Автономная некоммерческая профессиональная образовательная организация
«МЕЖДУНАРОДНЫЙ ВОСТОЧНО-ЕВРОПЕЙСКИЙ КОЛЛЕДЖ»
Пушкинская ул., д. 268, 426008, г. Ижевск. Тел.: (3412) 77-68-24. E-mail: mveu@mveu.ru, www. mveu.ru
ИНН 1831200089. ОГРН 1201800020641

20.02.2026 г.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

по выполнению практических работ

профессионального модуля

ПМ.02 АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ

по специальности

09.02.13 Интеграция решений с применением технологий искусственного интеллекта

2026

Практическая работа – небольшой научный отчет, обобщающий проведенную учащимся работу, которую представляют для защиты преподавателю.

В процессе практического занятия учащиеся выполняют одну или несколько практических работ (заданий) под руководством преподавателя в соответствии с изучаемым содержанием учебного материала.

Состав и содержание практических занятий направлены на реализацию Государственных требований.

Наряду с формированием умений и навыков в процессе практических занятий обобщаются, систематизируются, углубляются и конкретизируются теоретические знания, вырабатывается способность и готовность использовать теоретические знания на практике, развиваются интеллектуальные умения.

Практические занятия проводятся в форме практической подготовки в виде работ, связанных с будущей профессиональной деятельностью.

К практическим работам предъявляется ряд требований, основным из которых является полное, исчерпывающее описание всей проделанной работы, позволяющее судить о полученных результатах, степени выполнения заданий и профессиональной подготовке учащихся.

I. Практические работы:

МДК 02.01. Управление и автоматизация баз данных

Тема практической работы № 1. Установка СУБД MySQL и настройка службы на локальном сервере, объем часов: 2 часа.

У.1 Производить идентификацию проблем, связанных с нормальным функционированием базы данных

У.2 Принимать решения по локализации проблем

У.4 Осуществлять основные функции по администрированию баз данных

У.7 Производить регламентное обновление программного обеспечения

Цель практической работы: Изучить архитектуру инсталляции реляционной СУБД, выполнить развертывание серверной части MySQL и подготовить среду для интеграции с программными решениями на базе ИИ.

Задание(я):

1. Скачать и выполнить установку дистрибутива MySQL Community Server (версии 8.x).
2. Сконфигурировать системную службу MySQL и настроить её автоматический запуск.
3. Настроить учетную запись администратора (root) и проверить доступ к серверу через командную строку.
4. Выполнить проверку статуса сервера и сетевого порта.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** При выборе типа установки рекомендуется использовать «Server only» или «Full», если требуется графический интерфейс Workbench. Обратите внимание на выбор кодировки: для корректной работы с текстами в задачах ИИ (Natural Language Processing) обязательно выбирайте **UTF8mb4**.
- **К заданию 2:** В процессе конфигурации («Windows Service») задайте уникальное имя службы. Убедитесь, что галочка «Start the MySQL Server at System Startup» активна. Это критично для серверов, где работают скрипты сбора данных.
- **К заданию 3:** На этапе «Authentication Method» выберите рекомендуемый метод шифрования паролей. После установки добавьте путь к папке `C:\Program Files\MySQL\MySQL Server 8.0\bin` в системную переменную **PATH**, чтобы команда `mysql` была доступна в любом окне терминала.
- **К заданию 4:** Используйте команду `status;` внутри консоли MySQL. В отчете необходимо зафиксировать значение порта (по умолчанию 3306) и версию протокола. При возникновении ошибок доступа проверьте, не занят ли порт другими веб-серверами (например, XAMPP или Docker).

Требования к отчету 1:

Отчет служит доказательством корректной базовой инсталляции и готовности среды к работе. Обязательные элементы:

1. **Скриншот окна инсталлятора** на этапе выбора компонентов (Product Selection), подтверждающий установку *MySQL Server* и выбранных клиентских утилит.
2. **Скриншот этапа конфигурации «Authentication Method»**, подтверждающий выбор метода шифрования (Strong Password Encryption).

3. **Скриншот командной строки (cmd/PowerShell)**, демонстрирующий успешный вход в систему командой `mysql -u root -p` и вывод приветствия сервера.
4. **Скриншот окна «Переменные среды»**, где в переменной `Path` отображен добавленный путь к папке `bin` установленной СУБД.
5. **Результат выполнения команды `STATUS;`** в консоли MySQL (скриншот), на котором видны:
   - Версия сервера (Uptime);
   - Текущий пользователь (root@localhost);
   - Используемая кодировка (Character set).
6. **Вывод:** краткое подтверждение того, что СУБД успешно развернута, служба запущена и доступ к ней через терминал настроен.

Тема практической работы № 2. Установка PostgreSQL и настройка параметров конфигурации (порт, логирование), объем часов: 2 часа.
У.1 Производить идентификацию проблем, связанных с нормальным функционированием базы данных
У.2 Принимать решения по локализации проблем
У.4 Осуществлять основные функции по администрированию баз данных
У.7 Производить регламентное обновление программного обеспечения.

Цель практической работы: Освоить методику развертывания объектно-реляционной СУБД PostgreSQL, научиться вносить изменения в конфигурацию сервера через специализированные файлы и настраивать систему журналирования (логов).
Задание(я):
1. Произвести установку PostgreSQL (версия 15/16) с инициализацией кластера баз данных.
2. Изменить стандартный сетевой порт сервера с 5432 на альтернативный (например, 5433).
3. Настроить параметры журналирования: активировать запись всех SQL-запросов в лог-файл.
4. Проверить применение настроек путем подключения к серверу через утилиту `psql` или `pgAdmin`.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** При установке важно запомнить пароль суперпользователя `postgres` и указать локаль (Locale) «Russian, Russia», чтобы избежать проблем с кодировкой сообщений об ошибках. По завершении установки откажитесь от запуска *Stack Builder*, если дополнительные компоненты не требуются.
- **К заданию 2:** Для смены порта необходимо найти в каталоге данных (обычно `C:\Program Files\PostgreSQL\...\data`) файл **postgresql.conf**. Найдите строку `#port = 5432`, уберите символ комментария `#` и замените значение на 5433.
- **К заданию 3:** Настройка логирования критически важна для отладки приложений ИИ. В том же файле **postgresql.conf** установите параметры: `log_statement = 'all'` и `logging_collector = on`. Это позволит отслеживать, какие именно запросы отправляет модель ИИ к базе.
- **К заданию 4:** После внесения изменений в файлы конфигурации службу PostgreSQL необходимо **перезапустить** через оснастку «Службы» (services.msc), иначе настройки не вступят в силу. Для проверки порта используйте команду в консоли: `netstat -an | findstr 5433`.

Требования к отчету:

Отчет должен документально подтверждать изменение конфигурации сервера и активацию системы логирования. Обязательные элементы:

1. **Скриншот фрагмента файла `postgresql.conf`**, на котором видно:
   - Измененную строку порта: `port = 5433` (без символа комментария `#`).
   - Активированные параметры логирования: `log_statement = 'all'` и `logging_collector = on`.
2. **Скриншот оснастки «Службы» (services.msc)**, демонстрирующий статус «Выполняется» для службы PostgreSQL после внесения изменений.
3. **Скриншот консоли или терминала**, подтверждающий прослушивание нового порта. Используйте команду:
   - `netstat -ano | findstr :5433`
4. **Скриншот содержимого папки с логами** (обычно директория `log` внутри папки данных), где видно создание нового файла журнала после перезапуска службы.

5. **Листинг (текст) или скриншот** фрагмента лог-файла, в котором зафиксированы тестовые запросы (например, результат выполнения `SELECT now();`).
6. **Вывод:** краткое описание того, как изменение порта влияет на безопасность системы и зачем ИИ-интегратору может понадобиться детальное логирование запросов (мониторинг обращений моделей к данным).

Тема практической работы № 3. Установка Oracle Database и настройка окружения (переменные PATH, ORACLE_HOME), объем часов: 2 часа.

У.1 Производить идентификацию проблем, связанных с нормальным функционированием базы данных
У.2 Принимать решения по локализации проблем
У.4 Осуществлять основные функции по администрированию баз данных
У.7 Производить регламентное обновление программного обеспечения

Цель практической работы: Изучить особенности развертывания Oracle Database (версии Express Edition или Enterprise), освоить правила формирования переменных среды для корректной работы утилит администрирования.

Задание(я):
1. Произвести установку Oracle Database (рекомендуется версия XE 21c/18c).
2. Выполнить ручную настройку системных переменных окружения: `ORACLE_HOME`, `ORACLE_SID`, `PATH`.
3. Проверить запуск консольной утилиты `SQL*Plus` и выполнить подключение к базе под учетной записью `SYS`.
4. Получить информацию о версии установленного экземпляра.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):
- **К заданию 1:** При установке Oracle на ОС Windows имя компьютера должно быть написано латиницей. На этапе задания паролей для системных аккаунтов (SYS, SYSTEM) используйте пароль, соответствующий требованиям сложности, так как Oracle блокирует простые комбинации.

- **К заданию 2:** Это ключевой этап. В «Свойствах системы» -> «Переменные среды» создайте:
    - `ORACLE_HOME` — путь к корневой папке установленного ПО (например, `C:\app\product\21c\dbhomeXE`).
    - `ORACLE_SID` — идентификатор экземпляра (по умолчанию `XE`).
    - В переменную `PATH` добавьте путь к папке с исполняемыми файлами: `%ORACLE_HOME%\bin`.
- **К заданию 3:** Чтобы зайти под суперпользователем, используйте команду в командной строке: `sqlplus / as sysdba`. Если переменные настроены верно, пароль не потребуется (используется авторизация ОС).
- **К заданию 4:** После входа выполните запрос: `SELECT * FROM v$version;`. Полученный результат (список компонентов и версий) является подтверждением успешной настройки и должен быть включен в отчет.

Требования к отчету:

Отчет должен быть выполнен в электронном виде (формат .docx или .pdf) и содержать следующие обязательные элементы:

1. **Титульный лист:** оформленный по стандарту учебного заведения с указанием темы, ФИО студента и номера группы.
2. **Ход выполнения заданий:**
    - **Скриншот** окна завершения установки Oracle Database с системным сообщением об успешной конфигурации и указанием локальных ссылок (например, на Oracle Enterprise Manager).
    - **Скриншот** окна «Переменные среды» (Environment Variables), где четко видны созданные записи для `ORACLE_HOME` и `ORACLE_SID`.
    - **Скриншот** редактирования переменной `PATH`, подтверждающий добавление пути к директории `bin`.
3. **Результаты проверки:**
    - Текстовый лог или **скриншот** консоли `SQL*Plus`, демонстрирующий успешное выполнение команды `sqlplus / as sysdba`.

- Скриншот с таблицей, полученной в результате запроса `SELECT * FROM v$version;`.

4. **Вывод:** краткое резюме о проделанной работе, подтверждающее готовность сервера к приему внешних подключений.

Тема практической работы № 4. Установка MongoDB и настройка репликации для отказоустойчивости, объем часов: 2 часа.

У1: Производить идентификацию проблем, связанных с нормальным функционированием базы данных.
У2: Принимать решения по локализации проблем.
У4: Осуществлять основные функции по администрированию баз данных.
У7: Производить регламентное обновление программного обеспечения.

Цель практической работы: Изучить процесс развертывания нереляционной СУБД MongoDB и освоить базовые навыки настройки Replica Set (набора реплик) для обеспечения высокой доступности данных в системах искусственного интеллекта.

Задание(я):
1. Произвести установку MongoDB Community Server и графического клиента MongoDB Compass.
2. Подготовить конфигурационный файл для активации режима репликации.
3. Инициализировать набор реплик (Replica Set) через интерфейс командной строки `mongosh`.
4. Проверить статус репликации и работоспособность узла.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):
- **К заданию 1:** При установке на Windows снимите галочку «Install MongoDB as a Service», если планируете запускать несколько узлов на одном ПК вручную для тестов. Если установка идет как сервис, убедитесь, что путь к данным (`dbPath`) создан заранее.
- **К заданию 2:** Откройте файл конфигурации `mongod.cfg`. Найдите секцию `replication:`. Вам необходимо раскомментировать её и добавить имя набора реплик, например:

```
yaml
```

```
replication:
  replSetName: "rs0"
```
Используйте код с осторожностью.

- **К заданию 3:** Запустите оболочку MongoDB Shell (`mongosh`). Для активации репликации выполните команду `rs.initiate()`. Если вы настраиваете кластер из одного узла для практики, команда выполнится мгновенно, назначив ваш узел основным (PRIMARY).
- **К заданию 4:** Для проверки состояния используйте команду `rs.status()`. Обратите внимание на поле `"ok": 1` и статус узла `"stateStr": "PRIMARY"`. В задачах ИИ репликация позволяет считывать данные для обучения с второстепенных узлов (SECONDARY), снижая нагрузку на основной сервер.

Требования к отчету:
Отчет должен быть выполнен в электронном виде (формат .docx или .pdf) и содержать следующие обязательные элементы:

1. **Титульный лист:** оформленный по стандарту учебного заведения.
2. **Ход выполнения заданий:**
   - **Скриншот** содержимого файла `mongod.cfg` с настроенным параметром `replSetName`.
   - **Скриншот** окна консоли с процессом запуска службы MongoDB.
3. **Результаты проверки:**
   - **Скриншот** вывода команды `rs.initiate()`, подтверждающий успешную инициализацию.
   - **Скриншот** окна MongoDB Compass, где в строке статуса указано «Replica Set» и роль узла.
   - **Скриншот** (фрагмент) вывода `rs.status()`, где виден идентификатор набора и состояние узла.
4. **Вывод:** краткое резюме о том, как настроенная репликация помогает избежать потери данных при сбоях в работе ИИ-приложений.

Тема практической работы № 5. Установка Microsoft SQL Server и настройка параметров аутентификации, объем часов: 4 часа.

У1: Устанавливать и конфигурировать системы управления базами данных.
У4: Осуществлять основные функции по администрированию баз данных.

У5: Управлять учетными записями и правами доступа пользователей.
У6: Обеспечивать информационную безопасность на уровне СУБД.

Цель практической работы: Изучить архитектуру безопасности Microsoft SQL Server, выполнить установку экземпляра СУБД и настроить смешанный режим аутентификации для обеспечения доступа внешних программных продуктов.

Задание(я):

1. Произвести установку MS SQL Server (версия Express или Developer) и среды управления SQL Server Management Studio (SSMS).
2. Выбрать и настроить режим проверки подлинности (Authentication Mode).
3. Активировать системную учетную запись администратора sa и задать пароль.
4. Проверить возможность подключения к серверу под доменной и под системной учетной записью.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** При установке выберите «Новая установка изолированного экземпляра». На этапе выбора компонентов обязательно отметьте «Службы ядра СУБД». Помните, что SSMS с 2016 года устанавливается как отдельный пакет программ.
- **К заданию 2:** На этапе «Настройка ядра СУБД» (Database Engine Configuration) выберите **«Смешанный режим» (Mixed Mode)**. Это позволит подключаться к базе как через учетные записи Windows, так и через внутренние логины SQL Server, что необходимо для большинства интеграций с ИИ-платформами.
- **К заданию 3:** Если при установке был выбран только режим Windows, включить смешанный режим можно в свойствах сервера в SSMS (раздел «Безопасность»). После смены режима учетную запись sa (system administrator) необходимо перевести в состояние «Enabled» в узле «Безопасность -> Имена входа».
- **К заданию 4:** После изменения настроек аутентификации требуется **перезагрузка службы** SQL Server через «Диспетчер

конфигурации SQL Server». Попробуйте войти в систему, выбрав в поле Authentication пункт «SQL Server Authentication» и введя логин sa.

Требования к отчету:
Отчет должен быть выполнен в электронном виде (формат .docx или .pdf) и содержать следующие обязательные элементы:

1. **Титульный лист:** оформленный по стандарту учебного заведения.
2. **Ход выполнения заданий:**
    - **Скриншот** окна установки на этапе «Database Engine Configuration» с выбранным режимом «Mixed Mode».
    - **Скриншот** из среды SSMS, демонстрирующий статус учетной записи sa (свойства логина).
3. **Результаты проверки:**
    - **Скриншот** окна Object Explorer в SSMS, где видно успешное подключение пользователя sa.
    - **Скриншот** из «Диспетчера конфигурации» (SQL Server Configuration Manager), подтверждающий работу службы.
4. **Вывод:** краткое обоснование, почему смешанный режим аутентификации является предпочтительным для интеграции автономных решений с применением технологий ИИ.

Тема практической работы № 6. Установка и настройка клиента SQL Workbench для работы с базой данных MySQL, объем часов: 2 часа.

У.4: Осуществлять основные функции по администрированию баз данных.
У.10: Добавлять, удалять и изменять данные в базе данных.
У.11: Производить операции по импорту и экспорту данных в различных форматах.

Цель практической работы: Приобрести навыки работы с графическими интерфейсами администрирования СУБД, научиться настраивать удаленное подключение к серверу и освоить инструменты миграции данных (импорт/экспорт).

Задание(я):

1. Установить приложение MySQL Workbench и создать новое соединение с локальным сервером.
2. Изучить интерфейс администрирования (раздел Server Status и Client Connections).
3. Создать новую базу данных (схему) и наполнить её тестовыми данными через SQL-редактор.
4. Выполнить экспорт созданной базы данных в SQL-дамп и осуществить обратный импорт данных из файла.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** При создании подключения (Setup New Connection) укажите Hostname: 127.0.0.1 и Port: 3306. Используйте кнопку **«Test Connection»** для проверки связи с сервером, установленным в ПР №1. Если соединение не устанавливается, проверьте, запущена ли служба MySQL в системе.
- **К заданию 2:** В левой панели «Administration» обратите внимание на вкладку **Users and Privileges**. Здесь администратор может визуально управлять правами, которые мы разбирали в теории. Посмотрите также «Dashboard» для мониторинга нагрузки на сервер.
- **К заданию 3:** Используйте визуальный конструктор (кнопка «Create a new schema») для создания БД. Для изменения данных (У.10) примените встроенный редактор таблиц (Grid View), который позволяет вносить правки в ячейки аналогично Excel, с последующим нажатием кнопки **«Apply»**.
- **К заданию 4:** Для выполнения У.11 используйте инструменты **Data Export** и **Data Import/Restore**. Экспорт рекомендуется делать в формате «Self-Contained File». Это стандартный способ резервного копирования данных перед обновлением моделей ИИ или переносом датасетов на другой сервер.

Требования к отчету:
Отчет должен быть выполнен в электронном виде и содержать:

1. **Титульный лист** установленного образца.
2. **Ход выполнения заданий:**

- **Скриншот** окна «Connect to Database» с успешно пройденным тестом соединения (сообщение «Successfully made the MySQL connection»).
- **Скриншот** созданной схемы в дереве объектов (Schemas).

3. **Результаты выполнения операций с данными:**
   - **Скриншот** таблицы с добавленными записями.
   - **Скриншот** окна «Data Export» в процессе генерации SQL-файла.
   - **Скриншот** консоли лога импорта, подтверждающий успешное восстановление базы (сообщение «Import completed»).
4. **Вывод:** описание преимуществ использования GUI-клиентов перед командной строкой при выполнении массового импорта данных для обучения ИИ.

Тема практической работы № 7. Установка и настройка pgAdmin для работы с базой данных PostgreSQL, объем часов: 2 часа.

У.4: Осуществлять основные функции по администрированию баз данных.
У.10: Добавлять, удалять и изменять данные в базе данных.
У.11: Производить операции по импорту и экспорту данных в различных форматах.

Цель практической работы: Освоить инструментарий графической среды pgAdmin для эффективного администрирования серверов PostgreSQL, управления объектами БД и выполнения операций по миграции данных.

Задание(я):

1. Произвести установку актуальной версии pgAdmin 4.
2. Настроить подключение к локальному серверу PostgreSQL (созданному в ПР №2).
3. Создать новую базу данных и таблицу с использованием графического конструктора.
4. Выполнить экспорт таблицы в формат CSV (подготовка датасета) и импорт данных из внешнего файла.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** При первом запуске pgAdmin предложит установить «Master Password». Это пароль для защиты ваших сохраненных учетных данных к серверам. Обязательно зафиксируйте его.
- **К заданию 2:** В окне «Register - Server» на вкладке «Connection» укажите: Host — localhost, Port — 5433 (или тот, что был настроен в ПР №2), Maintenance database — postgres, Username — postgres.
- **К заданию 3:** При создании таблицы через контекстное меню (Create -> Table) обратите внимание на вкладку **Columns**. Задайте типы данных. Для задач ИИ часто используются типы JSONB (для неструктурированных метаданных) или REAL[] (для векторов), попробуйте добавить такие поля.
- **К заданию 4:** PostgreSQL отлично работает с форматом CSV. Для экспорта используйте инструмент **Import/Export Data** в контекстном меню таблицы. Убедитесь, что кодировка (Encoding) установлена как UTF-8, а разделитель (Delimiter) соответствует формату файла. Это базовый навык для подготовки данных перед подачей в Python-скрипты (библиотека Pandas).

