

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Северо-Кавказский федеральный университет»

Колледж СКФУ в г. Ставрополе

**Методические указания
к практическим занятиям**

ПМ.01 РАЗРАБОТКА, АДМИНИСТРИРОВАНИЕ И ЗАЩИТА БАЗ ДАННЫХ

**МДК 01.01 Технология разработки баз данных
МДК 01.02 Обслуживание и защита баз данных**

Специальность 09.02.11 Разработка и управление программным обеспечением

Форма обучения очная

Ставрополь

Пояснительная записка

Методические указания по организации и проведению практических занятий составлены в соответствии с ФГОС СПО по специальности 09.02.11 Разработка и управление программным обеспечением и рабочей программой МДК 01.01 Технология разработки баз данных.

Целью выполнения практических занятий является систематизация и закрепление теоретических знаний и формирование практических умений.

Особое значение для усвоения содержания МДК 01.01 Технология разработки баз данных и привития практических навыков имеет правильная и четкая организация проведения и выполнения студентами практических работ под контролем преподавателя.

Перед началом выполнения каждой работы студенты должны ознакомиться с ее основными положениями, порядком выполнения работы.

По каждому практическому занятию предусматривается индивидуальный отчет перед преподавателями.

Выполнение студентами практических работ направленно на:

- обобщение, систематизацию, углубление, закрепление полученных теоретических знаний по конкретным темам;
- формирование умений применять полученные знания на практике и реализацию единства интеллектуальной и практической деятельности;
- развитие интеллектуальных умений у будущих специалистов: аналитических, проектировочных, конструктивных и др.;
- выработку при решении поставленных задач таких профессиональных качеств, как самостоятельность, ответственность, точность, творческая инициатива.

В результате изучения учебной дисциплины МДК 01.01 Технология разработки баз данных обучающийся должен:

знать:

- модели данных, основные операции и ограничения; технологию установки и настройки сервера баз данных;
- требования к безопасности сервера базы данных;
- государственные стандарты и требования к обслуживанию баз данных.

уметь:

- проектировать и создавать базы данных;
- выполнять запросы по обработке данных на языке SQL;
- осуществлять основные функции по администрированию баз данных;
- разрабатывать политику безопасности SQL сервера, базы данных и отдельных объектов базы данных;
- владеть технологиями проведения сертификации программного средства.

Иметь практический опыт в:

- участии в соадминистрировании серверов;
- разработке политики безопасности SQL сервера, базы данных и отдельных объектов базы данных.

Правила выполнения практических работ.

1. Студент должен прийти на практическое занятие подготовленным к выполнению практической работы.
2. Каждый студент после проведения работы должен представить отчет о проделанной работе с анализом полученных результатов и выводом по работе.
3. Таблицы и рисунки следует выполнять с помощью чертежных инструментов (линейки, циркуля, и т.д.) карандашом с соблюдением ЕСКД.
4. Расчет следует проводить с точностью до двух значащих цифр.
5. Исправления проводить на обратной стороне листа. При мелких исправлениях

неправильное слово (буква, число и т.п.) аккуратно зачеркивается и над ним пишут правильное пропущенное слово (букву, число и т.п.).

6. Вспомогательные расчеты можно выполнять на отдельных листах, а при необходимости на листах отчета.

7. Если студент не выполнит практическую работу или часть работы, то он выполнит ее во внеурочное время, согласованное с преподавателем.

8. Оценку по практической работе студент получает с учетом срока выполнения работы, если:

- расчеты выполнены правильно и в полном объеме;
- сделан анализ проделанной работы и вывод по результатам работы;
- студент может пояснить выполнение любого этапа работы;
- отчет выполнен в соответствии с требованиями к выполнению работы.

Практическое занятие №1. Индексы: создание, использование и оптимизация

Тема: Создание и использование индексов для ускорения поиска. Удаление и пересоздание индексов. Оптимизация запросов с использованием EXPLAIN. Применение индексов в сложных запросах. Использование частичных индексов и индексов по выражениям. Работа с составными индексами

Цель работы: Научиться создавать различные типы индексов, анализировать их влияние на производительность запросов, использовать команду SET STATISTICS TIME для измерения производительности и осваивать техники оптимизации с помощью индексов .

Теоретическая часть

Индексы — это структуры базы данных, которые ускоряют операции поиска и выборки данных за счёт организации быстрого доступа к строкам таблицы .

Типы индексов в SQL Server:

- **Кластеризованный индекс (Clustered)** — определяет физический порядок данных в таблице. Таблица может иметь только один кластеризованный индекс.
- **Некластеризованный индекс (Nonclustered)** — содержит ключевые значения и указатели на строки данных. Таблица может иметь несколько некластеризованных индексов.
- **Составной индекс (Composite index)** — индекс, созданный по нескольким столбцам.
- **Уникальный индекс (Unique index)** — гарантирует уникальность значений в индексируемых столбцах.
- **Индекс по выражению (Index on computed column)** — индекс, созданный на вычисляемом столбце .
- **Частичный (фильтрованный) индекс (Filtered index)** — индекс, который содержит только подмножество строк таблицы .

Ход работы

Задание 1. Подготовка тестовой среды

Создадим базу данных и таблицу с большим объёмом данных для тестирования индексов.

```
sql
```

```
CREATE DATABASE IndexLab;
```

```
GO
```

```
USE IndexLab;
```

```
GO
```

```

-- Создадим таблицу с большим количеством записей
CREATE TABLE Products (
    ProductID INT IDENTITY(1,1) PRIMARY KEY, -- кластеризованный индекс по
умолчанию
    ProductName NVARCHAR(100),
    CategoryID INT,
    Price DECIMAL(10,2),
    CreatedDate DATETIME DEFAULT GETDATE(),
    Description NVARCHAR(MAX)
);
GO

```

```

-- Наполним таблицу 100 000 тестовых записей
DECLARE @i INT = 1;
WHILE @i <= 100000
BEGIN
    INSERT INTO Products (ProductName, CategoryID, Price, Description)
    VALUES (
        'Товар ' + CAST(@i AS NVARCHAR(10)),
        @i % 10 + 1,
        ROUND(RAND() * 1000, 2),
        'Описание товара номер ' + CAST(@i AS NVARCHAR(10))
    );
    SET @i = @i + 1;
END;
GO

```

Задание 2. Измерение производительности без индексов

Выполним поиск без индекса и измерим время выполнения.

sql

```
-- Включим вывод статистики времени
```

```
SET STATISTICS TIME ON;
```

```
GO
```

```
-- Поиск по цене (индекс отсутствует)
```

```
SELECT * FROM Products WHERE Price > 500;
```

```
GO
```

```
SET STATISTICS TIME OFF;
```

```
GO
```

Обратите внимание на время выполнения и количество операций чтения (во вкладке "Messages").

Задание 3. Создание и тестирование простого некластеризованного индекса

Создадим индекс по полю Price и проверим производительность.

sql

```
-- Создание индекса
```

```
CREATE INDEX IX_Products_Price ON Products(Price);
```

```
GO
```

```
-- Повторный поиск с индексом
```

```
SET STATISTICS TIME ON;
```

```
GO
```

```
SELECT * FROM Products WHERE Price > 500;
GO
SET STATISTICS TIME OFF;
GO
```

Сравните время выполнения с предыдущим результатом.

Задание 4. Анализ плана выполнения запроса

Включим отображение фактического плана выполнения и проанализируем, используется ли индекс .

1. Нажмите **Ctrl+M** (или в меню **Query** → **Include Actual Execution Plan**).
2. Выполните запрос:

```
sql
SELECT ProductName, Price FROM Products WHERE Price BETWEEN 100 AND 200;
GO
```

3. В дополнительной вкладке **Execution plan** изучите:
 - Используется ли Index Seek (поиск по индексу) или Index Scan (сканирование).
 - Стоимость операции в процентах.

(Рисунок 1: План выполнения с Index Seek)

Задание 5. Создание составного индекса

Создадим составной индекс для запросов, которые фильтруют по нескольким полям.

```
sql
-- Запрос, который фильтрует по категории и цене
SELECT * FROM Products WHERE CategoryID = 5 AND Price < 300;
GO
```

```
-- Создадим составной индекс
CREATE INDEX IX_Products_Category_Price ON Products(CategoryID, Price);
GO
```

```
-- Повторим запрос и проанализируем план
SELECT * FROM Products WHERE CategoryID = 5 AND Price < 300;
GO
```

Важно: Порядок столбцов в составном индексе имеет значение. Индекс будет эффективен для условий, которые используют первые столбцы индекса.

Задание 6. Создание фильтрованного (частичного) индекса

Фильтрованные индексы полезны, когда мы часто ищем только определённое подмножество данных .

```
sql
-- Создадим индекс только для товаров категории 10 с ценой выше 500
CREATE INDEX IX_Products_Category10_HighPrice ON Products(Price)
WHERE CategoryID = 10 AND Price > 500;
GO
```

```
-- Запрос, который будет использовать этот индекс
SELECT ProductName, Price FROM Products
WHERE CategoryID = 10 AND Price > 500 AND Price < 800;
GO
```

Задание 7. Индекс по выражению (вычисляемому столбцу)

Допустим, нам часто нужно искать по году создания. Создадим вычисляемый столбец и индекс на нём .

```
sql
-- Добавим вычисляемый столбец
ALTER TABLE Products ADD YearCreated AS YEAR(CreatedDate);
```

GO

```
-- Создадим индекс на вычисляемом столбце
CREATE INDEX IX_Products_YearCreated ON Products(YearCreated);
GO
```

```
-- Запрос, использующий индекс
SELECT * FROM Products WHERE YearCreated = YEAR(GETDATE());
GO
```

Задание 8. Удаление и пересоздание индекса

Иногда требуется перестроить индекс для устранения фрагментации.

sql

```
-- Удаление индекса
DROP INDEX IX_Products_Price ON Products;
GO
```

```
-- Пересоздание индекса
CREATE INDEX IX_Products_Price ON Products(Price);
GO
```

```
-- Альтернатива: перестроение существующего индекса
ALTER INDEX IX_Products_Price ON Products REBUILD;
GO
```

Задание 9. Анализ использования индексов с помощью DMV

Используем динамические представления для анализа использования индексов.

sql

```
SELECT
    OBJECT_NAME(i.object_id) AS TableName,
    i.name AS IndexName,
    i.type_desc,
    us.user_seeks,
    us.user_scans,
    us.user_lookups,
    us.user_updates
FROM sys.indexes i
LEFT JOIN sys.dm_db_index_usage_stats us
    ON i.object_id = us.object_id AND i.index_id = us.index_id
WHERE OBJECT_NAME(i.object_id) = 'Products'
ORDER BY us.user_seeks DESC;
GO
```

Этот запрос покажет, какие индексы действительно используются, а какие только замедляют операции обновления .

