условиях, служащие ориентиром для сравнения.

Упреждающий мониторинг — система непрерывного наблюдения за ключевыми метриками с целью выявления отклонений до возникновения критических сбоев.

Хранилище запросов (Query Store) — механизм, сохраняющий историю компиляции и метрики выполнения запросов для анализа и оптимизации.

Ход работы

1. Настройка параметров производительности

Шаг 1. Определите базовые параметры системы:

- загрузка процессора (`% Processor Time`);
- использование памяти (`Buffer Cache Hit Ratio`);
- количество транзакций в секунду (`Transactions/sec`);
- время ожидания блокировок (`Average Wait Time`).

Шаг 2. Настройте ключевые параметры СУБД:

- размер буферного пула (`buffer_cache_size`);
- максимальное число одновременных подключений (`max_connections`);
- порог длительности «медленных» запросов (`long_query_threshold`).

Пример для PostgreSQL (в файле `postgresql.conf`):

```
shared_buffers = 4GB
max_connections = 100
work_mem = 16MB
maintenance_work_mem = 512MB
```

2. Мониторинг активных запросов

Шаг 1. Включите хранилище запросов (если доступно):

```
sql
ALTER DATABASE [YourDB] SET QUERY_STORE = ON;
```

Шаг 2. Получите список текущих запросов:

```
sql
SELECT
    session_id,
```

```
    status,
    command,
    cpu_time,
    total_elapsed_time,
    text
FROM sys.dm_exec_requests
CROSS APPLY sys.dm_exec_sql_text(sql_handle);
```

Шаг 3. Проанализируйте длительные запросы:

```
sql
SELECT TOP 10
    qt.query_sql_text,
    q.query_id,
    p.plan_id,
    rs.last_execution_time,
    rs.avg_duration
FROM sys.query_store_query_text AS qt
INNER JOIN sys.query_store_query AS q ON qt.query_text_id = q.query_text_id
INNER JOIN sys.query_store_plan AS p ON q.query_id = p.query_id
INNER JOIN sys.query_store_runtime_stats AS rs ON p.plan_id = rs.plan_id
ORDER BY rs.avg_duration DESC;
```

3. Оптимизация производительности

Шаг 1. Изучите план выполнения «медленного» запроса:

- В SSMS: нажмите «Показать предполагаемый план выполнения».
- В pgAdmin: используйте команду `EXPLAIN ANALYZE`.

Шаг 2. Примените оптимизацию:

- Добавьте индексы для часто фильтруемых столбцов:

```
sql
CREATE INDEX idx_customer_name ON customers (last_name);
```

- Перепишите запрос с использованием JOIN вместо подзапросов.
- Разбейте сложный запрос на несколько простых.

Шаг 3. Принудительно примените оптимальный план (для SQL Server):

```
sql
EXEC sp_query_store_force_plan @query_id = 48, @plan_id = 49;
```

4. Упреждающий мониторинг

Шаг 1. Настройте сбор метрик:

- Интервал сбора: 15 сек.
- Хранилище данных: локальный журнал или удалённый сервер.

Шаг 2. Установите пороги предупреждений:

- Загрузка CPU > 80 % → уведомление низкого приоритета.
- Время ожидания блокировки > 5 сек → уведомление высокого приоритета.

Шаг 3. Создайте дашборд для визуализации:

- График загрузки процессора.
- Диаграмма количества активных соединений.
- Таблица топ-10 длительных запросов.

Контрольные вопросы

1. Что такое базовые параметры (baseline) и как их определить?
2. В чём отличие упреждающего мониторинга от реактивного?
3. Как использовать хранилище запросов для оптимизации производительности?
4. Какие метрики СУБД критичны для мониторинга?
5. Как принудительно применить план выполнения запроса?

Отчётность

Подготовьте отчёт, включающий:

1. Скриншоты настроек СУБД.
2. Примеры «медленных» запросов и их оптимизированных версий.
3. Графики метрик производительности до и после оптимизации.
4. Выводы о эффективности применённых методов.

Критерии оценки

- Корректность настройки параметров СУБД.
- Полнота мониторинга активных запросов.
- Качество анализа планов выполнения.
- Эффективность предложенных оптимизаций.
- Чёткость оформления отчёта.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Тема 2.3. Организация защиты данных в хранилищах

Практическая работа 21 Настройка шифрования данных в MySQL с использованием встроенных функций (например, AES_ENCRYPT, AES_DECRYPT).

Цель работы изучить, как управлять учётными записями пользователей и контролировать доступ к базам данных

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

MySQL поддерживает **симметричное шифрование** по алгоритму AES (Advanced Encryption Standard, также известному как Rijndael). Ключевые функции:

- `AES_ENCRYPT(str, key)` — шифрует строку `str` с использованием ключа `key`.
- `AES_DECRYPT(crypt_str, key)` — расшифровывает зашифрованную строку `crypt_str` при помощи ключа `key`.

Важные особенности:

- Результат шифрования — двоичные данные, поэтому хранить их следует в полях типа `BLOB` (например, `TINYBLOB`, `BLOB`, `MEDIUMBLOB`, `LONGBLOB`).
- Ключ шифрования передаётся в открытом виде в запросе (если не используется SSL), поэтому рекомендуется:
  - применять SSL-соединение для защиты трафика;
  - выполнять шифрование/дешифрование на стороне приложения, а в БД передавать только зашифрованные данные;
  - использовать хранимые процедуры для сокрытия ключа в коде приложения.

Ход работы

1. Подготовка базы данных и таблицы

Создайте базу данных и таблицу с полем типа `BLOB` для хранения зашифрованных данных:

```
sql
CREATE DATABASE IF NOT EXISTS crypto_db;
USE crypto_db;

CREATE TABLE IF NOT EXISTS users (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name BLOB,
   email BLOB
);
```

2. Шифрование данных при вставке

Вставьте запись, зашифровав поля `name` и `email` с помощью `AES_ENCRYPT`:

```
sql
INSERT INTO users (name, email)
VALUES (
   AES_ENCRYPT('Иван Иванов', 'my_secret_key'),
   AES_ENCRYPT('ivan@example.com', 'my_secret_key')
);
```

Пояснения:

- `'Иван Иванов'` и `'ivan@example.com'` — исходные данные.
- `'my_secret_key'` — ключ шифрования (в реальной системе должен быть надёжно защищён).

3. Дешифрование данных при выборе

Получите расшифрованные данные с помощью `AES_DECRYPT`:

```
sql
SELECT
   AES_DECRYPT(name, 'my_secret_key') AS name,
   AES_DECRYPT(email, 'my_secret_key') AS email
```

```
FROM users;
```

Результат:

```
+------------+-----------------+
| name       | email           |
+------------+-----------------+
| Иван Иванов | ivan@example.com |
+------------+-----------------+
```

4. Использование переменных для ключа

Для повышения безопасности можно использовать пользовательские переменные:

```
sql
SET @crypto_key = 'my_secret_key';

INSERT INTO users (name, email)
VALUES (
   AES_ENCRYPT('Анна Петрова', @crypto_key),
   AES_ENCRYPT('anna@example.com', @crypto_key)
);

SELECT
   AES_DECRYPT(name, @crypto_key) AS name,
   AES_DECRYPT(email, @crypto_key) AS email
FROM users;
```

5. Работа с NULL-значениями

Если ключ или данные равны `NULL`, функции возвращают `NULL`:

```
sql
SELECT AES_ENCRYPT(NULL, 'key'); -- Результат: NULL
SELECT AES_ENCRYPT('data', NULL); -- Результат: NULL
```

6. Проверка корректности дешифрования

Убедитесь, что дешифрование работает только с правильным ключом:

```
sql
-- Неверный ключ — результат NULL
SELECT AES_DECRYPT(name, 'wrong_key') FROM users;

-- Верный ключ — корректное значение
SELECT AES_DECRYPT(name, 'my_secret_key') FROM users;
```

Рекомендации по безопасности

1. **Хранение ключей:**
   - Не храните ключи в БД или в коде приложения.
   - Используйте внешние системы управления ключами (KMS) или конфигурационные файлы с ограниченным доступом.
2. **Передача данных:**
   - Включайте SSL/TLS для защиты данных при передаче.
   - Шифруйте данные на стороне клиента перед отправкой в БД.
3. **Управление доступом:**
   - Ограничьте права доступа к таблице с зашифрованными данными.
   - Мониторьте запросы к функциям шифрования.
4. **Резервное копирование:**
   - Регулярно создавайте резервные копии зашифрованных данных и ключей.
   - Храните копии ключей в защищённом месте.

Контрольные вопросы

1. Какой тип данных рекомендуется использовать для хранения результатов AES_ENCRYPT?
2. Почему важно защищать ключ шифрования?
3. Какие риски возникают при передаче ключа в открытом SQL-запросе?
4. Как можно повысить безопасность при работе с AES_ENCRYPT/AES_DECRYPT?
5. Что произойдёт, если использовать неверный ключ при дешифровании?

Вывод

В ходе работы освоены:

- механизм шифрования данных с помощью AES_ENCRYPT;
- процесс дешифрования через AES_DECRYPT;
- рекомендации по безопасному хранению и использованию ключей.

Итоговый результат: данные в БД хранятся в зашифрованном виде, доступ к ним возможен только при наличии корректного ключа.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 22 Реализация ролевой модели безопасности в PostgreSQL (создание ролей и управление их правами).

Цель работы: освоить механизмы создания ролей и управления их правами в СУБД PostgreSQL.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная

карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

В PostgreSQL **роль** — единый механизм аутентификации на уровне кластера, который может выступать:

- как **пользователь** (с правом входа в систему — атрибут `LOGIN`);
- как **группа** (без права входа — атрибут `NOLOGIN`).

Суперпользователь (например, роль `postgres`) имеет неограниченные права и обходит все проверки доступа.

Основные команды:

- `CREATE ROLE` — создание роли;
- `GRANT` — назначение прав;
- `REVOKE` — отзыв прав;
- `DROP ROLE` — удаление роли.

Ход работы

1. Подключение к серверу PostgreSQL

Подключитесь как суперпользователь (обычно `postgres`):

```
sql
psql -U postgres
```

2. Просмотр существующих ролей

Выведите список всех ролей:

```
sql
SELECT rolname FROM pg_roles;
```

Или в psql-клиенте:

```
\du
```

3. Создание пользовательских ролей

Создайте роль с правом входа и паролем:

```
sql
CREATE ROLE user1 WITH LOGIN PASSWORD 'secure_pass1';
```

Создайте ещё одну роль аналогичным образом:

```
sql
CREATE ROLE user2 WITH LOGIN PASSWORD 'secure_pass2';
```

4. Создание групповых ролей

Создайте группу без права входа:

```sql
CREATE ROLE developers NOLOGIN;
```

Добавьте пользователей в группу:

```sql
GRANT developers TO user1, user2;
```

5. Управление правами на уровне базы данных

Предположим, есть база данных `company_db`. Предоставьте права:

- **Подключение к БД**:

```sql
GRANT CONNECT ON DATABASE company_db TO developers;
```

- **Права на схему** `public`:

```sql
GRANT USAGE ON SCHEMA public TO developers;
```

6. Управление правами на таблицы

Создайте таблицу для тестирования:

```sql
CREATE TABLE public.employees (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    salary DECIMAL(10,2)
);
```

Предоставьте разные права группам/пользователям:

- **Группа** `developers` **— полные права на таблицу**:

```sql
GRANT ALL PRIVILEGES ON TABLE public.employees TO developers;
```

- **Пользователь** `user2` **— только чтение**:

```sql
GRANT SELECT ON TABLE public.employees TO user2;
```

7. Проверка назначенных прав

Выведите список прав для таблицы:

```sql
\z public.employees
```

Или детально:

```sql
SELECT grantee, privilege_type
FROM information_schema.table_privileges
WHERE table_name = 'employees';
```

8. Отзыв прав

Отозовите право `INSERT` у `user2`:

```sql
REVOKE INSERT ON TABLE public.employees FROM user2;
```

9. Создание роли с расширенными привилегиями

Создайте администратора с правом создавать роли и БД:

```sql
CREATE ROLE admin_user WITH
  LOGIN
  PASSWORD 'admin_pass'
  CREATEDB
  CREATEROLE;
```

10. Удаление роли

Удалите тестовую роль (если не используется):

```sql
DROP ROLE IF EXISTS temp_user;
```

Контрольные вопросы

1. Чем отличается роль с `LOGIN` от роли с `NOLOGIN`?
2. Какие привилегии даёт атрибут `CREATEDB`?
3. Как проверить, какие права назначены роли на конкретную таблицу?
4. Можно ли отозвать только часть прав (например, `INSERT`, оставив `SELECT`)?
5. Почему важно использовать групповые роли вместо назначения прав каждому пользователю отдельно?

Отчёт по работе

Подготовьте отчёт, включающий:

1. Цель работы.
2. Последовательность выполненных команд (с пояснениями).
3. Скриншоты результатов ключевых операций (список ролей, права на таблицу).
4. Ответы на контрольные вопросы.
5. Выводы о преимуществах ролевой модели доступа.