Требования к отчету:
Отчет должен быть выполнен в электронном виде и содержать:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
    - **Скриншот** главного окна pgAdmin со списком подключенных серверов (вкладка Browser).
    - **Скриншот** вкладки «Dashboard» для подключенного сервера, демонстрирующий график активности (Transactions per second).
3. **Результаты работы с данными:**
    - **Скриншот** конструктора таблицы с перечнем созданных столбцов.
    - **Скриншот** окна настройки экспорта (параметры Format и Filename).

- **Скриншот** (или фрагмент текста) полученного CSV-файла с данными.

4. **Вывод:** сравнительная оценка удобства использования pgAdmin по сравнению с консольной утилитой psql при работе с импортом больших объемов данных.

Тема практической работы № 8. Установка и настройка Microsoft Management Studio (SSMS) для работы с SQL Server, объем часов: 2 часа.

У.4: Осуществлять основные функции по администрированию баз данных.
У.10: Добавлять, удалять и изменять данные в базе данных.
У.11: Производить операции по импорту и экспорту данных в различных форматах.

Цель практической работы: Изучить интерфейс и возможности среды SSMS, настроить клиентское подключение к экземпляру SQL Server и освоить инструменты визуального редактирования данных и генерации скриптов.

Задание(я):

1. Произвести установку актуальной версии SQL Server Management Studio.
2. Выполнить подключение к локальному экземпляру сервера, используя различные типы аутентификации (Windows и SQL Server).
3. С помощью графического дизайнера создать базу данных и таблицу, настроив первичные ключи и типы данных.
4. Выполнить экспорт данных таблицы в плоский файл (Flat File) и создать SQL-скрипт на создание всей структуры БД.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Обратите внимание, что SSMS является независимым приложением и обновляется чаще, чем сам движок базы данных. При установке закройте все запущенные компоненты SQL Server для корректной регистрации библиотек.
- **К заданию 2:** В окне «Connect to Server» проверьте имя сервера. Если вы использовали экземпляр по умолчанию, введите (local) или имя ПК.

Для входа под системным администратором выберите «SQL Server Authentication» и используйте логин sa (настроенный в ПР №5).

- **К заданию 3:** В SSMS создание таблиц через контекстное меню позволяет сразу настраивать свойства столбцов (Allow Nulls, Identity Specification). Для задач ИИ-интеграции важно правильно выбирать типы: например, nvarchar(max) для больших текстов или float для весовых коэффициентов.
- **К заданию 4:** Для генерации скрипта нажмите правой кнопкой на БД -> Tasks -> **Generate Scripts**. Это позволяет сохранить всю структуру базы в один .sql файл для быстрой репликации окружения. Для экспорта данных используйте «Export Data Wizard», выбрав в качестве назначения «Flat File Destination».

Требования к отчету:
Отчет должен быть выполнен в электронном виде и содержать:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
    - **Скриншот** окна «Connect to Server» с заполненными параметрами подключения.
    - **Скриншот** панели «Object Explorer», где виден развернутый список папок сервера.
3. **Результаты работы с данными:**
    - **Скриншот** окна дизайна таблицы (Table Designer) с заданными полями.
    - **Скриншот** процесса экспорта данных через «SQL Server Import and Export Wizard».
    - **Скриншот** сгенерированного SQL-скрипта (фрагмент кода CREATE TABLE...).
4. **Вывод:** описание роли SSMS в задачах локализации проблем и преимуществ использования SQL-скриптов для регламентного обновления ПО.

Тема практической работы № 9. Установка и настройка DBeaver для подключения к различным типам баз данных, объем часов: 2 часа.

У.4: Осуществлять основные функции по администрированию баз данных.
У.10: Добавлять, удалять и изменять данные в базе данных.
У.11: Производить операции по импорту и экспорту данных в различных форматах.

Цель практической работы: Изучить принципы работы универсальных менеджеров баз данных, настроить кросс-платформенные подключения к различным СУБД и освоить визуализацию структур данных.

Задание(я):

1. Установить DBeaver Community Edition и выполнить первичную настройку интерфейса.
2. Создать подключения к двум разным СУБД (например, MySQL и PostgreSQL), установленным в предыдущих работах.
3. Изучить инструмент визуализации связей (ER-диаграммы) для существующих баз данных.
4. Выполнить консолидированную выгрузку данных из одной СУБД для последующего использования в другой.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** При установке DBeaver может потребовать установку Java (JRE). Рекомендуется использовать версию, включающую JRE в комплект. При первом подключении к любой СУБД программа предложит **скачать драйверы** — подтвердите загрузку, это необходимо для корректного протокола связи.
- **К заданию 2:** Используйте «Мастер создания соединений». Для MySQL и PostgreSQL укажите соответствующие порты (3306 и 5433), настроенные ранее. Обратите внимание, что DBeaver позволяет группировать подключения по папкам (например, «Production», «Test»), что удобно при администрировании множества ИИ-серверов.
- **К заданию 3:** Одной из мощнейших функций DBeaver является вкладка **ER Diagram**. Выберите схему базы данных и перейдите на эту вкладку — программа автоматически построит связи между таблицами.

Это незаменимо для понимания структуры данных перед написанием скриптов обработки.

- **К заданию 4:** Используйте функцию «Export Data». DBeaver позволяет экспортировать данные напрямую в формат **JSON** или **Markdown**, что часто требуется для вставки примеров данных в документацию ИИ-проектов или передачи в веб-сервисы.

Требования к отчету:
Отчет должен быть выполнен в электронном виде и содержать:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
    - **Скриншот** окна «Database Navigator», где видны одновременно два активных подключения (MySQL и PostgreSQL).
    - **Скриншот** процесса загрузки драйверов (Driver Manager).
3. **Результаты работы:**
    - **Скриншот** автоматически сгенерированной ER-диаграммы одной из баз данных.
    - **Скриншот** окна настроек экспорта в формат JSON.
    - **Скриншот** панели «SQL Editor» с выполненным простым запросом к любой из баз.
4. **Вывод:** анализ преимуществ использования универсального клиента (DBeaver) по сравнению со специализированными (SSMS, pgAdmin) в условиях работы с гетерогенными (разнородными) данными.

Тема практической работы № 10. Установка и настройка библиотек Python для взаимодействия с базами данных (pymysql, psycopg2), объем часов: 4 часа.

У.4: Осуществлять основные функции по администрированию баз данных.
У.10: Добавлять, удалять и изменять данные в базе данных.
У.11: Производить операции по импорту и экспорту данных в различных форматах.

Цель практической работы: Научиться настраивать программное окружение Python для работы с реляционными СУБД, устанавливать драйверы соединений и выполнять базовые операции администрирования данных через программный код.

Задание(я):

1. Настроить виртуальное окружение Python и установить библиотеки `pymysql` (для MySQL) и `psycopg2` (для PostgreSQL).
2. Написать скрипт для проверки соединения с обеими СУБД.
3. Реализовать через Python автоматизированное создание таблицы и добавление в неё тестовой записи.
4. Выполнить экспорт данных из таблицы БД в структуру данных Python (список или словарь).

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Используйте менеджер пакетов `pip`. Для установки введите в терминале: `pip install pymysql psycopg2-binary`. Использование версии `-binary` для PostgreSQL рекомендуется для учебных сред, так как она содержит скомпилированные зависимости.
- **К заданию 2:** При создании подключения (Connection object) используйте параметры `host`, `user`, `password`, `database` и `port`, настроенные в работах №1 и №2. Помните о закрытии соединения методом `.close()` после завершения работы — это критично для предотвращения утечек памяти на сервере.
- **К заданию 3:** Все SQL-команды передаются через объект «курсор» (`cursor`). Для фиксации изменений в базе (при операциях `INSERT`, `CREATE`) обязательно вызывайте метод `connection.commit()`, иначе изменения останутся во временном буфере и исчезнут.
- **К заданию 4:** Для получения данных используйте метод `cursor.fetchall()`. В задачах ИИ полученные данные затем преобразуются в массивы `numpy` или датафреймы `pandas`, поэтому важно убедиться, что типы данных из БД (например, `INT` или `FLOAT`) корректно распознаются Python.

Требования к отчету:
Отчет должен быть выполнен в электронном виде и содержать:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
   - **Скриншот** терминала с подтверждением успешной установки библиотек (команда `pip list`).
   - **Листинг кода** скрипта подключения (чувствительные пароли можно заменить на `***`).
3. **Результаты работы:**
   - **Скриншот** консоли Python с сообщением «Connection successful» или выводом версии сервера БД.
   - **Скриншот** из графического клиента (DBeaver или MySQL Workbench), подтверждающий, что таблица была создана программно.
   - **Скриншот** вывода данных из БД в консоль Python (результат выполнения `SELECT`).
4. **Вывод:** обоснование преимуществ использования Python для массовой загрузки данных в базу по сравнению с ручным вводом через GUI.

Тема практической работы № 11. Создание пользователей и групп в MySQL и назначение прав доступа (GRANT, REVOKE), объем часов: 2 часа.

У.5: Настраивать политики безопасности при работе с сервером баз данных.
У.6: Давать независимую оценку уровня безопасности.
У.8: Разрабатывать рекомендации по дальнейшей эксплуатации БД с максимальной защитой.

Цель практической работы: Освоить механизм управления доступом на основе ролей (RBAC), научиться разграничивать полномочия пользователей и проводить аудит безопасности привилегий в среде MySQL.
Задание(я):

5. Создать структуру пользователей: «Администратор безопасности», «Аналитик данных» и «Сервисный аккаунт ИИ».
6. Сформировать политики доступа: выдать полные права администратору и ограничить права остальных пользователей только необходимыми операциями.

7. Отработать сценарий отзыва привилегий (REVOKE) при подозрении на компрометацию учетной записи.
8. Провести аудит безопасности: сформировать отчет о текущих правах всех пользователей в системе.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Используйте команду `CREATE USER 'username'@'localhost' IDENTIFIED BY 'password';`. Обратите внимание на политику сложности паролей. Для «Сервисного аккаунта ИИ», который будет подключаться программно, рекомендуется ограничить хост подключения конкретным IP-адресом.
- **К заданию 2:** Используйте команду `GRANT`. Для аналитика данных достаточно прав `SELECT` на конкретные таблицы. Для сервисного аккаунта ИИ может потребоваться `SELECT` и `INSERT` (для сохранения результатов работы модели). Избегайте использования `GRANT ALL PRIVILEGES` для обычных пользователей.
- **К заданию 3:** Протестируйте команду `REVOKE`. Например, отзовите право на удаление данных: `REVOKE DELETE ON db_name.* FROM 'analyst'@'localhost';`. После этого попробуйте выполнить `DELETE` под этим пользователем и зафиксируйте ошибку `Access denied`.
- **К заданию 4:** Для оценки уровня безопасности используйте системную таблицу `mysql.user` или команду `SHOW GRANTS FOR 'username'@'localhost';`. Проанализируйте, нет ли у рядовых пользователей избыточных прав (например, доступа к системной базе `mysql` или прав `SHUTDOWN`).

Требования к отчету:

Отчет должен быть выполнен в электронном виде и содержать:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
   - **Листинг команд** создания пользователей и назначения им прав.
   - **Скриншот** ошибки доступа при попытке выполнить запрещенную операцию (после REVOKE).

3. **Результаты аудита:**
   - **Таблица (Матрица доступа)**, где указано: Имя пользователя — Роль — Набор разрешенных команд (SELECT, UPDATE и т.д.).
   - **Скриншот** вывода команды `SHOW GRANTS` для самого ограниченного пользователя.
4. **Вывод:** сформулируйте 3-4 правила безопасной эксплуатации БД (например: использование принципа наименьших привилегий, регулярная смена паролей сервисных аккаунтов, запрет на работу под `root` в приложениях).

Тема практической работы № 12. Настройка ролей и прав доступа в PostgreSQL для различных групп пользователей, объем часов: 2 часа.

У.5: Настраивать политики безопасности при работе с сервером баз данных.
У.6: Давать независимую оценку уровня безопасности.
У.8: Разрабатывать рекомендации по дальнейшей эксплуатации БД с максимальной защитой.

Цель практической работы: Изучить ролевую модель доступа PostgreSQL, научиться создавать групповые роли для коллективной работы и настраивать гранулярный доступ к объектам базы данных.
Задание(я):

1. Создать групповые роли `ai_developers` (разработчики) и `ai_readonly` (аудиторы).
2. Создать индивидуальные учетные записи пользователей и включить их в соответствующие группы.
3. Настроить права доступа: группе разработчиков разрешить управление схемами, а аудиторам — только чтение данных.
4. Выполнить проверку иерархии прав и протестировать наследование полномочий.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** В PostgreSQL роль создается командой `CREATE ROLE`. Для групповой роли используйте атрибут `NOLOGIN`, чтобы под ней нельзя было войти в систему напрямую: `CREATE ROLE ai_developers NOLOGIN;`.

- **К заданию 2:** При создании персональных ролей добавьте атрибут `LOGIN` и пароль: `CREATE ROLE ivan LOGIN PASSWORD 'secure_pass';`. Для включения пользователя в группу используйте `GRANT ai_developers TO ivan;`.
- **К заданию 3:** Права в PostgreSQL назначаются последовательно. Сначала разрешите использование схемы: `GRANT USAGE ON SCHEMA public TO ai_readonly;`, затем — доступ к таблицам: `GRANT SELECT ON ALL TABLES IN SCHEMA public TO ai_readonly;`. Для группы разработчиков используйте `GRANT ALL PRIVILEGES`.
- **К заданию 4:** Используйте команду `SET ROLE ivan;` для переключения контекста безопасности внутри сессии. Это позволит проверить, может ли пользователь выполнять операции, разрешенные его группе. Для аудита используйте системные команды `\du` (список ролей) и `\z` (права доступа к таблицам) в консоли `psql`.

Требования к отчету:

Отчет должен быть выполнен в электронном виде и содержать:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
   - **Скриншот или листинг кода** создания ролей и процесса их связывания (GRANT ROLE).
   - **Скриншот** настройки прав доступа через интерфейс pgAdmin (вкладка "Privileges" в свойствах таблицы).
3. **Результаты проверки:**
   - **Скриншот консоли** с результатом команды `\du`, подтверждающий иерархию ролей (столбец "Member of").
   - **Текстовое подтверждение** успешного/неуспешного выполнения запроса `INSERT` от лица пользователя из группы `ai_readonly`.
4. **Вывод:** подготовьте краткую рекомендацию по использованию схемы **`search_path`** и ограничению прав на схему `public` для повышения безопасности корпоративных данных ИИ.

Тема практической работы № 13. Управление правами доступа в Microsoft SQL Server с использованием SQL Server Management Studio (SSMS), объем часов: 2 часа.

У.5: Настраивать политики безопасности при работе с сервером баз данных.
У.6: Давать независимую оценку уровня безопасности.
У.8: Разрабатывать рекомендации по дальнейшей эксплуатации БД с максимальной защитой.

Цель практической работы: Изучить модель безопасности MS SQL Server, научиться создавать серверные имена входа, сопоставлять их с пользователями баз данных и управлять полномочиями через стандартные роли.

Задание(я):

1. Создать новое «Имя входа» (Login) с использованием аутентификации SQL Server.
2. Выполнить сопоставление (Mapping) имени входа с конкретной базой данных и создать соответствующего пользователя.
3. Включить пользователя в предопределенные роли базы данных (db_datareader, db_datawriter).
4. Настроить индивидуальные разрешения на выполнение хранимых процедур (гранулярный доступ).

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** В обозревателе объектов (Object Explorer) перейдите в раздел **Security -> Logins**. При создании логина снимите галочку «Enforce password expiration», если это тестовая среда, но в реальных ИИ-проектах это обязательное требование безопасности.
- **К заданию 2:** Перейдите на страницу **User Mapping**. Выберите базу данных и укажите имя пользователя (по умолчанию совпадает с логином). Помните: логин дает право подключиться к *серверу*, а пользователь — зайти в *базу*.
- **К заданию 3:** На этой же странице выберите роли. Роль db_datareader позволяет только читать данные (SELECT), а db_owner дает полные права. Для интеграции ИИ-моделей чаще всего достаточно комбинации reader + writer.

- **К заданию 4:** Для реализации более тонкой настройки щелкните правой кнопкой на объекте (например, таблице) -> **Properties -> Permissions**. Нажмите «Search», добавьте пользователя и отметьте галочкой право Execute или Alter. Это предотвращает изменение структуры базы случайным скриптом.

Требования к отчету:
Отчет должен содержать:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
    - **Скриншот** окна создания «Login» с выбранным методом аутентификации.
    - **Скриншот** окна «User Mapping», где видно, к каким базам привязан пользователь.
3. **Результаты настройки безопасности:**
    - **Скриншот** списка «Database Roles» в SSMS, подтверждающий членство пользователя в группах.
    - **Скриншот** успешного подключения к серверу под созданной учетной записью.
4. **Вывод:** Объясните разницу между **Fixed Server Roles** и **Database Roles**. Сформулируйте рекомендацию, почему не следует использовать роль sysadmin для работы ИИ-приложений.

Тема практической работы № 14. Настройка аутентификации и шифрования соединения в MySQL, объем часов: 2 часа.

У.5: Настраивать политики безопасности при работе с сервером баз данных.
У.6: Давать независимую оценку уровня безопасности.
У.8: Разрабатывать рекомендации по дальнейшей эксплуатации БД с максимальной защитой.

Цель практической работы: Изучить механизмы защиты данных при передаче по сети, научиться настраивать обязательное SSL/TLS шифрование для учетных записей и управлять плагинами аутентификации.
Задание(я):

1. Проверить текущий статус поддержки SSL на стороне сервера MySQL.
2. Создать пользователя с требованием обязательного безопасного соединения (SSL).
3. Настроить использование плагина аутентификации `caching_sha2_password`.
4. Проверить параметры шифрования текущего сеанса через командную строку.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Выполните в консоли запрос: `SHOW VARIABLES LIKE '%ssl%';`. Убедитесь, что значение `have_ssl` установлено в `YES`. В современных версиях MySQL (8.0+) сертификаты генерируются автоматически при установке.
- **К заданию 2:** Создайте «защищенного» пользователя командой: `CREATE USER 'ai_secure'@'%' IDENTIFIED BY 'Pass123!' REQUIRE SSL;`

  Теперь этот пользователь не сможет подключиться к базе по открытому (нешифрованному) каналу, что предотвращает перехват данных (Sniffing).
- **К заданию 3:** Проверьте тип используемого плагина в таблице `mysql.user`. Плагин `caching_sha2_password` обеспечивает более быстрое и безопасное хеширование паролей по сравнению с устаревшим `mysql_native_password`. Это критично для защиты от брутфорс-атак на серверы ИИ.
- **К заданию 4:** Подключитесь под созданным пользователем и выполните команду `\s` (status). Найдите строку **SSL:**. Если там указан алгоритм шифрования (например, `Cipher in use is TLS_AES_256_GCM_SHA384`), значит соединение защищено.

Требования к отчету:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
   - **Скриншот** вывода переменных SSL, подтверждающий поддержку шифрования сервером.

- **Листинг SQL-кода** создания пользователя с директивой `REQUIRE SSL`.

3. **Результаты проверки:**
   - **Скриншот** вывода команды `status` (`\s`) в консоли, где виден активный SSL-шифр.
   - **Скриншот** ошибки входа, если попробовать подключиться этим пользователем через клиент, в котором шифрование принудительно отключено.
4. **Вывод:** Объясните, в каких случаях необходимо использовать сертификаты CA (Certificate Authority) для проверки подлинности сервера со стороны клиента ИИ.

Тема практической работы № 15. Использование встроенных ролей в Oracle Database для управления доступом, объем часов: 4 часа.

У.5: Настраивать политики безопасности при работе с сервером баз данных.
У.6: Давать независимую оценку уровня безопасности.
У.8: Разрабатывать рекомендации по дальнейшей эксплуатации БД с максимальной защитой.

Цель практической работы: Изучить иерархию прав в Oracle, научиться использовать встроенные роли для быстрого назначения полномочий и разграничивать доступ между администраторами и пользователями данных.
Задание(я):

1. Создать новую учетную запись пользователя в подключаемой базе данных (PDB).
2. Назначить пользователю базовые роли `CONNECT` и `RESOURCE`, а также квоту на табличное пространство.
3. Исследовать состав встроенных ролей через системные представления.
4. Проверить ограничение прав: попытка выполнения административных действий под учетной записью обычного пользователя.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Используйте SQL*Plus или SQL Developer. Команда создания: `CREATE USER ai_user IDENTIFIED BY password;`. Помните, что в Oracle 12c и выше при работе в контейнерной

архитектуре (CDB/PDB) к именам локальных пользователей часто не требуется префикс `C##`.