Контрольные вопросы

1. В чём разница между кластеризованным и некластеризованным индексом?
2. Когда следует использовать составной индекс?
3. Что такое фильтрованный индекс и для каких сценариев он подходит?
4. Как с помощью плана выполнения определить, используется ли индекс?
5. Почему избыточное количество индексов может замедлить операции вставки и обновления?

Практическое занятие №2. Разработка представлений для различных групп пользователей

Тема: Разработка необходимых для различных групп пользователей представлений

Цель работы: Научиться создавать представления (VIEW) для упрощения доступа к данным и обеспечения безопасности, разграничивая доступ к данным для разных категорий пользователей .

Теоретическая часть

Представление (VIEW) — это виртуальная таблица, содержимое которой определяется запросом. Представления используются для:

- **Безопасности:** скрытие конфиденциальных столбцов или строк от пользователей .
- **Упрощения:** предоставление пользователям удобного интерфейса для сложных запросов.
- **Согласованности:** единый способ доступа к данным для разных приложений.

Ограничения представлений :

- Представление может быть создано только в текущей базе данных.
- Максимальное количество столбцов — 1024.
- Не все представления поддерживают обновление данных (например, с агрегатными функциями).

Ход работы

Задание 1. Подготовка тестовой базы данных

Создадим базу данных для отдела кадров и продаж.

```
sql
```

```
CREATE DATABASE CompanyDB;
```

```
GO
```

```
USE CompanyDB;
```

```
GO
```

```
-- Таблица сотрудников (конфиденциальные данные)
```

```
CREATE TABLE Employees (
```

```
    EmployeeID INT PRIMARY KEY IDENTITY(1,1),
```

```
    FullName NVARCHAR(100),
```

```
    Position NVARCHAR(50),
```

```
    Department NVARCHAR(50),
```

```
    Salary DECIMAL(10,2),
```

```
    PassportNumber NVARCHAR(20),
```

```
    HomeAddress NVARCHAR(200),
```

```
    Phone NVARCHAR(20)
```

```
);
```

```
GO
```

```
-- Таблица заказов
```

```
CREATE TABLE Orders (
```

```
    OrderID INT PRIMARY KEY IDENTITY(1,1),
```

```
    OrderDate DATETIME DEFAULT GETDATE(),
```

```
    CustomerName NVARCHAR(100),
```

```
    EmployeeID INT FOREIGN KEY REFERENCES Employees(EmployeeID),
```

```
    TotalAmount DECIMAL(10,2)
```

```
);
```

```
GO
```

```
-- Наполним данными
INSERT INTO Employees (FullName, Position, Department, Salary, PassportNumber,
HomeAddress, Phone) VALUES
('Иванов Иван', 'Менеджер', 'Продажи', 50000, '1234 567890', 'ул. Ленина, 1', '+7-123-456-78-90'),
('Петрова Мария', 'Бухгалтер', 'Финансы', 60000, '2345 678901', 'ул. Мира, 2', '+7-234-567-89-01'),
('Сидоров Петр', 'Разработчик', 'IT', 70000, '3456 789012', 'ул. Гагарина, 3', '+7-345-678-90-12'),
('Козлова Анна', 'Директор', 'Управление', 100000, '4567 890123', 'ул. Пушкина, 4', '+7-456-789-01-23');
```

```
INSERT INTO Orders (CustomerName, EmployeeID, TotalAmount) VALUES
('ООО "Ромашка"', 1, 15000),
('ЗАО "Весна"', 1, 23000),
('ИП Петров', 3, 8500),
('ООО "Лето"', 3, 12700);
GO
```

Задание 2. Создание представления для отдела кадров (без конфиденциальных данных)

Для отдела кадров нужно видеть контактные данные, но не зарплату .

```
sql
CREATE VIEW HR.EmployeeContactInfo
AS
SELECT
    EmployeeID,
    FullName,
    Position,
    Department,
    HomeAddress,
    Phone
FROM Employees;
GO
```

```
-- Проверка
SELECT * FROM HR.EmployeeContactInfo;
GO
```

Задание 3. Создание представления через графический интерфейс SSMS

1. В **Object Explorer** раскройте базу данных **CompanyDB**.
2. Щёлкните правой кнопкой на папке **Views** → **New View**.
3. В диалоговом окне **Add Table** выберите таблицу **Orders** и нажмите **Add**, затем **Close**.
4. На **Diagram** **pane** отметьте столбцы: OrderID, OrderDate, CustomerName, TotalAmount.
5. На **Criteria pane** в поле **Filter** для столбца TotalAmount введите > 10000 (чтобы показывать только крупные заказы).
6. В меню **File** выберите **Save View**, введите имя LargeOrders.

(Рисунок 2: Создание представления в конструкторе)

Задание 4. Создание представления с объединением таблиц

Создадим представление для менеджеров, показывающее заказы с именем сотрудника.

```
sql
```

```
CREATE VIEW Sales.OrderDetails
AS
SELECT
    o.OrderID,
    o.OrderDate,
    o.CustomerName,
    e.FullName AS EmployeeName,
    o.TotalAmount
FROM Orders o
INNER JOIN Employees e ON o.EmployeeID = e.EmployeeID;
GO
```

```
-- Проверка
SELECT * FROM Sales.OrderDetails ORDER BY OrderDate DESC;
GO
```

Задание 5. Создание представления с агрегацией

Для отчётов директору — статистика по сотрудникам .

```
sql
CREATE VIEW Sales.EmployeeSalesSummary
AS
SELECT
    e.EmployeeID,
    e.FullName,
    COUNT(o.OrderID) AS OrdersCount,
    ISNULL(SUM(o.TotalAmount), 0) AS TotalSales,
    ISNULL(AVG(o.TotalAmount), 0) AS AvgOrderAmount
FROM Employees e
LEFT JOIN Orders o ON e.EmployeeID = o.EmployeeID
GROUP BY e.EmployeeID, e.FullName;
GO
```

```
-- Проверка
SELECT * FROM Sales.EmployeeSalesSummary ORDER BY TotalSales DESC;
GO
```

Задание 6. Защита данных с помощью представлений

Создадим логины и пользователей для разных ролей.

```
sql
USE master;
GO
CREATE LOGIN HR_Login WITH PASSWORD = 'HRPass123!';
CREATE LOGIN Manager_Login WITH PASSWORD = 'ManagerPass123!';
GO
```

```
USE CompanyDB;
GO
CREATE USER HR_User FOR LOGIN HR_Login;
CREATE USER Manager_User FOR LOGIN Manager_Login;
GO
```

```
-- Дадим права только на представления, не на базовые таблицы
GRANT SELECT ON HR.EmployeeContactInfo TO HR_User;
GRANT SELECT ON Sales.OrderDetails TO Manager_User;
```

```
GRANT SELECT ON Sales.EmployeeSalesSummary TO Manager_User;  
GO
```

Проверка: Подключитесь от имени HR_User и попробуйте выполнить:

```
sql  
SELECT * FROM Employees; -- Ошибка: нет доступа  
SELECT * FROM HR.EmployeeContactInfo; -- Успешно
```

Задание 7. Модификация и удаление представления

```
sql  
-- Изменение представления (добавление столбца)
```

```
ALTER VIEW HR.EmployeeContactInfo  
AS
```

```
SELECT  
    EmployeeID,  
    FullName,  
    Position,  
    Department,  
    HomeAddress,  
    Phone,  
    -- Добавим отдел  
    Department  
FROM Employees;  
GO
```

```
-- Удаление представления  
DROP VIEW IF EXISTS Sales.LargeOrders;  
GO
```

Контрольные вопросы

1. В каких целях используются представления в базах данных?
2. Какие ограничения существуют при создании представлений?
3. Может ли представление обновлять данные в базовых таблицах? При каких условиях?
4. Как представления помогают обеспечить безопасность данных?

Практическое занятие №3. Анализ и оптимизация производительности запросов

Тема: Анализ логов ошибок и медленных запросов. Оптимизация запросов. Построение и анализ плана выполнения запросов. Оптимизация структуры таблиц и индексов. Профилирование запросов. Мониторинг и анализ производительности запросов

Цель работы: Освоить методы анализа производительности запросов, научиться читать планы выполнения, выявлять узкие места и оптимизировать структуру базы данных .

Теоретическая часть

План выполнения запроса — это последовательность операций, которую выбирает оптимизатор для выполнения запроса. Анализ плана позволяет понять, почему запрос выполняется медленно .

Ключевые метрики:

- **Index Scan** — полное сканирование таблицы (медленно).
- **Index Seek** — выборочное чтение по индексу (быстро).
- **Key Lookup** — подтягивание данных из кластеризованного индекса.
- **Cost** — относительная стоимость операции в процентах.

Ход работы

Задание 1. Подготовка тестовой среды

```
sql
```

```
CREATE DATABASE PerformanceLab;  
GO
```

```
USE PerformanceLab;  
GO
```

```
-- Создадим большую таблицу
```

```
CREATE TABLE Sales (  
    SaleID INT IDENTITY(1,1) PRIMARY KEY,  
    SaleDate DATETIME,  
    ProductID INT,  
    Quantity INT,  
    UnitPrice DECIMAL(10,2),  
    CustomerID INT,  
    SalesPerson NVARCHAR(100)  
);  
GO
```

```
-- Наполним 100 000 записей
```

```
WITH Numbers AS (  
    SELECT TOP 100000 ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) AS n  
    FROM sys.objects a CROSS JOIN sys.objects b  
)  
INSERT INTO Sales (SaleDate, ProductID, Quantity, UnitPrice, CustomerID, SalesPerson)  
SELECT  
    DATEADD(day, -n % 1000, GETDATE()),  
    n % 100,  
    n % 10 + 1,  
    ROUND(RAND(CHECKSUM(NEWID())) * 100, 2),
```

```

n % 1000,
'SalesPerson' + CAST(n % 10 AS NVARCHAR(10))
FROM Numbers;
GO

```

Задание 2. Анализ "медленного" запроса

Выполним запрос и проанализируем его план.

```

sql
-- Включаем фактический план выполнения (Ctrl+M)
SET STATISTICS TIME ON;
GO

```

```

-- "Медленный" запрос без индексов
SELECT
    CustomerID,
    SUM(Quantity * UnitPrice) AS TotalSpent
FROM Sales
WHERE SaleDate > '2025-01-01'
GROUP BY CustomerID
ORDER BY TotalSpent DESC;
GO

```

```

SET STATISTICS TIME OFF;
GO

```

В плане выполнения вы увидите **Clustered Index Scan** (полное сканирование таблицы) — признак отсутствия подходящего индекса.

(Рисунок 3: План выполнения с Index Scan)

Задание 3. Создание индексов и повторный анализ

Создадим индексы для ускорения запроса.

```

sql
-- Индекс по дате (для фильтрации)
CREATE INDEX IX_Sales_SaleDate ON Sales(SaleDate);
GO

```

```

-- Составной индекс для группировки и агрегации
CREATE INDEX IX_Sales_Customer_Date ON Sales(CustomerID, SaleDate)
INCLUDE (Quantity, UnitPrice);
GO

```

Повторите запрос из Задания 2 и сравните план выполнения. Теперь должны использоваться **Index Seek** и **Index Scan** (но по индексу, а не по таблице).