Дополнительные задания (по желанию)

1. Создайте роль для резервного копирования с минимальными правами (`CONNECT` + `SELECT` на все таблицы).
2. Настройте права так, чтобы одна роль могла изменять структуру таблицы (`ALTER`), но не данные.
3. Используйте команду `\dp` для просмотра всех назначенных прав в текущей БД.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 23 Настройка аудита действий пользователей в Microsoft SQL Server.

Цель работы :освоить механизмы управления доступом в PostgreSQL на основе ролевой модели: создание ролей, назначение привилегий, управление членством.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретическая часть

В PostgreSQL **роль** — единый механизм аутентификации на уровне кластера, который может выступать:

- как **пользователь** (с правом входа в систему);
- как **группа** (без права входа, для объединения пользователей).

Ключевые атрибуты ролей:

- `LOGIN` — разрешает вход в систему;
- `SUPERUSER` — предоставляет полные права (обходит все проверки);
- `CREATEDB` — позволяет создавать базы данных;
- `CREATEROLE` — позволяет управлять другими ролями;
- `PASSWORD` — задаёт пароль для аутентификации.

Привилегии — права на действия с объектами БД (таблицы, схемы, базы данных). Основные операции:

- `SELECT` — чтение данных;

- `INSERT` — добавление записей;
- `UPDATE` — изменение данных;
- `DELETE` — удаление записей;
- `CREATE` — создание объектов;
- `ALL PRIVILEGES` — все права.

Практическая часть

1. Подключение к PostgreSQL

Подключитесь как суперпользователь (обычно `postgres`):

```
sql
psql -U postgres
```

2. Просмотр существующих ролей

Выведите список всех ролей:

```
sql
SELECT rolname FROM pg_roles;
```

Или в psql-клиенте:

```
\du
```

3. Создание пользовательской роли

Создайте роль с правом входа и паролем:

```
sql
CREATE ROLE user1 WITH LOGIN PASSWORD 'secure_password' VALID UNTIL '2026-01-01';
```

- `LOGIN` — разрешает аутентификацию;
- `PASSWORD` — устанавливает пароль;
- `VALID UNTIL` — задаёт срок действия пароля.

4. Создание групповой роли

Создайте роль без права входа (для группировки):

```
sql
CREATE ROLE developers NOLOGIN;
```

5. Назначение привилегий группе

Предоставьте группе права на схему и таблицы:

```
sql
-- Право создавать объекты в схеме public
GRANT CREATE ON SCHEMA public TO developers;

-- Права SELECT, INSERT, UPDATE, DELETE на все таблицы схемы public
GRANT SELECT, INSERT, UPDATE, DELETE
ON ALL TABLES IN SCHEMA public
TO developers;
```

6. Добавление пользователя в группу

Включите пользователя в групповую роль:

```
sql
GRANT developers TO user1;
```

Теперь `user1` наследует все привилегии `developers`.

7. Предоставление прав на конкретную таблицу

Дайте пользователю прямые права на таблицу:

```sql
GRANT SELECT, UPDATE ON TABLE employees TO user1;
```

8. Отзыв привилегий

Отозовите право на удаление из группы:

```sql
REVOKE DELETE ON ALL TABLES IN SCHEMA public FROM developers;
```

9. Создание роли с расширенными правами

Создайте администратора с правом управлять ролями:

```sql
CREATE ROLE admin_user WITH
   LOGIN
   PASSWORD 'admin_pass'
   CREATEROLE
   CREATEDB;
```

10. Проверка прав

Убедитесь, что права назначены корректно:

```sql
-- Просмотр привилегий на таблицы
SELECT * FROM information_schema.table_privileges
WHERE grantee = 'user1';

-- Просмотр членства в группах
SELECT rolname
FROM pg_roles
WHERE pg_has_role('user1', rolname, 'member');
```

11. Удаление роли

Удалите роль (если она не владеет объектами):

```sql
DROP ROLE IF EXISTS user1;
```

Контрольные вопросы

1. Чем отличается роль с атрибутом `LOGIN` от роли с `NOLOGIN`?
2. Какие привилегии даёт атрибут `SUPERUSER`?
3. Как проверить, какие права имеет пользователь на конкретную таблицу?
4. Почему рекомендуется использовать групповые роли вместо прямого назначения прав пользователям?
5. Какие команды используются для предоставления и отзыва привилегий?

Требования к отчёту

Отчёт должен содержать:

1. Цель работы.
2. Краткие теоретические сведения (2–3 абзаца).
3. Последовательность выполненных команд с комментариями.

4. Скриншоты результатов выполнения ключевых операций.
5. Ответы на контрольные вопросы.
6. Выводы по работе.

Дополнительные задания (по желанию)

1. Создайте роль для резервного копирования с минимальными привилегиями.
2. Настройте права так, чтобы пользователь мог только читать данные из определённых таблиц.
3. Используйте команду `ALTER DEFAULT PRIVILEGES` для автоматического назначения прав новым таблицам.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.
2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 24 Конфигурация шифрования трафика между клиентом и сервером базы данных (TLS/SSL).

Цель работы: Настроить защищённое соединение по протоколу TLS/SSL между клиентом и сервером СУБД, обеспечив шифрование передаваемых данных.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.

• Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).

• Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

TLS (Transport Layer Security) и **SSL** (Secure Sockets Layer) — криптографические протоколы для защиты сетевого взаимодействия. В современных системах используется TLS (SSL считается устаревшим, но термин часто применяется как синоним).

Основные этапы установления защищённого соединения:

1. Клиент инициирует соединение и запрашивает защищённый канал.
2. Стороны согласовывают алгоритм шифрования.
3. Сервер передаёт свой цифровой сертификат.
4. Клиент проверяет валидность сертификата.
5. Генерируется сеансовый ключ для шифрования.

Ключевые компоненты:

- **Сертификат сервера** (server.crt) — содержит открытый ключ и информацию о владельце.
- **Закрытый ключ** (server.key) — хранится на сервере, не передаётся.
- **Корневой сертификат** — используется для подписи серверных сертификатов.

Практическая часть

1. Генерация сертификатов

Шаг 1. Создание корневого сертификата (CA):

bash

```
openssl req -new -x509 -days 365 -keyout root.key -out root.crt \
-subj "/C=RU/ST=Moscow/L=Moscow/O=MyOrg/CN=RootCA"
```

Шаг 2. Создание запроса на сертификат сервера:

bash

```
openssl req -new -keyout server.key -out server.csr \
-subj "/C=RU/ST=Moscow/L=Moscow/O=MyOrg/CN=dbserver.example.com"
```

Шаг 3. Подписание серверного сертификата корневым CA:

bash

```
openssl x509 -req -in server.csr -CA root.crt -CAkey root.key \
-CAcreateserial -out server.crt -days 365
```

2. Настройка сервера СУБД (пример для PostgreSQL)

Шаг 1. Размещение файлов в каталоге данных сервера:

- server.crt — сертификат сервера;
- server.key — закрытый ключ (права 0600);
- root.crt — корневой сертификат (для проверки клиентов).

Шаг 2. Редактирование postgresql.conf:

ini

```
ssl = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

Шаг 3. Настройка pg_hba.conf для требований SSL:

ini

```
# Требовать SSL для всех подключений
hostssl all all 0.0.0.0/0 md5

# Или только для определённых сетей
hostssl all all 192.168.1.0/24 md5
```

Шаг 4. Перезагрузка сервера:

bash

```
pg_ctl reload
```

3. Настройка клиентского подключения

Вариант 1. Через строку подключения (например, для psql):

bash

```
psql "host=dbserver.example.com port=5432 dbname=mydb \
user=myuser sslmode=require sslrootcert=root.crt"
```

Вариант 2. Через переменные окружения:

bash

```
export PGSSLMODE=require
export PGSSLCERT=client.crt
export PGSSLKEY=client.key
export PGSSLROOTCERT=root.crt
```

Вариант 3. Через файл .pgpass и параметры подключения.

4. Проверка работоспособности

Шаг 1. Проверка статуса SSL в сервере:

sql

```
SELECT ssl, client_addr, client_port FROM pg_stat_ssl;
```

Шаг 2. Тестирование подключения:

bash

```
openssl s_client -connect dbserver.example.com:5432 -CAfile root.crt
```

Ожидаемый результат: успешное установление соединения и вывод информации о сертификате.

Возможные ошибки и их устранение

1. **Ошибка «certificate verify failed»**
   - Проверить путь к корневому сертификату.
   - Убедиться, что sslrootcert указан корректно.
2. **Ошибка «permission denied» для server.key**
   - Установить права: chmod 600 server.key.
3. **Сервер не принимает SSL-подключения**
   - Проверить ssl = on в postgresql.conf.
   - Убедиться, что в pg_hba.conf есть правила hostssl.

Контрольные вопросы

1. В чём разница между SSL и TLS?
2. Для чего нужен корневой сертификат (CA)?
3. Почему закрытый ключ нельзя передавать третьим лицам?
4. Как проверить, что подключение использует SSL?

5. Какие параметры конфигурации PostgreSQL отвечают за SSL?

Отчёт по работе

В отчёте необходимо отразить:

1. Пошаговые команды генерации сертификатов.
2. Изменённые параметры конфигурационных файлов.
3. Результаты тестирования подключения.
4. Ответы на контрольные вопросы.
5. Описание возникших проблем и способов их решения.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 25 Организация резервного копирования с шифрованием в Oracle Database.

Цель работы изучить, как управлять учётными записями пользователей и контролировать доступ к базам данных

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.

- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Для организации резервного копирования с шифрованием в Oracle Database можно использовать несколько подходов, включая инструменты Oracle Recovery Manager (RMAN) и Transparent Data Encryption (TDE). Ниже приведены ключевые шаги и рекомендации для выполнения практической работы.

Подготовка к резервному копированию

Перед началом резервного копирования необходимо подготовить базу данных. В Oracle для этого используется команда ALTER DATABASE/TABLESPACE BEGIN BACKUP, которая переводит базу в режим резервного копирования. Это гарантирует, что все изменения, произошедшие до начала резервного копирования, будут зафиксированы в журналах, а при восстановлении не потребуется применять эти журналы.

По окончании резервного копирования базу данных нужно вернуть в обычное состояние с помощью команды ALTER DATABASE/TABLESPACE END BACKUP.

Шифрование с помощью RMAN

RMAN позволяет шифровать резервные копии с использованием явного указания ключа или прозрачного шифрования через Oracle Wallet. Для настройки шифрования необходимо:

1. **Настроить параметры шифрования в RMAN**:

sql

 SET ENCRYPTION ON FOR ALL TABLESPACES;

 Указать алгоритм шифрования (например, AES256):

sql

 SET ENCRYPTION ALGORITHM 'AES256';

 Использовать Oracle Wallet для управления ключами:

- Создать или открыть кошелек с помощью команды ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "password".
- Указать кошелек в параметрах RMAN:

sql

-
- SET ENCRYPTION WALLET LOCATION '/path/to/wallet';

 Выполнить резервное копирование:

sql

4. BACKUP AS BACKUPSET DATABASE TAG "encrypted_backup";
5.

Шифрование с помощью TDE

Transparent Data Encryption (TDE) позволяет шифровать данные на уровне табличных пространств или столбцов таблиц. Для шифрования резервных копий с использованием TDE необходимо:

1. **Настроить Oracle Wallet** для хранения ключей шифрования.
2. **Шифровать табличные пространства**:

sql

2. ALTER TABLESPACE users ENCRYPTION USING 'AES256';

3.

4. **Убедиться, что резервные копии будут зашифрованы автоматически**, так как TDE шифрует данные в файлах табличных пространств, включая резервные копии. fors.ru +1

Сравнение методов

Метод	Преимущества	Недостатки
RMAN с явным указанием ключа	Гибкость в выборе алгоритмов и ключей	Требует ручного управления ключами
RMAN с Oracle Wallet	Автоматизация управления ключами	Зависимость от корректной настройки кошелька
TDE	Прозрачное шифрование данных и резервных копий	Ограничения на шифрование индексированных столбцов

Дополнительные рекомендации

- **Резервное копирование журналов транзакций**: Для возможности восстановления на любой момент времени (Point-in-Time Recovery) необходимо регулярно архивировать журналы транзакций. habr.com +1
- **Тестирование восстановления**: Регулярно проверяйте возможность восстановления резервных копий, чтобы убедиться в их целостности и корректности шифрования.
- **Управление ключами**: Обеспечьте безопасное хранение и ротацию ключей шифрования. Для TDE используйте централизованное хранилище ключей, например, Oracle Key Vault.

Пример сценария резервного копирования с шифрованием

1. Подготовка базы данных:

sql

 ALTER DATABASE BEGIN BACKUP;

 Настройка шифрования в RMAN:

sql

 SET ENCRYPTION ON FOR ALL TABLESPACES ALGORITHM 'AES256';
SET ENCRYPTION WALLET LOCATION '/path/to/wallet';

 Выполнение резервного копирования:

sql

 BACKUP AS BACKUPSET DATABASE PLUS ARCHIVELOG TAG "encrypted_backup";

 Завершение режима резервного копирования:

sql

4. ALTER DATABASE END BACKUP;

5.

Для более детальной настройки и решения специфических задач рекомендуется обратиться к официальной документации Oracle и дополнительным источникам, таким как

и

.

Если у вас есть конкретные вопросы по настройке или примеры ошибок, уточните их — помогу разобраться!

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 26 Разработка политики управления доступом к данным на уровне таблиц и столбцов.

Цель работы: освоить принципы и инструменты разграничения прав доступа к данным в СУБД на уровне таблиц и отдельных столбцов, сформировать навыки настройки политик безопасности.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

Управление доступом в современных СУБД базируется на **модели привилегий** (разрешений), которые назначаются:

- пользователям;
- ролям (группам пользователей).

Ключевые инструкции SQL для управления доступом:

- GRANT — предоставление разрешений;
- REVOKE — отзыв ранее предоставленных разрешений;
- DENY — явный запрет доступа (перекрывает GRANT).

Разрешения для таблиц

Основные типы разрешений на таблицу:

- SELECT — чтение данных;
- INSERT — добавление строк;
- UPDATE — изменение данных;
- DELETE — удаление строк;
- ALTER — изменение структуры таблицы;
- REFERENCES — создание внешних ключей;
- TAKE OWNERSHIP — передача владения;
- VIEW DEFINITION — доступ к метаданным.

Разрешения для столбцов

На уровне столбцов доступны:

- SELECT — чтение значений столбца;
- UPDATE — изменение значений столбца.

Важно: разрешения на столбцы действуют *дополнительно* к разрешениям на таблицу. Если запрещено SELECT на таблицу, доступ к отдельным столбцам тоже будет заблокирован.

Порядок выполнения работы

Шаг 1. Создание тестовой схемы

1. Создайте базу данных и таблицу:

sql