- **К заданию 2:** Назначьте права: `GRANT CONNECT, RESOURCE TO ai_user;`. Роль **RESOURCE** позволяет создавать таблицы и индексы. **Важно:** обязательно выдайте квоту на запись данных, иначе создание таблиц будет невозможным: `ALTER USER ai_user QUOTA UNLIMITED ON USERS;`.
- **К заданию 3:** Чтобы понять, какие именно привилегии входят в роль, выполните запрос к системному словарю: `SELECT privilege FROM dba_sys_privs WHERE grantee = 'RESOURCE';`. Это поможет оценить, не избыточны ли права для конкретной задачи ИИ.
- **К заданию 4:** Подключитесь как `ai_user` и попробуйте создать нового пользователя или изменить системные параметры. Зафиксируйте ошибку `ORA-01031: insufficient privileges`.

Требования к отчету:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
   - **Скриншот** выполнения команд создания пользователя и назначения ролей в SQL*Plus.
   - **Скриншот** окна свойств пользователя в SQL Developer (вкладка "Roles").
3. **Результаты проверки:**
   - **Таблица** со списком системных привилегий, входящих в роль `RESOURCE`.
   - **Скриншот** с текстом ошибки при попытке выполнения запрещенной команды.
4. **Вывод:** Сформулируйте рекомендацию о переходе от стандартных ролей к созданию **пользовательских ролей** (Custom Roles) с минимально необходимым набором прав для сервисов ИИ.

Тема практической работы № 16. Создание резервной копии базы данных MySQL с использованием утилиты mysqldump, объем часов: 2 часа.

У.4: Осуществлять основные функции по администрированию баз данных.
У.3: Документировать внештатные ситуации.
У.7: Производить регламентное обновление программного обеспечения.

Цель практической работы: Освоить методику логического резервного копирования данных, научиться использовать консольные утилиты для создания дампов и подготовить регламент восстановления базы данных после сбоя.

Задание(я):

1. Создать полную резервную копию выбранной базы данных в формате .sql файла.
2. Выполнить экспорт только структуры таблиц (без данных) для целей разработки.
3. Имитировать «внештатную ситуацию» (удаление базы данных) и задокументировать её.
4. Произвести восстановление данных из ранее созданного дампа и проверить целостность информации.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Используйте системную утилиту в командной строке:
  mysqldump -u root -p db_name > backup_full.sql.
  Обратите внимание: если в пароле есть спецсимволы, его лучше вводить по запросу, а не прописывать в команде.
- **К заданию 2:** Для обновления ПО или переноса структуры (У.7) часто нужен дамп без самих данных. Используйте флаг --no-data:
  mysqldump -u root -p --no-data db_name > structure_only.sql.
- **К заданию 3:** В рамках отработки У.3 создайте краткий «Протокол инцидента», где укажите время «сбоя» (удаления БД командой DROP DATABASE), причину и принятое решение о восстановлении.
- **К заданию 4:** Для восстановления сначала создайте пустую базу с тем же именем, затем выполните:
  mysql -u root -p db_name < backup_full.sql.

После завершения проверьте количество записей в таблицах, чтобы убедиться в отсутствии потерь.

Требования к отчету:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
    - **Скриншот** окна консоли с командами экспорта.
    - **Скриншот** созданных файлов в проводнике с указанием их размера.
3. **Документирование:**
    - **Фрагмент «Журнала внештатных ситуаций»**: описание удаления данных и фиксация ошибки подключения приложения к пустой базе.
4. **Результаты восстановления:**
    - **Скриншот** команды импорта.
    - **Скриншот** результата SQL-запроса SELECT COUNT(*), подтверждающий восстановление данных.
5. **Вывод:** Опишите, почему регулярный запуск mysqldump по расписанию является обязательным регламентным действием при эксплуатации ИИ-систем.

Тема практической работы № 17. Резервное копирование базы данных PostgreSQL с помощью pg_dump и pg_dumpall, объем часов: 2 часа.

У.4: Осуществлять основные функции по администрированию баз данных.
У.3: Документировать внештатные ситуации.
У.7: Производить регламентное обновление программного обеспечения.

Цель практической работы: Изучить методы логического архивирования данных в PostgreSQL, научиться экспортировать глобальные объекты кластера (роли, права) и отработать технологию восстановления из архивных копий.
Задание(я):

1. Создать резервную копию конкретной базы данных в текстовом формате SQL с помощью `pg_dump`.
2. Создать полную резервную копию всего серверного кластера (включая пользователей и роли) через `pg_dumpall`.
3. Выполнить экспорт базы данных в сжатом кастомном формате (`-Fc`) для последующего восстановления через `pg_restore`.
4. Отработать сценарий восстановления данных и составить отчет о выявленной "внештатной ситуации".

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Используйте команду: `pg_dump -U postgres -d db_name -f backup.sql`. Обратите внимание, что `pg_dump` делает копию только одной конкретной базы. Это удобно для регламентного обновления отдельных модулей ИИ-системы.
- **К заданию 2:** Для сохранения глобальных настроек (ролей, созданных в ПР №12) используйте `pg_dumpall -U postgres > full_cluster.sql`. Это критично для переноса всей среды администрирования на новый сервер.
- **К заданию 3:** Кастомный формат (`-Fc`) позволяет восстанавливать объекты выборочно и работает быстрее на больших объемах данных. Команда: `pg_dump -U postgres -F c -b -v -f db_backup.dump db_name`.
- **К заданию 4:** Имитируйте потерю данных (удаление таблицы). В случае использования текстового SQL-файла восстановление идет через `psql`. В случае кастомного формата используйте: `pg_restore -U postgres -d db_name db_backup.dump`. Для фиксации внештатной ситуации опишите в отчете симптомы "порчи данных" до восстановления.

Требования к отчету:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
   - **Скриншоты** командной строки с процессом выполнения `pg_dump` и `pg_dumpall`.
   - **Скриншот** папки с файлами бэкапов, демонстрирующий разницу в размерах между текстовым и сжатым форматом.

3. **Документирование:**
   - **Описание инцидента:** "Случайное удаление схемы данных разработчиком". Укажите время обнаружения и метод восстановления.
4. **Результаты проверки:**
   - **Скриншот** лога восстановления (вывод `pg_restore`).
   - **Скриншот** из pgAdmin, подтверждающий наличие восстановленных объектов и данных.
5. **Вывод:** Объясните преимущество использования `pg_dumpall` перед обновлением версии СУБД.

Тема практической работы № 18. Настройка и выполнение резервного копирования в Microsoft SQL Server с использованием SSMS, объем часов: 2 часа.

У.4: Осуществлять основные функции по администрированию баз данных.
У.3: Документировать внештатные ситуации.
У.7: Производить регламентное обновление программного обеспечения.

Цель практической работы: Изучить стратегии резервного копирования в среде MS SQL Server, научиться создавать полные бэкапы через графический интерфейс и восстанавливать базу данных с контролем состояния файлов.

Задание(я):

1. Выполнить полное резервное копирование (Full Backup) базы данных в файл формата .bak.
2. Настроить параметры проверки целостности данных перед созданием копии.
3. Провести симуляцию аварии (удаление базы) и задокументировать инцидент.
4. Осуществить восстановление базы данных из файла и проверить доступность данных.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** В SSMS щелкните правой кнопкой на БД -> **Tasks -> Back Up**. Убедитесь, что выбран тип «Full». В поле «Destination» удалите

старые пути и добавьте актуальный путь к папке (например, C:\Backups\db.bak). Это базовое действие перед любым регламентным обновлением ПО.

- **К заданию 2:** На вкладке «Media Options» установите галочку **«Verify backup when finished»**. Это критически важно: поврежденный бэкап бесполезен при реальной аварии. Для ИИ-проектов с большими объемами данных также рекомендуется включать «Set backup compression».
- **К заданию 3:** Зафиксируйте в отчете сценарий: «Потеря данных вследствие ошибки дисковой подсистемы». Удалите базу данных (предварительно убедившись, что бэкап создан).
- **К заданию 4:** Щелкните правой кнопкой на папку «Databases» -> **Restore Database**. Выберите «Device» и укажите ваш .bak файл. Если вы восстанавливаете базу поверх существующей, на вкладке «Options» выберите «Overwrite the existing database (WITH REPLACE)».

Требования к отчету:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
    - **Скриншот** окна «Back Up Database» с настроенным путем и типом бэкапа.
    - **Скриншот** окна «Restore Database», где виден план восстановления (Restore Plan).
3. **Документирование:**
    - **Запись в журнале**: «Инцидент №18-А. Удаление БД. Источник восстановления: локальный диск, файл db.bak».
4. **Результаты проверки:**
    - **Скриншот** уведомления системы: «Database restored successfully».
    - **Скриншот** списка таблиц восстановленной базы в Object Explorer.
5. **Вывод:** Обоснуйте выбор между полным и разностным резервным копированием для систем, работающих в режиме реального времени.

Тема практической работы № 19. Автоматизация резервного копирования базы данных MongoDB с использованием скриптов, объем часов: 2 часа.

У.4: Осуществлять основные функции по администрированию баз данных.
У.3: Документировать внештатные ситуации.
У.7: Производить регламентное обновление программного обеспечения.

Цель практической работы: Изучить методы создания бинарных дампов в MongoDB, научиться автоматизировать процесс архивации данных с помощью командных файлов (скриптов) и настроить систему именования копий по датам.

Задание(я):

1. Выполнить ручное резервное копирование базы данных с помощью утилиты `mongodump`.
2. Разработать командный файл (`.bat` для Windows или `.sh` для Linux) для автоматического запуска бэкапа.
3. Настроить в скрипте динамическое создание имени папки, содержащей дату и время бэкапа.
4. Проверить корректность восстановления данных из созданного автоматизированного дампа.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Используйте команду: `mongodump --db=имя_базы --out=C:\backup_folder`. Если в MongoDB настроена авторизация, добавьте параметры `--username` и `--password`. Утилита создает бинарные файлы BSON, которые восстанавливаются быстрее, чем текстовые SQL-дампы.
- **К заданию 2:** Создайте текстовый файл `backup_script.bat`. Пропишите в нем путь к исполняемому файлу `mongodump.exe`. Это первый шаг к регламентному обслуживанию системы.
- **К заданию 3:** Чтобы бэкапы не перезаписывали друг друга, используйте переменные окружения для формирования имени. Пример для Windows:
  `set backup_dir=backup_%date%_%time:~0,2%-%time:~3,2%`

`mongodump --out=C:\backups\%backup_dir%`

Это позволяет хранить историю версий обучающих датасетов.

- **К заданию 4:** Имитируйте потерю коллекции (команда `db.collection.drop()`). Задокументируйте «внештатную ситуацию». Выполните восстановление: `mongorestore --db=имя_базы C:\backups\путь_к_дампу\имя_базы`.

Требования к отчету:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
   - **Листинг (текст)** созданного скрипта автоматизации.
   - **Скриншот** папки с результатами работы скрипта, где видно несколько папок с разными временными метками.
3. **Документирование:**
   - **Запись в журнале**: «Сбой: случайное удаление коллекции документов ИИ. Действие: восстановление из последнего автоматического дампа».
4. **Результаты проверки:**
   - **Скриншот** консоли с выводом процесса `mongorestore`.
   - **Скриншот** интерфейса MongoDB Compass, подтверждающий возврат данных.
5. **Вывод:** Объясните, как автоматизация бэкапов помогает при регламентном обновлении моделей машинного обучения.

Тема практической работы № 20. Создание и управление резервными копиями Oracle Database с помощью RMAN (Recovery Manager), объем часов: 4 часа.

У.4: Осуществлять основные функции по администрированию баз данных.
У.3: Документировать внештатные ситуации.
У.7: Производить регламентное обновление программного обеспечения.

Цель практической работы: Изучить принципы физического резервного копирования в Oracle, научиться работать в консоли RMAN и освоить процедуры проверки (валидации) бэкапов для обеспечения стопроцентной сохранности данных.

Задание(я):

1. Запустить утилиту RMAN и выполнить подключение к целевой базе данных (Target Database).
2. Создать полную резервную копию базы данных и управляющего файла (Control File).
3. Проверить целостность созданного бэкапа с помощью команды VALIDATE.
4. Имитировать критический сбой (удаление файла данных) и выполнить процедуру восстановления.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** В командной строке введите rman target /. Эта команда подключает вас к локальному экземпляру под правами SYSDBA. RMAN — это отдельная среда, работающая параллельно с SQL*Plus.
- **К заданию 2:** Для создания бэкапа выполните: BACKUP DATABASE PLUS ARCHIVELOG;. Включение архивных логов (archivelog) критически важно для восстановления данных на конкретный момент времени (Point-in-Time Recovery), что часто требуется при откате моделей ИИ к предыдущим состояниям данных.
- **К заданию 3:** Создание копии не гарантирует её читаемость. Выполните LIST BACKUP;, чтобы увидеть список копий, а затем VALIDATE BACKUPSET [номер];. Это регламентная процедура проверки, исключающая внештатные ситуации в будущем.
- **К заданию 4:** Имитируйте аварию, удалив один из файлов .dbf в папке oradata при выключенном сервере. При запуске база выдаст ошибку. Используйте RMAN для автоматического исправления: RESTORE DATABASE; и затем RECOVER DATABASE;.

Требования к отчету:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
    - **Скриншот** консоли RMAN с результатами выполнения команды BACKUP DATABASE.
    - **Скриншот** вывода команды LIST BACKUP, подтверждающий регистрацию бэкапа в репозитории.

3. **Документирование:**
    - **Отчет об инциденте**: «Критический сбой — повреждение физического файла данных. Статус базы: MOUNTED (не открывается)».
4. **Результаты проверки:**
    - **Скриншот** процесса RECOVER DATABASE, демонстрирующий накат логов.
    - **Скриншот** успешного открытия базы командой ALTER DATABASE OPEN;.
5. **Вывод:** Обоснуйте преимущество физического бэкапа RMAN перед логическим дампом (expdp) при работе с базами данных объемом в сотни гигабайт.

Тема практической работы № 21. Восстановление базы данных MySQL из резервной копии, созданной с помощью mysqldump, объем часов: 2 часа.

У.1: Идентифицировать проблемы функционирования базы данных.
У.2: Принимать решения по локализации проблем.
У.3: Документировать внештатные ситуации.
У.4: Осуществлять администрирование баз данных.

Цель практической работы: Приобрести практические навыки обнаружения сбоев в работе СУБД, научиться проводить первичную диагностику и реализовывать алгоритм полного восстановления работоспособности базы данных из логического дампа.

Задание(я):

1. Выполнить диагностику «проблемной» базы данных и выявить факт повреждения или отсутствия данных.
2. Разработать план локализации проблемы и подготовить среду к восстановлению.
3. Провести восстановление данных из ранее созданного SQL-дампа (ПР №16).
4. Оформить документацию по результатам ликвидации последствий внештатной ситуации.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Имитируйте проблему (удалите ключевую таблицу или всю базу). Попытайтесь выполнить SQL-запрос из приложения или консоли. Получив ошибку Table '...' doesn't exist или Unknown database, зафиксируйте её код. Это и есть идентификация проблемы.
- **К заданию 2:** Принятие решения по локализации включает в себя остановку внешних приложений (или скриптов ИИ), чтобы они не записывали новые данные в процессе восстановления. Если база повреждена частично, примите решение — восстанавливать её поверх или удалить и создать заново. Рекомендуется: DROP DATABASE IF EXISTS db_name; и CREATE DATABASE db_name;.
- **К заданию 3:** Выполните импорт через командную строку:
  `mysql -u root -p db_name < backup_full.sql`
  Обратите внимание на кодировку. Если дамп был в UTF8mb4, убедитесь, что клиент использует ту же кодировку, чтобы данные для моделей ИИ не превратились в «кракозябры».
- **К заданию 4:** Заполните отчетную форму, указав: время обнаружения, тип ошибки, источник бэкапа и время, затраченное на восстановление (RTO — Recovery Time Objective).

Требования к отчету:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
   - **Скриншот** с текстом ошибки, возникшей при попытке обращения к данным.
   - **Скриншот** процесса подготовки (создание пустой базы данных).
3. **Документирование инцидента:**
   - **Краткий отчет:** «Обнаружена недоступность обучающей выборки в 14:05. В 14:10 принято решение о восстановлении из дампа от [дата]. Инцидент исчерпан в 14:20».
4. **Результаты проверки:**
   - **Скриншот** консоли в процессе загрузки дампа.

- **Скриншот** результата контрольного запроса SELECT, подтверждающий наличие данных в восстановленных таблицах.

5. **Вывод:** Опишите, как наличие актуальных бэкапов влияет на минимизацию рисков в проектах по интеграции ИИ.

Тема практической работы № 22. Восстановление PostgreSQL базы данных из дампа (pg_restore), объем часов: 2 часа.

У.1: Идентифицировать проблемы функционирования базы данных
У.2: Принимать решения по локализации проблем
У.3: Документировать внештатные ситуации
У.4: Осуществлять основные функции по администрированию баз данных

Цель практической работы: Отработать технологию восстановления базы данных из кастомного архивного формата, научиться использовать инструменты выборочного восстановления объектов и фиксировать результаты устранения сбоев в системном журнале.

Задание(я):

1. Идентифицировать потерю конкретных объектов базы данных (таблиц или схем), необходимых для работы ИИ-моделей.
2. Выполнить предварительную подготовку целевой БД (очистка или создание новой) для исключения конфликтов при восстановлении.
3. Провести полное восстановление из файла кастомного формата (.dump), созданного в ходе выполнения предыдущих практических работ.
4. Проверить целостность связей и ограничений после восстановления и оформить отчет о проделанной работе.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Используйте pgAdmin или консольную утилиту psql для проверки наличия данных. Например, попытка обращения к таблице с обучающей выборкой может вернуть ошибку: ERROR: relation "dataset" does not exist. Зафиксируйте код ошибки и время инцидента в черновике отчета.
- **К заданию 2:** Если база данных существует, но повреждена, используйте флаг --clean в команде восстановления (это автоматически

удалит существующие объекты перед их воссозданием) или предварительно удалите базу вручную: DROP DATABASE ai_storage; CREATE DATABASE ai_storage;.

- **К заданию 3:** Используйте утилиту pg_restore. Пример команды для восстановления:
  `pg_restore -U postgres -d ai_storage -v "C:\backups\ai_db_backup.dump"`
  Флаг -v (verbose) позволяет видеть детальный лог процесса, что помогает быстро определить проблемные объекты, если файл бэкапа был поврежден частично.
- **К заданию 4:** Документирование — обязательная часть регламента администрирования. Составьте краткий "Акт восстановления", указав: имя файла архива, дату его создания и подтверждение того, что все триггеры, индексы и связи (Foreign Keys) восстановились корректно.

Требования к отчету:
Отчет должен быть выполнен в электронном виде и содержать следующие элементы:

1. **Титульный лист:** установленного образца.
2. **Ход выполнения заданий:**
   - **Скриншот** сообщения об ошибке, подтверждающего неработоспособность или неполноту данных в БД.
   - **Скриншот** окна командной строки с процессом выполнения команды pg_restore.
3. **Документирование инцидента:**
   - **Журнал событий:** краткая хронология (время обнаружения сбоя, время принятия решения о восстановлении, время завершения операции).
4. **Результаты проверки:**
   - **Скриншот** интерфейса pgAdmin, демонстрирующий структуру восстановленных таблиц.
   - **Скриншот** результата выполнения тестового SQL-запроса к восстановленным данным.

5. **Вывод:** краткое описание преимуществ использования сжатых дампов (.dump) при необходимости быстрого восстановления систем искусственного интеллекта.

Тема практической работы № 23. Восстановление базы данных Microsoft SQL Server из полной резервной копии с использованием SSMS, объем часов: 2 часа.

У.1: Идентифицировать проблемы функционирования базы данных
У.2: Принимать решения по локализации проблем
У.3: Документировать внештатные ситуации
У.4: Осуществлять основные функции по администрированию баз данных

Цель практической работы: Изучить алгоритм восстановления базы данных из файла резервной копии (.bak) в среде SQL Server Management Studio, отработать навыки локализации последствий сбоев и проверки целостности данных.

Задание(я):

1. Выполнить идентификацию программного или физического сбоя, приведшего к недоступности базы данных.
2. Провести подготовку сервера к восстановлению (анализ доступных бэкапов и закрытие активных соединений).
3. Выполнить полное восстановление базы данных из резервной копии с заменой существующей БД.
4. Проверить работоспособность базы и задокументировать этапы устранения внештатной ситуации.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Имитируйте критический сбой, удалив базу данных или повредив её файлы при остановленной службе. При попытке подключения зафиксируйте статус базы (например, «Recovery Pending» или её отсутствие в списке). Это этап идентификации проблемы.
- **К заданию 2:** Для успешного восстановления необходимо убедиться, что к базе нет активных подключений. В процессе восстановления в SSMS используйте опцию «Close existing connections to destination

database». Это решение по локализации проблемы, исключающее конфликты доступа.