Задание 4. Поиск самых ресурсоёмких запросов

Используем динамические представления для анализа статистики выполнения запросов .

```

sql
SELECT TOP 5
    qs.total_worker_time/1000 AS TotalCPUTime_ms,
    qs.execution_count,
    qs.total_elapsed_time/1000 AS TotalElapsedTime_ms,
    qs.total_logical_reads,
    qs.total_logical_writes,
    SUBSTRING(st.text, qs.statement_start_offset/2 + 1,
        (CASE WHEN qs.statement_end_offset = -1 THEN LEN(st.text)
            ELSE qs.statement_end_offset/2 - qs.statement_start_offset/2 + 1
        END)) AS QueryText

```

```
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) st
ORDER BY qs.total_worker_time DESC;
GO
```

Задание 5. Анализ плана с помощью SET SHOWPLAN_XML

Включим режим отображения плана без фактического выполнения.

```
sql
SET SHOWPLAN_XML ON;
GO
```

```
-- Этот запрос не выполняется, только показывается план
SELECT CustomerID, SUM(Quantity * UnitPrice) AS TotalSpent
FROM Sales
WHERE SaleDate > '2025-01-01'
GROUP BY CustomerID
ORDER BY TotalSpent DESC;
GO
```

```
SET SHOWPLAN_XML OFF;
GO
```

В результате появится ссылка на XML-файл с планом, который можно открыть и детально изучить.

Задание 6. Оптимизация структуры таблицы

Иногда лучше изменить структуру таблицы, чем создавать множество индексов. Например, выделим часто используемые поля в отдельную таблицу.

```
sql
-- Создадим справочник товаров
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName NVARCHAR(100),
    Category NVARCHAR(50)
);
GO
```

```
-- Добавим внешний ключ
ALTER TABLE Sales ADD CONSTRAINT FK_Sales_Product
FOREIGN KEY (ProductID) REFERENCES Products(ProductID);
GO
```

Задание 7. Использование Database Engine Tuning Advisor

1. В SSMS выберите меню **Tools** → **Database Engine Tuning Advisor**.
2. Подключитесь к серверу.
3. Выберите базу данных PerformanceLab.
4. В окне **Workload** вставьте текст "медленного" запроса.
5. Запустите анализ и изучите рекомендации по созданию индексов.

Контрольные вопросы

1. В чём разница между Index Scan и Index Seek?
2. Какую информацию предоставляет фактический план выполнения?
3. Для чего используется Database Engine Tuning Advisor?
4. Какие метрики в sys.dm_exec_query_stats помогают найти самые "тяжёлые" запросы?
5. Почему избыточное количество индексов может замедлить операции вставки?

Практическое занятие №4. Пользовательские функции

Тема: Создание и использование простых пользовательских функций. Создание пользовательских функций для работы с текстовыми данными и датами. Вложенные пользовательские функции. Обработка ошибок в пользовательских функциях. Использование пользовательских функций в запросах. Создание пользовательских функций для работы с JSON-данными

Цель работы: Научиться создавать пользовательские функции различных типов, использовать их в запросах и обрабатывать JSON-данные с помощью встроенных функций .

Теоретическая часть

Типы пользовательских функций в SQL Server:

- **Скалярные функции (Scalar Functions)** — возвращают одно значение.
- **Табличные функции (Inline Table-Valued Functions)** — возвращают таблицу, определяются одним оператором SELECT.
- **Многооператорные табличные функции (Multi-statement Table-Valued Functions)** — возвращают таблицу, могут содержать сложную логику.

JSON-функции в SQL Server :

- ISJSON() — проверяет, является ли строка допустимым JSON.
- JSON_VALUE() — извлекает скалярное значение из JSON.
- JSON_QUERY() — извлекает объект или массив из JSON.
- JSON_MODIFY() — изменяет значение в JSON.
- OPENJSON() — преобразует JSON в табличный формат.

Ход работы

Задание 1. Подготовка базы данных

```
sql
```

```
CREATE DATABASE FunctionsLab;  
GO
```

```
USE FunctionsLab;  
GO
```

```
CREATE TABLE Employees (  
    EmployeeID INT PRIMARY KEY IDENTITY(1,1),  
    FullName NVARCHAR(100),  
    BirthDate DATE,  
    Email NVARCHAR(100),  
    Salary DECIMAL(10,2),  
    AdditionalInfo NVARCHAR(MAX) -- для JSON-данных  
);  
GO
```

```
INSERT INTO Employees (FullName, BirthDate, Email, Salary, AdditionalInfo) VALUES  
(Иванов Иван, '1990-05-15', 'ivanov@company.com', 50000,  
'{"skills": ["SQL", "C#"], "projects": [{"name": "ProjectA", "hours": 120}]}'),  
(Петрова Мария, '1985-08-22', 'petrova@company.com', 60000,  
'{"skills": ["Python", "SQL"], "projects": [{"name": "ProjectB", "hours": 200}]}'),  
(Сидоров Петр, '1992-11-03', 'sidorov@company.com', 45000,
```

```
'{"skills": ["Java", "JavaScript"], "projects": []}');
```

```
GO
```

Задание 2. Создание скалярной функции для расчёта возраста

```
sql
```

```
CREATE FUNCTION dbo.CalculateAge (@BirthDate DATE)
```

```
RETURNS INT
```

```
AS
```

```
BEGIN
```

```
    DECLARE @Age INT;
```

```
    SET @Age = DATEDIFF(YEAR, @BirthDate, GETDATE()) -
```

```
        CASE WHEN DATEADD(YEAR, DATEDIFF(YEAR, @BirthDate, GETDATE()),  
@BirthDate) > GETDATE()
```

```
            THEN 1 ELSE 0 END;
```

```
    RETURN @Age;
```

```
END;
```

```
GO
```

```
-- Использование функции
```

```
SELECT
```

```
    FullName,
```

```
    BirthDate,
```

```
    dbo.CalculateAge(BirthDate) AS Age
```

```
FROM Employees;
```

```
GO
```

Задание 3. Создание функции для форматирования ФИО

```
sql
```

```
CREATE FUNCTION dbo.FormatFullName (@FullName NVARCHAR(100))
```

```
RETURNS NVARCHAR(100)
```

```
AS
```

```
BEGIN
```

```
    -- Простейший пример: перевод в верхний регистр и удаление лишних пробелов
```

```
    RETURN UPPER(LTRIM(RTRIM(@FullName)));
```

```
END;
```

```
GO
```

```
SELECT
```

```
    dbo.FormatFullName(FullName) AS FormattedName
```

```
FROM Employees;
```

```
GO
```

Задание 4. Создание табличной функции для фильтрации по зарплате

```
sql
```

```
CREATE FUNCTION dbo.GetEmployeesBySalary (@MinSalary DECIMAL(10,2),  
@MaxSalary DECIMAL(10,2))
```

```
RETURNS TABLE
```

```
AS
```

```
RETURN
```

```
(
```

```
    SELECT EmployeeID, FullName, Email, Salary
```

```
    FROM Employees
```

```
    WHERE Salary BETWEEN @MinSalary AND @MaxSalary
```

```
);
```

```
GO
```

-- Использование

```
SELECT * FROM dbo.GetEmployeesBySalary(40000, 55000);
```

```
GO
```

Задание 5. Вложенные функции

Функции можно вызывать внутри других функций.

sql

```
SELECT
```

```
    FullName,
```

```
    dbo.CalculateAge(BirthDate) AS Age,
```

```
    CASE
```

```
        WHEN dbo.CalculateAge(BirthDate) < 30 THEN 'Молодой специалист'
```

```
        WHEN dbo.CalculateAge(BirthDate) < 40 THEN 'Опытный специалист'
```

```
        ELSE 'Старший специалист'
```

```
    END AS Category
```

```
FROM Employees;
```

```
GO
```

Задание 6. Многооператорная табличная функция со сложной логикой

sql

```
CREATE FUNCTION dbo.GetEmployeeStatistics ()
```

```
RETURNS @Stats TABLE
```

```
(
```

```
    Category NVARCHAR(50),
```

```
    EmployeeCount INT,
```

```
    AvgSalary DECIMAL(10,2),
```

```
    MinSalary DECIMAL(10,2),
```

```
    MaxSalary DECIMAL(10,2)
```

```
)
```

```
AS
```

```
BEGIN
```

```
    INSERT INTO @Stats
```

```
    SELECT
```

```
        CASE
```

```
            WHEN Salary < 50000 THEN 'До 50000'
```

```
            WHEN Salary < 70000 THEN '50000-70000'
```

```
            ELSE 'Более 70000'
```

```
        END AS Category,
```

```
        COUNT(*) AS EmployeeCount,
```

```
        AVG(Salary) AS AvgSalary,
```

```
        MIN(Salary) AS MinSalary,
```

```
        MAX(Salary) AS MaxSalary
```

```
    FROM Employees
```

```
    GROUP BY CASE
```

```
        WHEN Salary < 50000 THEN 'До 50000'
```

```
        WHEN Salary < 70000 THEN '50000-70000'
```

```
        ELSE 'Более 70000'
```

```
    END;
```

```
    RETURN;
```

```
END;
```

```
GO
```

```
SELECT * FROM dbo.GetEmployeeStatistics();
```

```
GO
```

Задание 7. Работа с JSON-данными через функции

Проверка валидности JSON

```
sql
```

```
SELECT
```

```
    EmployeeID,
```

```
    FullName,
```

```
    AdditionalInfo,
```

```
    ISJSON(AdditionalInfo) AS IsValidJSON
```

```
FROM Employees;
```

```
GO
```

Извлечение данных из JSON

```
sql
```

```
SELECT
```

```
    FullName,
```

```
    JSON_VALUE(AdditionalInfo, '$.skills[0]') AS FirstSkill,
```

```
    JSON_QUERY(AdditionalInfo, '$.projects') AS ProjectsJSON
```

```
FROM Employees
```

```
WHERE ISJSON(AdditionalInfo) > 0;
```

```
GO
```

Создание функции для извлечения списка навыков

```
sql
```

```
CREATE FUNCTION dbo.GetEmployeeSkills (@EmployeeID INT)
```

```
RETURNS TABLE
```

```
AS
```

```
RETURN
```

```
(
```

```
    SELECT
```

```
        value AS Skill
```

```
    FROM Employees e
```

```
    CROSS APPLY OPENJSON(e.AdditionalInfo, '$.skills')
```

```
    WHERE e.EmployeeID = @EmployeeID
```

```
);
```

```
GO
```

```
-- Использование
```

```
SELECT * FROM dbo.GetEmployeeSkills(1);
```

```
GO
```

Функция для преобразования данных в JSON

```
sql
```

```
CREATE FUNCTION dbo.GetEmployeesJSON ()
```

```
RETURNS NVARCHAR(MAX)
```

```
AS
```

```
BEGIN
```

```
    DECLARE @JSON NVARCHAR(MAX);
```

```
    SELECT @JSON = (
```

```
        SELECT EmployeeID, FullName, Email, Salary
```

```
        FROM Employees
```

```
        FOR JSON PATH, ROOT('Employees')
```

```
    );
```

```
    RETURN @JSON;
END;
GO
```

```
SELECT dbo.GetEmployeesJSON();
GO
```

Задание 8. Обработка ошибок в функциях

В функциях нельзя использовать конструкцию TRY...CATCH, но можно проверять входные параметры.

```
sql
CREATE FUNCTION dbo.SafeDivide (@a DECIMAL(10,2), @b DECIMAL(10,2))
RETURNS DECIMAL(10,2)
AS
BEGIN
    IF @b = 0
        RETURN NULL; -- Возвращаем NULL вместо ошибки

    RETURN @a / @b;
END;
GO
```

```
SELECT
    Salary,
    dbo.SafeDivide(Salary, 1000) AS SalaryInThousands
FROM Employees;
GO
```

Контрольные вопросы

1. Какие типы пользовательских функций существуют в SQL Server?
2. В чём разница между скалярной и табличной функцией?
3. Для чего используется функция JSON_VALUE и чем она отличается от JSON_QUERY?
4. Можно ли в пользовательской функции изменять данные в таблицах?
5. Какие ограничения существуют при обработке ошибок в функциях?