```
CREATE DATABASE AccessTest;
USE AccessTest;

CREATE TABLE Employees (
   ID INT PRIMARY KEY,
   Name NVARCHAR(100),
   Salary DECIMAL(10,2),
   Department NVARCHAR(50)
);
```

2. Добавьте тестовые данные:

sql

```
INSERT INTO Employees VALUES
(1, 'Иванов И.И.', 50000, 'IT'),
(2, 'Петрова А.С.', 45000, 'HR'),
(3, 'Сидоров В.В.', 60000, 'Finance');
```

Шаг 2. Создание пользователей и ролей

1. Создайте пользователей:

sql

```
CREATE LOGIN User1 WITH PASSWORD = 'Password1!';
CREATE LOGIN User2 WITH PASSWORD = 'Password2!';

CREATE USER User1 FOR LOGIN User1;
CREATE USER User2 FOR LOGIN User2;
```

2. Создайте роль для отдела кадров:

sql

```
CREATE ROLE HR_Role;
ALTER ROLE HR_Role ADD MEMBER User1;
```

Шаг 3. Настройка доступа на уровне таблиц

1. Предоставьте роль HR_Role право читать и обновлять таблицу:

sql

```
GRANT SELECT, UPDATE ON Employees TO HR_Role;
```

2. Запретите пользователю User2 доступ к таблице:

sql

```
DENY SELECT ON Employees TO User2;
```

3. Проверьте действующие разрешения:

sql

```
SELECT * FROM sys.database_permissions
WHERE major_id = OBJECT_ID('Employees');
```

Шаг 4. Настройка доступа на уровне столбцов

1. Разрешите User1 видеть только столбцы Name и Department:

sql

```
GRANT SELECT (Name, Department) ON Employees TO User1;
```

2. Запретите User1 изменять столбец Salary:

sql

```
DENY UPDATE (Salary) ON Employees TO User1;
```

3. Проверьте доступ:

sql

```
-- От имени User1 (успешно)
SELECT Name, Department FROM Employees;

-- От имени User1 (ошибка)
UPDATE Employees SET Salary = 55000 WHERE ID = 1;
```

Шаг 5. Отзыв и корректировка прав

1. Отозовите право на обновление таблицы у HR_Role:

sql

```
REVOKE UPDATE ON Employees FROM HR_Role;
```

2. Проверьте, что User1 больше не может обновлять таблицу:

sql

```
-- От имени User1 (ошибка)
UPDATE Employees SET Department = 'IT' WHERE ID = 2;
```

3. Восстановите частичное право на обновление (только столбец Department):

sql

```
GRANT UPDATE (Department) ON Employees TO User1;
```

Шаг 6. Анализ и документирование политики

Составьте таблицу текущих разрешений:

Пользователь/Роль	Таблица	Разрешения	Столбцы	Ограничения
HR_Role	Employees	SELECT	Все	Нет
User1	Employees	SELECT	Name, Department	Нет
User1	Employees	UPDATE	Department	Запрет на Salary
User2	Employees	—	—	DENY SELECT

Контрольные вопросы

1. В чём разница между REVOKE и DENY?
2. Какие разрешения необходимы для выполнения UPDATE на столбец?
3. Как проверить текущие права доступа к объекту?
4. Можно ли назначить DENY на столбец, если есть GRANT на таблицу?
5. Какова иерархия применения разрешений (таблица → столбец)?

Требования к отчёту

Отчёт должен содержать:

1. Текст всех выполненных SQL-запросов.
2. Скриншоты результатов выполнения ключевых команд.
3. Таблицу текущих разрешений (см. Шаг 6).
4. Ответы на контрольные вопросы.
5. Выводы о принципах разграничения доступа.

Критерии оценки

- Корректность выполнения SQL-запросов.
- Полнота настройки разрешений.
- Точность анализа действующих прав.
- Грамотность оформления отчёта.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 27 Настройка защиты конфиденциальных данных с использованием маскирования данных (Data Masking) в Microsoft SQL Server.

Цель работы изучить, как управлять учётными записями пользователей и контролировать доступ к базам данных

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Динамическое маскирование данных (Dynamic Data Masking, DDM) в Microsoft SQL Server — это функция безопасности, которая позволяет скрывать чувствительные данные от неавторизованных пользователей, не изменяя при этом сами данные в базе. Она применяется к отдельным столбцам таблиц и работает на уровне запросов, автоматически маскируя данные для пользователей без соответствующих прав.

Типы маскирования

В SQL Server доступны несколько типов маскирования:

1. **Default** — полное маскирование в соответствии с типом данных поля:
   - Для строковых типов (char, varchar и др.) используется «XXXX» (или меньше, если размер поля меньше 4 символов).
   - Для числовых типов (int, money и др.) применяется значение 0.
   - Для двоичных типов (binary, varbinary) используется однобайтовое значение 0.
2. **Email** — маскирует адреса электронной почты, оставляя только первый символ имени пользователя и домен «.com» (например,

).

Random — заменяет числовые значения случайными числами в заданном диапазоне (например, random(100, 300)).

Partial (Custom String) — позволяет настроить маску, указав префикс, суффикс и заполнение (например, partial(3, "XXXX-XXXX-", 4)).

DateTime (доступно в SQL Server 2022) — маскирует отдельные части данных типа datetime (часы, месяц, день и т. д.).

Настройка маскирования

Чтобы применить маскирование к столбцу, используется синтаксис:

```
sql
ALTER TABLE [ИмяТаблицы]
ALTER COLUMN [ИмяСтолбца]
ADD MASKED WITH (FUNCTION = '[функция_маскирования]');
```

Пример создания таблицы с маскированием:

```
sql
CREATE TABLE Employee (
   ID INT IDENTITY (1,1) PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50),
   Email VARCHAR(100) MASKED WITH (FUNCTION = 'email()'),
   WorkPhoneNumber VARCHAR(50) MASKED WITH (FUNCTION =
'partial(1,"XXXXXXX",0)'),
   ServicePeriodInYears INT MASKED WITH (FUNCTION = 'random(100, 300)'),
   BirthDate DATE MASKED WITH (FUNCTION = 'datetime("Y")'),
   Salary MONEY MASKED WITH (FUNCTION = 'default()')
);
```

Управление доступом

Для просмотра незамаскированных данных пользователю необходимо предоставить разрешение **UNMASK**. Это можно сделать с помощью команды:

```
sql
GRANT UNMASK TO [имя_пользователя];
```

Важно: до SQL Server 2022 разрешение UNMASK применялось глобально для всей базы данных. С SQL Server 2022 появилась возможность предоставлять это разрешение на уровне столбцов.

Пример создания пользователя и проверки маскирования:

```
sql
CREATE USER TestUser WITHOUT LOGIN;
GRANT SELECT ON Employee TO TestUser;

EXECUTE AS USER = 'TestUser';
SELECT * FROM Employee;
REVERT;
```

Ограничения и рекомендации

- Маскирование не заменяет шифрование и не защищает данные в всех сценариях. Оно предназначено для ограничения доступа на уровне запросов.

Маскирование не поддерживается для некоторых типов данных, например, COLUMN_SET, FILESTREAM, зашифрованных столбцов. Для изменения или удаления маски используется команда ALTER TABLE ... DROP MASKED. Рекомендуется комбинировать

DDM с другими функциями безопасности SQL Server (аудит, шифрование, безопасность на уровне строк).

Пример полного сценария

1. Создание таблицы с маскированием.
2. Вставка данных.
3. Создание пользователя без прав UNMASK.
4. Проверка маскирования.
5. Предоставление прав UNMASK и повторная проверка.

sql

```
-- Шаг 1: Создание таблицы
CREATE TABLE CustomerData (
    ID int IDENTITY PRIMARY KEY,
    FullName nvarchar(100) MASKED WITH (FUNCTION = 'default()') NULL,
    Email nvarchar(100) MASKED WITH (FUNCTION = 'email()') NULL,
    CreditCard varchar(50) MASKED WITH (FUNCTION = 'partial(0,"XXXX-XXXX-XXXX-",4)') NULL,
    BirthDate date MASKED WITH (FUNCTION = 'default()') NULL
);

-- Шаг 2: Вставка данных
INSERT INTO CustomerData (FullName, Email, CreditCard, BirthDate)
VALUES ('John Doe', 'johndoe@example.com', '1234-5678-9012-3456', '1980-01-01');

-- Шаг 3: Создание пользователя
CREATE USER NonPrivilegedUser WITHOUT LOGIN;
GRANT SELECT ON CustomerData TO NonPrivilegedUser;

-- Шаг 4: Проверка маскирования
EXECUTE AS USER = 'NonPrivilegedUser';
SELECT * FROM CustomerData;
REVERT;

-- Шаг 5: Предоставление прав UNMASK
GRANT UNMASK TO NonPrivilegedUser;

-- Повторная проверка
EXECUTE AS USER = 'NonPrivilegedUser';
SELECT * FROM CustomerData;
REVERT;