- **К заданию 3:** В окне «Restore Database» выберите источник (Device) и укажите путь к вашему .bak файлу. На вкладке **Options** обязательно отметьте пункт **«Overwrite the existing database (WITH REPLACE)»**. Если вы восстанавливаете базу на другом сервере, проверьте вкладку «Files», чтобы пути к файлам .mdf и .ldf соответствовали новой файловой системе.
- **К заданию 4:** После сообщения «Database restored successfully» проверьте консистентность данных. Документирование ситуации должно включать описание причины сбоя и подтверждение того, что база переведена в статус «Online».

Требования к отчету:

1. **Титульный лист:** оформленный по стандарту.
2. **Ход выполнения заданий:**
    - **Скриншот** окна Object Explorer с ошибкой доступа к базе или отсутствием объекта.
    - **Скриншот** окна настроек восстановления (вкладки General и Options).
3. **Документирование инцидента:**
    - **Запись в журнале**: указание типа сбоя, времени восстановления и версии использованного бэкапа.
4. **Результаты проверки:**
    - **Скриншот** системного сообщения об успешном завершении операции восстановления.
    - **Скриншот** окна запроса (Query), показывающий содержимое одной из таблиц восстановленной базы.
5. **Вывод:** описание значения регулярной проверки бэкапов на «восстанавливаемость» для стабильной работы сервисов ИИ.

Тема практической работы № 24. Восстановление базы данных MongoDB из резервного архива, объем часов: 2 часа.

У.1: Идентифицировать проблемы функционирования базы данных
У.2: Принимать решения по локализации проблем
У.3: Документировать внештатные ситуации
У.4: Осуществлять основные функции по администрированию баз данных

Цель практической работы: Освоить технологию восстановления данных из бинарных дампов MongoDB, научиться использовать утилиту mongorestore для возврата работоспособности нереляционных систем и документировать процесс ликвидации последствий потери данных.

Задание(я):

1. Идентифицировать отсутствие или порчу данных в коллекции MongoDB (например, случайное удаление документов через Mongo Shell).
2. Подготовить целевую среду к импорту данных и локализовать проблему, ограничив доступ к поврежденной коллекции.
3. Произвести полное восстановление базы данных или отдельной коллекции из бинарного архива (дампа).
4. Проверить целостность документов и восстановить работоспособность интеграционных шлюзов ИИ.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Используйте MongoDB Compass или оболочку mongosh. Выполните команду db.collection.countDocuments({}). Если результат равен нулю или отличается от ожидаемого, зафиксируйте факт потери данных. Это начальный этап идентификации проблемы.
- **К заданию 2:** В отличие от реляционных СУБД, MongoDB автоматически создает базу при записи. Чтобы избежать дублирования или конфликтов идентификаторов (_id), перед восстановлением рекомендуется удалить поврежденную базу командой db.dropDatabase(). Это радикальное, но эффективное решение по локализации проблемы.
- **К заданию 3:** Используйте утилиту mongorestore. Если бэкап был создан в ПР №19, выполните команду: mongorestore --db=имя_базы C:\backups\путь_к_папке_дампа\имя_базы. Если вы восстанавливаете данные на сервер с включенной авторизацией,

не забудьте добавить флаги --username и --password. При использовании Replica Set восстановление должно производиться на **Primary** узле.

- **К заданию 4:** После завершения импорта проверьте наличие индексов. По умолчанию mongorestore восстанавливает индексы, но важно убедиться в их работоспособности через db.collection.getIndexes(), так как от них зависит скорость работы ИИ-приложений.

Требования к отчету:

1. **Титульный лист:** стандартного образца.
2. **Ход выполнения заданий:**
    - **Скриншот** консоли mongosh с результатом поиска, подтверждающим отсутствие данных.
    - **Скриншот** окна командной строки в процессе работы утилиты mongorestore (лог восстановления).
3. **Документирование инцидента:**
    - **Журнал внештатных ситуаций**: краткое описание (дата, причина — человеческий фактор/сбой ПО, время простоя системы).
4. **Результаты проверки:**
    - **Скриншот** интерфейса MongoDB Compass с восстановленными документами в коллекции.
    - **Скриншот** вывода команды проверки количества документов после восстановления.
5. **Вывод:** описание особенностей восстановления NoSQL-баз (скорость, работа с BSON-форматом) по сравнению с реляционными системами.

Тема практической работы № 25. Восстановление Oracle Database с использованием RMAN (Recovery Manager), объем часов: 4 часа.

У.1: Идентифицировать проблемы функционирования базы данных
У.2: Принимать решения по локализации проблем
У.3: Документировать внештатные ситуации
У.4: Осуществлять основные функции по администрированию баз данных

Цель практической работы: Изучить процедуры восстановления базы данных на физическом уровне, научиться использовать Recovery Manager (RMAN) для ликвидации последствий критических сбоев и отработать навыки возврата БД в состояние «Open».

Задание(я):

1. Идентифицировать критический сбой (отсутствие или повреждение файла данных), препятствующий запуску экземпляра БД.
2. Провести диагностику через RMAN и принять решение о методе локализации проблемы (восстановление конкретного файла или всей базы).
3. Выполнить процедуру восстановления (Restore) и наката изменений (Recover) из архивных копий.
4. Документировать процесс устранения внештатной ситуации и проверить целостность данных.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** При попытке перевести базу в режим открытия (ALTER DATABASE OPEN;) в SQL*Plus возникнет ошибка, например, ORA-01157: cannot identify/lock data file. Зафиксируйте номер отсутствующего файла. Это основной признак проблемы функционирования.
- **К заданию 2:** Подключитесь к RMAN (rman target /). Используйте команду LIST FAILURE; и ADVISE FAILURE;. Oracle сам проанализирует повреждения и предложит скрипт для локализации и исправления ситуации. Это интеллектуальный способ принятия решений администратором.
- **К заданию 3:** В консоли RMAN выполните последовательность команд:
    - STARTUP MOUNT; (если база была выключена);
    - RESTORE DATABASE; (извлечение файлов из бэкапа);
    - RECOVER DATABASE; (применение архивных логов для актуализации данных).
      Для ИИ-систем критически важно довести базу до самого последнего транзакционного состояния, чтобы не потерять результаты обучения моделей.

- **К заданию 4:** После завершения наката выполните ALTER DATABASE OPEN;. В рамках документирования внештатной ситуации опишите тип повреждения и подтвердите, что база переведена в штатный режим работы.

Требования к отчету:

1. **Титульный лист:** по установленному образцу.
2. **Ход выполнения заданий:**
    - **Скриншот** окна SQL*Plus с ошибкой открытия базы данных.
    - **Скриншот** вывода команды ADVISE FAILURE в консоли RMAN.
3. **Документирование инцидента:**
    - **Журнал инцидентов**: описание причины сбоя (например, «физическое удаление datafile 4»), хронометраж действий администратора.
4. **Результаты проверки:**
    - **Скриншот** лога RMAN, подтверждающий успешное выполнение команд RESTORE и RECOVER.
    - **Скриншот** финального статуса базы данных (команда SELECT status FROM v$instance;).
5. **Вывод:** Опишите, почему использование RMAN является более надежным методом для крупных СУБД по сравнению с ручным копированием файлов.

Тема практической работы № 26. Настройка и использование утилиты MySQL Performance Schema для мониторинга работы базы данных, объем часов: 4 часа.

У.1: Идентифицировать проблемы функционирования базы данных
У.2: Локализовать проблемы
У.6: Давать независимую оценку уровня безопасности

Цель практической работы: Изучить архитектуру инструментального мониторинга MySQL, научиться активировать сбор метрик

производительности и использовать системные таблицы для поиска «узких мест» и потенциальных угроз безопасности.

Задание(я):

1. Проверить статус активации Performance Schema и настроить параметры сбора событий.
2. Провести идентификацию проблем производительности с помощью анализа ожиданий (Wait Events).
3. Локализовать наиболее ресурсоемкие запросы, используя системное представление `sys.statement_analysis`.
4. Выполнить аудит текущих соединений для независимой оценки безопасности подключений.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Проверьте включение схемы запросом `SHOW VARIABLES LIKE 'performance_schema';`. Если значение `OFF`, его необходимо изменить в файле `my.ini` и перезапустить службу. Performance Schema хранит данные в памяти, что обеспечивает высокую скорость мониторинга.
- **К заданию 2:** Для идентификации проблем используйте таблицу `events_waits_summary_global_by_event_name`. Она покажет, какие операции (чтение диска, блокировки) заставляют сервер простаивать. Это важно для понимания задержек в конвейерах обработки данных ИИ.
- **К заданию 3:** Чтобы локализовать проблему, обратитесь к схеме **SYS**. Выполните запрос: `SELECT query, exec_count, avg_latency FROM sys.statement_analysis ORDER BY avg_latency DESC LIMIT 5;`. Это выявит 5 самых медленных запросов, которые требуют оптимизации.
- **К заданию 4:** Для оценки безопасности проанализируйте таблицу `users`. Проверьте наличие подключений с подозрительных хостов или под учетными записями с избыточными правами. Используйте таблицу `host_cache` для выявления ошибок аутентификации, которые могут свидетельствовать о попытках подбора пароля.

Требования к отчету:

1. **Титульный лист:** по стандарту.
2. **Ход выполнения заданий:**

- **Скриншот** результата проверки переменной `performance_schema`.

- **Скриншот** SQL-запроса к таблице ожиданий с пояснением самого частого события.

3. **Анализ производительности и безопасности:**

- **Таблица** (или скриншот) с топ-5 медленных запросов.
- **Скриншот** списка активных сессий пользователей из таблицы `processlist` или `threads`.

4. **Оценка безопасности:**

- **Краткое заключение**: «В ходе мониторинга аномальных подключений не выявлено» или «Обнаружено превышение лимита соединений для сервисного аккаунта».

5. **Вывод:** опишите, как использование Performance Schema помогает администратору предотвратить отказ системы при резком росте нагрузки со стороны ИИ-приложений.

Тема практической работы № 27. Использование утилиты pg_stat_activity в PostgreSQL для отслеживания активных соединений и запросов, объем часов: 2 часа.

У.1: Идентифицировать проблемы функционирования базы данных
У.2: Локализовать проблемы
У.6: Давать независимую оценку уровня безопасности

Цель практической работы: Научиться проводить оперативный мониторинг текущего состояния СУБД PostgreSQL, выявлять блокировки и длительные запросы, а также проводить аудит активных сессий для обеспечения безопасности данных.

Задание(я):

1. Изучить структуру системного представления pg_stat_activity.
2. Выполнить идентификацию активных сессий и определить их текущий статус.
3. Локализовать «проблемные» запросы (длительные транзакции и ожидания).
4. Провести независимую оценку безопасности соединений, проанализировав сетевые адреса и учетные записи подключенных клиентов.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Откройте Query Tool в pgAdmin и выполните базовый запрос: `SELECT * FROM pg_stat_activity;`. Ознакомьтесь с ключевыми полями: `datname` (имя базы), `usename` (пользователь), `client_addr` (IP клиента), `state` (статус сессии) и `query` (текст запроса).
- **К заданию 2:** Для идентификации проблем обратите внимание на статус active. Если сессия находится в этом статусе слишком долго, это может указывать на зацикливание или крайне неэффективный алгоритм обработки данных в ИИ-приложении.
- **К заданию 3:** Чтобы локализовать проблему, используйте фильтрацию по времени выполнения запроса. Например, запрос для поиска сессий, работающих дольше 5 минут:
  `SELECT pid, now() - query_start AS duration, query, state FROM pg_stat_activity WHERE state != 'idle' AND (now() - query_start) > interval '5 minutes';`.
  Узнайте `pid` (идентификатор процесса) для возможности его принудительной остановки.
- **К заданию 4:** При оценке безопасности проверьте поле `client_addr`. Наличие соединений с неизвестных IP-адресов или использование учетной записи `postgres` внешними скриптами должно быть зафиксировано как нарушение политики безопасности.

Требования к отчету:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
   - **Скриншот** результата выполнения общего запроса к pg_stat_activity.
   - **Скриншот** SQL-запроса для выявления длительных транзакций.
3. **Анализ активности:**
   - **Таблица**, содержащая список активных пользователей и количество открытых ими соединений.

- **Скриншот** с примером запроса, находящегося в состоянии ожидания (wait_event).

4. **Оценка безопасности:**
    - **Краткое заключение**: подтверждение легитимности всех активных IP-адресов и используемых учетных записей.
5. **Вывод:** опишите значимость мониторинга активных сессий для предотвращения критических перегрузок сервера БД при интенсивной работе ИИ-интеграций.

Тема практической работы № 28. Мониторинг событий в Microsoft SQL Server с помощью SQL Server Profiler, объем часов: 2 часа.

У.1: Идентифицировать проблемы функционирования базы данных
У.2: Локализовать проблемы
У.6: Давать независимую оценку уровня безопасности

Цель практической работы: Изучить методику трассировки событий в реальном времени, научиться настраивать фильтры событий для выявления аномалий и использовать полученные данные для оценки безопасности и производительности сервера.

Задание(я):

1. Запустить SQL Server Profiler и создать новый шаблон трассировки.
2. Выполнить идентификацию событий выполнения SQL-запросов и логических чтений.
3. Локализовать наиболее ресурсоемкие команды, проанализировав столбцы CPU, Duration и Reads.
4. Провести независимую оценку безопасности, отследив события входа (Login) и неудачные попытки аутентификации.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Откройте Profiler (меню Tools в SSMS). При создании трассировки выберите шаблон «Standard». На вкладке «Events Selection» убедитесь, что выбраны события RPC:Completed и SQL:BatchCompleted. Это позволит видеть все запросы, приходящие от ИИ-модулей.

- **К заданию 2:** Для идентификации проблем запустите трассировку и выполните в SSMS несколько разных запросов (простых и сложных). В окне Profiler вы увидите, как сервер обрабатывает каждый из них. Обратите внимание на столбец TextData.
- **К заданию 3:** Чтобы локализовать проблему, настройте фильтр (кнопка «Column Filters»). Установите значение Duration > 1000 (миллисекунд). Теперь Profiler будет показывать только те запросы, которые тормозят систему. Анализ показателя Reads поможет понять, какие запросы заставляют сервер выполнять избыточное сканирование диска.
- **К заданию 4:** Для оценки безопасности добавьте в трассировку события из категории «Security Audit»: Audit Login и Audit Logout. Если вы видите частые ошибки входа с одного IP, это может указывать на атаку или неверно настроенный сервис интеграции.

Требования к отчету:

1. **Титульный лист:** по стандарту.
2. **Ход выполнения заданий:**
    - **Скриншот** окна настройки трассировки (выбранные события и фильтры).
    - **Скриншот** окна Profiler в процессе активной трассировки запросов.
3. **Анализ событий:**
    - **Листинг (текст)** самого «дорогого» запроса (с максимальным CPU или Reads), зафиксированного профилировщиком.
    - **Скриншот** событий аудита безопасности (успешные входы в систему).
4. **Оценка безопасности:**
    - **Краткое заключение**: оценка легитимности зафиксированных подключений и отсутствия подозрительной активности в логах трассировки.
5. **Вывод:** опишите преимущества использования Profiler для отладки приложений искусственного интеллекта на этапе их интеграции с БД.

Тема практической работы № 29. Установка и настройка Prometheus для сбора метрик производительности базы данных MySQL, объем часов: 2 часа.

У.1: Идентифицировать проблемы функционирования базы данных
У.2: Локализовать проблемы
У.6: Давать независимую оценку уровня безопасности

Цель практической работы: Изучить принципы работы современных систем мониторинга, научиться настраивать экспорт метрик из СУБД MySQL и конфигурировать Prometheus для анализа производительности и стабильности базы данных.

Задание(я):

1. Произвести установку сервера Prometheus и специализированного экспортера mysqld_exporter.
2. Создать в MySQL выделенную учетную запись для мониторинга с ограниченными правами.
3. Настроить конфигурационный файл Prometheus для регулярного сбора (скрейпинга) данных из базы.
4. Провести идентификацию аномалий в нагрузке на СУБД через веб-интерфейс Prometheus.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Скачайте бинарные файлы Prometheus и mysqld_exporter. Экспортер служит «прослойкой»: он забирает данные из MySQL и отдает их в формате, понятном Prometheus. Запустите экспортер, указав параметры подключения к базе через переменную окружения DATA_SOURCE_NAME.
- **К заданию 2:** Для обеспечения безопасности (У.6) не используйте root. Создайте пользователя mysqld_exporter и выдайте ему только необходимые права: PROCESS, SELECT, REPLICATION CLIENT. Это позволит собирать метрики, не создавая угрозы несанкционированного изменения данных.
- **К заданию 3:** В файле prometheus.yml добавьте новый блок в секцию scrape_configs. Укажите job_name: 'mysql' и адрес экспортера (по умолчанию localhost:9104). После этого перезапустите Prometheus.

- **К заданию 4:** Откройте веб-интерфейс (порт 9090). Используйте язык запросов PromQL для идентификации проблем (У.1). Например, запрос mysql_global_status_threads_connected покажет количество активных соединений. Резкие скачки на графике помогут локализовать моменты пиковой нагрузки от ИИ-сервисов.

Требования к отчету:

1. **Титульный лист:** стандартного образца.
2. **Ход выполнения заданий:**
    - **Скриншот** процесса запуска mysqld_exporter в консоли.
    - **Листинг кода** (фрагмент) файла prometheus.yml с настройками для MySQL.
3. **Анализ метрик:**
    - **Скриншот** веб-интерфейса Prometheus (раздел Status -> Targets), подтверждающий состояние «UP» для MySQL.
    - **Скриншот графика** любой метрики (например, количество запросов или загрузка CPU), полученного через встроенный визуализатор.
4. **Оценка безопасности:**
    - **Краткое заключение**: подтверждение того, что учетная запись мониторинга имеет минимально достаточные права доступа.
5. **Вывод:** описание преимуществ использования внешних систем мониторинга (Prometheus) для раннего обнаружения сбоев в работе высоконагруженных баз данных.

Тема практической работы № 30. Анализ журнала событий (log files) в Oracle Database для выявления ошибок и проблем, объем часов: 4 часа.

У.1: Идентифицировать проблемы функционирования базы данных
У.2: Локализовать проблемы
У.6: Давать независимую оценку уровня безопасности

Цель практической работы: Изучить структуру и местоположение журналов событий Oracle Database, научиться интерпретировать сообщения об ошибках

в Alert Log и использовать ADRCI (Automatic Diagnostic Repository Command Interpreter) для диагностики системы.

Задание(я):

1. Определить местоположение Alert Log и трассировочных файлов через системные представления.
2. Провести идентификацию системных событий и ошибок запуска/остановки экземпляра.
3. Локализовать ошибки в файлах трассировки (Trace files) при возникновении критических исключений.
4. Выполнить аудит журнала на предмет подозрительных действий и нарушений целостности данных для оценки безопасности.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Для поиска пути к журналам выполните запрос в SQL*Plus: SELECT value FROM v$parameter WHERE name = 'background_dump_dest';. Также можно использовать утилиту командной строки **ADRCI**, введя команду show alert.
- **К заданию 2:** Alert Log содержит хронологию всех значимых событий. Для идентификации проблем (У.1) найдите коды ошибок с префиксом **ORA-**. Например, ошибки серии ORA-00600 указывают на внутренние критические сбои, а ошибки контрольных сумм — на проблемы с дисковой подсистемой.
- **К заданию 3:** Если произошла критическая ошибка процесса (например, фонового процесса DBWn), Oracle создает файл трассировки (.trc). Чтобы локализовать проблему, необходимо проанализировать содержимое файла, на который ссылается Alert Log. В нем будет указан конкретный SQL-запрос или блок памяти, вызвавший сбой.
- **К заданию 4:** При анализе безопасности обратите внимание на сообщения о неудачных попытках запуска базы, изменении параметров аутентификации или несанкционированном использовании команд ALTER SYSTEM. Это позволяет выявить попытки дестабилизации работы ИИ-сервисов на уровне ядра СУБД.

Требования к отчету:

1. **Титульный лист:** стандартного образца.
2. **Ход выполнения заданий:**
    - **Скриншот** результата SQL-запроса, определяющего путь к папке логов.
    - **Скриншот** окна утилиты ADRCI с выведенными последними сообщениями журнала.
3. **Анализ логов:**
    - **Фрагмент текста (листинг)** из Alert Log, содержащий записи о последнем успешном запуске (Startup) базы данных.
    - **Пример идентификации ошибки**: описание любой найденной ошибки ORA- или фиксация её отсутствия («Clean log»).
4. **Оценка безопасности:**
    - **Краткое заключение**: результат проверки журнала на наличие несанкционированных административных действий.
5. **Вывод:** описание роли регулярного анализа журналов событий в предотвращении длительных простоев ИИ-инфраструктуры.