Практическое занятие №5. Хранимые процедуры

Тема: Создание простой хранимой процедуры для вставки данных. Создание хранимой процедуры для обновления определенного поля в таблице на основании некоторого критерия. Создание хранимой процедуры, принимающую параметры для фильтрации данных и возвращающую результат в виде набора строк. Создание хранимой процедуры с использованием курсора для последовательной обработки записей. Создание хранимой процедуры со встроенной обработкой ошибок. Создание сложной хранимой процедуры с несколькими параметрами, выполняющую несколько операций над данными. Оптимизация хранимых процедур.

Цель работы: Научиться создавать хранимые процедуры различной сложности, использовать параметры, обрабатывать ошибки и оптимизировать производительность.

Теоретическая часть

Хранимая процедура — это скомпилированный набор инструкций T-SQL, который хранится на сервере. Преимущества:

- Повторное использование кода.
- Уменьшение сетевого трафика.
- Повышение безопасности.
- Улучшение производительности за счёт кэширования планов выполнения.

Ход работы

Задание 1. Подготовка базы данных

sql

```
CREATE DATABASE ProceduresLab;  
GO
```

```
USE ProceduresLab;  
GO
```

```
CREATE TABLE Products (  
    ProductID INT PRIMARY KEY IDENTITY(1,1),  
    ProductName NVARCHAR(100) NOT NULL,  
    CategoryID INT,  
    Price DECIMAL(10,2),  
    StockQuantity INT,  
    CreatedDate DATETIME DEFAULT GETDATE()  
);  
GO
```

```
CREATE TABLE Sales (  
    SaleID INT PRIMARY KEY IDENTITY(1,1),  
    ProductID INT FOREIGN KEY REFERENCES Products(ProductID),  
    SaleDate DATETIME DEFAULT GETDATE(),  
    Quantity INT,  
    TotalPrice DECIMAL(10,2)  
);  
GO
```

-- Наполним начальными данными

```
INSERT INTO Products (ProductName, CategoryID, Price, StockQuantity) VALUES
('Ноутбук', 1, 45000, 10),
('Мышь', 1, 800, 50),
('Клавиатура', 1, 1500, 30),
('Монитор', 2, 12000, 15);
GO
```

Задание 2. Процедура для вставки данных

```
sql
CREATE PROCEDURE sp_InsertProduct
    @ProductName NVARCHAR(100),
    @CategoryID INT,
    @Price DECIMAL(10,2),
    @StockQuantity INT
AS
BEGIN
    INSERT INTO Products (ProductName, CategoryID, Price, StockQuantity)
    VALUES (@ProductName, @CategoryID, @Price, @StockQuantity);

    -- Возвращаем ID новой записи
    SELECT SCOPE_IDENTITY() AS NewProductID;
END;
GO
```

```
-- Тестирование
EXEC sp_InsertProduct 'Принтер', 2, 8500, 8;
GO
```

Задание 3. Процедура для обновления данных с условием

```
sql
CREATE PROCEDURE sp_UpdateProductPrice
    @ProductID INT,
    @NewPrice DECIMAL(10,2)
AS
BEGIN
    UPDATE Products
    SET Price = @NewPrice
    WHERE ProductID = @ProductID;

    -- Возвращаем количество обновлённых строк
    SELECT @@ROWCOUNT AS RowsAffected;
END;
GO
```

```
EXEC sp_UpdateProductPrice @ProductID = 5, @NewPrice = 9000;
GO
```

Задание 4. Процедура с параметрами для фильтрации

```
sql
CREATE PROCEDURE sp_GetProductsByCategory
    @CategoryID INT = NULL, -- если NULL, то все категории
    @MinPrice DECIMAL(10,2) = 0,
    @MaxPrice DECIMAL(10,2) = 999999
AS
BEGIN
```

```

SELECT
    ProductID,
    ProductName,
    CategoryID,
    Price,
    StockQuantity
FROM Products
WHERE (@CategoryID IS NULL OR CategoryID = @CategoryID)
    AND Price BETWEEN @MinPrice AND @MaxPrice
ORDER BY Price;
END;
GO

```

```

-- Тестирование
EXEC sp_GetProductsByCategory @CategoryID = 1, @MinPrice = 1000;
EXEC sp_GetProductsByCategory; -- все товары
GO

```

Задание 5. Процедура с курсором для последовательной обработки

Создадим процедуру для пересчёта цен с наценкой для каждой категории .
sql

```

CREATE PROCEDURE sp_ApplyPriceMarkup
    @CategoryID INT,
    @MarkupPercent DECIMAL(5,2)
AS
BEGIN
    DECLARE @ProductID INT, @CurrentPrice DECIMAL(10,2);

    -- Объявляем курсор
    DECLARE product_cursor CURSOR FOR
        SELECT ProductID, Price
        FROM Products
        WHERE CategoryID = @CategoryID;

    OPEN product_cursor;

    FETCH NEXT FROM product_cursor INTO @ProductID, @CurrentPrice;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Обновляем цену с наценкой
        UPDATE Products
        SET Price = @CurrentPrice * (1 + @MarkupPercent/100)
        WHERE CURRENT OF product_cursor;

        FETCH NEXT FROM product_cursor INTO @ProductID, @CurrentPrice;
    END;

    CLOSE product_cursor;
    DEALLOCATE product_cursor;

    SELECT 'Обновление завершено' AS Message;
END;

```

GO

-- Тестирование

EXEC sp_ApplyPriceMarkup @CategoryID = 1, @MarkupPercent = 10;

GO

Важно: Курсоры медленные, их следует использовать только когда нельзя обойтись массовыми операциями .

Задание 6. Процедура с обработкой ошибок (TRY...CATCH)

sql

CREATE PROCEDURE sp_SafeInsertSale

 @ProductID INT,

 @Quantity INT

AS

BEGIN

 BEGIN TRY

 BEGIN TRANSACTION;

 -- Проверка наличия товара

 DECLARE @Stock INT, @Price DECIMAL(10,2);

 SELECT @Stock = StockQuantity, @Price = Price
 FROM Products WITH (UPDLOCK, HOLDLOCK)
 WHERE ProductID = @ProductID;

 IF @Stock IS NULL

 BEGIN

 RAISERROR('Товар не найден', 16, 1);

 ROLLBACK TRANSACTION;

 RETURN;

 END;

 IF @Stock < @Quantity

 BEGIN

 RAISERROR('Недостаточно товара на складе', 16, 1);

 ROLLBACK TRANSACTION;

 RETURN;

 END;

 -- Обновляем остаток

 UPDATE Products

 SET StockQuantity = StockQuantity - @Quantity

 WHERE ProductID = @ProductID;

 -- Вставляем запись о продаже

 INSERT INTO Sales (ProductID, Quantity, TotalPrice)

 VALUES (@ProductID, @Quantity, @Price * @Quantity);

 COMMIT TRANSACTION;

 SELECT 'Продажа оформлена' AS Status, @ProductID AS ProductID, @Quantity AS
Quantity;
 END TRY

```

BEGIN CATCH
    IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;

    -- Возвращаем информацию об ошибке
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_MESSAGE() AS ErrorMessage,
        ERROR_SEVERITY() AS ErrorSeverity;
END CATCH;
END;
GO

-- Тестирование
EXEC sp_SafeInsertSale @ProductID = 1, @Quantity = 2; -- Успех
EXEC sp_SafeInsertSale @ProductID = 1, @Quantity = 100; -- Ошибка
GO

Задание 7. Сложная процедура с несколькими операциями
sql
CREATE PROCEDURE sp_GenerateSalesReport
    @StartDate DATETIME,
    @EndDate DATETIME,
    @CategoryID INT = NULL,
    @IncludeProductDetails BIT = 1
AS
BEGIN
    -- Создаём временную таблицу для отчёта
    CREATE TABLE #ReportData (
        CategoryName NVARCHAR(50),
        ProductName NVARCHAR(100),
        TotalQuantity INT,
        TotalRevenue DECIMAL(10,2),
        AvgPrice DECIMAL(10,2)
    );

    -- Вставляем агрегированные данные
    INSERT INTO #ReportData
    SELECT
        CASE
            WHEN p.CategoryID = 1 THEN 'Электроника'
            WHEN p.CategoryID = 2 THEN 'Периферия'
            ELSE 'Прочее'
        END AS CategoryName,
        p.ProductName,
        SUM(s.Quantity) AS TotalQuantity,
        SUM(s.TotalPrice) AS TotalRevenue,
        AVG(p.Price) AS AvgPrice
    FROM Sales s
    INNER JOIN Products p ON s.ProductID = p.ProductID
    WHERE s.SaleDate BETWEEN @StartDate AND @EndDate
        AND (@CategoryID IS NULL OR p.CategoryID = @CategoryID)
    GROUP BY p.CategoryID, p.ProductName;

```

```

-- Возвращаем результат
SELECT * FROM #ReportData ORDER BY TotalRevenue DESC;

-- Если нужны детали, возвращаем второй набор
IF @IncludeProductDetails = 1
BEGIN
    SELECT
        s.SaleID,
        s.SaleDate,
        p.ProductName,
        s.Quantity,
        s.TotalPrice
    FROM Sales s
    INNER JOIN Products p ON s.ProductID = p.ProductID
    WHERE s.SaleDate BETWEEN @StartDate AND @EndDate
        AND (@CategoryID IS NULL OR p.CategoryID = @CategoryID)
    ORDER BY s.SaleDate;
END;

-- Удаляем временную таблицу
DROP TABLE #ReportData;
END;
GO

-- Тестирование
EXEC sp_GenerateSalesReport
    @StartDate = '2026-01-01',
    @EndDate = '2026-12-31',
    @IncludeProductDetails = 1;
GO

```

Задание 8. Оптимизация хранимых процедур

Советы по оптимизации :

1. Используйте SET NOCOUNT ON для уменьшения сетевого трафика.
2. Избегайте курсоров, используйте массовые операции.
3. Используйте временные таблицы вместо табличных переменных для больших объёмов данных.