-- Очистка
DROP USER NonPrivilegedUser;
DROP TABLE CustomerData;
```

Этот пример демонстрирует, как динамическое маскирование позволяет контролировать доступ к чувствительным данным без изменения структуры базы данных.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 28 Организация двухфакторной аутентификации для доступа к базам данных.

Цель работы изучить, как управлять учётными записями пользователей и контролировать доступ к базам данных

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Двухфакторная аутентификация (2FA) — это метод защиты доступа к системам и данным, при котором для подтверждения личности требуется два независимых способа

проверки. В контексте баз данных 2FA может использоваться для усиления безопасности при подключении пользователей, сочетая традиционные методы аутентификации (логин/пароль) с дополнительным фактором, таким как SMS-код, push-уведомление, аппаратный токен или биометрические данные. hi-tech.mail.ru +2

Основные методы двухфакторной аутентификации

1. **SMS-коды и email-уведомления**

После ввода логина и пароля пользователь получает одноразовый код на телефон или email, который необходимо ввести для завершения аутентификации. rt-solar.ru +1

2. **Push-уведомления**

Используются в мобильных приложениях. При попытке входа пользователь получает push-уведомление, которое нужно подтвердить или отклонить.

 Аппаратные токены (например, YubiKey)

Физические устройства, генерирующие одноразовые коды или использующие криптографические ключи для аутентификации. microsoft.com +1

 Биометрические данные

Отпечаток пальца, распознавание лица или голоса. Требуют специального оборудования или интеграции с биометрическими системами. rt-solar.ru +1

 Сертификаты и SSL/TLS

Проверка сертификата клиента, содержащего логин и, опционально, IP-адрес или DNS-имя. Используется в сочетании с парольной аутентификацией.

6. **RADIUS и PAM (Pluggable Authentication Modules)**

Внешние сервисы, которые могут интегрироваться с базами данных для реализации 2FA. Например, PostgreSQL можно настроить на использование RADIUS-сервера для проверки второго фактора. pro100vps.ru +1

Реализация 2FA для PostgreSQL

PostgreSQL поддерживает несколько методов аутентификации, включая интеграцию с внешними системами. Для реализации 2FA можно использовать:

- **RADIUS-сервер** в сочетании с PAM-модулем. PostgreSQL делегирует проверку имени и пароля внешнему RADIUS-серверу, который, в свою очередь, может требовать второй фактор аутентификации (например, SMS-код или токен). pro100vps.ru +1
- **Сертификаты SSL/TLS**. В этом случае вторым фактором является проверка сертификата клиента. Сертификат должен содержать поле CN с логином пользователя и, опционально, поле SubjectAltName с IP-адресом или DNS-именем.
- **Интеграция с системами вроде LoginTC или miniOrange**, которые предоставляют готовые решения для 2FA с поддержкой PostgreSQL. miniorange.com +1

Пример настройки через RADIUS и PAM:

1. Настройте PostgreSQL на использование PAM, добавив запись в файл pg_hba.conf:

text

1. host all all 192.168.0.0/24 pam pamservice=postgresql
2.
3. Настройте PAM-модуль для использования RADIUS, указав сервер RADIUS и секретный ключ.
4. Настройте RADIUS-сервер для запроса второго фактора (например, SMS-кода или токена).

Рекомендации по выбору решения

При выборе системы 2FA учитывайте:

- **Объекты защиты**: определите, какие данные требуют повышенной защиты (финансовые системы, персональные данные и т.д.).

Требования регулятора: соответствие стандартам, таким как PCI DSS, GDPR, ФЗ №152-ФЗ «О персональных данных». **Удобство для пользователей**: простота регистрации, время прохождения аутентификации, кросс-платформенность. **Поддерживаемые методы аутентификации**: SMS, push-уведомления, аппаратные токены, биометрия.

Сравнение методов 2FA

Метод	Преимущества	Недостатки
SMS	Простота, доступность	Риск перехвата, зависимость от оператора связи
Push-уведомления	Быстрота, удобство	Требует интернет, зависимость от приложения
Аппаратные токены	Высокая безопасность	Стоимость, риск потери
Биометрия	Удобство, скорость	Сложность восстановления при компрометации
Сертификаты SSL/TLS	Шифрование, гибкость	Сложность настройки

Вывод

Двухфакторная аутентификация значительно повышает безопасность доступа к базам данных, сочетая несколько методов проверки личности. Для PostgreSQL можно использовать RADIUS, PAM, SSL/TLS или готовые решения вроде LoginTC и miniOrange. При выборе метода учитывайте требования к безопасности, удобство для пользователей и соответствие регуляторным стандартам.

Если у вас есть конкретные вопросы по настройке 2FA для определённой СУБД или требуется помощь в выборе решения, уточните детали.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 29 . Анализ и устранение уязвимостей базы данных с использованием встроенных инструментов безопасности PostgreSQL.

Цель работы изучить, как управлять учётными записями пользователей и контролировать доступ к базам данных

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Практическая работа по анализу и устранению уязвимостей в PostgreSQL включает несколько ключевых направлений: настройку конфигурации, управление пользователями и ролями, шифрование данных, аудит и мониторинг, а также устранение известных уязвимостей. Рассмотрим основные шаги и инструменты.

Настройка конфигурации

1. **Файл postgresql.conf**:

- Ограничьте параметр listen_addresses конкретными IP-адресами или оставьте значение localhost, чтобы предотвратить удалённый доступ с неавторизованных машин.

Включите SSL для шифрования трафика между клиентом и сервером, установив ssl = on и указав пути к SSL-сертификатам и ключам (ssl_cert_file, ssl_key_file). Используйте метод хеширования паролей scram-sha-256 вместо менее безопасного md5.

Файл pg_hba.conf:

- Настройте правила аутентификации, ограничивая доступ по IP-адресам и используя надёжные методы, такие как scram-sha-256. Избегайте метода trust, который позволяет подключение без проверки пароля.

Для доверенных хостов можно использовать md5, но для критически важных систем предпочтительнее scram-sha-256.

Файл pg_ident.conf:

- Используйте для сопоставления системных пользователей с ролями PostgreSQL при аутентификации методами peer или ident. Это помогает ограничить доступ к критическим ролям.

Управление пользователями и ролями

- Создавайте роли с паролями, используя команду CREATE ROLE user_name WITH LOGIN PASSWORD 'secure_password'.
- Внедрите политику паролей, требующую их регулярного изменения, и используйте сложные пароли.
- Используйте расширение pgaudit для аудита операций, выполняемых пользователями. Это помогает своевременно обнаруживать подозрительную активность.

Шифрование данных

- Включите SSL для шифрования данных при передаче между клиентом и сервером.

Для шифрования данных на уровне таблиц или столбцов используйте расширение pgcrypto. Например, для хеширования паролей можно применять функцию crypt() с солью, генерируемой gen_salt(). Рассмотрите шифрование диска, на котором размещена база данных, для защиты данных в случае физического доступа к серверу.

Аудит и мониторинг

- Расширение pgaudit позволяет отслеживать различные события, связанные с безопасностью, и записывать их в журналы сервера. Это помогает выявлять аномалии и несанкционированные действия.

В сертифицированной редакции Postgres Pro Standard доступно расширение pg_proaudit с расширенными возможностями аудита.

Устранение известных уязвимостей

Уязвимость	Описание	Рекомендации
CVE-2022-1552	Уязвимость в операциях автоочистки, переиндексации и других, позволяющая выполнение произвольных SQL-функций под учётной записью суперпользователя	Обновите PostgreSQL до последней версии или отключите автоочистку и избегайте ручного запуска уязвимых команд
CVE-2022-31197	Уязвимость в JDBC-драйвере PostgreSQL, позволяющая Java-программам выполнять команды от имени привилегированного пользователя	Обновите драйвер до версий 42.2.26 или 42.4.1
CVE-2025-8714	Уязвимость, позволяющая внедрять метакоманды psql в дампы базы данных, что может привести к выполнению произвольного кода	Используйте форматы custom или directory для создания резервных копий вместо plain-текстового. Перед восстановлением проверяйте дампы на наличие подозрительного кода

bigdataschool.ru +1

Дополнительные меры

- **Ф**

айрвол: заблокируйте доступ к порту PostgreSQL (по умолчанию 5432) с помощью брандмауэра, разрешив подключения только с доверенных IP-адресов.

 Регулярные обновления: своевременно обновляйте PostgreSQL и все используемые расширения до последних версий, чтобы закрыть известные уязвимости. bigdataschool.ru +1

 Проверка расширений: регулярно проверяйте расширения на наличие потенциально опасных объектов и отключайте неиспользуемые учётные записи. Для глубокого анализа

уязвимостей можно использовать инструменты пентестинга, такие как Metasploit, но их применение требует осторожности и знаний.

Важно: перед внесением изменений в конфигурацию или выполнение потенциально опасных операций создайте резервную копию базы данных.

Если у вас есть конкретные вопросы по настройке или устранению уязвимостей, уточните их — помогу разобраться!

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 30 Разработка и реализация стратегии защиты данных от несанкционированного доступа в корпоративной базе данных.

Цель работы: разработать комплексную стратегию защиты корпоративной базы данных (БД) от несанкционированного доступа, реализовать ключевые механизмы безопасности и оценить их эффективность.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.

• Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).

• Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Исходные данные

- Тип БД: реляционная (например, PostgreSQL, Oracle, MS SQL Server).
- Объём данных: средний/крупный (от 100 ГБ).
- Количество пользователей: 50–200 сотрудников.
- Типы данных: персональные данные, финансовая информация, коммерческая тайна.
- Инфраструктура: гибридная (локальные серверы + облако).

3. Этапы разработки стратегии

Этап 1. Аудит и оценка рисков

1. **Инвентаризация данных**:
   - классификация по уровню конфиденциальности (Конфиденциально, Для служебного пользования, Общедоступные);
   - определение владельцев данных и потоков информации.
2. **Анализ угроз**:
   - внешние атаки (SQL-инъекции, DDoS, фишинг);
   - внутренние угрозы (злоупотребление правами, утечка через сотрудников);
   - технические уязвимости (необновлённое ПО, слабые пароли).
3. **Оценка рисков** по методике Вероятность×Ущерб.

Этап 2. Разработка политики безопасности

1. **Правила доступа**:
   - принцип «минимальных привилегий» (Need-to-know);
   - ролевая модель (RBAC): Администратор, Аналитик, Оператор).
2. **Требования к аутентификации**:
   - сложные пароли (≥12 символов, смена каждые 90 дней);
   - многофакторная аутентификация (MFA) для администраторов.
3. **Процедуры реагирования** на инциденты (план IRP).

Этап 3. Технические меры защиты

1. **Шифрование**:
   - данные «в покое»: TDE (Transparent Data Encryption);
   - данные в передаче: TLS 1.3 (AES-256).
2. **Контроль доступа**:
   - белые списки IP-адресов;
   - ограничение подключений по времени/географии.
3. **Мониторинг и аудит**:
   - журналы событий (Audit Logs) с защитой от изменений;
   - SIEM-система для анализа аномалий.
4. **Резервное копирование**:
   - ежедневное инкрементное + еженедельное полное;
   - хранение в зашифрованном виде (3-2-1 rule).

Этап 4. Организационные меры

1. **Обучение персонала**:
   - тренинги по кибергигиене (раз в 6 месяцев);
   - тестирование на фишинг.
2. **Документооборот**:
   - соглашения о неразглашении (NDA);
   - регламенты работы с данными.
3. **Тестирование безопасности**:
   - пентесты раз в 12 месяцев;
   - сканирование уязвимостей (Nessus, OpenVAS) ежемесячно.

4. Реализация (на примере PostgreSQL)

Шаг 1. Настройка аутентификации

sql

```
-- Включение SCRAM-SHA-256
ALTER SYSTEM SET password_encryption = 'scram-sha-256';

-- Создание ролей
CREATE ROLE analyst WITH LOGIN PASSWORD 'complex_pass123!';
GRANT SELECT ON TABLE sales TO analyst;
```

Шаг 2. Шифрование трафика

1. Сгенерировать SSL-сертификаты.
2. В postgresql.conf:

ini

```
ssl = on
ssl_cert_file = '/path/to/server.crt'
ssl_key_file = '/path/to/server.key'
```

Шаг 3. Аудит действий

sql

```
-- Включить логирование
ALTER SYSTEM SET log_statement = 'all';
ALTER SYSTEM SET logging_collector = on;
```

Шаг 4. Резервное копирование

bash