Тема практической работы № 31. Настройка и анализ журнала ошибок (error log) в MySQL, объем часов: 2 часа.

У.3: Документировать внештатные ситуации
У.6: Давать независимую оценку уровня безопасности
У.8: Разрабатывать рекомендации по дальнейшей эксплуатации БД

Цель практической работы: Изучить механизмы логирования ошибок в СУБД MySQL, научиться настраивать уровни детализации журнала и использовать данные логов для расследования инцидентов безопасности и разработки стратегии защиты данных.

Задание(я):

1. Определить текущее местоположение и параметры настройки Error Log в конфигурации сервера.

2. Имитировать внештатную ситуацию (некорректный запуск или критический сбой плагина) и задокументировать её отражение в журнале.
3. Провести независимый аудит безопасности, выявив в логе попытки несанкционированного доступа (Access Denied).
4. Разработать рекомендации по настройке ротации логов для долгосрочной эксплуатации БД.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Путь к журналу ошибок задается параметром log_error. Проверьте его значение через SQL-запрос: `SHOW VARIABLES LIKE 'log_error';`. Убедитесь, что параметр log_error_verbosity установлен в значение **3** (максимальная детализация), что необходимо для глубокого анализа работы ИИ-интеграций.
- **К заданию 2:** Для документирования внештатной ситуации попробуйте внести заведомо неверный параметр в файл my.ini и перезапустить службу. После неудачного старта откройте файл лога, найдите записи со статусом [ERROR] и скопируйте описание причины сбоя.
- **К заданию 3:** При оценке безопасности ищите записи вида Access denied for user.... Большое количество таких записей с одного IP-адреса свидетельствует о попытках брутфорс-атаки. Зафиксируйте данные инциденты в отчете.
- **К заданию 4:** На основе анализа объема логов разработайте рекомендации. Например, использование утилиты logrotate (в Linux) или ручного архивирования (в Windows), чтобы файл журнала не занимал все дисковое пространство, что критично для стабильности работы серверов баз данных.

Требования к отчету:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
    - **Скриншот** результата проверки системных переменных log_error и log_error_verbosity.

- **Скриншот фрагмента лога** с зафиксированной ошибкой запуска (результат задания 2).

3. **Аудит безопасности:**
    - **Выписка из журнала**: перечень подозрительных попыток входа или подтверждение «чистоты» лога.
4. **Разработка рекомендаций:**
    - **Текстовый блок**: предложения по периодичности проверки логов и методам их хранения для обеспечения максимальной защиты.
5. **Вывод:** описание важности Error Log как первичного источника информации при расследовании причин отказа систем искусственного интеллекта.

Тема практической работы № 32. Конфигурация и просмотр логов событий в PostgreSQL с использованием параметра logging_collector, объем часов: 2 часа.

У.3: Документировать внештатные ситуации
У.6: Давать независимую оценку уровня безопасности
У.8: Разрабатывать рекомендации по дальнейшей эксплуатации БД

Цель практической работы: Изучить подсистему сбора сообщений PostgreSQL, научиться активировать и настраивать параметры журналирования событий, а также проводить анализ полученных логов для повышения безопасности и стабильности работы СУБД.

Задание(я):

1. Активировать подсистему сбора логов в конфигурационном файле сервера.
2. Настроить параметры ротации журналов событий (размер файла и время жизни).
3. Произвести идентификацию подозрительной активности и ошибок аутентификации через анализ лог-файлов.
4. Разработать регламент мониторинга журналов для обеспечения бесперебойной эксплуатации БД.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Откройте файл `postgresql.conf`. Для включения сбора сообщений установите параметр `logging_collector = on`. Укажите директорию для хранения файлов через параметр `log_directory` (обычно 'log'). После внесения изменений **перезапустите службу** PostgreSQL.
- **К заданию 2:** Для предотвращения переполнения диска настройте ротацию. Установите `log_rotation_age = 1d` (создание нового файла каждые сутки) и `log_rotation_size = 10MB` (создание нового файла при достижении размера 10 МБ). Это гарантирует наличие актуальных данных без риска остановки сервера из-за нехватки места.
- **К заданию 3:** Проведите эксперимент: попробуйте подключиться к базе с неверным паролем несколько раз. Откройте последний созданный файл в папке `log` и найдите записи со статусом `FATAL: password authentication failed for user`. Наличие таких записей с внешних IP-адресов позволяет дать независимую оценку безопасности системы.
- **К заданию 4:** На основе проведенной работы сформулируйте рекомендации. Например, включение параметра `log_duration = on` для отслеживания времени выполнения всех запросов ИИ-моделей или настройка `log_min_messages` для фильтрации незначительных уведомлений.

Требования к отчету:

1. **Титульный лист:** установленного образца.
2. **Ход выполнения заданий:**
    - **Скриншот** фрагмента файла postgresql.conf с настроенными параметрами logging_collector и ротации.
    - **Скриншот** содержимого директории log, подтверждающий создание файлов журналов в заданном формате.
3. **Документирование и аудит:**
    - **Листинг (фрагмент лога)** с зафиксированной ошибкой доступа или системным предупреждением.

- **Описание инцидента**: краткое пояснение найденной записи (тип события, время, влияние на систему).

4. **Рекомендации по эксплуатации:**
    - **Текстовый блок**: перечень предлагаемых настроек для долгосрочной защиты сервера.
5. **Вывод:** оценка эффективности использования logging_collector для оперативного выявления проблем в высоконагруженных проектах.

Тема практической работы № 33. Настройка протоколирования аудита в Microsoft SQL Server с использованием Extended Events, объем часов: 2 часа.

У.3: Документировать внештатные ситуации
У.6: Давать независимую оценку уровня безопасности
У.8: Разрабатывать рекомендации по дальнейшей эксплуатации БД

Цель практической работы: Освоить инструменты расширенных событий (Extended Events) для создания гибких систем аудита, научиться отслеживать критические события безопасности и настраивать автоматическое протоколирование действий пользователей.

Задание(я):

1. Создать сессию расширенных событий для мониторинга неудачных попыток входа в систему.
2. Настроить хранилище (Target) для записи событий в файл и буфер обмена.
3. Провести идентификацию и документирование попыток несанкционированного доступа.
4. Сформулировать рекомендации по внедрению постоянного аудита для защиты конфиденциальных данных ИИ.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** В среде SSMS перейдите в раздел **Management -> Extended Events -> Sessions**. Используйте «New Session Wizard». Выберите событие login_failed из библиотеки событий. В отличие от

трассировок, сессии XEvents потребляют значительно меньше ресурсов CPU, что критично для серверов, обрабатывающих данные в реальном времени.

- **К заданию 2:** Настройте параметры «Target». Выберите event_file для сохранения истории на диск и ring_buffer для оперативного просмотра последних событий в памяти. Укажите максимальный размер файла и количество файлов в цепочке ротации для предотвращения переполнения диска.
- **К заданию 3:** Запустите созданную сессию (Start Session). Имитируйте внештатную ситуацию: попробуйте подключиться к серверу с заведомо неверным логином. Откройте окно «Watch Live Data» в SSMS. Зафиксируйте детали события: имя пользователя, время, ошибку и IP-адрес источника. Эти данные необходимы для независимой оценки безопасности системы.
- **К заданию 4:** Проанализируйте полученные данные и разработайте рекомендации. Например, предложите автоматическую блокировку IP-адреса на уровне брандмауэра при достижении порога в 10 неудачных попыток входа за минуту, зафиксированных через Extended Events.

Требования к отчету:

1. **Титульный лист.**
2. **Ход выполнения заданий:**
    - **Скриншот** окна настройки сессии Extended Events с выбранным событием login_failed.
    - **Скриншот** настройки целевого хранилища (Data Storage).
3. **Аудит и документирование:**
    - **Скриншот окна Watch Live Data**, отображающий зафиксированную попытку неудачного входа.
    - **Описание инцидента**: краткий отчет на основе полей события (клиентское приложение, код ошибки).
4. **Рекомендации по эксплуатации:**
    - **Текстовый блок**: перечень дополнительных событий (например, database_xml_deadlock_report), которые следует включить в постоянный аудит.

5. **Вывод:** обоснование перехода на Extended Events как на более современный и безопасный метод мониторинга по сравнению с классическими логами.

Тема практической работы № 34. Включение и настройка логирования операций в MongoDB с использованием параметра profilingLevel, объем часов: 4 часа.

У.3: Документировать внештатные ситуации
У.6: Давать независимую оценку уровня безопасности
У.8: Разрабатывать рекомендации по дальнейшей эксплуатации БД

Цель практической работы: Изучить механизмы профилирования в NoSQL-системах, научиться настраивать уровни логирования операций в MongoDB и использовать системную коллекцию system.profile для аудита безопасности и анализа эффективности запросов.

Задание(я):

1. Настроить уровень профилирования базы данных (profilingLevel) и установить порог фиксации медленных операций.
2. Провести идентификацию и документирование ресурсоемких операций (чтение/запись), влияющих на стабильность системы.
3. Выполнить аудит выполненных команд для независимой оценки безопасности доступа к коллекциям.
4. Разработать рекомендации по настройке профилирования на продуктовых серверах для предотвращения деградации производительности.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Используйте оболочку mongosh. Для включения профилирования выполните команду db.setProfilingLevel(2). Уровень **2** фиксирует все операции, уровень **1** — только медленные (превышающие slowms). Для задач отладки интеграций ИИ рекомендуется уровень 1 с порогом slowms: 50, чтобы видеть запросы, замедляющие обучение моделей.

- **К заданию 2:** Все данные профилирования записываются в системную коллекцию system.profile. Выполните несколько запросов к БД, затем найдите самые длительные из них: db.system.profile.find().sort({millis:-1}).limit(5). Задокументируйте найденные операции как потенциальные внештатные ситуации, требующие оптимизации.
- **К заданию 3:** В рамках оценки безопасности проанализируйте поле user и client в коллекции профиля. Убедитесь, что операции удаления данных (remove, drop) выполнялись только авторизованными администраторами. Наличие аномальных массовых чтений может свидетельствовать о попытке выгрузки (кражи) обучающих датасетов.
- **К заданию 4:** На основе анализа нагрузки сформулируйте рекомендации. Учтите, что уровень профилирования 2 создает высокую нагрузку на диск и CPU. Рекомендуйте использовать его только для отладки, а для постоянной эксплуатации — уровень 1 с адекватным порогом фильтрации.

Требования к отчету:

1. **Титульный лист:** по установленному образцу.
2. **Ход выполнения заданий:**
    - **Скриншот** выполнения команды db.setProfilingLevel и проверки текущего статуса через db.getProfilingStatus().
    - **Скриншот** структуры документов в коллекции system.profile.
3. **Аудит и документирование:**
    - **Листинг (фрагмент JSON)** документа профиля, содержащего информацию о медленном запросе.
    - **Описание инцидента**: анализ полей millis (время), keysExamined (индексы) и ns (коллекция).
4. **Рекомендации по эксплуатации:**
    - **Текстовый блок**: обоснование выбора уровня профилирования для различных этапов жизненного цикла ИИ-проекта (разработка/продакшн).
5. **Вывод:** оценка полезности профилирования для обеспечения прозрачности работы NoSQL-базы.

Тема практической работы № 35. Настройка и просмотр журнала аудита (Audit Trail) в Oracle Database, объем часов: 4 часа.

У.3: Документировать внештатные ситуации
У.6: Давать независимую оценку уровня безопасности
У.8: Разрабатывать рекомендации по дальнейшей эксплуатации БД

Цель практической работы: Изучить механизмы стандартного и расширенного аудита в Oracle Database, научиться активировать сбор статистических данных о действиях пользователей и анализировать журнал аудита для выявления нарушений политик безопасности.

Задание(я):

1. Проверить текущие настройки параметров аудита сервера и активировать запись журнала в базу данных.
2. Настроить аудит конкретных действий (например, попыток входа и операций изменения таблиц).
3. Провести идентификацию и документирование событий доступа к конфиденциальным данным.
4. Выполнить независимую оценку безопасности на основе анализа системного представления DBA_AUDIT_TRAIL.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **К заданию 1:** Проверьте параметр `audit_trail` запросом: `SELECT name, value FROM v$parameter WHERE name = 'audit_trail';`. Если аудит отключен (`NONE`), установите его значение в `DB` командой `ALTER SYSTEM SET audit_trail=db SCOPE=SPFILE;` и **перезапустите экземпляр** базы данных. В Oracle аудит в БД позволяет удобно анализировать логи с помощью SQL.
- **К заданию 2:** Включите аудит для мониторинга подозрительной активности. Например, для отслеживания всех неудачных попыток входа: `AUDIT SESSION WHENEVER NOT SUCCESSFUL;`. Для контроля изменений в таблицах с данными ИИ используйте: `AUDIT INSERT, UPDATE, DELETE ON имя_таблицы BY ACCESS;`.

- **К заданию 3:** Имитируйте внештатную ситуацию: войдите под созданным ранее пользователем и выполните несколько операций с таблицей. Затем, от лица администратора, задокументируйте эти действия, выгрузив данные из таблицы SYS.AUD$. Зафиксируйте время, имя пользователя и тип выполненной команды (SQL-код).
- **К заданию 4:** Для независимой оценки безопасности проанализируйте представление DBA_AUDIT_TRAIL. Ищите записи, где RETURNCODE отличен от нуля (ошибки доступа). Большое количество таких записей может свидетельствовать о попытках несанкционированного сбора данных (Data Scraping) со стороны внешних интеграций.

Требования к отчету:

1. **Титульный лист:** стандартного образца.
2. **Ход выполнения заданий:**
    - **Скриншот** результата проверки и изменения параметра audit_trail.
    - **Листинг команд** AUDIT, примененных в ходе работы.
3. **Аудит и документирование:**
    - **Скриншот** результата запроса к DBA_AUDIT_TRAIL, показывающий зафиксированные действия.
    - **Описание инцидента**: анализ одной строки аудита (кто, когда и какое действие совершил).
4. **Рекомендации по эксплуатации:**
    - **Текстовый блок**: предложения по очистке журнала аудита (Purging), чтобы избежать переполнения системного табличного пространства SYSTEM.
5. **Вывод:** обоснование необходимости включения аудита в корпоративных системах, использующих технологии ИИ для работы с персональными данными.

МДК 02.02. Технология разработки и защиты баз данных

Тема практической работы № 1. Создание концептуальной модели базы данных с использованием диаграммы «сущность-связь» (ER-диаграмма), объем часов: 4 часа.

У.9: Формировать требования к обработке данных и их извлечению.
У.10: Добавлять, удалять и изменять данные в базе данных.

Цель практической работы: Научиться анализировать предметную область, выделять ключевые сущности и атрибуты, а также проектировать логические связи между ними для последующей реализации в СУБД.

Материальное обеспечение: LibreOffice Draw, Draw io

Задание:

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

Варианты заданий:

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

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

Краткие теоретические сведения:

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

Основные элементы ER-диаграммы:

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

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

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

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

Алгоритм проектирования ER-диаграммы

1. **Определить сущности** - выделить основные объекты предметной области.
2. **Определить атрибуты** - для каждой сущности описать ключевые характеристики.
3. **Определить связи** - установить, как сущности взаимодействуют друг с другом (один к одному, один ко многим и многие ко многим).
4. **Назначить ключи** - для каждой сущности выберать первичный ключ, для связей - внешние ключи, если требуется.
5. **Визуализировать модель** - создать ER-диаграмму с помощью Draw.io, разместить сущности, атрибуты и связи.

Пример: ER-диаграмма для учебного процесса

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

– Один студент может быть записан на несколько курсов (1:M).
– Один преподаватель может вести несколько курсов (1:M).
– Один курс могут посещать несколько студентов (M:N).

Порядок выполнения работы:

1. **Запуск Draw.io**
   - Перейдите на сайт [draw.io (diagrams.net)].
   - Выберите место для сохранения файла (на устройстве или в облаке).
2. **Создание новой диаграммы**
   - Выберите "Создать новую диаграмму".
   - В открывшемся окне выберите шаблон "Entity Relationship" или начните с пустого холста.
3. **Добавление сущностей**
   - На панели слева выберите фигуру "Прямоугольник" (или специальную форму для сущностей).
   - Разместите на холсте несколько сущностей (например, "Студент", "Курс", "Преподаватель").
4. **Добавление атрибутов**
   - Для каждой сущности добавьте овалы или текстовые блоки с атрибутами (например, для "Студент": ID, ФИО, Дата рождения).
   - Свяжите атрибуты с сущностями линиями или разместите их внутри прямоугольников.
5. **Установка связей**
   - Используйте стрелки или линии для отображения связей между сущностями (например, "Студент" - "записан на" - "Курс").
   - Укажите тип связи (1:1, 1:N, M:N), подписав линии.

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

Требования к отчету:

1. **Титульный лист:** стандартного образца.
2. **Ход выполнения заданий:**
   - **Текстовое описание** предметной области и перечень функциональных требований к БД.
   - **Таблица атрибутов** для каждой сущности (Имя, Тип, Описание, Ключ).
3. **Графическая часть:**
   - **Изображение ER-диаграммы**, на котором четко видны сущности, их атрибуты и типы связей.
4. **Описание манипуляций:**
   - Краткий перечень правил (бизнес-логика) по добавлению и изменению данных.
5. **Вывод:** оценка влияния качества проектирования на сложность последующей разработки ИИ-решений.

Тема практической работы № 2. Разработка логической модели базы данных на основе ER-диаграммы, объем часов: 4 часа.

У.9: Формировать требования к обработке данных и их извлечению.
У.10: Добавлять, удалять и изменять данные в базе данных.

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

Материальное обеспечение: LibreOffice Draw, Draw io

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

Варианты заданий:

Вариант	Наименование базы данных, описание

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

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

Краткие теоретические сведения:

Проектирование базы данных заключается в ее многоступенчатом описании с различной степенью детализации и формализации, в ходе которого производится уточнение и оптимизация структуры базы данных. Проектирование начинается с описания предметной области и задач информационной системы, идет к более абстрактному уровню логического описания данных и далее – к схеме физической (внутренней) модели базы

данных. Трем основным уровням моделирования системы – **концептуальному, логическому и физическому** соответствуют три последовательных этапа детализации описания объектов базы данных и их взаимосвязей.

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

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

Другими показателями, влияющими на выбор СУБД, являются:

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

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

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

На **физическом уровне** производиться выбор рациональной структуры хранения данных и методов доступа к ним, которые обеспечивает выбранная

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

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

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

   При проектировании таблиц рекомендуется руководствоваться следующими основными принципами:

   - каждая таблица должна содержать данные только на одну тему;

   - данные не должны дублироваться.

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

Форма представления результата:

Оформленная логическая схема базы данных.

Тема практической работы № 3. Нормализация базы данных: приведение таблиц к третьей нормальной форме (3НФ), объем часов: 4 часа.

У.9: Формировать требования к обработке данных и их извлечению.
У.10: Добавлять, удалять и изменять данные в базе данных.

Цель практической работы: Освоить нормализацию базы данных.

Материальное обеспечение: MySQL Workbench

Задание:

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

Краткие теоретические сведения:

Нормализация баз данных

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

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

1 Первая нормальная форма (1НФ)

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

Первичный ключ

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

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

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

Атомарность

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

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

Таблица 1 - Горизонтальное дублирование данных – плохая практика.

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

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

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

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

Таблица 2 - Множественные значения в одной ячейке.

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

Порядок записей не должен иметь значение

Правило: порядок записей таблицы не должен иметь значения.

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

2 Вторая нормальная форма

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

Избыточность данных

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

Может звучать немного заумно. А означает это то, что можно хранить в таблице только данные, которые напрямую связаны с ней и не имеют отношения к другой сущности. Следование второй нормальной форме – это вопрос нахождения данных, которые часто дублируются в записях таблицы и которые могут принадлежать другой сущности.

Таблица 3 - Дублирование данных среди записей в поле store.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3 Третья нормальная форма

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

Транзитивные зависимости

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

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

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

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

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

В данном случае устранение транзитивной зависимости из таблицы может быть достигнуто путем удаления полей города и провинции из таблицы и хранение их в отдельной таблице, содержащей почтовый код (первичный ключ), имя провинции и имя города. Получение комбинации почтовый код-город-провинция для целой страны может быть весьма нетривиальным занятием.

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

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

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

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

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

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

Форма представления результата:
Оформленная схема базы данных.

Тема практической работы № 4 Создание базы данных с использованием языка SQL (CREATE DATABASE, CREATE TABLE), объем часов: 4 часов.