4. Перекомпилируйте процедуры при изменении структуры таблиц.

sql

```

-- Пример оптимизированной процедуры
CREATE PROCEDURE sp_GetTopProducts
    @TopCount INT = 10
AS
BEGIN
    SET NOCOUNT ON;

    SELECT TOP (@TopCount)
        ProductName,
        Price,
        StockQuantity
    FROM Products
    ORDER BY Price DESC;

```

END;
GO

Контрольные вопросы

1. Какие преимущества дают хранимые процедуры по сравнению с обычными запросами?
2. Как обрабатывать ошибки в хранимых процедурах?
3. Почему курсоры считаются неэффективными и когда их всё же приходится использовать?
4. Для чего нужна конструкция WITH (UPDLOCK, HOLDLOCK) в процедуре продажи?
5. Как оптимизировать производительность хранимых процедур?

Практическое занятие №6. Триггеры

Тема: Создание простого триггера для аудита изменений. Проверка корректности данных с помощью триггеров. Автоматическое заполнение полей с помощью триггера. Создание триггера, запрещающий удаление записей из таблицы, если они связаны с другими таблицами. Создание триггера, который реализует каскадное обновление связанной информации. Создание триггера со сложной логикой, включающей обработку ошибок. Оптимизация триггера с использованием временных таблиц

Цель работы: Научиться создавать триггеры различных типов (AFTER, INSTEAD OF) для автоматизации бизнес-логики, обеспечения целостности данных и аудита изменений .

Теоретическая часть

Триггер — это специальный тип хранимой процедуры, которая автоматически выполняется при наступлении определённого события (INSERT, UPDATE, DELETE).

Типы триггеров в SQL Server:

- **AFTER триггеры** — выполняются после завершения операции.
- **INSTEAD OF триггеры** — выполняются вместо операции.
- **DDL триггеры** — срабатывают на события изменения структуры (CREATE, ALTER, DROP).

Специальные таблицы:

- **inserted** — содержит новые/изменённые строки.
- **deleted** — содержит удалённые/старые строки .

Ход работы

Задание 1. Подготовка базы данных

sql

```
CREATE DATABASE TriggersLab;  
GO
```

```
USE TriggersLab;  
GO
```

-- Основные таблицы

```
CREATE TABLE Employees (  
    EmployeeID INT PRIMARY KEY IDENTITY(1,1),  
    FullName NVARCHAR(100),  
    Position NVARCHAR(50),  
    Department NVARCHAR(50),  
    Salary DECIMAL(10,2),  
    LastModified DATETIME,  
    ModifiedBy NVARCHAR(100)  
);  
GO  
  
CREATE TABLE Departments (  
    DepartmentID INT PRIMARY KEY IDENTITY(1,1),  
    DepartmentName NVARCHAR(50),  
    ManagerID INT NULL FOREIGN KEY REFERENCES Employees(EmployeeID),  
    Budget DECIMAL(15,2)  
);
```

GO

```
CREATE TABLE Orders (  
    OrderID INT PRIMARY KEY IDENTITY(1,1),  
    CustomerName NVARCHAR(100),  
    OrderDate DATETIME DEFAULT GETDATE(),  
    Status NVARCHAR(20) DEFAULT 'NEW',  
    TotalAmount DECIMAL(10,2)  
);  
GO
```

```
CREATE TABLE OrderItems (  
    OrderItemID INT PRIMARY KEY IDENTITY(1,1),  
    OrderID INT FOREIGN KEY REFERENCES Orders(OrderID),  
    ProductName NVARCHAR(100),  
    Quantity INT,  
    UnitPrice DECIMAL(10,2)  
);  
GO
```

-- Таблица для аудита

```
CREATE TABLE AuditLog (  
    AuditID INT PRIMARY KEY IDENTITY(1,1),  
    TableName NVARCHAR(50),  
    Action NVARCHAR(10),  
    RecordID INT,  
    OldValue NVARCHAR(MAX),  
    NewValue NVARCHAR(MAX),  
    ChangedBy NVARCHAR(100),  
    ChangeDate DATETIME DEFAULT GETDATE()  
);  
GO
```

-- Начальные данные

```
INSERT INTO Employees (FullName, Position, Department, Salary) VALUES  
(Иванов Иван', 'Разработчик', 'IT', 70000),  
(Петрова Мария', 'Менеджер', 'Продажи', 60000),  
(Сидоров Петр', 'Директор', 'Управление', 100000);
```

```
INSERT INTO Departments (DepartmentName, ManagerID, Budget) VALUES  
(IT', 1, 500000),  
(Продажи', 2, 300000),  
(Управление', 3, 800000);  
GO
```

Задание 2. Триггер для аудита изменений (AFTER UPDATE)

Создадим триггер, который записывает в журнал все изменения зарплат сотрудников.
sql

```
CREATE TRIGGER trg_Employees_Audit  
ON Employees  
AFTER UPDATE  
AS  
BEGIN
```

```
SET NOCOUNT ON;
```

```
INSERT INTO AuditLog (TableName, Action, RecordID, OldValue, NewValue, ChangedBy)
SELECT
    'Employees',
    'UPDATE',
    i.EmployeeID,
    'Salary: ' + CAST(d.Salary AS NVARCHAR(20)),
    'Salary: ' + CAST(i.Salary AS NVARCHAR(20)),
    SUSER_NAME()
FROM inserted i
INNER JOIN deleted d ON i.EmployeeID = d.EmployeeID
WHERE i.Salary != d.Salary; -- логируем только изменение зарплаты
END;
GO
```

-- Тестирование

```
UPDATE Employees SET Salary = 75000 WHERE EmployeeID = 1;
GO
```

```
SELECT * FROM AuditLog;
GO
```

Задание 3. Триггер для автоматического заполнения полей

sql

```
CREATE TRIGGER trg_Employees_LastModified
ON Employees
AFTER INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE e
    SET
        LastModified = GETDATE(),
        ModifiedBy = SUSER_NAME()
    FROM Employees e
    INNER JOIN inserted i ON e.EmployeeID = i.EmployeeID;
END;
GO
```

-- Тестирование

```
UPDATE Employees SET Position = 'Старший разработчик' WHERE EmployeeID = 1;
SELECT * FROM Employees WHERE EmployeeID = 1;
GO
```

Задание 4. Триггер для проверки корректности данных

Запретим вставку заказов с отрицательной суммой.

sql

```
CREATE TRIGGER trg_Orders_CheckAmount
ON Orders
AFTER INSERT, UPDATE
AS
BEGIN
```

```

IF EXISTS (SELECT 1 FROM inserted WHERE TotalAmount < 0)
BEGIN
    RAISERROR('Сумма заказа не может быть отрицательной', 16, 1);
    ROLLBACK TRANSACTION;
    RETURN;
END;
END;
GO

```

```

-- Тестирование (должна быть ошибка)
INSERT INTO Orders (CustomerName, TotalAmount) VALUES ('Тест', -100);
GO

```

Задание 5. Триггер, запрещающий удаление связанных записей

Запретим удаление отдела, если в нём есть сотрудники.

```

sql
CREATE TRIGGER trg_Departments_PreventDelete
ON Departments
INSTEAD OF DELETE
AS
BEGIN
    -- Проверяем, есть ли сотрудники в удаляемых отделах
    IF EXISTS (
        SELECT 1
        FROM deleted d
        INNER JOIN Employees e ON e.Department = d.DepartmentName
    )
    BEGIN
        RAISERROR('Невозможно удалить отдел, в котором есть сотрудники', 16, 1);
        RETURN;
    END;

    -- Если сотрудников нет, удаляем
    DELETE FROM Departments
    WHERE DepartmentID IN (SELECT DepartmentID FROM deleted);
END;
GO

```

```

-- Тестирование
DELETE FROM Departments WHERE DepartmentID = 1; -- Должна быть ошибка
GO

```

Задание 6. Триггер для каскадного обновления

При изменении статуса заказа автоматически обновляем связанные позиции.

```

sql
CREATE TRIGGER trg_Orders_CascadeStatus
ON Orders
AFTER UPDATE
AS
BEGIN
    IF UPDATE(Status)
    BEGIN
        -- Можно добавить логику, например, блокировку изменения позиций
        -- для закрытых заказов
    END
END;

```

```
UPDATE oi
SET oi.UnitPrice = oi.UnitPrice -- триггер, который ничего не меняет, но может
проверять
FROM OrderItems oi
INNER JOIN inserted i ON oi.OrderID = i.OrderID
WHERE i.Status = 'CLOSED';
```

```
-- Или записывать в аудит
INSERT INTO AuditLog (TableName, Action, RecordID, NewValue, ChangedBy)
SELECT
    'Orders',
    'STATUS_CHANGE',
    i.OrderID,
    'New Status: ' + i.Status,
    SUSER_NAME()
FROM inserted i;
END;
END;
GO
```

```
-- Тестирование
UPDATE Orders SET Status = 'PROCESSING' WHERE OrderID = 1;
GO
```

Задание 7. Сложный триггер со вложенной логикой и обработкой ошибок

Триггер для расчёта итоговой суммы заказа при добавлении позиций.

sql

```
CREATE TRIGGER trg_OrderItems_UpdateTotal
ON OrderItems
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    BEGIN TRY
        SET NOCOUNT ON;

        -- Собираем все заказы, которые были изменены
        DECLARE @AffectedOrders TABLE (OrderID INT);

        INSERT INTO @AffectedOrders
        SELECT OrderID FROM inserted
        UNION
        SELECT OrderID FROM deleted;

        -- Обновляем итоговые суммы в заказах
        UPDATE o
        SET TotalAmount = (
            SELECT ISNULL(SUM(Quantity * UnitPrice), 0)
            FROM OrderItems
            WHERE OrderID = o.OrderID
        )
        FROM Orders o
        INNER JOIN @AffectedOrders ao ON o.OrderID = ao.OrderID;
```

```

-- Логируем изменение
INSERT INTO AuditLog (TableName, Action, RecordID, ChangedBy)
SELECT
    'Orders',
    'TOTAL_UPDATED',
    ao.OrderID,
    SUSER_NAME()
FROM @AffectedOrders ao;
END TRY
BEGIN CATCH
    -- В случае ошибки откатываем транзакцию
    IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;

    -- Логируем ошибку (можно в отдельную таблицу)
    INSERT INTO AuditLog (TableName, Action, NewValue, ChangedBy)
    VALUES (
        'OrderItems',
        'ERROR',
        ERROR_MESSAGE(),
        SUSER_NAME()
    );

    -- Пробрасываем ошибку дальше
    THROW;
END CATCH;
END;
GO

-- Тестирование
INSERT INTO Orders (CustomerName, TotalAmount) VALUES ('Тестовый заказ', 0);
DECLARE @OrderID INT = SCOPE_IDENTITY();

INSERT INTO OrderItems (OrderID, ProductName, Quantity, UnitPrice) VALUES
(@OrderID, 'Товар 1', 2, 100),
(@OrderID, 'Товар 2', 1, 250);

SELECT * FROM Orders WHERE OrderID = @OrderID; -- TotalAmount должен быть 450
GO

```

Задание 8. Оптимизация триггеров

Проблема: Триггеры могут замедлять операции. Важно:

- Использовать SET NOCOUNT ON
- Избегать курсоров в триггерах
- Минимизировать количество операций

sql

```

-- Мониторинг выполнения триггеров
SELECT
    OBJECT_NAME(object_id) AS TriggerName,
    execution_count,
    total_worker_time/1000 AS TotalCPUTime_ms,
    total_elapsed_time/1000 AS TotalElapsedTime_ms
FROM sys.dm_exec_trigger_stats

```

```
ORDER BY total_elapsed_time DESC;  
GO
```

Контрольные вопросы

1. В чём разница между AFTER и INSTEAD OF триггерами?
2. Какие специальные таблицы доступны в триггерах и что они содержат?
3. Как избежать бесконечного цикла срабатывания триггеров?
4. Почему триггеры могут снижать производительность и как это оптимизировать?
5. Как отследить выполнение триггеров с помощью DMV?