```
pg_dump -U backup_user -h localhost -Fc db_name > backup.dump
gpg --encrypt --recipient "backup@company.com" backup.dump
```

5. Тестирование эффективности

1. **Проверка доступа**:
   - попытка входа с неверным MFA → ошибка 403;
   - запрос запрещённой таблицы → Permission denied.
2. **Сканирование уязвимостей**:
   - отсутствие открытых портов (nmap -p 5432 localhost);
   - валидность SSL-сертификата.
3. **Восстановление из бэкапа**:
   - время восстановления <30 мин.

6. Отчётность

1. **Ежедневные отчёты**:
   - количество неудачных входов;
   - аномальные запросы (SIEM-алерты).
2. **Ежемесячные аудиты**:
   - соответствие RBAC политике;
   - актуальность паролей.
3. **Годовой отчёт**:
   - статистика инцидентов;
   - рекомендации по улучшению.

7. Критерии успешности

- Снижение инцидентов на 70% за 6 месяцев.
- Время реагирования на угрозу <15 мин.
- Соответствие требованиям GDPR, ФЗ-152.

8. Выводы

Разработанная стратегия обеспечивает:

1. **Конфиденциальность** — шифрование и контроль доступа.
2. **Целостность** — аудит и резервное копирование.
3. **Доступность** — защита от DDoS и быстрое восстановление.

Рекомендации по развитию:

- Внедрение Zero Trust-архитектуры.
- Автоматизация реагирования на инциденты (SOAR).
- Переход на PostgreSQL 16+ с улучшенными механизмами безопасности.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Тема 2.4. Векторные базы данных

Практическая работа 31 Установка и настройка векторной базы данных (например, Milvus, Pinecone или Weaviate).

Цель работы изучить, как управлять учётными записями пользователей и контролировать доступ к базам данных

Задачи:

- Систематизировать подходы к изучению предмета.

- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Практическая работа по установке и настройке векторной базы данных может включать работу с Milvus, Pinecone или Weaviate. Ниже приведены краткие инструкции для каждого из них.

Milvus

Установка через Docker:

1. Убедитесь, что Docker установлен.
2. Скачайте скрипт установки: Invoke-WebRequest https://raw.githubusercontent.com/milvus-io/milvus/refs/heads/master/scripts/standalone_embed.bat -OutFile standalone.bat.
3. Запустите скрипт: standalone.bat start.
4. После запуска контейнер будет доступен по порту 19530.

Настройка через VK Cloud:

1. Зарегистрируйтесь в VK Cloud и перейдите в личный кабинет.
2. В разделе «Магазин приложений» найдите Milvus и нажмите «Подключить».
3. Укажите параметры сервера (тип размещения, резервное копирование, мониторинг) и виртуальной машины (сеть, зона доступности, тип ВМ, размер диска).
4. После развёртывания получите данные для доступа (milvus_url, логин, пароль, приватный ключ).

Pinecone

Регистрация и настройка:

1. Перейдите на сайт Pinecone и зарегистрируйтесь.
2. Создайте проект, выбрав регион.
3. Получите API-ключ в консоли Pinecone.

Работа с Python-клиентом:

1. Установите клиент: pip install pinecone-client.
2. Инициализируйте соединение:

python

 import pinecone
pinecone.init(api_key="ваш_ключ", environment="ваш_регион")

 Создайте индекс:

python

3. pinecone.create_index(name="индекс", dimension=1536, metric="cosine")
4.
5. Вставляйте векторы и запрашивайте данные через методы upsert и query. sparkcodehub.com +1

Weaviate

Установка через Docker:

1. Убедитесь, что Docker установлен.
2. Получите файл конфигурации: curl -o docker-compose.yml "https://configuration.semi.technology/v2/docker-compose/docker-compose.yml?enterprise_usage_collector=false&modules=standalone&runtime=docker-compose&weaviate_version=v15.0.0".
3. Запустите Docker Compose: docker-compose up -d.
4. Доступ к веб-интерфейсу будет по адресу http://localhost:8080. ipv6.rs +1

Настройка через Kubernetes:

1. Используйте Helm для установки оператора Weaviate:

bash

 helm install weaviate weaviate/weaviate -f weaviate-values.yaml
 Настройте модули (например, text2vec-openai) в файле weaviate-values.yaml.

Работа с Python-клиентом:

1. Установите клиент: pip install weaviate-client.
2. Подключитесь к Weaviate:

python

 from weaviate import Client
client = Client("http://localhost:8080")

 Создавайте коллекции, вставляйте данные и выполняйте запросы через API клиента.

Сравнение и рекомендации

Критерий	Milvus	Pinecone	Weaviate
Тип	Открытый исходный код	SaaS/PaaS	Открытый исходный код
Сложность установки	Средняя (Docker/K8s)	Простая (API-ключ)	Средняя (Docker/K8s)
Интеграция	RESTful API, клиентские библиотеки	Python SDK, REST API	REST API, GraphQL, SDK
Масштабируемость	Высокая (распределённое хранение)	Высокая (облако)	Высокая (Kubernetes)

Рекомендации:

• Для быстрого старта и прототипирования подойдёт Pinecone из-за простоты настройки.

• Для проектов с требованиями к приватности и контролю инфраструктуры лучше выбрать Milvus или Weaviate с локальным развёртыванием.

• Weaviate удобен для сложных сценариев с интеграцией генеративных моделей и семантическим поиском.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 32 Создание и управление коллекциями данных в векторной базе (создание индексов и добавление векторов).

Цель работы: освоить базовые операции по работе с векторной базой данных: создание коллекций, настройку индексов, добавление векторов и метаданных, выполнение поисковых запросов.

Задачи:

• Систематизировать подходы к изучению предмета.

• Научить правильному плану работы ИИ

• Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

• Для выполнения данного задания необходимо изучить теоретический материал по данной теме.

- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

• Предъявить преподавателю результаты работы.

• Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).

• Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

Векторная база данных — специализированная СУБД для хранения и поиска векторных эмбеддингов (числовых представлений объектов).

Ключевые понятия:

- **эмбеддинг** — вектор фиксированной длины, кодирующий семантику объекта (текста, изображения и др.);
- **индекс** — структура данных для ускорения поиска похожих векторов (HNSW, LSH, PQ);
- **метрика сходства** — способ измерения близости векторов (cosine, L2-норма);
- **метаданные** — дополнительная информация об объекте (теги, дата, категория).

Преимущества векторных БД:

- семантический поиск (по смыслу, а не ключам);
- высокая скорость поиска в больших датасетах;
- поддержка неструктурированных данных.

Практическая часть

1. Подготовка окружения

Установите зависимости (пример для Python):

bash

```
pip install chromadb sentence-transformers
```

2. Создание клиентской сессии

python

```
import chromadb
from sentence_transformers import SentenceTransformer

# Инициализация клиента
client = chromadb.PersistentClient(path="./vectordb")

# Загрузка модели для генерации эмбеддингов
model = SentenceTransformer('all-MiniLM-L6-v2')
```

3. Создание коллекции

python

```
collection = client.create_collection(
    name="documents",
    metadata={"hnsw:space": "cosine"}  # Метрика сходства
)
```

Параметры:

- name — уникальное имя коллекции;
- metadata — настройка индекса (здесь: косинусное сходство).

4. Генерация и добавление векторов

python

```
# Примеры документов
```

```
documents = [
    "Машинное обучение помогает анализировать данные",
    "Векторные базы ускоряют поиск по смыслу",
    "Эмбеддинги кодируют семантику текста"
]

# Генерация эмбеддингов
embeddings = model.encode(documents)

# Добавление в коллекцию
collection.add(
    documents=documents,
    embeddings=embeddings,
    metadatas=[{"category": "ML"}, {"category": "DB"}, {"category": "NLP"}],
    ids=["id1", "id2", "id3"]
)
```

Важные моменты:

- embeddings должны соответствовать размерности модели;
- ids — уникальные идентификаторы (повторение приведёт к ошибке);
- metadatas — опциональные метаданные для фильтрации.

5. Создание индекса

Индекс создаётся автоматически при добавлении данных. Для настройки:

python

```
# Пересоздание коллекции с оптимизированным индексом
collection = client.create_collection(
    name="optimized_docs",
    metadata={
        "hnsw:space": "l2",          # Евклидово расстояние
        "hnsw:ef_construction": 100, # Параметр поиска
        "hnsw:M": 16                 # Количество связей в графе
    }
)
```

6. Выполнение поисковых запросов

python

```
# Поиск по тексту (автоматическая генерация эмбеддинга)
results = collection.query(
    query_texts=["Как работает векторный поиск?"],
    n_results=2,
    include=["documents", "metadatas", "distances"]
)

# Вывод результатов
for doc, meta, dist in zip(
    results['documents'][0],
    results['metadatas'][0],
```

```
    results['distances'][0]
):
    print(f"Текст: {doc}")
    print(f"Метаданные: {meta}")
    print(f"Расстояние: {dist:.3f}\n")
```

7. Управление коллекциями

```
python
# Получение информации
print(f"Количество записей: {collection.count()}")

# Просмотр первых 5 элементов
print(collection.peek(limit=5))

# Фильтрация по метаданным
filtered = collection.get(
    where={"category": "ML"},
    include=["documents"]
)

# Удаление коллекции
client.delete_collection("documents")
```

Контрольные задания

1. Создайте коллекцию с метрикой L2-расстояния.
2. Добавьте 5 документов с метаданными (категория, дата).
3. Выполните поиск по ключевому слову, выведите 3 ближайших результата.
4. Проверьте количество записей в коллекции.
5. Удалите коллекцию после завершения работы.

Требования к отчёту

Отчёт должен содержать:

1. Листинги кода с комментариями.
2. Скриншоты результатов выполнения запросов.
3. Ответы на контрольные вопросы:
   - В чём отличие векторных БД от реляционных?
   - Какие метрики сходства векторов существуют?
   - Зачем нужны метаданные в векторной коллекции?
4. Выводы по работе.

Возможные ошибки и их устранение

- **Ошибка размерности векторов:** убедитесь, что модель эмбеддингов и коллекция используют одинаковую размерность.
- **Дублирование ids:** проверяйте уникальность идентификаторов перед добавлением.
- **Медленный поиск:** настройте параметры индекса (например, hnsw:ef_construction).

Рекомендуемые инструменты

- **ChromaDB** — простая в настройке векторная БД для Python;

- **Milvus** — масштабируемое решение для промышленных задач;
- **Sentence-Transformers** — библиотека для генерации эмбеддингов текста.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 33 Реализация функции поиска ближайших соседей (Nearest Neighbor Search) на примере текстовых или изображений.

Цель работы: реализовать алгоритм поиска ближайших соседей (**K-Nearest Neighbors, KNN**) для классификации текстовых данных или изображений, изучить ключевые этапы работы алгоритма и оценить его эффективность.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретическая основа

KNN — метрический алгоритм классификации и регрессии, работающий по принципу «большинство среди k ближайших соседей». Основные шаги:

1. Задать число соседей k.
2. Вычислить расстояние от нового объекта до всех объектов обучающей выборки.
3. Выбрать k ближайших соседей.
4. Присвоить классу нового объекта:
   - в классификации — класс большинства среди k соседей;
   - в регрессии — среднее значение целевых признаков соседей.

Метрики расстояния:

- Евклидово: $\rho(x,y)=\sum i(xi-yi)2$
- Манхэттенское: $\rho(x,y)=\sum i|xi-yi|$
- Косинусное: $\rho(x,y)=1-\|x\|\cdot\|y\|x\cdot y$ (часто используется для текстов)

Реализация на Python (на примере классификации изображений/текстов)

Шаг 1. Подготовка окружения

python

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_olivetti_faces, fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
```

Шаг 2. Загрузка и предобработка данных

Вариант A. Изображения (Olivetti Faces)

python

```
# Загрузка данных
faces = fetch_olivetti_faces(shuffle=True, random_state=42)
X, y = faces.data, faces.target

# Разделение на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# Нормализация признаков
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Вариант B. Тексты (20 Newsgroups)

python

```
# Загрузка данных
newsgroups = fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'))
X, y = newsgroups.data, newsgroups.target
```

```
# Векторизация текстов (TF-IDF)
vectorizer = TfidfVectorizer(max_features=5000, stop_words='english')
X_vec = vectorizer.fit_transform(X)

# Разделение на выборки
X_train, X_test, y_train, y_test = train_test_split(
    X_vec, y, test_size=0.3, random_state=42
)
```

Шаг 3. Обучение модели KNN

python

```
# Подбор оптимального k с помощью кросс-валидации
k_range = range(1, 11)
cv_scores = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=5, scoring='accuracy')
    cv_scores.append(scores.mean())

# Выбор лучшего k
best_k = k_range[np.argmax(cv_scores)]
print(f"Оптимальное k: {best_k}")

# Обучение финальной модели
knn_model = KNeighborsClassifier(n_neighbors=best_k)
knn_model.fit(X_train, y_train)
```

Шаг 4. Прогноз и оценка качества

python

```
# Предсказание на тестовой выборке
y_pred = knn_model.predict(X_test)

# Отчёт о классификации
print("Отчёт о классификации:")
print(classification_report(y_test, y_pred, target_names=newsgroups.target_names))

# Матрица ошибок
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Матрица ошибок")
plt.ylabel("Истинные классы")
plt.xlabel("Предсказанные классы")
plt.show()
```

Шаг 5. Визуализация результатов (для изображений)

python

```
# Отображение нескольких тестовых изображений и их предсказаний
fig, axes = plt.subplots(2, 5, figsize=(12, 6))
for i, ax in enumerate(axes.flat):
    if i < len(X_test):
        image = X_test[i].reshape(64, 64)  # для Olivetti Faces
        ax.imshow(image, cmap='gray')
        ax.set_title(f"Истинный: {y_test[i]}\nПредсказанный: {y_pred[i]}")
        ax.axis('off')
plt.suptitle("Примеры классификации изображений")
plt.show()
```

Анализ результатов

1. **Выбор k**:
   - Маленькое k (1–3) ведёт к переобучению (высокая вариативность).
   - Большое k сглаживает границы классов, но может упустить локальные закономерности.
   - Оптимальное k подбирается по кросс-валидации.
2. **Метрика расстояния**:
   - Для изображений часто используют евклидово расстояние.
   - Для текстов — косинусное (учитывает семантическую близость).
3. **Масштабирование признаков**:
   - Обязательно для KNN, так как алгоритм чувствителен к масштабу признаков.

Вывод

В ходе работы:

- Реализован алгоритм KNN для классификации изображений и текстов.
- Изучены этапы: предобработка данных, подбор k, обучение, оценка качества.
- Показано, что KNN прост в реализации, но требует тщательного выбора метрики и параметра k.
- Для текстов важна векторизация (TF-IDF), для изображений — нормализация.

Рекомендации по улучшению:

- Использовать алгоритмы ускоренного поиска (например, KD-tree или HNSW) для больших данных.
- Применять ансамблевые методы (например, взвешенный KNN) для повышения точности.
- Тестировать разные метрики расстояния в зависимости от типа данных.

Контрольные вопросы

1. Почему KNN называют «алгоритмом без обучения»?
2. Как выбор k влияет на качество классификации?
3. В чём преимущество косинусного расстояния для текстовых данных?
4. Какие проблемы возникают при применении KNN к большим наборам данных?

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 34 Интеграция векторной базы данных с Python для загрузки и обработки векторов.

Цель работы: освоить интеграцию векторной базы данных (на примере **ChromaDB**) с Python: создание коллекции, загрузка векторов, выполнение запросов, обработка результатов.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

Векторная база данных — специализированная СУБД для хранения векторных представлений (эмбеддингов) данных и эффективного поиска по сходству.

Ключевые понятия:

- **Эмбеддинг** — векторное представление текста/изображения/объекта (массив чисел).
- **Метрика сходства** — способ измерения расстояния между векторами (cosine, L2, inner product).
- **Коллекция** — контейнер для хранения векторов с метаданными.

Подготовка окружения

1. Установите зависимости:

bash