У.9: Формировать требования к обработке данных и их извлечению.
У.10: Добавлять, удалять и изменять данные в базе данных.

Цель практической работы: освоить операции создания баз данных с использованием языка SQL.

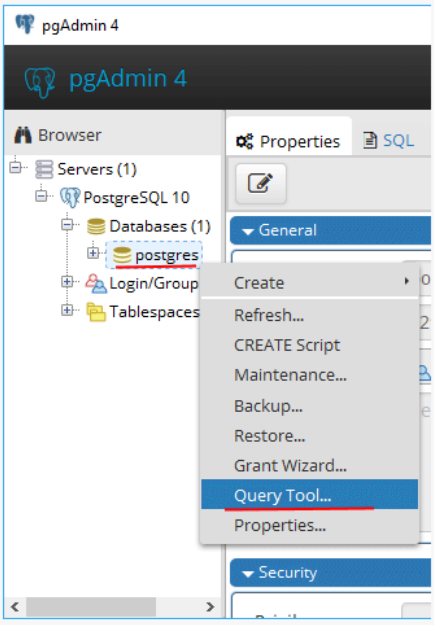
Материальное обеспечение: PostgreSQL pgAdmin, DBeaver

Задание:

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

Краткие теоретические сведения:

Создание базы данных в среде PostgreSQL



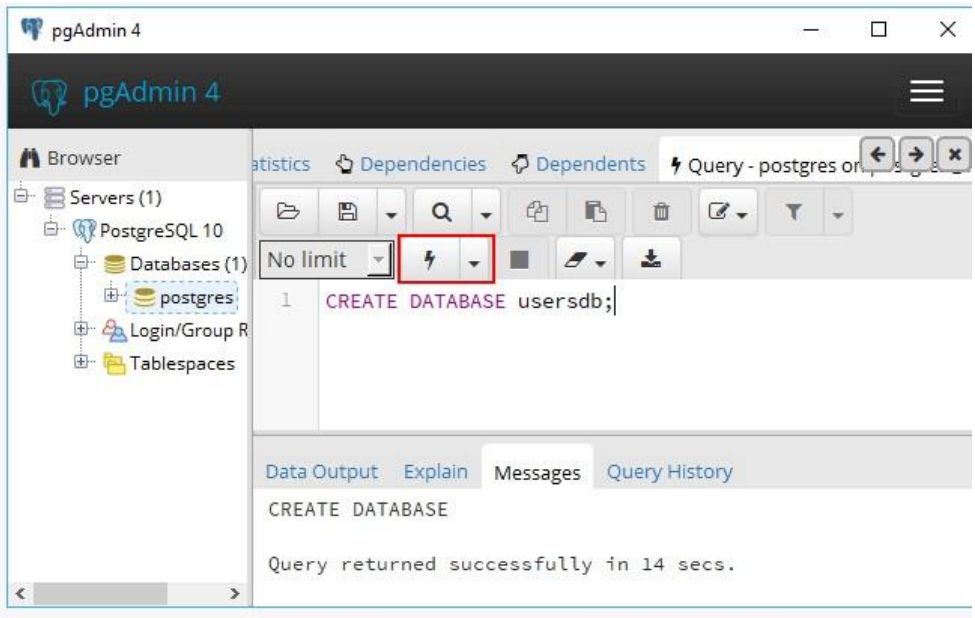
Для создания базы данных используется команда CREATE DATABASE, после которой указывается название базы данных.

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

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

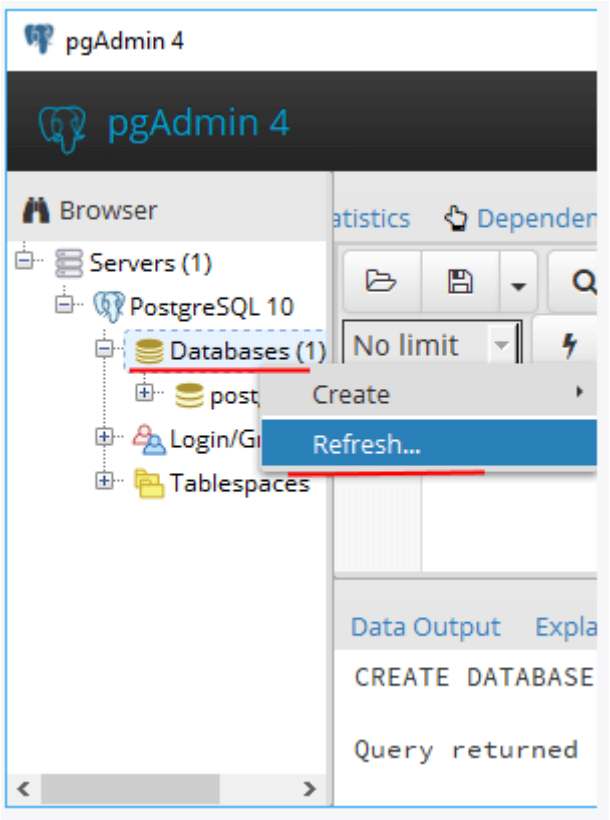
В появившемся меню выберем пункт Query Tool..., и в центральной части программы откроется поле для ввода кода SQL. В это поле введем следующий код:

```
CREATE DATABASE usersdb;
```

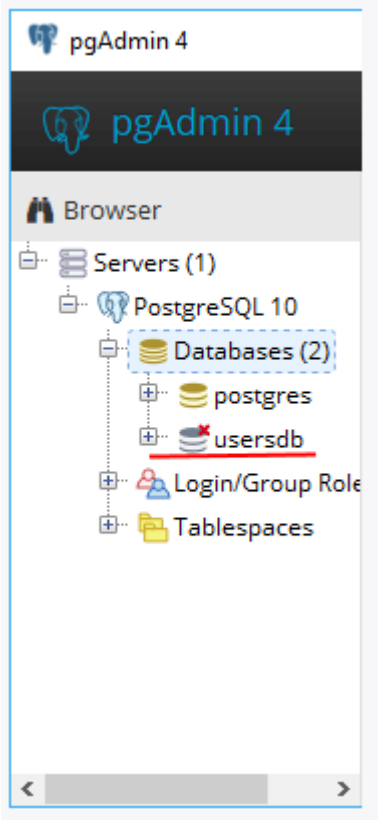


Для выполнения кода нажмем на значок молнии, и после этого будет создана база данных usersdb.

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



Произойдет обновление, и мы увидим созданную базу данных.



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

Удаление базы данных

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

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

Например, удаление базы данных usersdb:

DROP DATABASE usersdb;

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

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

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

Например, для Иванова Петра Сидоровича база данных будет иметь имя **storeIPS**: postgres=# **CREATE DATABASE storeIPS**;
CREATE DATABASE

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

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

SELECT datname FROM pg_database;

2 вариант: метакоманда **\l**

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

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

Синтаксис команды CREATE TABLE:

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

Для получения справки о синтаксисе SQL-команды:

\h CREATE TABLE

Способы ввода команд

- **Способ 1**

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

- **Способ 2**

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

CREATE TABLE

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

```
demo=# CREATE TABLE aircrafts ... \g
```

Прервать ввод команды можно клавишами Ctrl-C:

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

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

\d aircrafts

Таблица "public.aircrafts"

Колонка | Тип | Модификаторы

aircraft_code | character(3) | NOT NULL

model | text | NOT NULL

range | integer | NOT NULL

Индексы:

"aircrafts_pkey" PRIMARY KEY, btree (aircraft_code)

Ограничения-проверки:

"aircrafts_range_check" CHECK (range > 0)

Примечания:

public означает имя так называемой схемы.

- Для реализации первичного ключа (PRIMARY KEY) всегда автоматически создается индекс. В данном случае тип индекса — btree, т. е. B-дерево. Можно задать свои собственные имена для всех ограничений.

Удаление таблицы

Упрощенный синтаксис:

DROP TABLE имя_таблицы;

Например:

DROP TABLE aircrafts;

Форма представления результата:

Отчет по выполненной работе.

Тема практической работы № 5. Определение индексов для оптимизации запросов к базе данных, объем часов: 6 часов.

У.9: Формировать требования к обработке данных и их извлечению.
У.10: Добавлять, удалять и изменять данные в базе данных.

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

Материальное обеспечение: PostgreSQL pgAdmin, DBeaver

Задание:

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

Краткие теоретические сведения:

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

Индексы нужны для:

- ускорение запросов: индексы значительно ускоряют выполнение запросов, особенно на больших объемах данных;
- улучшение производительности: они помогают оптимизировать операции поиска, сортировки и фильтрации;

– поддержка уникальности: индексы могут использоваться для обеспечения уникальности значений в столбцах.

Создание индексов

Индексы создаются с помощью команды CREATE INDEX: CREATE INDEX имя_индекса ON имя_таблицы (имя_столбца);

Удаление индексов

Индексы можно удалить с помощью команды DROP INDEX: DROP INDEX имя_индекса;

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

1. **Подготовка среды**

Подключитесь к PostgreSQL через psql:
psql -U postgres

Создайте новую базу данных:
CREATE DATABASE performance_db;
\c performance_db

2. **Создание тестовой таблицы**

Создадим таблицу users с большим количеством записей:

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

Заполним её 100 000 строк случайными данными:

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

3. **Анализ производительности без индекса**

Выполним медленный запрос:

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

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

4. **Создание индекса**

Создадим индекс по полю username:

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

Теперь повторим тот же запрос:

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

Теперь используется Index Scan — гораздо быстрее!

5. **Составной индекс**

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

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

Пример использования:

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

6. **Уникальный индекс**

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

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

Попробуйте добавить дублирующий email:

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

Получите ошибку: duplicate key value violates unique constraint.

7. **Удаление индекса**

Если индекс больше не нужен:

```
DROP INDEX idx_username;
```

8. Анализ существующих индексов

Посмотреть все индексы в текущей базе данных:

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

Задание для самостоятельной работы

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

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

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

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

Форма представления результата:

Отчет по выполненной работе.

Тема практической работы №6. Создание базы данных с использованием языка SQL (CREATE DATABASE, CREATE TABLE), объем часов: 4 часов.

У.1: Производить идентификацию проблем, связанных с нормальным функционированием базы данных
У.2: Принимать решения по локализации проблем, связанных с нормальным функционированием базы данных
У.4: Осуществлять администрирование баз данных
У.10: Добавлять, удалять и изменять данные
У.11: Импорт и экспорт данных

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

Материальное обеспечение: PostgreSQL pgAdmin, DBeaver

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

Краткие теоретические сведения:

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

Основные команды DDL (Data Definition Language):

- CREATE DATABASE — создание новой базы данных.
- CREATE TABLE — создание новой таблицы.
- ALTER TABLE — изменение существующей таблицы.
- DROP TABLE / DATABASE — удаление таблицы или базы данных.

Команда CREATE DATABASE

Используется для создания новой базы данных:

```
CREATE DATABASE имя_базы;
```

Команда CREATE TABLE

Синтаксис:

```
CREATE TABLE имя_таблицы (
    столбец1 тип_данных ограничения,
```

столбец2 тип_данных ограничения,
...
);

Типы данных в PostgreSQL:

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

Ограничения (Constraints):

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

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

1. **Подключение к PostgreSQL**

Откройте psql или клиент, например, pgAdmin .

2. **Создание новой базы данных**

Подключитесь через psql:
psql -U postgres
Создайте новую базу данных:
-- Создаем базу данных
CREATE DATABASE school_db;
-- Подключаемся к ней
\c school_db
Теперь все дальнейшие команды будут выполняться в контексте этой БД.

3. Создание таблиц

Пример 1: Таблица students — студенты

```
CREATE TABLE students(

        student_id SERIAL
        PRIMARY KEY,
        first_name
        VARCHAR(50) NOT
        NULL, last_name
        VARCHAR(50) NOT
        NULL, birth_date
        DATE,

        email VARCHAR(100) UNIQUE,

        is_active BOOLEAN DEFAULT TRUE

    );
```

Комментарий:

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

Пример 2: Таблица courses — курсы

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

Комментарий:

Поле duration_weeks должно быть положительным числом благодаря CHECK.

Пример 3: Таблица enrollments — зачисления студентов на курсы

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

Комментарий:

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

4. **Вставка тестовых данных**

Добавим несколько записей:

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

-- Добавляем курсы
INSERT INTO courses (course_name, description,
duration_weeks, start_date)
VALUES ('Введение в программирование', 'Основы
программирования на Python', 12, '2025-02-01'),
('Английский для IT', 'Развитие технической грамотности
на английском', 16, '2025-02-05');
-- Регистрируем зачисления
INSERT INTO enrollments
(student_id, course_id) VALUES (1,
1), (1, 2), (2, 1);
```

5. **Выполнение простых запросов**

Посмотреть всех студентов:

```
SELECT * FROM students;
```

Посмотреть активных студентов:

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

Посмотреть оценки студентов по курсам:

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

Задание для самостоятельной работы

1. Создайте новую таблицу teachers со следующими полями:
- teacher_id — уникальный ID (первичный ключ),
- full_name — ФИО преподавателя,
- subject — предмет,
- hire_date — дата найма.
2. Добавьте связь между courses и teachers (например, один преподаватель может вести один курс).
3. Создайте таблицу grades_log, которая будет хранить историю изменения оценок студентов:
- log_id,
- enrollment_id,
- old_grade,
- new_grade,
- changed_at TIMESTAMP DEFAULT NOW().
4. Напишите триггер, который будет добавлять запись в grades_log при обновлении оценки в enrollments

Форма представления результата:

Отчет по выполненной работе.

Тема практической работы №7. Реализация ограничений целостности (PRIMARY KEY, FOREIGN KEY, UNIQUE) в таблицах базы данных, объем часов: 4 часа.

У.1: Производить идентификацию проблем, связанных с нормальным функционированием базы данных
У.2: Принимать решения по локализации проблем, связанных с нормальным функционированием базы данных
У.4: Осуществлять администрирование баз данных
У.10: Добавлять, удалять и изменять данные
У.11: Импорт и экспорт данных

Цель практической работы: Научиться реализовывать и использовать основные механизмы обеспечения целостности данных в PostgreSQL.

Материальное обеспечение: PostgreSQL pgAdmin, DBeaver

Задание:
Реализовать и использовать основные механизмы обеспечения целостности данных в PostgreSQL.

Краткие теоретические сведения:

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

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

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

Ограничения столбцов и таблиц

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

PRIMARY KEY

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

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

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

Установка первичного ключа на уровне таблицы:

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

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

```
CREATE TABLE OrderLines (
    OrderId INTEGER,
    ProductId INTEGER,
    Quantity INTEGER,
```

```
    Price MONEY,
    PRIMARY KEY(OrderId, ProductId)
);
```

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

UNIQUE

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

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

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

Также мы можем определить этот атрибут на уровне таблицы:

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

Или так:

```
CREATE TABLE Customers (
    Id SERIAL PRIMARY KEY,
```

```
    FirstName CHARACTER
    VARYING(20),
    LastName CHARACTER
    VARYING(20), Email
    CHARACTER
    VARYING(30), Phone
    CHARACTER
    VARYING(30),
    Age
    INTEGER,
    UNIQUE(E
    mail),
    UNIQUE(Phone)
);
```

NULL и NOT NULL

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

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

DEFAULT

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

```
CREATE
TABLE
```

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

Здесь для столбца Age предусмотрено значение по умолчанию 18.

CHECK

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

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

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

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

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

```
CREATE
```

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

Оператор CONSTRAINT. Установка имени ограничений.

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

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

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

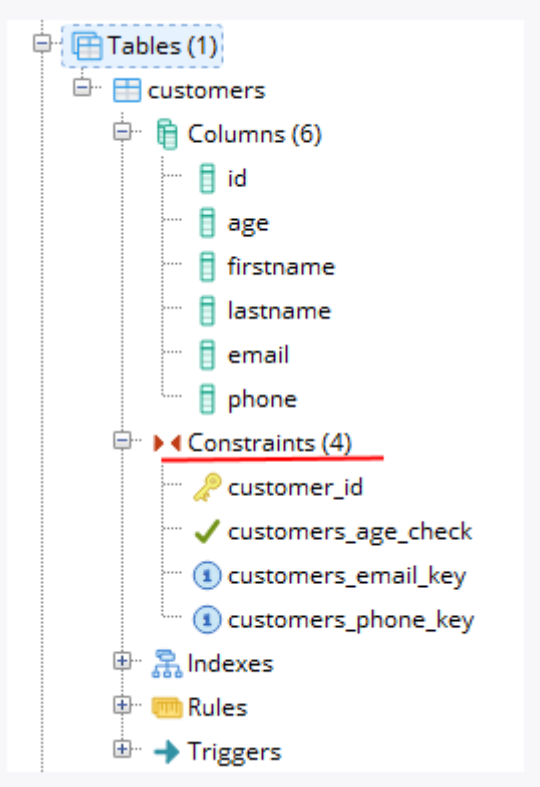
В принципе необязательно задавать имена ограничений, при установке соответствующих атрибутов SQL Server автоматически определяет их имена. Но, зная имя ограничения, мы можем к нему обращаться, например, для его удаления.

И также можно задать все имена ограничений через атрибуты таблицы:

```
CREATE
TABLE
Customers
(
    Id SERIAL, Age INTEGER,
```

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

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



Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

1. Подготовка среды

Подключитесь к PostgreSQL через psql:

psql -U postgres

Создайте новую базу данных:

CREATE DATABASE integrity_db;

\c integrity_db

2. Создание таблиц с ограничениями Таблица users — пользователи системы

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

Комментарий:

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

Таблица orders — заказы пользователей

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

Комментарий:

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

Таблица emails — дополнительная проверка UNIQUE

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

3. Попытка нарушить ограничения

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

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

- Ошибка: duplicate key value violates unique constraint.

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

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

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

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

- Ошибка: duplicate key value violates unique constraint.

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

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

---

4. Работа с существующими таблицами

Добавление ограничения после создания таблицы

Можно добавить ограничение UNIQUE к уже существующей таблице:

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

Удаление ограничения

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

Переименование ограничения

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

---

5. Использование составного PRIMARY KEY

Иногда требуется использовать несколько полей в качестве первичного ключа.

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

Теперь комбинация user_id и role должна быть уникальной.

Задание для самостоятельной работы

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

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

Установите ограничения:

- employee_id — PRIMARY KEY,

- full_name — UNIQUE и NOT NULL,
- salary — CHECK больше 0.

4. Попробуйте вручную нарушить каждое из этих ограничений и зафиксируйте возникающие ошибки.
5. Напишите запросы, которые:

- Выводят всех сотрудников с зарплатой выше средней,
- Выводят количество заказов по каждому продукту.

Форма представления результата:
Отчет по выполненной работе.

Тема практической работы №8 Написание и выполнение SQL-запросов для добавления, изменения и удаления данных (INSERT, UPDATE, DELETE), объем часов: 4 часа.

У.1: Производить идентификацию проблем, связанных с нормальным функционированием базы данных
У.2: Принимать решения по локализации проблем, связанных с нормальным функционированием базы данных
У.4: Осуществлять администрирование баз данных
У.10: Добавлять, удалять и изменять данные
У.11: Импорт и экспорт данных

Цель практической работы: освоить операции создания баз данных с использованием языка SQL.

Материальное обеспечение: PostgreSQL pgAdmin, DBeaver

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

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

Используя метод нормальных форм спроектировать базу данных. Процесс проектирования должен быть представлен в форме отчета, показан процесс

формирования таблиц под выделенные сущности и их последовательная нормализация методом нормальных форм

Форма представления результата:
Оформленная схема базы данных.

Тема практической работы №9. Настройка индексов для оптимизации производительности запросов (CREATE INDEX), объем часов: 4 час.

У.1: Производить идентификацию проблем, связанных с нормальным функционированием базы данных
У.2: Принимать решения по локализации проблем, связанных с нормальным функционированием базы данных
У.4: Осуществлять администрирование баз данных
У.10: Добавлять, удалять и изменять данные
У.11: Импорт и экспорт данных

Цель практической работы: освоить операции создания баз данных с использованием языка SQL.

Материальное обеспечение: PostgreSQL pgAdmin, DBeaver

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

Краткие теоретические сведения:
Основные DML-команды SQL:

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

1. **Команда INSERT**

Добавляет новые данные в таблицу.

Синтаксис:

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

Можно добавить несколько строк за раз:

```
INSERT INTO имя_таблицы (столбец1,
столбец2) VALUES
(значение1,
значение2),
(значение3,
значение4);
```

2. Команда UPDATE

Изменяет значения существующих строк.

Синтаксис:

```
UPDATE имя_таблицы
SET столбец1 = значение1, столбец2 =
значение2 WHERE условие;
```

Важно: всегда используйте WHERE, иначе будут изменены все строки в таблице!

3. Команда DELETE

Удаляет строки из таблицы.

Синтаксис:

```
DELETE FROM
имя_таблицы
WHERE условие;
```

Важно: без WHERE будут удалены все данные из таблицы.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

1. Подготовка среды

Подключитесь к PostgreSQL через psql или клиент (например, pgAdmin):

```
psql -U postgres
```

Создайте новую базу данных и подключитесь к ней:

```
CREATE DATABASE practice_db;
\c practice_db
```

2. Создание тестовой таблицы

Создадим таблицу students:

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

3. Добавление данных — INSERT

Добавим несколько студентов:

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

Проверим результат:

```
SELECT * FROM students;
```

4. Изменение данных — UPDATE

Обновим возраст одного студента:

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

Теперь обновим email:

```
UPDATE students
SET email = 'ivan_new@example.com'
WHERE student_id = 1;
```

5. Удаление данных — DELETE

Удалим студента по ID:

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

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

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

6. Работа с несколькими таблицами (связанные данные)

Создадим таблицу courses:

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

Создадим связь через внешний ключ:

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

Добавим курсы:

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

Запишем студентов на курсы:

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

Обновим дату зачисления:

```
UPDATE enrollments
SET enrollment_date = '2025-
```

```
03-01' WHERE enrollment_id =
1;
```

Удалим запись о зачислении:

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

Задание для самостоятельной работы

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

Добавьте туда не менее 5 записей.

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

Форма представления результата:

Отчет по выполненной работе.

Тема практической работы №10. Реализация хранимых процедур и триггеров для автоматизации работы с базой данных, объем часов: 6 часов.

У.1: Производить идентификацию проблем, связанных с нормальным функционированием базы данных

У.2: Принимать решения по локализации проблем, связанных с нормальным функционированием базы данных

У.4: Осуществлять администрирование баз данных

У.10: Добавлять, удалять и изменять данные

У.11: Импорт и экспорт данных

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

Материальное обеспечение: PostgreSQL pgAdmin, DBeaver

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

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

– повторное использование кода;

– автоматизация логики;

– уменьшение сетевого трафика между приложением и базой данных.

– повышение производительности за счет предварительной компиляции.

Синтаксис

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

Описание

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

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

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

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

Параметры

имя

Имя создаваемой процедуры (возможно, дополненное схемой).

режим_аргумента

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

имя_аргумента

Имя аргумента.

тип_аргумента

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

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

выражение_по_умолчанию

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

имя_языка

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

параметр_конфигурации значение

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

определение

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

объектный_файл, объектный_символ

Эта форма предложения AS применяется для динамически загружаемых процедур на языке C, когда имя процедуры в коде C не совпадает с именем процедуры в SQL. Строка объектный_файл задаёт имя файла, содержащего скомпилированную процедуру на C (данная команда воспринимает эту строку так же, как и LOAD). Строка объектный_символ задаёт символ скомпонованной процедуры, то есть имя процедуры в исходном коде на языке C. Если объектный символ не указан, предполагается, что он совпадает с именем определяемой SQL-процедуры.

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

тело_sql

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

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

Оно определяется подобно телу, задаваемому строковой константой (см. определение выше), но есть и некоторые различия. Эта форма работает только с функциями в стиле LANGUAGE SQL, тогда как форма со строковой константой поддерживается для всех языков. Она разбирается во время

определения процедуры, тогда как форма со строковой константой — во время выполнения; как следствие, эта форма не поддерживает полиморфные типы аргументов и другие конструкции, которые нельзя обработать во время определения процедуры. С данной формой отслеживаются зависимости процедуры от объектов, используемых в её теле, так что команда DROP ... CASCADE выполнится корректно, тогда как в случае определения тела в строковой константе после такого удаления могут остаться неполноценные процедуры. Наконец, данная форма в большей степени соответствует стандарту SQL и совместима с другими реализациями SQL.

Чтобы выполнить процедуру, воспользуйтесь командой CALL.

Примеры

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

или

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

и пример вызова:

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

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

Где применяются:

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

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

1. Подготовка среды

Подключитесь к PostgreSQL через psql:

```
psql -U postgres
```

Создайте новую базу данных:

```
CREATE DATABASE automation_db;
\c automation_db
```

2. Создание тестовой таблицы

Создадим таблицу employees:

```
CREATE TABLE employees (
    employee_id SERIAL PRIMARY
    KEY, full_name
    VARCHAR(100),
    salary NUMERIC(10,
    2), department
    VARCHAR(50)
);
```

Добавим несколько записей:

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

3. Создание простой функции

Напишем функцию, которая увеличивает зарплату сотрудника на заданное количество процентов:

```
CREATE OR REPLACE FUNCTION
    increase_salary( emp_id INT,
    percent NUMERIC
)
RETURNS
VOID AS $$
BEGIN
    UPDATE employees
    SET salary = salary * (1 +
```

```
    percent / 100) WHERE employee_id
    = emp_id;
END;
$$ LANGUAGE plpgsql;
```

RETURNS VOID — функция ничего не возвращает. Вызов функции:

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

4. Функция с возвратом значения

Создадим функцию, возвращающую среднюю зарплату по отделу:

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

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

Вызов функции:

```
SELECT avg_salary_by_department('IT');
```

5. Создание триггера

Создадим таблицу для хранения истории изменений зарплат:

```
CREATE TABLE salary_log (
    log_id SERIAL PRIMARY
    KEY,
    employee_id INT,
```

```
    old_salary NUMERIC(10,2),
    new_salary NUMERIC(10,2),
    changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Теперь создадим функцию-обработчик для триггера:

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

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

Теперь привяжем её к таблице employees как триггер:

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

Протестируем:

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

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

6. Дополнительный пример: запрет обновления имени

Создадим триггер, который запрещает обновление поля full_name :

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

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

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

Проверяем:

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

Задание для самостоятельной работы

1. Создайте таблицу users со следующими полями:
   - user_id,
   - username,
   - email,
   - created_at.
2. Напишите функцию, которая добавляет нового пользователя, если email уникален.
3. Создайте таблицу user_logs, куда будут записываться действия (добавление, удаление, обновление).
4. Настройте триггеры:
   - при добавлении пользователя — записывается событие в user_logs;

- при удалении — тоже записывается событие;
- при обновлении `email` — логируется старый и новый `email`;
- протестируйте работу всех функций и триггеров.

Форма представления результата:
Отчет по выполненной работе.

Тема практической работы №11. Настройка шифрования данных в MySQL с использованием встроенных функций (например, AES_ENCRYPT, AES_DECRYPT), объем часов: 4 часа.

У.5: Настраивать политики безопасности при работе с сервером баз данных
У.6: Давать независимую оценку уровня безопасности
У.8: Разрабатывать перечень рекомендаций по дальнейшей эксплуатации БД с максимальной защитой хранящейся информации
У.3: Документировать внештатные ситуации, связанные с нормальным функционированием базы данных
У.7: Производить регламентное обновление программного обеспечения

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

Материальное обеспечение: MySQL

Задание:

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

Краткие теоретические сведения:

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

Важность шифрования данные в базе данных

Причина	**Описание**

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

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

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

Шаги по настройке шифрования

1. Подготовка:

– создайте таблицу или столбцы, в которых будут храниться данные, подлежащие шифрованию;

– решите, какие данные нужно зашифровать;

– выберете длину ключа aes (обычно 128 или 256 бит) в зависимости от требований безопасности.

2. Шифрование данных при вставке:

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

Например:

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

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

3. Извлечение и расшифровка данных:

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

Например:

```
SELECT username, AES_DECRYPT(encrypted_password,
```

```
'my_secret_key') AS decrypted_password FROM users WHERE
username = 'john_doe';
```

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

4. Обеспечение безопасности ключа:

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

Пример настройки шифрования для столбца

Создайте таблицу с зашифрованным столбцом:

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

Вставьте данные:

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

Извлеките и расшифруйте данные:

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

Важные замечания:

- AES_ENCRYPT() и AES_DECRYPT() возвращают двоичные строки (BLOB). Если вы хотите сохранить расшифрованные данные в текстовом столбце, необходимо преобразовать их в текстовый формат.
- Для обеспечения безопасности, рекомендуется использовать AES_ENCRYPT() и AES_DECRYPT() при выполнении операций с

данными, а не хранить зашифрованные данные прямо в столбце. Это поможет защитить данные от несанкционированного доступа, если кто-то получит доступ к базе данных.

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

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

1. Подготовка среды

Подключитесь к вашему серверу MySQL:

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

Создайте тестовую базу данных:

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

2. Простейший пример использования AES_ENCRYPT / AES_DECRYPT

Проверим работу функций шифрования:

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

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

3. Создание таблицы с зашифрованными полями

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

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

VARBINARY используется для хранения бинарных данных (результат шифрования)

4. Вставка зашифрованных данных

Добавим пользователя:

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

5. Получение и расшифровка данных

Выберем пользователя и расшифруем его пароль:

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

Результат будет в виде строки, если ключ совпадает.

6. Использование шифрования для полей типа TEXT

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

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

Вставляем данные:

```
INSERT INTO personal_data (full_name, phone_number)
VALUES
(AES_ENCRYPT('Иван   Петров', 'data_key_123'),
AES_ENCRYPT('+79123456789', 'data_key_123'));
```

Получаем обратно:

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

7. Хранение ключа шифрования безопасно

Важные рекомендации:

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

Использование ключа в приложении (на Python):

```
import mysql.connector

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

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

Задание для самостоятельной работы

1. Создайте таблицу employees со следующими полями:
   - employee_id,
   - full_name (зашифрованное),
   - salary (зашифрованное),
   - passport_number (зашифрованное).
2. Добавьте несколько записей сотрудников с реальными данными.
3. Напишите запрос, который выводит расшифрованные данные только для пользователей с зарплатой выше 100 000.
4. Реализуйте простое приложение (на Python), которое:
   - добавляет новых сотрудников в таблицу с шифрованием данных;

- выводит список сотрудников с расшифровкой.

5. Настройте систему аудита (логирование) всех операций добавления/обновления/удаления зашифрованных данных.

Форма представления результата:
Оформленная схема базы данных.

Тема практической работы №12. Реализация ролевой модели безопасности в PostgreSQL (создание ролей и управление их правами), объем часов: 4 часа.

У.5: Настраивать политики безопасности при работе с сервером баз данных
У.6: Давать независимую оценку уровня безопасности
У.8: Разрабатывать перечень рекомендаций по дальнейшей эксплуатации БД с максимальной защитой хранящейся информации
У.3: Документировать внештатные ситуации, связанные с нормальным функционированием базы данных
У.7: Производить регламентное обновление программного обеспечения

Цель практической работы: Изучить ролевую модель доступа в PostgreSQL, научиться разграничивать полномочия пользователей через создание групповых и индивидуальных ролей, а также настраивать безопасную среду для работы приложений искусственного интеллекта.

Материальное обеспечение: Персональный компьютер, установленная СУБД PostgreSQL, графический интерфейс pgAdmin или терминал psql.

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

1. Спроектировать и реализовать в PostgreSQL трехуровневую ролевую модель: «Администратор», «Разработчик ИИ» (чтение/запись), «Аналитик» (только чтение).
2. Настроить права доступа к конкретной базе данных и её схемам, используя принципы минимальных привилегий.

3. Провести аудит безопасности созданных ролей и проверить систему на устойчивость к несанкционированным операциям.
4. Задокументировать действия при возникновении ошибок доступа и составить регламент защиты данных.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **Создание ролей:** Используйте команду CREATE ROLE. Помните, что в PostgreSQL роли с правом входа создаются с атрибутом LOGIN. Для групп (ролей без права входа) используйте NOLOGIN.
- **Иерархия прав:** Реализуйте наследование прав. Сначала назначьте права групповой роли (например, GRANT SELECT ON ALL TABLES...), а затем включите в неё конкретных пользователей командой GRANT group_role TO user_name.
- **Безопасность схем:** По умолчанию все пользователи имеют доступ к схеме public. Для обеспечения максимальной защиты (У.8) рекомендуется отозвать эти права: REVOKE ALL ON SCHEMA public FROM PUBLIC;.
- **Документирование внештатных ситуаций:** Если пользователь не может выполнить запрос из-за отсутствия прав, зафиксируйте текст ошибки (ERROR: permission denied for table...) и опишите способ её локализации без предоставления избыточных прав суперпользователя.
- **Регламент обновления:** В рамках У.7 опишите процесс переназначения прав (например, использование ALTER DEFAULT PRIVILEGES), который гарантирует сохранение политики безопасности при добавлении новых таблиц в процессе обновления ПО.

Форма представления результата:

1. **Сводная таблица (матрица доступа)** со списком созданных ролей и их полномочий.
2. **Листинг SQL-кода**, содержащий команды создания ролей и распределения прав.
3. **Отчет по аудиту**: подтверждение того, что роль «Аналитик» не может изменять данные (скриншот ошибки).
4. **Перечень рекомендаций** по защите базы данных при её эксплуатации в промышленных ИИ-системах.

Тема практической работы №13. Настройка аудита действий пользователей в Microsoft SQL Server, объем часов: 4 часа.

У.5: Настраивать политики безопасности при работе с сервером баз данных
У.6: Давать независимую оценку уровня безопасности
У.8: Разрабатывать перечень рекомендаций по дальнейшей эксплуатации БД с максимальной защитой хранящейся информации
У.3: Документировать внештатные ситуации, связанные с нормальным функционированием базы данных
У.7: Производить регламентное обновление программного обеспечения

Цель практической работы: Научиться проектировать и внедрять системы тотального контроля действий пользователей в MS SQL Server, обеспечивать фиксацию критических операций и проводить аудит безопасности данных.

Материальное обеспечение: ПК, СУБД Microsoft SQL Server, среда SQL Server Management Studio (SSMS).

Задание:

1. Создать объект аудита сервера (Server Audit) с хранением логов в файловой системе.
2. Сконфигурировать аудит спецификации базы данных для отслеживания операций изменения данных (DML) и изменения структуры (DDL).
3. Провести симуляцию несанкционированного доступа к таблицам и зафиксировать это в журнале.
4. Проанализировать результаты аудита и сформировать отчет о состоянии безопасности.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **Создание аудита сервера:** В SSMS перейдите в раздел Security -> Audits. При настройке укажите путь к папке для логов. Установите параметр On Audit Log Failure в значение Fail Operation, если требуется максимальная защита (запрет действий при невозможности их записи).
- **Настройка спецификации:** В конкретной базе данных создайте Database Audit Specification. Выберите действия, которые критичны для ИИ-проектов: SCHEMA_OBJECT_ACCESS_GROUP (доступ к данным) и SCHEMA_OBJECT_CHANGE_GROUP (изменение моделей данных).

- **Идентификация угроз:** Попробуйте выполнить запрос SELECT или DROP от лица пользователя с ограниченными правами. Проверьте, отобразилось ли это событие в просмотре логов (View Audit Logs).
- **Документирование внештатных ситуаций:** Если файл аудита переполнился или папка недоступна, зафиксируйте ошибку сервера и опишите способ восстановления работоспособности службы.
- **Регламент эксплуатации:** Подготовьте рекомендации по архивации файлов аудита (.sqlaudit), чтобы обеспечить требования регуляторов по хранению истории доступа к персональным данным, используемым в ИИ.

Форма представления результата:

1. **Скриншоты** настроенных объектов Server Audit и Database Audit Specification.
2. **Листинг (фрагмент лога)**, подтверждающий фиксацию тестовых обращений к базе.
3. **Аналитическая записка** с независимой оценкой уровня безопасности системы.
4. **Рекомендации** по настройке аудита при проведении регламентного обновления программного обеспечения.

Тема практической работы №14. Конфигурация шифрования трафика между клиентом и сервером базы данных (TLS/SSL), объем часов: 4 час.

У.5: Настраивать политики безопасности при работе с сервером баз данных
У.6: Давать независимую оценку уровня безопасности
У.8: Разрабатывать перечень рекомендаций по дальнейшей эксплуатации БД с максимальной защитой хранящейся информации
У.3: Документировать внештатные ситуации, связанные с нормальным функционированием базы данных
У.7: Производить регламентное обновление программного обеспечения

Цель практической работы: Изучить механизмы криптографической защиты данных при их транспортировке, научиться генерировать сертификаты и настраивать принудительное шифрование соединений в современных СУБД.

Материальное обеспечение: ПК, СУБД (MySQL или PostgreSQL), утилита OpenSSL, клиентское ПО (DBeaver или консольный клиент).

Задание:

1. Выполнить генерацию самоподписанных SSL-сертификатов для сервера и клиента.
2. Сконфигурировать сервер базы данных для поддержки протокола TLS.
3. Настроить учетную запись пользователя с требованием обязательного шифрования (REQUIRE SSL).
4. Проверить статус шифрования сессии и провести аудит защищенности канала передачи данных.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **Работа с сертификатами:** Используйте OpenSSL для создания цепочки: CA (корневой сертификат), Server Certificate и Client Certificate. Помните, что срок действия сертификатов должен учитываться при **регламентном обновлении ПО** .
- **Настройка сервера:** В конфигурационном файле (my.ini или postgresql.conf) укажите пути к файлам .crt и .key. После перезагрузки службы убедитесь, что сервер «видит» сертификаты, проверив системные переменные (have_ssl).
- **Принудительное шифрование:** Для реализации политики максимальной защиты настройте пользователя так, чтобы вход без SSL был невозможен. Это предотвратит атаки типа «человек посередине» (MITM).
- **Документирование внештатных ситуаций:** Имитируйте попытку подключения с просроченным сертификатом или по незащищенному каналу. Зафиксируйте ошибку соединения (SSL connection error) и опишите порядок действий для восстановления связи.
- **Оценка безопасности:** С помощью утилиты Wireshark или встроенных команд СУБД убедитесь, что передаваемый текст SQL-запросов зашифрован и не читается в открытом виде.

Форма представления результата:

1. **Набор сгенерированных файлов** сертификатов (копии).
2. **Скриншоты** конфигурационных файлов сервера с внесенными изменениями.
3. **Скриншот статуса соединения**, подтверждающий использование конкретного шифра (например, TLS_AES_256_GCM).
4. **Рекомендации** по управлению жизненным циклом SSL-сертификатов в инфраструктуре ИИ.

Тема практической работы №15. Организация резервного копирования с шифрованием в Oracle Database, объем часов: 6 часов.

У.5: Настраивать политики безопасности при работе с сервером баз данных
У.6: Давать независимую оценку уровня безопасности
У.8: Разрабатывать перечень рекомендаций по дальнейшей эксплуатации БД с максимальной защитой хранящейся информации
У.3: Документировать внештатные ситуации, связанные с нормальным функционированием базы данных
У.7: Производить регламентное обновление программного обеспечения

Цель практической работы: Изучить технологии защиты резервных копий, научиться настраивать шифрование в среде RMAN (Recovery Manager) с использованием кошельков (Wallets) и обеспечивать целостность данных при их хранении вне СУБД.

Материальное обеспечение: ПК, СУБД Oracle Database, утилита RMAN.

Задание:

1. Создать и настроить хранилище ключей (Keystore/Wallet) для управления паролями шифрования.
2. Сконфигурировать параметры RMAN для выполнения резервного копирования с использованием алгоритма AES256.
3. Выполнить тестовое шифрованное резервирование базы данных и проверить статус зашифрованных наборов (Backupsets).
4. Отработать сценарий восстановления из зашифрованного бэкапа и провести аудит защищенности архивов.

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **Настройка Keystore:** Перед шифрованием необходимо создать программный кошелек. Используйте команду ADMINISTER KEY MANAGEMENT CREATE KEYSTORE.... Помните, что потеря пароля от кошелька делает восстановление данных невозможным — это критическая **внештатная ситуация**.
- **Конфигурация RMAN:** Настройте шифрование для сессии или на постоянной основе: CONFIGURE ENCRYPTION FOR DATABASE ON;. Выберите метод шифрования (Transparent или Password mode).

- **Регламентное обновление:** В рамках опишите процедуру ротации ключей шифрования, которая должна проводиться регулярно при обновлении программных компонентов системы.
- **Оценка безопасности:** Попробуйте прочитать содержимое файла бэкапа через текстовый редактор. Убедитесь, что данные нечитаемы. Выполните команду LIST BACKUP;, чтобы убедиться, что в поле Encryption стоит статус YES.
- **Рекомендации:** Сформируйте рекомендации по раздельному хранению файлов бэкапов и файлов кошелька (Wallet) для исключения возможности расшифровки данных злоумышленником при получении доступа к серверу.

Форма представления результата:

1. **Скриншоты** процесса создания и открытия Keystore в SQL*Plus.
2. **Листинг команд RMAN**, использованных для настройки и запуска шифрованного копирования.
3. **Отчет о проверке**: подтверждение успешного восстановления данных из зашифрованной копии.
4. **Регламент** по безопасному хранению ключей шифрования для обеспечения максимальной защиты информации.

Тема практической работы №16. Установка и настройка векторной базы данных (например, Milvus, Pinecone или Weaviate), объем часов: 4 часа.

У.9 Производить формирование требований к обработке данных и их извлечению
У.11 Производить операции по импорту и экспорту данных в различных форматах
У.4 Осуществлять основные функции по администрированию баз данных
У.5 Настраивать политики безопасности при работе с сервером баз данных

Цель практической работы: Изучить архитектуру векторных систем управления данными, научиться развертывать векторную СУБД в контейнеризированной среде и настраивать параметры извлечения данных на основе семантического поиска.

Материальное обеспечение: ПК, Docker Desktop, Python (с библиотеками pymilvus или weaviate-client).

Задание:

1. Сформировать технические требования к структуре коллекции для хранения текстовых эмбеддингов (размерность вектора, тип метрики расстояния).
2. Выполнить установку и запуск векторной СУБД с использованием Docker Compose.
3. Осуществить импорт тестового набора векторных данных и выполнить поиск по сходству (Similarity Search).
4. Настроить базовые параметры безопасности (аутентификация и ограничение прав доступа к API).

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **Формирование требований:** При проектировании коллекции важно выбрать правильную метрику расстояния (L2 — евклидово расстояние, IP — внутреннее произведение или Cosine — косинусное сходство). От этого зависит точность извлечения данных ИИ-моделью.
- **Администрирование (Docker):** Используйте готовый YAML-конфигуратор. После запуска проверьте статус контейнеров командой docker ps. Убедитесь, что порты (например, 19530 для Milvus) доступны для внешних подключений.
- **Импорт/Экспорт:** Подготовьте данные в формате JSON или CSV. Перед импортом векторы должны быть нормализованы. Отработайте экспорт результатов поиска в текстовый формат для анализа качества выдачи.
- **Безопасность:** Векторные БД часто поставляются с отключенной по умолчанию аутентификацией. Включите режим RBAC (Role-Based Access Control) и создайте пользователя с правами только на чтение для клиентского приложения ИИ.

Форма представления результата:

1. **Техническая спецификация** коллекции (описание полей, размерность векторов, выбранный индекс).
2. **Скриншот консоли** со статусом запущенных сервисов векторной СУБД.
3. **Листинг Python-скрипта**, демонстрирующий импорт данных и результат выполнения семантического поиска (Top-K результатов).

4. **Отчет о настройке безопасности**: описание созданных ролей и методов аутентификации.

Тема практической работы №17. Создание и управление коллекциями данных в векторной базе (создание индексов и добавление векторов), объем часов: 4 часа.

У.9 Производить формирование требований к обработке данных и их извлечению
У.11 Производить операции по импорту и экспорту данных в различных форматах
У.4 Осуществлять основные функции по администрированию баз данных
У.5 Настраивать политики безопасности при работе с сервером баз данных

Цель практической работы: Научиться проектировать схемы коллекций в векторных СУБД, освоить алгоритмы индексирования высокоразмерных данных и приобрести навыки управления жизненным циклом векторов.

Материальное обеспечение: ПК, развернутая векторная СУБД (Milvus, Weaviate или Qdrant), Python, библиотека NumPy.

Краткие теоретические сведения:
Создание коллекции в векторной среде (на примере Milvus/Weaviate)
В векторных СУБД вместо таблиц используются **коллекции**. Для создания коллекции необходимо описать её схему: указать, какие поля будут хранить обычные данные (ID, текст, категория), а какое — вектор (эмбеддинг).
Для выполнения операций используется программный интерфейс (Python SDK) или графические консоли (например, Attu для Milvus). Чтобы создать коллекцию через код, необходимо определить поля и их типы.

Пример создания схемы коллекции на Python:

```
python
from pymysql import schema
# Определение полей: ID и векторное поле размерностью
768
fields = [
    FieldSchema(name="id", dtype=DataType.INT64,
is_primary=True, auto_id=True),
```

```
    FieldSchema(name="embeddings",
dtype=DataType.FLOAT_VECTOR, dim=768)
]
schema = CollectionSchema(fields, "Коллекция для
хранения эмбеддингов ИИ")
collection = Collection("ai_data_collection", schema)
Используйте код с осторожностью.
```

Создание индексов

Без индекса поиск будет идти методом полного перебора, что крайне медленно. Индекс «упорядочивает» векторы в пространстве. Самым популярным для ИИ является индекс **HNSW** (иерархический граф).

Для создания индекса в консоли или коде указываются параметры (метрика и тип):

```
python
index_params = {
    "metric_type": "L2",
    "index_type": "HNSW",
    "params": {"M": 8, "efConstruction": 64}
}
collection.create_index(field_name="embeddings",
index_params=index_params)
Используйте код с осторожностью.
```

После выполнения команды база данных построит граф связей, что позволит мгновенно находить «ближайших соседей» для поискового запроса.

Удаление коллекции

Для удаления коллекции и всех накопленных векторов применяется команда **DROP**. Будьте осторожны: эта операция необратима и удаляет данные вместе с построенными индексами.

```
python
from pymilvus import utility
utility.drop_collection("ai_data_collection")
Используйте код с осторожностью.
```

Перед удалением убедитесь, что коллекция не используется активными ИИ-сервисами, так как это вызовет ошибку в работе приложения.

Задание:

1. Разработать схему коллекции (Schema), включающую векторные поля и метаданные (например, ID, теги, текст).
2. Реализовать программное добавление (Insert) набора векторов в созданную коллекцию.
3. Настроить различные типы векторных индексов (Flat, IVF_FLAT или HNSW) и сравнить их влияние на скорость извлечения данных.
4. Выполнить настройку прав доступа на уровне конкретной коллекции (Collection-level security).

Методические указания по ходу выполнения работы (комментарии к заданиям, к выполнению заданий):

- **Проектирование схемы:** При формировании требований определите, какие метаданные необходимы для фильтрации. Например, при поиске по документам полезно хранить timestamp или category. Укажите auto_id для упрощения администрирования.
- **Индексирование:** Это ключевой этап администрирования. Индекс **FLAT** дает 100% точность, но медленен; **HNSW** (Hierarchical Navigable Small World) — современный стандарт для ИИ, обеспечивающий высокую скорость поиска. Протестируйте параметры индекса (например, M и efConstruction для HNSW).
- **Добавление данных:** Используйте пакетную вставку (Batch Insert). Загрузка данных по одному вектору неэффективна для больших систем. Отработайте экспорт части коллекции в формат JSON для проверки корректности записи метаданных.
- **Безопасность:** Настройте доступ так, чтобы сервисный аккаунт ИИ мог выполнять поиск (Search), но не имел прав на удаление коллекции (Drop) или изменение параметров индекса.

Форма представления результата:

1. **Описание схемы коллекции** с обоснованием выбранной размерности векторов.
2. **Листинг кода** на языке Python для создания индекса и пакетной загрузки данных.
3. **Сравнительная таблица** времени выполнения поискового запроса при разных типах индексов.
4. **Скриншот консоли** управления, подтверждающий успешное создание и наполнение коллекции

Тема практической работы №18. Реализация функции поиска ближайших соседей (Nearest Neighbor Search) на примере текстовых или изображений, объем часов: 4 часа.

У.9 Производить формирование требований к обработке данных и их извлечению
У.11 Производить операции по импорту и экспорту данных в различных форматах
У.4 Осуществлять основные функции по администрированию баз данных
У.5 Настраивать политики безопасности при работе с сервером баз данных

Цель практической работы: Научиться реализовывать алгоритмы поиска по сходству в векторных СУБД, проводить настройку параметров точности извлечения данных и интегрировать поиск в программный код на Python.

Материальное обеспечение: ПК, СУБД Milvus/Weaviate, Python, библиотека sentence-transformers или PIL для работы с изображениями.

Задание:

1. Подготовить набор данных (текстовые описания или миниатюры изображений) и требования к их извлечению.
2. Реализовать скрипт для генерации эмбеддингов и их загрузки в векторную базу.
3. Выполнить серию поисковых запросов («Найди похожие тексты») и проанализировать точность выдачи при разных метриках.
4. Проверить логи безопасности, чтобы убедиться, что запросы выполняются от имени авторизованного пользователя.

Краткие теоретические сведения:
Поиск ближайших соседей (Nearest Neighbor Search)
В векторных базах данных поиск выполняется не по точному совпадению текста или ID, а по математической близости векторов. Процесс поиска заключается в том, чтобы найти в базе объекты, чьи векторы (эмбеддинги) находятся на минимальном расстоянии от вектора-запроса.

Пример выполнения поиска на Python (библиотека PyMilvus):
Для поиска используется метод `search()`. Основные параметры: вектор-запрос, имя поля, метрика и количество результатов (Limit).
```
python
```

```
# Вектор, полученный из запроса пользователя "Как
настроить ИИ?"
query_vector = [0.12, 0.05, ..., 0.22]

search_params = {"metric_type": "L2", "params":
{"nprobe": 10}}

results = collection.search(
    data=[query_vector],
    anns_field="embeddings",
    param=search_params,
    limit=3, # Вернуть 3 самых похожих результата
    output_fields=["text_content"] # Вывести связанные
данные
)
Используйте код с осторожностью.
```

Результаты поиска

После выполнения кода база возвращает список объектов с указанием дистанции (дистанции или коэффициента схожести). Чем меньше значение при метрике **L2**, тем ближе объекты друг к другу.

Пример вывода результатов:

```
text
- ID: 101, Distance: 0.15, Content: "Инструкция по
конфигурации нейросети"
- ID: 405, Distance: 0.24, Content: "Базовые настройки
моделей ИИ"
- ID: 212, Distance: 0.88, Content: "История развития
баз данных"
Используйте код с осторожностью.
```

Примечание: третий результат имеет большую дистанцию (0.88), что говорит о низкой релевантности запросу.

Фильтрация по метаданным

Часто при поиске требуется извлечь данные только из определенной категории. Для этого применяются выражения фильтрации:

```
python
# Поиск только в категории "Техническая документация"
results = collection.search(
    data=[query_vector],
```

```
    anns_field="embeddings",
    param=search_params,
    limit=3,
    expr="category == 'tech_docs'"
)
Используйте код с осторожностью.
```

Удаление нерелевантных данных

Если в ходе мониторинга (У.4) выявлено, что векторные данные устарели или загружены ошибочно, их можно удалить по первичному ключу:

```
python
collection.delete("id in [101, 405]")
Используйте код с осторожностью.
```

Методические указания по ходу выполнения работы:

- **Генерация векторов:** Используйте библиотеку sentence-transformers для текстов. Важно, чтобы модель генерации векторов для базы и для запроса была **одинаковой**, иначе поиск не даст результатов.
- **Параметры поиска:** Экспериментируйте с параметром limit (Top-K). Обратите внимание на значение distance или score в результатах — чем меньше расстояние (L2) или больше косинусное сходство, тем точнее результат.
- **Администрирование :** Используйте инструменты мониторинга базы, чтобы оценить нагрузку на CPU во время выполнения тяжелых поисковых запросов.

Форма представления результата:

1. **Листинг Python-скрипта** (преобразование данных + запрос к БД).
2. **Сводная таблица результатов** поиска (Запрос -> Топ-3 найденных объекта -> Оценка близости).
3. **Скриншот консоли администратора**, подтверждающий выполнение поисковых операций.
4. **Вывод** о выборе оптимальной метрики для вашей предметной области

Тема практической работы №19. Интеграция векторной базы данных с Python для загрузки и обработки векторов, объем часов: 4 час.

У.9 Производить формирование требований к обработке данных и их извлечению
У.11 Производить операции по импорту и экспорту данных в различных форматах
У.4 Осуществлять основные функции по администрированию баз данных
У.5 Настраивать политики безопасности при работе с сервером баз данных

Цель практической работы: Освоить программные методы управления векторными базами данных через Python, научиться автоматизировать процесс подготовки и импорта эмбеддингов и настраивать безопасное соединение.

Краткие теоретические сведения:
Взаимодействие Python с векторными СУБД
Для интеграции приложений искусственного интеллекта с базами данных используются специализированные библиотеки-драйверы (SDK). Например, для Milvus это `pymilvus`, для Weaviate — `weaviate-client`. Основная задача Python-скрипта — преобразовать «сырые» данные (текст, изображения) в числовые векторы и организовать их пакетную загрузку.
Процесс интеграции включает три этапа:

1. **Подключение:** Установка сессии с сервером БД.
2. **Обработка (Embedding):** Использование моделей (например, из библиотеки `sentence-transformers`) для генерации векторов.
3. **Загрузка (Upsert):** Передача данных в коллекцию.

Пример кода интеграции (загрузка данных):

```
python
import numpy as np
from pymilvus import connections, Collection

# 1. Подключение к серверу
connections.connect("default", host="localhost",
port="19530")

# 2. Подготовка данных (текст -> вектор)
# В реальной задаче здесь используется ML-модель
vectors = [[0.1] * 768, [0.2] * 768]  # Пример векторов
```

```
metadatas = ["Инструкция №1", "Справочник №2"]

# 3. Загрузка в коллекцию
data = [vectors, metadatas]
collection = Collection("ai_collection")
collection.insert(data)
collection.flush() # Фиксация данных в БД
Используйте код с осторожностью.
```

Обработка данных перед загрузкой

При формировании требований важно учитывать лимиты СУБД. Если данные поступают в формате **CSV** или **JSON**, Python-скрипт должен выполнить:

- **Очистку:** удаление лишних символов и дублей.
- **Нормализацию:** приведение векторов к единичной длине (требуется для некоторых метрик сходства).
- **Пакетную обработку (Batching):** загрузку данных частями по 100–1000 записей для снижения нагрузки на сеть и CPU.

Материальное обеспечение: ПК, развернутая векторная СУБД (Docker), Python 3.10+, библиотека sentence-transformers или openai.

Задание:

1. Разработать Python-скрипт для автоматического подключения к векторной БД с использованием аутентификации.
2. Реализовать функцию импорта данных из файла формата **JSON**, содержащего текстовые описания.
3. Добавить в скрипт блок обработки: преобразование текстов в векторы и их пакетную загрузку в коллекцию.
4. Выполнить проверку загрузки: извлечь (экспортировать) первые 5 записей из базы для контроля целостности метаданных.

Методические указания по ходу выполнения работы:

- **Безопасность:** Не прописывайте пароли открытым текстом в коде. Используйте переменные окружения (.env файлы) или getpass.
- **Импорт:** При чтении JSON используйте стандартную библиотеку json или pandas. Проверьте, чтобы размерность сгенерированных векторов строго совпадала с размерностью, заданной при создании коллекции в ПР №17.
- **Администрирование:** Используйте метод collection.num_entities для контроля фактического количества загруженных объектов.

Форма представления результата:

1. Листинг завершенного Python-скрипта с комментариями.
2. Скриншот консоли со статистикой загруженных данных.
3. Образец файла JSON, использованного для импорта.
4. Вывод о преимуществах автоматизированной загрузки перед ручным управлением данными.

Тема практической работы №20. Проведение кластеризации данных в векторной базе с использованием встроенных функций, объем часов: 6 часов.

У.9 Производить формирование требований к обработке данных и их извлечению
У.11 Производить операции по импорту и экспорту данных в различных форматах
У.4 Осуществлять основные функции по администрированию баз данных
У.5 Настраивать политики безопасности при работе с сервером баз данных

Цель практической работы: Научиться использовать механизмы автоматической группировки данных в векторных СУБД, проводить настройку параметров разбиения пространства и анализировать результаты извлечения сгруппированных данных.

Материальное обеспечение: ПК, СУБД Milvus/Weaviate, Python, библиотека Scikit-learn (для внешней валидации кластеров).
Задание:

1. Сформировать требования к обработке данных: определить количество предполагаемых групп (кластеров) для тестового набора данных.
2. Реализовать загрузку векторов и создание индекса типа **IVF_FLAT** с заданным числом разделов (nlist).
3. Провести эксперимент по извлечению данных из разных кластеров, используя поисковые запросы к центроидам групп.
4. Выполнить экспорт сформированных кластеров в формат CSV для визуализации и анализа.

Краткие теоретические сведения:
Кластеризация в векторных СУБД
Кластеризация — это процесс автоматической группировки векторов на

основе их близости в многомерном пространстве. В отличие от поиска ближайших соседей (NNS), где мы ищем объекты относительно одного запроса, кластеризация анализирует всю совокупность данных для выявления скрытых структур и закономерностей.

Зачем нужна кластеризация в ИИ:

- **Группировка контента:** Автоматическое разделение тысяч статей по темам без участия человека.
- **Очистка данных:** Выявление аномалий (объектов, которые не попали ни в один крупный кластер).
- **Оптимизация поиска:** Некоторые типы индексов (например, **IVF**) используют кластеризацию для разделения базы на сегменты, чтобы ускорить извлечение данных.

Пример реализации логики кластеризации (алгоритм IVF):
Векторная база данных разбивает пространство на k кластеров (центроидов). При поиске система сначала определяет ближайший кластер, а затем ищет только внутри него.

python
```
# Настройка параметров кластеризации при создании индекса
index_params = {
    "index_type": "IVF_FLAT",
    "metric_type": "L2",
    "params": {"nlist": 128} # Количество кластеров (центроидов)
}
collection.create_index("embeddings", index_params)
```
Используйте код с осторожностью.

Извлечение результатов кластеризации
Хотя большинство векторных СУБД проводят кластеризацию «под капотом» для ускорения поиска, администратор может извлекать данные по группам, используя фильтрацию по метаданным или специализированные функции анализа распределения векторов.

Методические указания по ходу выполнения работы:

- **Выбор параметров:** Параметр nlist (число кластеров) должен соответствовать объему данных. Для малых выборок (до 10 000 векторов) достаточно 128–256 кластеров.
- **Администрирование:** Мониторьте потребление оперативной памяти при перестроении индекса. Процесс кластеризации — ресурсоемкая операция, требующая значительных мощностей CPU.
- **Экспорт:** При выгрузке данных добавьте поле с расстоянием до центроида кластера. Это поможет оценить «плотность» групп — чем меньше среднее расстояние, тем качественнее проведена кластеризация.
- **Безопасность:** Убедитесь, что функции управления индексами (перестроение кластеров) доступны только под учетной записью администратора.

Форма представления результата:

1. Скриншот параметров индекса с настроенным количеством кластеров.
2. Листинг кода выполнения запросов к группам данных.
3. Таблица с примерами объектов, попавших в один кластер (подтверждение семантической близости).
4. Вывод о влиянии количества кластеров на скорость извлечения данных.

II. Общие рекомендации

По всем вопросам, связанным с изучением дисциплины (включая самостоятельную работу), консультироваться с преподавателем.

III. Контроль и оценка результатов

Оценка за выполнение практической работы выставляется по пятибалльной системе и учитывается как показатель текущей успеваемости обучающегося.

Качественная оценка индивидуальных образовательных достижений		**Критерии оценки результата**
балл (оценка)	**вербальный аналог**	
5	отлично	Представленные работы высокого качества, уровень выполнения отвечает всем требованиям. Студент демонстрирует свободное владение синтаксисом SQL и NoSQL, успешно разворачивает серверы СУБД и векторные хранилища (Milvus/PostgreSQL/Oracle). Теоретическое содержание освоено полностью: студент обосновывает выбор индексов (HNSW/IVF) и политик безопасности. Практические навыки

		сформированы: реализовано безошибочное подключение через Python, настроено шифрование и репликация. Все задания выполнены в полном объеме, отчеты содержат корректные скриншоты и листинги кода.
4	хорошо	Уровень выполнения работы отвечает всем требованиям. Теоретическое содержание освоено полностью, однако некоторые практические навыки (например, настройка сложных параметров SSL/TLS или тонкий тюнинг векторных индексов) сформированы недостаточно. Все задания выполнены, но в коде Python-интеграции или в SQL-скриптах настройки прав доступа (GRANT/REVOKE) встречаются незначительные ошибки, не влияющие на общую работоспособность системы. Отчеты оформлены правильно.
3	удовлетворительно	Уровень выполнения отвечает большинству основных требований. Теоретическое содержание (основы администрирования, принципы репликации и бэкапа) освоено частично, но пробелы не носят существенного характера. Студент испытывает затруднения при ручной конфигурации файлов (postgresql.conf, mongod.cfg), требуя помощи преподавателя. Большинство заданий выполнено, но допущены ошибки в реализации ролевой модели безопасности или в процедурах восстановления данных. Навыки работы с векторными СУБД сформированы на базовом уровне.
2	не удовлетворительно	Теоретическое содержание модуля не освоено, базовые принципы функционирования СУБД не поняты. Необходимые практические навыки работы (установка СУБД, выполнение базовых запросов, создание бэкапов) не сформированы. Большинство предусмотренных заданий не выполнено или выполнено с критическими ошибками, приводящими к неработоспособности сервера или потере данных. Студент не может объяснить логику работы настроенных им инструментов.