Практическое занятие №7. Транзакции и уровни изоляции

Тема: Управление транзакциями. Настройка уровней изоляции транзакций. Анализ и решение проблемы грязного чтения. Неповторяемое чтение и фантомное чтение: диагностика и исправление. Автоматическое и ручное управление блокировками в SQL

Цель работы: Изучить механизм транзакций, научиться настраивать уровни изоляции, диагностировать и решать проблемы параллельного доступа .

Теоретическая часть

Уровни изоляции транзакций определяют, как изменения, сделанные одной транзакцией, видны другим транзакциям .

Уровень изоляции	Грязное чтение	Неповторяемое чтение	Фантомы
READ UNCOMMITTED	Да	Да	Да
READ COMMITTED	Нет	Да	Да
REPEATABLE READ	Нет	Нет	Да
SNAPSHOT	Нет	Нет	Нет
SERIALIZABLE	Нет	Нет	Нет

Проблемы параллельного доступа:

- **Грязное чтение (Dirty Read)** — чтение данных, изменённых незавершённой транзакцией.
- **Неповторяемое чтение (Non-repeatable Read)** — при повторном чтении в рамках одной транзакции данные изменяются.
- **Фантомное чтение (Phantom Read)** — при повторном чтении появляются новые строки.

Ход работы

Задание 1. Подготовка базы данных

```
sql
CREATE DATABASE TransactionsLab;
GO
```

```
USE TransactionsLab;
GO
```

```
CREATE TABLE Accounts (
    AccountID INT PRIMARY KEY,
    AccountName NVARCHAR(100),
    Balance DECIMAL(10,2)
);
GO
```

```
INSERT INTO Accounts VALUES
(1, 'Иванов Иван', 1000),
(2, 'Петрова Мария', 500);
GO
```

Задание 2. Демонстрация грязного чтения (READ UNCOMMITTED)

Сессия 1 (изменяет данные, но не завершает транзакцию):

```
sql
BEGIN TRANSACTION;
UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = 1;
```

```
-- Не завершаем транзакцию
SELECT * FROM Accounts WHERE AccountID = 1;
-- ROLLBACK или COMMIT позже
```

Сессия 2 (читает неподтверждённые данные):

```
sql
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT * FROM Accounts WHERE AccountID = 1;
```

Решение: Использовать уровень READ COMMITTED (по умолчанию) .

Задание 3. Неповторяемое чтение (READ COMMITTED vs REPEATABLE READ)

Сессия 1 (читает данные дважды):

```
sql
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN TRANSACTION;
    SELECT * FROM Accounts WHERE AccountID = 1;
    -- Пауза для выполнения Сессии 2
    WAITFOR DELAY '00:00:10';
    SELECT * FROM Accounts WHERE AccountID = 1;
COMMIT;
```

Сессия 2 (изменяет данные между чтениями):

```
sql
UPDATE Accounts SET Balance = Balance + 200 WHERE AccountID = 1;
```

Решение: Использовать REPEATABLE READ.

```
sql
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN TRANSACTION;
    SELECT * FROM Accounts WHERE AccountID = 1;
    WAITFOR DELAY '00:00:10';
    SELECT * FROM Accounts WHERE AccountID = 1;
COMMIT;
```

Задание 4. Фантомное чтение (REPEATABLE READ vs SERIALIZABLE)

Сессия 1:

```
sql
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN TRANSACTION;
    SELECT * FROM Accounts WHERE Balance > 600;
    WAITFOR DELAY '00:00:10';
    SELECT * FROM Accounts WHERE Balance > 600;
COMMIT;
```

Сессия 2:

```
sql
INSERT INTO Accounts VALUES (3, 'Сидоров Петр', 700);
```

Решение: Использовать SERIALIZABLE.

```
sql
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN TRANSACTION;
    SELECT * FROM Accounts WHERE Balance > 600;
    WAITFOR DELAY '00:00:10';
    SELECT * FROM Accounts WHERE Balance > 600;
COMMIT;
```

Задание 5. Уровень изоляции SNAPSHOT

Включим SNAPSHOT изоляцию на уровне базы данных.

```
sql
```

```
ALTER DATABASE TransactionsLab
SET ALLOW_SNAPSHOT_ISOLATION ON;
GO
```

```
-- Сессия 1
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
BEGIN TRANSACTION;
    SELECT * FROM Accounts;
    WAITFOR DELAY '00:00:10';
    -- Данные будут такими же, как в начале транзакции
    SELECT * FROM Accounts;
COMMIT;
GO
```

```
-- Сессия 2 (между чтениями)
UPDATE Accounts SET Balance = 2000 WHERE AccountID = 1;
GO
```

Задание 6. Анализ блокировок

```
sql
-- Просмотр текущих блокировок
SELECT
    request_session_id,
    resource_type,
    resource_description,
    request_mode,
    request_status
FROM sys.dm_tran_locks
WHERE request_session_id > 50;
GO
```

```
-- Поиск блокирующих сессий
SELECT
    blocking.session_id AS BlockingSession,
    blocked.session_id AS BlockedSession,
    blocked.wait_type,
    blocked.wait_time
FROM sys.dm_exec_requests blocked
JOIN sys.dm_exec_requests blocking ON blocked.blocking_session_id = blocking.session_id;
GO
```

Контрольные вопросы

1. Что такое грязное чтение и как его предотвратить?
2. В чём разница между REPEATABLE READ и SERIALIZABLE?
3. Как уровень изоляции SNAPSHOT решает проблемы параллелизма?
4. Как обнаружить блокировки в SQL Server?

Практическое занятие №8. Введение в NoSQL

Тема: Работа с различными типами NoSQL систем управления базами данных

Цель работы: Изучить основные типы NoSQL баз данных, их особенности и сценарии использования .

Теоретическая часть

NoSQL (Not Only SQL) — подход к проектированию баз данных, обеспечивающий высокую масштабируемость и гибкость при работе с неструктурированными данными .

Типы NoSQL баз данных :

1. **Документо-ориентированные (MongoDB, CouchDB)** — данные хранятся в формате JSON-подобных документов.
2. **Ключ-значение (Redis, DynamoDB)** — простейшая модель с высокой производительностью.
3. **Колоночные (Cassandra, HBase)** — оптимизированы для запросов к большим массивам данных.
4. **Графовые (Neo4j)** — для работы со связанными данными.

Сравнение SQL и NoSQL :

- SQL: ACID-транзакции, строгая схема, вертикальное масштабирование.
- NoSQL: высокая масштабируемость, гибкость схемы, горизонтальное масштабирование.

Ход работы

Задание 1. Моделирование документо-ориентированной БД в SQL Server

Хотя SQL Server — реляционная СУБД, она поддерживает работу с JSON, что позволяет моделировать документо-ориентированный подход .

```
sql
```

```
CREATE DATABASE NoSQL_Lab;  
GO
```

```
USE NoSQL_Lab;  
GO
```

```
-- Таблица для хранения документов в формате JSON  
CREATE TABLE Documents (  
    DocumentID INT PRIMARY KEY IDENTITY(1,1),  
    DocumentType NVARCHAR(50),  
    Content NVARCHAR(MAX),  
    CreatedDate DATETIME DEFAULT GETDATE(),  
    CONSTRAINT CK_JSON CHECK (ISJSON(Content) = 1)  
);  
GO
```

```
-- Вставка различных документов
```

```
INSERT INTO Documents (DocumentType, Content) VALUES  
(  
    'customer', '{"id": 1, "name": "Иванов Иван", "email": "ivanov@mail.com", "address": {"city":  
"Москва", "street": "Ленина"}}'),  
(  
    'customer', '{"id": 2, "name": "Петрова Мария", "email": "petrova@mail.com", "phone": "+7-  
123-456-78-90"}'),  
(  
    'order', '{"id": 101, "customerId": 1, "items": [{"product": "Ноутбук", "price": 45000},  
{"product": "Мышь", "price": 800}], "total": 45800}'),
```

```
('product', '{"id": 1001, "name": "Ноутбук", "category": "Электроника", "price": 45000, "inStock": true}');
```

```
GO
```

Задание 2. Запросы к документной базе

Извлечение данных из JSON-документов .

```
sql
```

```
-- Поиск клиентов из Москвы
```

```
SELECT
```

```
    JSON_VALUE(Content, '$.name') AS CustomerName,
```

```
    JSON_VALUE(Content, '$.email') AS Email,
```

```
    JSON_VALUE(Content, '$.address.city') AS City
```

```
FROM Documents
```

```
WHERE DocumentType = 'customer'
```

```
    AND JSON_VALUE(Content, '$.address.city') = 'Москва';
```

```
GO
```

```
-- Поиск заказов с суммой > 40000
```

```
SELECT
```

```
    JSON_VALUE(Content, '$.id') AS OrderID,
```

```
    JSON_VALUE(Content, '$.customerId') AS CustomerID,
```

```
    JSON_VALUE(Content, '$.total') AS Total
```

```
FROM Documents
```

```
WHERE DocumentType = 'order'
```

```
    AND CAST(JSON_VALUE(Content, '$.total') AS DECIMAL(10,2)) > 40000;
```

```
GO
```

Задание 3. Работа с массивами в JSON

```
sql
```

```
-- Извлечение товаров из заказа
```

```
SELECT
```

```
    JSON_VALUE(d.Content, '$.id') AS OrderID,  
    item.*
```

```
FROM Documents d
```

```
CROSS APPLY OPENJSON(d.Content, '$.items') WITH (
```

```
    product NVARCHAR(100) '$.product',
```

```
    price DECIMAL(10,2) '$.price'
```

```
) AS item
```

```
WHERE d.DocumentType = 'order';
```

```
GO
```