```
pip install chromadb sentence-transformers numpy
```

2. Импортируйте библиотеки:

python

```
import chromadb
from chromadb.utils import embedding_functions
import numpy as np
```

Шаг 1. Инициализация клиентской части

Создайте клиент для работы с базой данных:

python

```
# Локальное хранение (файл на диске)
client = chromadb.PersistentClient(path="./chroma_db")


# Или временное хранение (в памяти)
# client = chromadb.Client()
```

Шаг 2. Создание коллекции

Определите коллекцию с функцией эмбеддинга:

python

```
# Функция для преобразования текста в вектор
sentence_transformer_ef = embedding_functions.SentenceTransformerEmbeddingFunction(
    model_name="all-MiniLM-L6-v2"
)

# Создание коллекции
collection = client.create_collection(
    name="my_documents",
    embedding_function=sentence_transformer_ef,
    metadata={"hnsw:space": "cosine"}  # Метрика сходства
)
```

Параметры:

- name — уникальное имя коллекции.
- embedding_function — модель для генерации векторов.
- metadata["hnsw:space"] — метрика (cosine, l2, ip).

Шаг 3. Загрузка данных

Добавьте документы с метаданными:

python

```
documents = [
    "Кошка любит молоко.",
    "Собака гоняется за мячом.",
    "Птицы летают в небе."
]
```

```
ids = ["id1", "id2", "id3"]

metadatas = [
    {"category": "животные", "lang": "ru"},
    {"category": "животные", "lang": "ru"},
    {"category": "природа", "lang": "ru"}
]

# Добавление в коллекцию
collection.add(
    documents=documents,
    metadatas=metadatas,
    ids=ids
)
```

Важно:

- Каждый id должен быть уникальным.
- metadatas — словарь для фильтрации.

Шаг 4. Выполнение запросов

Найдите ближайшие векторы к запросу:

python

```
results = collection.query(
    query_texts=["Что любят кошки?"],
    n_results=2,
    where={"lang": "ru"},  # Фильтр по метаданным
    include=["documents", "metadatas", "distances"]  # Что возвращать
)

print(results)
```

Параметры запроса:

- query_texts — текстовые запросы.
- n_results — количество возвращаемых результатов.
- where — фильтр по метаданным.
- include — какие данные возвращать (documents, metadatas, distances).

Шаг 5. Анализ результатов

Результат — словарь с ключами:

- ids — идентификаторы найденных документов.
- documents — тексты документов.
- metadatas — метаданные.
- distances — расстояния (чем меньше, тем ближе вектор).

Пример обработки:

python

```
for i, doc in enumerate(results['documents'][0]):
    print(f"Результат {i+1}: {doc}")
    print(f"Расстояние: {results['distances'][0][i]:.4f}")
```

Шаг 6. Дополнительные операции

1. **Просмотр первых элементов:**

python

```
print(collection.peek())
```

2. **Подсчёт записей:**

python

```
print(collection.count())
```

3. **Получение по ID:**

python

```
result = collection.get(ids=["id1"], include=["documents"])
print(result)
```

4. **Удаление коллекции:**

python

```
client.delete_collection("my_documents")
```

Контрольные вопросы

1. Для чего используются векторные базы данных?
2. Какие метрики сходства поддерживаются в ChromaDB?
3. Зачем нужны метаданные в коллекции?
4. Как фильтровать результаты запроса?
5. В чём разница между PersistentClient и Client?

Задания для самостоятельной работы

1. Создайте коллекцию с 10 документами на английском языке.
2. Выполните запрос на поиск синонимов к слову «happy».
3. Добавьте метаданные «author» и отфильтруйте результаты по автору.
4. Сравните результаты при разных метриках сходства (cosine vs l2).
5. Реализуйте функцию для добавления новых документов в существующую коллекцию.

Возможные ошибки и решения

- **Ошибка загрузки модели:** проверьте доступ к интернету или укажите локальный путь к модели.
- **Несовпадение размерностей векторов:** используйте одну модель для всех операций.
- **Дублирование ID:** убедитесь в уникальности идентификаторов.

Итоговый отчёт

Подготовьте отчёт, включающий:

1. Код с комментариями.
2. Примеры входных данных и результатов запросов.
3. Анализ влияния метрики сходства на результаты.
4. Ответы на контрольные вопросы.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 35 Проведение кластеризации данных в векторной базе с использованием встроенных функций.

Цель работы: освоить методику кластеризации векторных данных с применением встроенных инструментов, понять принципы группировки объектов по семантической близости и оценить влияние кластеризации на скорость и качество поиска.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

Векторные базы данных — специализированные системы для хранения и извлечения неструктурированных данных (текст, изображения, аудио) через числовые векторные представления (эмбеддинги). В отличие от реляционных баз, они ищут не точные совпадения, а семантическое сходство, используя метрики расстояний.

Кластеризация — метод группировки объектов в кластеры так, чтобы:

- внутри кластера объекты были максимально похожи (малое расстояние в векторном пространстве);
- между кластерами различия были значительны (большое расстояние).

Преимущества кластеризации:

- ускорение поиска (поиск ведётся внутри одного кластера);
- снижение нагрузки на память (меньше векторов для сравнения);
- повышение релевантности результатов.

Ключевой алгоритм: *K-Means* — итеративный метод разбиения на k кластеров с минимизацией суммарной ошибки относительно центроидов.

Ход работы

1. Подготовка среды и данных

1. Установите необходимые библиотеки:

bash

 pip install numpy scikit-learn faiss-cpu

 Загрузите векторные данные (пример — текстовые эмбеддинги):

python

```
import numpy as np
# Пример: матрица векторов размером (N, D)
vectors = np.random.rand(1000, 128)  # 1000 векторов размерности 128

```

2. Реализация кластеризации K-Means

Используйте sklearn.cluster.KMeans:

python

```
from sklearn.cluster import KMeans

# Параметры
k = 10  # число кластеров
n_init = 10  # число инициализаций
max_iter = 300  # макс. число итераций

# Инициализация и обучение
kmeans = KMeans(n_clusters=k, n_init=n_init, max_iter=max_iter, random_state=42)
labels = kmeans.fit_predict(vectors)  # метки кластеров для каждого вектора
centroids = kmeans.cluster_centers_  # координаты центроидов
```

Параметры для эксперимента:

- k∈{5,10,20};
- init='k-means++' (улучшенная инициализация);
- tol=1e-4 (критерий сходимости).

3. Оценка качества кластеризации

1. **Inertia (сумма квадратов расстояний до центроидов):**

python

 inertia = kmeans.inertia_

```
print(f"Inertia: {inertia}")
```

 Коэффициент силуэта (Silhouette Score):

python

 from sklearn.metrics import silhouette_score

```
score = silhouette_score(vectors, labels)
print(f"Silhouette Score: {score:.3f}")
```

 Визуализация (для 2D/3D проекции):

python

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Снижение размерности до 2D
pca = PCA(n_components=2)
reduced_vectors = pca.fit_transform(vectors)

    # График
    plt.scatter(reduced_vectors[:, 0], reduced_vectors[:, 1], c=labels, cmap='tab10')
    plt.title("K-Means Clustering (PCA)")
    plt.show()
```

4. Поиск в кластеризованной базе

1. Определите кластер для запроса:

python

```
 query_vector = np.random.rand(1, 128)  # вектор запроса
query_label = kmeans.predict(query_vector)[0]
```

 Извлеките векторы из целевого кластера:

python

```
 cluster_indices = np.where(labels == query_label)[0]
cluster_vectors = vectors[cluster_indices]
```

 Найдите ближайшие соседи внутри кластера (например, через FAISS):

python

```
import faiss

# Создание индекса для кластера
index = faiss.IndexFlatL2(128)  # L2-расстояние
index.add(cluster_vectors)

# Поиск 5 ближайших
    k_search = 5
    distances, indices = index.search(query_vector, k_search)

    # Индексы в исходной базе
    final_indices = cluster_indices[indices[0]]
    print("Ближайшие индексы:", final_indices)
```

5. Сравнение с полным перебором

1. Создайте индекс FAISS для всей базы:

python

```
full_index = faiss.IndexFlatL2(128)
full_index.add(vectors)
full_distances, full_indices = full_index.search(query_vector, 5)
```

4.

5. Сравните время поиска и релевантность результатов.

Содержание отчёта

1. **Введение**
   - Цель работы.
   - Краткое описание K-Means и векторных баз.
2. **Методика**
   - Схема алгоритма кластеризации.
   - Параметры экспериментов (k, метрики).
3. **Результаты**
   - Таблица: k vs Inertia vs Silhouette Score.
   - Графики кластеризации (PCA).
   - Время поиска: кластерный vs полный перебор.
4. **Анализ**
   - Как k влияет на качество и скорость?
   - Плюсы/минусы кластеризации для поиска.
5. **Выводы**
   - Оптимальное k для ваших данных.
   - Практическая значимость кластеризации.

Контрольные вопросы

1. Почему кластеризация ускоряет поиск в векторных базах?
2. Как выбрать оптимальное число кластеров k?
3. В чём отличие K-Means от иерархической кластеризации?
4. Какие метрики расстояния можно использовать вместо L2?
5. Как обработать новый вектор, если он далёк от всех центроидов?

Дополнительные задания (по желанию)

1. Реализуйте **иерархическую кластеризацию** (sklearn.cluster.AgglomerativeClustering) и сравните с K-Means.
2. Используйте **DBSCAN** для выявления шумовых векторов.
3. Оптимизируйте поиск через **IVF-индексы** в FAISS.
4. Оцените влияние **нормализации векторов** на качество кластеризации.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 36 Построение векторов для текстовых данных с использованием моделей преобразования (например, Word2Vec, BERT).

Цель работы: освоить методы векторизации текста с помощью современных NLP-моделей: изучить принципы работы Word2Vec и BERT, реализовать построение векторных представлений слов/предложений и проанализировать их свойства.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретическая часть

Векторизация текста — преобразование слов, фраз или документов в числовые векторы фиксированной размерности, сохраняющие семантическую и синтаксическую информацию.

Основные модели

1. **Word2Vec** (Google, 2013)
   - Принцип: контекстная близость слов → схожесть векторов.
   - Алгоритмы:
     - *Skip-Gram*: предсказывает окружающие слова по центральному.
     - *CBOW* (Continuous Bag of Words): предсказывает центральное слово по контексту.
   - Особенности:
     - Фиксированная размерность вектора (обычно 100–300).
     - Улавливает семантические аналогии (например, король−мужчина+женщина≈королева).
     - Не учитывает морфологию (разные формы слова могут иметь разные векторы).

2. **BERT** (Google, 2018)

- Принцип: двунаправленное трансформер-внимание, контекстно-зависимые векторы.
- Особенности:
  - Для одного слова вектор зависит от контекста предложения.
  - Использует маскированное предсказание слов и классификацию следующих предложений.
  - Выходная размерность: 768 (base) или 1024 (large).
  - Лучше учитывает синтаксис и многозначность.

Метрики сходства векторов

- **Косинусное сходство**:

similarity(A,B)=‖A‖·‖B‖A·B=∑i=1nAi2 ·∑i=1nBi2 ∑i=1nAiBi

  Евклидово расстояние: ∑i=1n(Ai−Bi)2

- .