Задание 4. Моделирование key-value хранилища

```
sql
```

```
CREATE TABLE KeyValueStore (
```

```
    [Key] NVARCHAR(255) PRIMARY KEY,
```

```
    [Value] NVARCHAR(MAX),
```

```
    ExpiryDate DATETIME NULL
```

```
);
```

```
GO
```

```
INSERT INTO KeyValueStore ([Key], [Value]) VALUES
```

```
('session:user:1', '{"userId": 1, "token": "abc123", "lastAccess": "2026-02-18T10:30:00"}'),
```

```
('config:app:theme', '"dark"'),
```

```
('cache:products:top10', '[{"id": 1, "name": "Ноутбук"}, {"id": 2, "name": "Мышь"}]');
```

```
GO
```

```
-- Быстрый доступ по ключу
SELECT * FROM KeyValueStore WHERE [Key] = 'session:user:1';
GO
```

Задание 5. Моделирование графовой базы

sql

```
CREATE TABLE Nodes (
    NodeID INT PRIMARY KEY,
    NodeType NVARCHAR(50),
    Properties NVARCHAR(MAX)
);
GO
```

```
CREATE TABLE Edges (
    EdgeID INT PRIMARY KEY IDENTITY(1,1),
    FromNode INT FOREIGN KEY REFERENCES Nodes(NodeID),
    ToNode INT FOREIGN KEY REFERENCES Nodes(NodeID),
    Relationship NVARCHAR(50),
    Properties NVARCHAR(MAX)
);
GO
```

```
INSERT INTO Nodes VALUES
(1, 'person', '{"name": "Иван", "age": 30}'),
(2, 'person', '{"name": "Мария", "age": 28}'),
(3, 'project', '{"name": "Project A", "budget": 100000}'),
(4, 'project', '{"name": "Project B", "budget": 150000}');
```

```
INSERT INTO Edges (FromNode, ToNode, Relationship) VALUES
(1, 3, 'works_on'),
(2, 3, 'works_on'),
(1, 4, 'manages');
GO
```

```
-- Поиск проектов, над которыми работает Иван
SELECT
    JSON_VALUE(n.Properties, '$.name') AS ProjectName
FROM Nodes n
INNER JOIN Edges e ON n.NodeID = e.ToNode
INNER JOIN Nodes p ON e.FromNode = p.NodeID
WHERE JSON_VALUE(p.Properties, '$.name') = 'Иван'
    AND e.Relationship = 'works_on';
GO
```

Контрольные вопросы

1. Какие основные типы NoSQL баз данных существуют?
2. В чём преимущества и недостатки NoSQL перед реляционными СУБД?
3. Как SQL Server поддерживает работу с JSON-данными?
4. Для каких сценариев подходят key-value хранилища?

Практическое занятие №9. Запросы к NoSQL данным
Тема: Создания запросов для работы с данными в NoSQL базах данных

Цель работы: Освоить методы создания запросов к данным, хранящимся в NoSQL-подобном формате, используя JSON-функции SQL Server .

Ход работы

Задание 1. Подготовка данных

```
sql
USE NoSQL_Lab;
GO

-- Создадим коллекцию документов с разнородной структурой
CREATE TABLE Library (
    DocumentID INT PRIMARY KEY IDENTITY(1,1),
    DocType NVARCHAR(50),
    Content NVARCHAR(MAX)
);
GO

INSERT INTO Library (DocType, Content) VALUES
('book', '{
    "isbn": "978-3-16-148410-0",
    "title": "SQL для профессионалов",
    "author": {"firstName": "Иван", "lastName": "Петров"},
    "year": 2020,
    "price": 1500,
    "tags": ["sql", "database", "programming"],
    "reviews": [
        {"user": "user1", "rating": 5, "comment": "Отличная книга"},
        {"user": "user2", "rating": 4}
    ]
}'),
('book', '{
    "isbn": "978-0-13-235088-4",
    "title": "NoSQL для начинающих",
    "author": {"firstName": "Мария", "lastName": "Сидорова"},
    "year": 2022,
    "price": 1200,
    "tags": ["nosql", "database"],
    "reviews": [
        {"user": "user3", "rating": 5}
    ]
}'),
('magazine', '{
    "title": "IT News",
    "issue": "January 2026",
    "articles": [
        {"title": "Будущее SQL", "pages": "10-15"},
        {"title": "NoSQL тренды", "pages": "16-20"}
    ],
    "price": 300
```

```

    }'),
    ('user', '{
      "login": "john_doe",
      "name": "John Doe",
      "preferences": {
        "theme": "dark",
        "notifications": true
      },
      "borrowed": [
        {"isbn": "978-3-16-148410-0", "dueDate": "2026-03-01"},
        {"isbn": "978-0-13-235088-4", "dueDate": "2026-03-15"}
      ]
    }');
GO

```

Задание 2. Простые запросы с JSON_VALUE

sql

-- Найти все книги, выпущенные после 2020 года

SELECT

JSON_VALUE(Content, '\$.title') AS Title,

JSON_VALUE(Content, '\$.year') AS Year,

JSON_VALUE(Content, '\$.price') AS Price

FROM Library

WHERE DocType = 'book'

AND CAST(JSON_VALUE(Content, '\$.year') AS INT) > 2020;

GO

-- Найти книги по автору

SELECT

JSON_VALUE(Content, '\$.title') AS Title,

JSON_VALUE(Content, '\$.author.firstName') + ' ' +

JSON_VALUE(Content, '\$.author.lastName') AS Author

FROM Library

WHERE JSON_VALUE(Content, '\$.author.lastName') = 'Петров';

GO

Задание 3. Запросы с JSON_QUERY

sql

-- Извлечь теги книг как JSON-массив

SELECT

JSON_VALUE(Content, '\$.title') AS Title,

JSON_QUERY(Content, '\$.tags') AS TagsJSON,

JSON_QUERY(Content, '\$.reviews') AS ReviewsJSON

FROM Library

WHERE DocType = 'book';

GO

Задание 4. Развертывание массивов с OPENJSON

sql

-- Развернуть теги книг в табличный формат

SELECT

JSON_VALUE(l.Content, '\$.title') AS BookTitle,

tag.value AS Tag

FROM Library l

CROSS APPLY OPENJSON(l.Content, '\$.tags') AS tag

```
WHERE l.DocType = 'book';
GO
```

```
-- Развернуть отзывы
```

```
SELECT
    JSON_VALUE(l.Content, '$.title') AS BookTitle,
    review.user,
    review.rating,
    review.comment
FROM Library l
CROSS APPLY OPENJSON(l.Content, '$.reviews') WITH (
    [user] NVARCHAR(50) '$.user',
    rating INT '$.rating',
    comment NVARCHAR(200) '$.comment'
) AS review
WHERE l.DocType = 'book';
GO
```

Задание 5. Вложенные развертывания

```
sql
```

```
-- Развернуть статьи в журналах
```

```
SELECT
    JSON_VALUE(l.Content, '$.title') AS MagazineTitle,
    article.title,
    article.pages
FROM Library l
CROSS APPLY OPENJSON(l.Content, '$.articles') WITH (
    title NVARCHAR(100) '$.title',
    pages NVARCHAR(10) '$.pages'
) AS article
WHERE l.DocType = 'magazine';
GO
```

Задание 6. Условная логика и модификация

```
sql
```

```
-- Обновить цену книги
```

```
DECLARE @isbn NVARCHAR(50) = '978-3-16-148410-0';
DECLARE @newPrice DECIMAL(10,2) = 1800;
```

```
UPDATE Library
SET Content = JSON_MODIFY(Content, '$.price', @newPrice)
WHERE DocType = 'book'
    AND JSON_VALUE(Content, '$.isbn') = @isbn;
GO
```

```
-- Добавить новый тег к книге
```

```
UPDATE Library
SET Content = JSON_MODIFY(
    Content,
    'append $.tags',
    'best-seller'
)
WHERE DocType = 'book'
    AND JSON_VALUE(Content, '$.isbn') = '978-3-16-148410-0';
```

GO

Задание 7. Агрегация JSON-данных

sql

-- Средний рейтинг книг

SELECT

JSON_VALUE(l.Content, '\$.title') AS BookTitle,
AVG(CAST(review.rating AS FLOAT)) AS AvgRating,
COUNT(*) AS ReviewCount

FROM Library l

CROSS APPLY OPENJSON(l.Content, '\$.reviews') WITH (
rating INT '\$.rating'

) AS review

WHERE l.DocType = 'book'

GROUP BY JSON_VALUE(l.Content, '\$.title');

GO

Контрольные вопросы

1. В чём разница между функциями JSON_VALUE и JSON_QUERY?
2. Как развернуть JSON-массив в реляционный формат?
3. Как добавить новый элемент в JSON-массив?
4. Можно ли выполнять агрегацию по данным, извлечённым из JSON?

Практическое занятие №10. Оптимизация NoSQL производительности

Тема: Оптимизации производительности NoSQL систем управления баз данных, используя индексы и другие техники

Цель работы: Научиться оптимизировать производительность запросов к JSON-данным с помощью индексов и других методов .

Теоретическая часть

Индексация JSON-данных в SQL Server:

- Индексы на вычисляемых столбцах
- Фильтрованные индексы
- Составные индексы для часто используемых путей

Ход работы

Задание 1. Создание базы с большим объёмом JSON-данных

sql

```
CREATE DATABASE NoSQL_Perf;  
GO
```

```
USE NoSQL_Perf;  
GO
```

```
CREATE TABLE Documents (  
    DocID INT IDENTITY(1,1) PRIMARY KEY,  
    DocType NVARCHAR(50),  
    Content NVARCHAR(MAX)  
);  
GO
```

-- Заполним 10000 документов

```
WITH Numbers AS (  
    SELECT TOP 10000 ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) AS n  
    FROM sys.objects a CROSS JOIN sys.objects b  
)  
INSERT INTO Documents (DocType, Content)  
SELECT  
    CASE WHEN n % 3 = 0 THEN 'customer'  
        WHEN n % 3 = 1 THEN 'order'  
        ELSE 'product' END,  
    CASE WHEN n % 3 = 0 THEN  
        '{ "id": ' + CAST(n AS NVARCHAR) + ',  
        "name": "Customer ' + CAST(n AS NVARCHAR) + "',  
        "email": "cust' + CAST(n AS NVARCHAR) + '@mail.com",  
        "city": "City ' + CAST(n % 100 AS NVARCHAR) + "',  
        "age": ' + CAST(20 + n % 50 AS NVARCHAR) + '}'  
    WHEN n % 3 = 1 THEN  
        '{ "id": ' + CAST(n AS NVARCHAR) + ',  
        "customerId": ' + CAST(n % 1000 AS NVARCHAR) + ',  
        "date": "2026-01-" + RIGHT('0' + CAST(1 + n % 30 AS NVARCHAR), 2) + "',  
        "total": ' + CAST(100 + n % 900 AS NVARCHAR) + '}'  
    ELSE  
        '{ "id": ' + CAST(n AS NVARCHAR) + ',  
        "name": "Product ' + CAST(n AS NVARCHAR) + "',
```

```
        "category": "Cat " + CAST(n % 10 AS NVARCHAR) + "",
        "price": ' ' + CAST(10 + n % 990 AS NVARCHAR) + '}'
    END
FROM Numbers;
GO
```