Практическая часть

Шаг 1. Подготовка окружения

Установите библиотеки:

bash

```
pip install transformers torch sentence-transformers gensim numpy matplotlib
```

Шаг 2. Работа с Word2Vec

1. **Обучение модели** (на своём корпусе):

python

```
from gensim.models import Word2Vec
import nltk

# Пример корпуса (список предложений, каждое — список слов)
corpus = [
    ["кот", "любит", "молоко"],
    ["собака", "гоняет", "кота"],
    # ...
]

model = Word2Vec(
    sentences=corpus,
    vector_size=100,      # размерность вектора
    window=5,          # окно контекста
    min_count=1,       # мин. частота слова
    workers=4,         # потоки
    sg=1               # 1=Skip-Gram, 0=CBOW
)
```

2. **Получение векторов и анализ**:

python

```
# Вектор слова
```

```
vector = model.wv['кот']

# Ближайшие слова
similar = model.wv.most_similar('кот', topn=5)

# Семантическая аналогия
result = model.wv.most_similar(
    positive=['король', 'женщина'],
    negative=['мужчина'],
    topn=1
)
```

Шаг 3. Работа с BERT

1. **Использование предобученной модели** (через sentence-transformers):

```
python
from sentence_transformers import SentenceTransformer

model = SentenceTransformer('all-MiniLM-L6-v2')  # лёгкая модель для предложений

# Векторизация предложений
sentences = [
    "Кот любит молоко.",
    "Собака гоняет кота."
]
embeddings = model.encode(sentences)  # массив (2, 384)
```

2. **Анализ сходства**:

```
python
from sklearn.metrics.pairwise import cosine_similarity

similarity = cosine_similarity([embeddings[0]], [embeddings[1]])
print(f"Сходство: {similarity[0][0]:.3f}")
```

Шаг 4. Визуализация

Постройте график ближайших слов (Word2Vec) с помощью PCA:

```
python
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

words = ['кот', 'собака', 'молоко', 'бежать', 'играть']
word_vectors = [model.wv[word] for word in words]

pca = PCA(n_components=2)
reduced = pca.fit_transform(word_vectors)

plt.figure(figsize=(8, 6))
for i, word in enumerate(words):
    plt.scatter(reduced[i, 0], reduced[i, 1])
```

```
    plt.annotate(word, (reduced[i, 0], reduced[i, 1]))
plt.show()
```

Задания для выполнения

1. Обучите модель Word2Vec на корпусе из 100+ предложений (можно использовать новостные статьи или отзывы).
2. Для 5 слов найдите 3 ближайших соседа по косинусному сходству.
3. Сравните векторы синонимов и антонимов (например, «хороший»/«плохой»).
4. Используя BERT, получите векторы для 5 предложений и постройте матрицу сходства.
5. Визуализируйте 10 слов из вашей модели Word2Vec в 2D-пространстве.

Контрольные вопросы

1. В чём ключевое отличие Word2Vec от BERT?
2. Почему косинусное сходство чаще используют для векторов слов, чем евклидово расстояние?
3. Как размер окна контекста (window) влияет на качество векторов в Word2Vec?
4. Что означает параметр vector_size в Word2Vec?
5. Приведите пример семантической аналогии, которую может решить Word2Vec.

Требования к отчёту

1. Текст работы с описанием шагов и кодом.
2. Таблицы с результатами (ближайшие слова, матрицы сходства).
3. Графики визуализации.
4. Ответы на контрольные вопросы.
5. Выводы о преимуществах и ограничениях каждой модели.

Рекомендуемые ресурсы

- Документация gensim (Word2Vec):

Репозиторий sentence-transformers:

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 37 Создание векторного хранилища для изображений и реализация поиска по сходству.

Цель работы: освоить технологию векторного поиска: создать хранилище векторных представлений изображений и реализовать механизм поиска по визуальному сходству.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

Векторный поиск — технология поиска по смысловому (визуальному) сходству, а не по точным совпадениям метаданных.

Эмбеддинг (embedding) — плотное векторное представление объекта (в данном случае изображения) в многомерном пространстве. Для изображений обычно используются нейросети-энкодеры (ResNet, EfficientNet, CLIP и др.).

Метрики сходства для сравнения векторов:

- косинусное сходство: cosine_similarity(u,v)=‖u‖‖v‖u·v;
- евклидово расстояние: euclidean_distance(u,v)=∑i=1n(ui−vi)2.

Инструменты и библиотеки

Рекомендуемые векторные базы данных:

- **Pinecone** — облачное решение с простым API;
- **Milvus** — open-source, поддерживает интеграцию с PyTorch/TensorFlow;
- **Qdrant** — open-source, оптимизирован для высокопроизводительного поиска;
- **Faiss** (Facebook AI Similarity Search) — библиотека для быстрого поиска в больших векторных пространствах.

Библиотеки для извлечения признаков:

- torchvision (с предобученными моделями ResNet, EfficientNet);
- clip (OpenAI CLIP для мультимодального представления);
- sentence-transformers (для текстовых описаний изображений).

Пошаговая реализация

Шаг 1. Подготовка данных

1. Соберите набор изображений (например, 1 000–10 000 файлов в форматах JPG/PNG).
2. Организуйте структуру папок:
3. /dataset/
4. ├── img_001.jpg
5. ├── img_002.jpg
6. └── ...
7. При необходимости проведите предобработку:
   - изменение размера (например, до 224×224 px);
   - нормализация пикселей.

Шаг 2. Извлечение векторных представлений

1. Выберите предобученную модель (например, ResNet50 из torchvision.models).
2. Напишите код для получения эмбеддингов:

```
python
import torch
import torchvision.models as models
import torchvision.transforms as transforms
from PIL import Image
import os
import numpy as np

# Загрузка модели
model = models.resnet50(weights='IMAGENET1K_V2')
model.fc = torch.nn.Identity()  # Удаляем последний слой
model.eval()

# Преобразования
preprocess = transforms.Compose([
   transforms.Resize(256),
   transforms.CenterCrop(224),
   transforms.ToTensor(),
   transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

# Функция извлечения эмбеддинга
def get_embedding(image_path):
   image = Image.open(image_path).convert('RGB')
   image_tensor = preprocess(image).unsqueeze(0)
   with torch.no_grad():
      embedding = model(image_tensor)
   return embedding.squeeze().numpy()
```

3. Пройдите по всем изображениям и сохраните эмбеддинги в массив/словарь.

Шаг 3. Создание векторного хранилища

Вариант А. Использование Qdrant (локально)

1. Установите Qdrant: pip install qdrant-client.

2. Запустите сервер (например, через Docker):

bash

```
docker run -p 6333:6333 -p 6334:6334 qdrant/qdrant
```

3. Создайте коллекцию и загрузите векторы:

python

```
from qdrant_client import QdrantClient
from qdrant_client.http.models import Distance, VectorParams

client = QdrantClient("http://localhost:6333")

# Создание коллекции
client.recreate_collection(
    collection_name="image_embeddings",
    vectors_config=VectorParams(size=2048, distance=Distance.COSINE)
)

# Загрузка данных
for img_id, embedding in embeddings_dict.items():
    client.upsert(
        collection_name="image_embeddings",
        points=[
            {
                "id": img_id,
                "vector": embedding.tolist(),
                "payload": {"image_path": f"/dataset/img_{img_id}.jpg"}
            }
        ]
    )
```

Вариант B. Использование Pinecone (облако)

1. Зарегистрируйтесь на .
2. Получите API-ключ и среду.
3. Загрузите векторы:

python

```
import pinecone

pinecone.init(api_key="YOUR_API_KEY", environment="us-west1-gcp")
index = pinecone.Index("image-search")


for img_id, embedding in embeddings_dict.items():
    index.upsert([(str(img_id), embedding.tolist())])
```

Шаг 4. Реализация поиска по сходству

1. Извлеките эмбеддинг для поискового изображения (аналогично Шагу 2).
2. Выполните запрос к векторной базе:

Для Qdrant:

python
```
results = client.search(
    collection_name="image_embeddings",
    query_vector=search_embedding.tolist(),
    limit=5  # Количество ближайших соседей
)
```
Для Pinecone:
python
```
results = index.query(search_embedding.tolist(), top_k=5)
```
3. Выведите результаты (пути к изображениям, оценки сходства).

Шаг 5. Визуализация результатов

1. Отобразите поисковое изображение.
2. Ниже покажите 5 наиболее похожих изображений из базы.
3. Подпишите оценки сходства (например, косинусное сходство от 0 до 1).

Пример кода (сводка)
python
```
# 1. Извлечение эмбеддинга для запроса
search_embedding = get_embedding("query_image.jpg")

# 2. Поиск ближайших векторов
results = client.search(
    collection_name="image_embeddings",
    query_vector=search_embedding.tolist(),
    limit=5
)

# 3. Вывод результатов
print("Найденные изображения:")
for hit in results:
    print(f"ID: {hit.id}, Сходство: {hit.score:.3f}, Путь: {hit.payload['image_path']}")
```
Контрольные вопросы

1. Что такое эмбеддинг изображения и как он получается?
2. В чём преимущество векторного поиска перед поиском по метаданным?
3. Какие метрики сходства векторов существуют? Приведите формулы.
4. Назовите 3 векторные базы данных и их особенности.
5. Как влияет размерность вектора на скорость и точность поиска?

Требования к отчёту

1. Описание выбранных инструментов (модель для эмбеддингов, векторная база).
2. Листинги ключевых фрагментов кода.
3. Примеры поисковых запросов и результатов (скриншоты).
4. Анализ точности поиска (например, доля релевантных результатов среди топ-5).
5. Выводы о преимуществах и ограничениях реализованного решения.

Дополнительные задания (по желанию)

1. Добавьте фильтрацию результатов по метаданным (например, по дате съёмки).

2. Реализуйте ранжирование по комбинации визуального сходства и текстового описания.

3. Оптимизируйте скорость поиска (например, используя приближённые методы в Faiss).

4. Создайте веб-интерфейс для загрузки запросов и отображения результатов.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 38 Оптимизация индексов в векторной базе данных для увеличения скорости поиска.

Цель работы: освоить методы оптимизации индексов в векторных базах данных, направленные на повышение скорости поиска сходных векторов.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

Векторный индекс — структура данных, ускоряющая поиск векторов, похожих на целевой. Без индекса поиск требует полного перебора (O(*n*)), что неприемлемо для больших наборов данных.

Ключевые компромиссы:

- скорость поиска ↔ точность;
- потребление памяти ↔ производительность;
- время построения индекса ↔ эффективность запросов.

Основные типы векторных индексов

1. **FLAT**
   - гарантирует 100 % recall (полную точность);
   - не сжимает векторы;
   - сравнивает запрос со всеми векторами в базе;
   - подходит для небольших наборов (до миллиона векторов);
   - самый медленный вариант.
2. **IVF_FLAT** (Inverted File + FLAT)
   - делит векторы на кластеры (*nlist*);
   - сравнивает запрос с центрами кластеров, затем исследует *nprobe* ближайших кластеров;
   - позволяет балансировать скорость и точность через *nprobe*;
   - ускоряет поиск за счёт частичного перебора.
3. **IVF_SQ8**
   - сжимает векторы (скалярное квантование);
   - быстрее IVF_FLAT, но менее точно;
   - экономит память.
4. **HNSW** (Hierarchical Navigable Small World)
   - граф-ориентированный алгоритм;
   - строит многоуровневую навигационную структуру;
   - эффективен для динамических данных и фильтрации;
   - параметры: *m* (максимальная степень узла), *ef* (диапазон поиска).

Методы оптимизации

1. **Квантование данных**
   - замена Float32 на BFloat16 сокращает объём данных вдвое;
   - компромисс: потеря точности ради скорости;
   - альтернатива: хранение двух копий (квантованной и полной точности).
2. **Настройка гранулярности (*GRANULARITY*)**
   - большое значение (по умолчанию — 100 млн) снижает частоту перестроения индекса;
   - уменьшать только при проблемах с потреблением памяти.
3. **Параллелизация построения индексов**
   - установить число потоков = числу ядер CPU;
   - ускоряет создание индекса.

4. **Отложенное построение индексов**
   - отключить создание индексов при вставках (materialize_skip_indexes_on_insert);
   - построить индексы позже (например, в период низкой нагрузки).
5. **Использование кэша индексов**
   - загрузка индекса в оперативную память один раз;
   - повторное использование для последующих запросов.
6. **Фильтрация данных**
   - **префильтрация**: применение условий WHERE до векторного поиска (риск полного перебора);
   - **постфильтрация**: векторный поиск → фильтрация (может вернуть меньше строк, чем LIMIT).
7. **Оптимизация хранения**
   - отключить сжатие для векторных столбцов: CODEC(NONE);
   - добавить ограничения на длину векторов: CONSTRAINT same_length CHECK length(vectors) = 256.

Практические рекомендации

1. **Для малых наборов (< 1 млн векторов)**
   - использовать FLAT для 100 % точности;
   - если допустима потеря точности — IVF_SQ8.
2. **Для больших наборов (> 1 млн векторов)**
   - IVF_FLAT с настройкой nprobe (начало с 10 – 20 % от nlist);
   - HNSW для динамических данных и сложной фильтрации.
3. **При частых вставках**
   - отложенное построение индексов;
   - параллелизация с числом потоков = числу CPU-ядер.
4. **Для экономии памяти**
   - квантование (BFloat16);
   - отключение сжатия (CODEC(NONE) только если это критично для скорости).
5. **При использовании фильтров**
   - индексировать только поля, участвующие в WHERE;
   - для строковых полей — полнотекстовый индекс с токенизацией.

Пример настройки индекса (ClickHouse)

sql

```
CREATE TABLE vectors (
   id UInt32,
   vec Array(Float32) CODEC(NONE),
   INDEX idx vec TYPE vector_similarity('hnsw', 'L2Distance', 2)
) ENGINE = MergeTree
ORDER BY id;
```

Контрольные вопросы

1. В чём основное отличие FLAT от IVF_FLAT?
2. Как параметр nprobe влияет на поиск в IVF_FLAT?
3. Почему HNSW эффективен для фильтрации данных?
4. Какие компромиссы возникают при квантовании векторов?
5. Когда целесообразно использовать отложенное построение индексов?

Задание для выполнения

1. Создайте таблицу с векторным индексом типа HNSW.
2. Загрузите тестовый набор векторов (100 тыс. записей, размерность 128).
3. Выполните поиск ближайших соседей без индекса и с индексом — сравните время.
4. Настройте nprobe для IVF_FLAT и оцените влияние на скорость/точность.
5. Примените квантование (BFloat16) и измерьте выигрыш в скорости.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 39 Обеспечение масштабируемости и высокой доступности векторной базы данных.

Цель работы изучить, как управлять учётными записями пользователей и контролировать доступ к базам данных

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.

- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Практическая работа по обеспечению масштабируемости и высокой доступности векторной базы данных включает несколько ключевых аспектов: выбор стратегии масштабирования, оптимизацию индексирования, балансировку нагрузки, репликацию данных и использование распределённых систем. Эти меры позволяют обрабатывать большие объёмы данных, обеспечивать быстрый поиск по сходству и минимизировать простои.

Стратегии масштабирования

1. **Вертикальное масштабирование** — увеличение ресурсов на одном сервере (процессор, память, хранилище). Подходит для небольших приложений с предсказуемым ростом, но имеет ограничения из-за аппаратных лимитов и риска единой точки отказа.

 Горизонтальное масштабирование — распределение данных по нескольким серверам. Позволяет обрабатывать большие объёмы данных и обеспечивает высокую доступность за счёт репликации и шардинга. itweek.ru +1

 Гибридный подход — комбинация вертикального и горизонтального масштабирования. Например, можно сначала модернизировать серверы, а затем распределять базу данных по нескольким узлам при дальнейшем росте.

Оптимизация индексирования

Для ускорения поиска в векторных базах данных используются специализированные алгоритмы индексирования:

- **HNSW (Hierarchical Navigable Small World)** — графовый алгоритм, который ускоряет поиск ближайших соседей за счёт иерархической структуры. itweek.ru +1
- **PQ (Product Quantization)** — метод сжатия векторов высокой размерности в компактные представления с минимальными потерями информации.

ANN (Approximate Nearest Neighbors) — приближённый поиск ближайших соседей, который снижает вычислительные затраты по сравнению с точным поиском.

Балансировка нагрузки и репликация

- **Балансировка нагрузки** обеспечивает равномерное распределение запросов по узлам, предотвращая перегрузку отдельных серверов.

Репликация создаёт копии данных на нескольких узлах, что повышает отказоустойчивость и доступность. При выходе из строя одного узла запросы могут обрабатываться другими узлами.

Распределённые системы и фреймворки

Использование фреймворков распределённых вычислений, таких как **Apache Spark** или **Hadoop**, позволяет масштабировать векторные базы данных за счёт параллельной обработки данных на нескольких узлах. Это особенно полезно для обработки сложных запросов и больших массивов данных.

Выбор векторной базы данных

При выборе системы стоит учитывать требования к масштабируемости, скорости поиска, поддержке сообщества и интеграцию с другими инструментами. Некоторые популярные варианты:

- **Milvus** — открытая система с поддержкой горизонтального масштабирования, репликации и различных алгоритмов индексирования. blog.milvus.io +1
- **Pinecone** — облачная векторная база данных с высокой производительностью и масштабируемостью.

Qdrant — модульная система с поддержкой HNSW и других методов индексирования. **Chroma** — открытая база данных, оптимизированная для работы с LLM-приложениями.

Дополнительные рекомендации

- **Кэширование часто используемых данных** для снижения нагрузки на основную базу данных.
- **Оптимизация запросов** — переписывание запросов для снижения сложности, использование планировщиков запросов, которые оптимизируют путь выполнения на основе распределения данных и шаблонов рабочей нагрузки.

Мониторинг и настройка — использование инструментов для анализа журналов, обнаружения узких мест и устранения неполадок.

Теорема CAP и трилемма векторных баз данных

При масштабировании векторных баз данных важно учитывать теорему CAP (Consistency, Availability, Partition tolerance), которая описывает компромиссы между согласованностью, доступностью и устойчивостью к разделению. В контексте векторных баз данных также актуальна трилемма, сформулированная Zilliz: баланс между экономической эффективностью, точностью и производительностью.

Вывод: Обеспечение масштабируемости и высокой доступности векторной базы данных требует комплексного подхода, включающего выбор стратегии масштабирования, оптимизацию индексирования, балансировку нагрузки, репликацию и использование распределённых систем. При выборе конкретной системы важно учитывать требования к производительности, объёму данных и интеграцию с другими инструментами.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Практическая работа 40 Интеграция векторной базы данных в приложение для рекомендаций или кластеризации пользователей.

Цель работы: освоить интеграцию векторной базы данных (Vector DB) в прикладное приложение для построения систем рекомендаций или кластеризации пользователей на основе семантического сходства.

Задачи:

- Систематизировать подходы к изучению предмета.
- Научить правильному плану работы ИИ
- Показать основные приемы эффективного использования возможностей ИИ

Оборудование, технические средства и инструменты:

- Методические рекомендации для практических занятий
- Компьютер с подключением к сети Интернет

Ход практического занятия:

- Для выполнения данного задания необходимо изучить теоретический материал по данной теме.
- Руководствуясь методическими рекомендациями (инструкционная карта работы), студенты выполняют практическую работу. Предъявить преподавателю результаты работы.

- Предъявить преподавателю результаты работы.
- Оформить отчет по практической работе в соответствии с требованиями (содержание отчета см. ниже).
- Подготовить ответы на контрольные вопросы.

Теоретические и учебно-методические материалы по теме практической работы

Теоретические основы

Векторная база данных — NoSQL-решение для хранения, индексирования и поиска похожих векторов (эмбеддингов).

Ключевые операции:

- векторизация объектов (тексты, изображения, профили пользователей);
- сохранение векторов + метаданные;
- поиск ближайших соседей (similarity search);
- кластеризация по векторным представлениям.

Почему векторные БД?

- поиск по смыслу, а не по ключам;
- поддержка неструктурированных данных;
- высокая скорость поиска среди миллионов записей;
- интеграция с ML-моделями.

Инструменты и технологии

Популярные векторные БД:

- **Milvus** (open-source, высокая производительность);
- **Pinecone** (облачная, простота API);
- **Chroma** (open-source, интеграция с LangChain);
- **Weaviate** (гибридный поиск, open-source);

- **Qdrant** (Rust, низкая задержка).

Модели векторизации:

- Sentence-Transformers (all-MiniLM-L6-v2 и др.);
- OpenAI Embeddings;
- Word2Vec, TF-IDF (для базовых сценариев).

Пошаговая реализация

Шаг 1. Подготовка данных

1. Соберите профили пользователей (текстовые описания, истории действий, теги).
2. Очистите и нормализуйте данные (удаление стоп-слов, лемматизация).
3. Определите признаки для векторизации (например, «интересы», «покупки», «отзывы»).

Шаг 2. Векторизация

1. Выберите модель эмбеддингов (например, all-MiniLM-L6-v2).
2. Преобразуйте каждый профиль в вектор фиксированной размерности (например, 384):

```
python
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('all-MiniLM-L6-v2')
vectors = model.encode(user_profiles)  # shape: (N, 384)
```

Шаг 3. Настройка векторной БД

Пример для Qdrant (локальный запуск):

1. Установите Qdrant: docker run -p 6333:6333 -p 6334:6334 qdrant/qdrant
2. Создайте коллекцию:

```
python
from qdrant_client import QdrantClient
client = QdrantClient("http://localhost:6333")

client.recreate_collection(
   collection_name="user_vectors",
   vectors_config=models.VectorParams(
      size=384,
      distance=models.Distance.COSINE
   )
)
```

Шаг 4. Загрузка данных

1. Сохраните векторы и метаданные (ID пользователя, теги):

```
python
client.upload_collection(
   collection_name="user_vectors",
   vectors=vectors,
   payload=[{"user_id": uid, "tags": tags} for uid, tags in user_data]
)
```

Шаг 5. Реализация рекомендаций/кластеризации

А. Рекомендации (поиск ближайших соседей):

```
python
```

```
query_vector = model.encode("интересы пользователя X")
results = client.search(
    collection_name="user_vectors",
    query_vector=query_vector,
    limit=5  # топ-5 похожих пользователей
)
```

Б. Кластеризация (на примере DBSCAN):

1. Извлеките все векторы из БД.
2. Примените алгоритм:

```
python
from sklearn.cluster import DBSCAN
clustering = DBSCAN(eps=0.5, min_samples=2).fit(vectors)
labels = clustering.labels_
```

3. Назначьте кластеры пользователям.

Шаг 6. Оценка качества

- **Для рекомендаций**: Precision@k, Recall@k, NDCG.
- **Для кластеризации**: Silhouette Score, Calinski-Harabasz Index.
- A/B-тестирование (если есть продакшен-трафик).

Шаг 7. Интеграция в приложение

1. Создайте API-эндпоинты:
   - /recommend?user_id=123 → список рекомендаций;
   - /cluster?user_id=123 → ID кластера.
2. Кешируйте результаты (Redis/Memcached).
3. Настройте мониторинг (время отклика, ошибки).

Пример сценария: рекомендательная система для соцсети

1. **Вход**: профиль пользователя (био, посты, лайки).
2. **Векторизация**: преобразование текста в вектор.
3. **Поиск**: нахождение 10 ближайших профилей в векторном пространстве.
4. **Вывод**: «Вам могут понравиться: UserA, UserB…».

Возможные сложности и решения

- **Масштабирование**: используйте шардирование в векторной БД.
- **Актуальность данных**: периодическое перестроение векторов.
- **Холодный старт**: комбинируйте с rule-based рекомендациями.
- **Качество эмбеддингов**: тестируйте разные модели векторизации.

Отчёт по работе

Включите:

1. Описание выбранных инструментов (БД, модель векторизации).
2. Схема архитектуры системы.
3. Код ключевых этапов (векторизация, загрузка, поиск).
4. Результаты оценки качества.
5. Анализ проблем и путей оптимизации.

Дополнительные возможности

- **Гибридные рекомендации**: сочетание векторных и collaborative filtering методов.
- **Реальное время**: потоковая обработка новых данных (Kafka + Vector DB).

- **Мультимодальность**: векторизация текста + изображений профилей.

Итог: вы освоили полный цикл интеграции векторной БД для интеллектуальных рекомендаций/кластеризации, что применимо в e-commerce, соцсетях, контент-платформах.

Оформление результатов работы

Оформить отчёт о проделанной работе, который должен содержать исчерпывающие текстовые ответы на поставленные вопросы с решениями, пояснениями, результатами решения:

Содержание отчёта

1. Цель.

2. Раскрытие темы работы с сопровождением необходимыми материалами по построению и кодированию.

Ответить на контрольные вопросы.(устно)

Сформулировать выводы по результатам работы.

Сдать и защитить работу.

Критерии оценки

Оценка «отлично» ставится в том случае, если студент:

- свободно применяет полученные знания при выполнении практических заданий;
- выполнил работу в полном объеме с соблюдением необходимой последовательности действий;
- в письменном отчете по работе правильно и аккуратно выполнены все записи;
- при ответах на контрольные вопросы правильно понимает их сущность, дает точное определение и истолкование основных понятий, использует специальную терминологию дисциплины, не затрудняется при ответах на видоизмененные вопросы, сопровождает ответ примерами.

Оценка «хорошо» ставится, если:

- выполнены требования к оценке «отлично», но допущены 2 – 3 недочета при выполнении практических заданий и студент может их исправить самостоятельно или при небольшой помощи преподавателя;
- в письменном отчете по работе делает незначительные ошибки;
- при ответах на контрольные вопросы не допускает серьезных ошибок, легко устраняет отдельные неточности, но затрудняется в применении знаний в новой ситуации, приведении примеров.

Оценка «удовлетворительно» ставится, если:

- практическая работа выполнена не полностью, но объем выполненной части позволяет получить правильные результаты и выводы;
- в ходе выполнения работы студент продемонстрировал слабые практические навыки, были допущены ошибки;
- студент умеет применять полученные знания при решении простых задач по готовому алгоритму;
- в письменном отчете по работе допущены ошибки;
- при ответах на контрольные вопросы правильно понимает их сущность, но в ответе имеются отдельные пробелы и при самостоятельном воспроизведении материала требует дополнительных и уточняющих вопросов преподавателя.

Оценка «неудовлетворительно» ставится, если:

- практическая работа выполнена не полностью и объем выполненной работы не позволяет сделать правильных выводов, у студента имеются лишь отдельные представления об изученном материале, большая часть материала не усвоена;

- в письменном отчете по работе допущены грубые ошибки, либо он вообще отсутствует;

- на контрольные вопросы студент не может дать ответов, так как не овладел основными знаниями и умениями в соответствии с требованиями программы.