Задание 2. Анализ производительности без индексов

Включим статистику и выполним запрос.

```
sql
SET STATISTICS TIME ON;
GO

-- Поиск клиентов по городу (медленно)
SELECT
    DocID,
    JSON_VALUE(Content, '$.name') AS Name,
    JSON_VALUE(Content, '$.city') AS City
FROM Documents
WHERE DocType = 'customer'
    AND JSON_VALUE(Content, '$.city') = 'City 42';
GO
```

```
SET STATISTICS TIME OFF;
GO
```

Задание 3. Создание индекса на вычисляемом столбце

```
sql
-- Добавим вычисляемые столбцы
ALTER TABLE Documents ADD
    CustomerCity AS CAST(JSON_VALUE(Content, '$.city') AS NVARCHAR(100)),
    CustomerAge AS CAST(JSON_VALUE(Content, '$.age') AS INT),
    OrderTotal AS CAST(JSON_VALUE(Content, '$.total') AS DECIMAL(10,2)),
    ProductCategory AS CAST(JSON_VALUE(Content, '$.category') AS NVARCHAR(50));
GO
```

```
-- Создадим индексы
CREATE INDEX IX_Documents_CustomerCity ON Documents(CustomerCity) WHERE
DocType = 'customer';
CREATE INDEX IX_Documents_OrderTotal ON Documents(OrderTotal) WHERE DocType =
'order';
CREATE INDEX IX_Documents_ProductCategory ON Documents(ProductCategory) WHERE
DocType = 'product';
GO
```

Задание 4. Повторный анализ с индексами

```
sql
SET STATISTICS TIME ON;
GO
```

```
-- Тот же запрос с индексом
SELECT
    DocID,
    JSON_VALUE(Content, '$.name') AS Name,
    CustomerCity AS City
FROM Documents
```

```
WHERE DocType = 'customer'
  AND CustomerCity = 'City 42';
GO
```

```
SET STATISTICS TIME OFF;
GO
```

Сравните время выполнения.

Задание 5. Составные индексы для сложных запросов

sql

-- Запрос: клиенты определённого возраста из конкретного города

```
SELECT
  DocID,
  JSON_VALUE(Content, '$.name') AS Name
FROM Documents
WHERE DocType = 'customer'
  AND CustomerCity = 'City 42'
  AND CustomerAge BETWEEN 30 AND 40;
```

-- Создадим составной индекс

```
CREATE INDEX IX_Documents_City_Age ON Documents(CustomerCity, CustomerAge)
WHERE DocType = 'customer';
GO
```

Задание 6. Индексы для JSON-массивов

Для поиска по элементам массивов требуется другой подход — денормализация.

sql

-- Таблица для поиска по тегам

```
CREATE TABLE DocumentTags (
  DocID INT FOREIGN KEY REFERENCES Documents(DocID),
  Tag NVARCHAR(50),
  PRIMARY KEY (DocID, Tag)
);
GO
```

-- Пример заполнения (для книг с тегами)

```
INSERT INTO DocumentTags (DocID, Tag)
SELECT
  d.DocID,
  tag.value
FROM Documents d
CROSS APPLY OPENJSON(d.Content, '$.tags') AS tag
WHERE d.DocType = 'book';
GO
```

Задание 7. Мониторинг использования индексов

sql

```
SELECT
  OBJECT_NAME(i.object_id) AS TableName,
  i.name AS IndexName,
  us.user_seeks,
  us.user_scans,
  us.user_lookups,
  us.user_updates
FROM sys.indexes i
```

```
LEFT JOIN sys.dm_db_index_usage_stats us
  ON i.object_id = us.object_id AND i.index_id = us.index_id
WHERE OBJECT_NAME(i.object_id) = 'Documents'
ORDER BY us.user_seeks DESC;
GO
```

Контрольные вопросы

1. Как индексировать JSON-поля в SQL Server?
2. Что такое вычисляемые столбцы и как они помогают при работе с JSON?
3. Когда фильтрованные индексы эффективнее обычных?
4. Как отследить эффективность созданных индексов?

Практическое занятие №11. Администрирование NoSQL систем
Тема: Настройка и управление NoSQL системами управления базами данных

Цель работы: Изучить основные задачи администрирования NoSQL-подобных хранилищ на примере SQL Server с JSON-данными .

Ход работы

Задание 1. Проектирование документ-ориентированной базы

sql

```
CREATE DATABASE NoSQL_Admin;  
GO
```

```
USE NoSQL_Admin;  
GO
```

-- Создадим файловую группу для документов

```
ALTER DATABASE NoSQL_Admin ADD FILEGROUP DocumentData;  
GO
```

```
ALTER DATABASE NoSQL_Admin ADD FILE (  
    NAME = 'DocumentData_File',  
    FILENAME = 'C:\SQLData\DocumentData.ndf',  
    SIZE = 100MB,  
    MAXSIZE = 1GB,  
    FILEGROWTH = 100MB  
) TO FILEGROUP DocumentData;  
GO
```

-- Создадим таблицу документов в этой файловой группе

```
CREATE TABLE DocumentCollection (  
    DocID UNIQUEIDENTIFIER DEFAULT NEWID() PRIMARY KEY,  
    DocType NVARCHAR(50),  
    Content NVARCHAR(MAX),  
    CreatedDate DATETIME DEFAULT GETDATE(),  
    ModifiedDate DATETIME,  
    Version INT DEFAULT 1  
) ON DocumentData;  
GO
```

Задание 2. Создание метаданных для документов

sql

```
CREATE TABLE DocumentMetadata (  
    MetadataID INT PRIMARY KEY IDENTITY(1,1),  
    DocID UNIQUEIDENTIFIER FOREIGN KEY REFERENCES DocumentCollection(DocID),  
    MetaKey NVARCHAR(100),  
    MetaValue NVARCHAR(MAX),  
    INDEX IX_Metadata_Key_Value (MetaKey, MetaValue)  
);  
GO
```

-- Триггер для автоматического обновления метаданных

```
CREATE TRIGGER trg_DocumentCollection_UpdateMetadata  
ON DocumentCollection
```

```

AFTER INSERT, UPDATE
AS
BEGIN
    UPDATE DocumentCollection
    SET ModifiedDate = GETDATE()
    WHERE DocID IN (SELECT DocID FROM inserted);
END;
GO

```

Задание 3. Создание процедур для управления документами

```

sql
-- Вставка документа с метаданными
CREATE PROCEDURE sp_InsertDocument
    @DocType NVARCHAR(50),
    @Content NVARCHAR(MAX),
    @Metadata NVARCHAR(MAX) = NULL -- JSON с метаданными
AS
BEGIN
    DECLARE @DocID UNIQUEIDENTIFIER = NEWID();

    BEGIN TRANSACTION;

    -- Вставляем документ
    INSERT INTO DocumentCollection (DocID, DocType, Content)
    VALUES (@DocID, @DocType, @Content);

    -- Вставляем метаданные, если есть
    IF @Metadata IS NOT NULL
    BEGIN
        INSERT INTO DocumentMetadata (DocID, MetaKey, MetaValue)
        SELECT
            @DocID,
            [key],
            value
        FROM OPENJSON(@Metadata);
    END;

    COMMIT TRANSACTION;

    SELECT @DocID AS NewDocID;
END;
GO

```

```

-- Пример использования
EXEC sp_InsertDocument
    @DocType = 'book',
    @Content = '{"title": "NoSQL Guide", "author": "John Doe", "year": 2026}',
    @Metadata = '{"author": "John Doe", "year": "2026", "category": "technical"}';
GO

```

Задание 4. Создание представлений для разных типов документов

```

sql
CREATE VIEW vw_Books
AS

```

```

SELECT
    DocID,
    JSON_VALUE(Content, '$.title') AS Title,
    JSON_VALUE(Content, '$.author') AS Author,
    JSON_VALUE(Content, '$.year') AS Year,
    Content
FROM DocumentCollection
WHERE DocType = 'book';
GO

```

```

CREATE VIEW vw_Customers
AS
SELECT
    DocID,
    JSON_VALUE(Content, '$.name') AS Name,
    JSON_VALUE(Content, '$.email') AS Email,
    JSON_VALUE(Content, '$.city') AS City,
    Content
FROM DocumentCollection
WHERE DocType = 'customer';
GO

```

Задание 5. Создание индексов для поиска по метаданным

sql

-- Поиск документов по метаданным

```

CREATE PROCEDURE sp_FindDocumentsByMetadata

```

```

    @MetaKey NVARCHAR(100),

```

```

    @MetaValue NVARCHAR(MAX)

```

```

AS

```

```

BEGIN

```

```

    SELECT

```

```

        dc.DocID,
```

```
        dc.DocType,
```

```
        dc.Content,
```

```
        dc.CreatedDate

```

```

    FROM DocumentCollection dc

```

```

    INNER JOIN DocumentMetadata dm ON dc.DocID = dm.DocID

```

```

    WHERE dm.MetaKey = @MetaKey AND dm.MetaValue = @MetaValue;

```

```

END;

```

```

GO

```

```

EXEC sp_FindDocumentsByMetadata @MetaKey = 'author', @MetaValue = 'John Doe';

```

```

GO

```

Задание 6. Мониторинг размера документов

sql

-- Анализ размера документов

```

SELECT

```

```

    DocType,
```

```
    COUNT(*) AS DocumentCount,
```

```
    AVG(DATALENGTH(Content)) AS AvgSizeBytes,
```

```
    SUM(DATALENGTH(Content)) / 1024 / 1024 AS TotalSizeMB

```

```

FROM DocumentCollection

```

```

GROUP BY DocType

```

```
ORDER BY TotalSizeMB DESC;
GO
```

```
-- Поиск самых больших документов
SELECT TOP 10
    DocID,
    DocType,
    DATALENGTH(Content) / 1024 AS SizeKB,
    LEFT(Content, 200) AS ContentPreview
FROM DocumentCollection
ORDER BY DATALENGTH(Content) DESC;
GO
```

Задание 7. Архивация старых документов

sql

```
-- Создание архивной таблицы
CREATE TABLE DocumentArchive (
    DocID UNIQUEIDENTIFIER PRIMARY KEY,
    DocType NVARCHAR(50),
    Content NVARCHAR(MAX),
    CreatedDate DATETIME,
    ArchivedDate DATETIME DEFAULT GETDATE()
);
GO
```

```
-- Процедура архивации документов старше N дней
CREATE PROCEDURE sp_ArchiveOldDocuments
    @DaysOld INT
AS
BEGIN
    BEGIN TRANSACTION;

    -- Копируем в архив
    INSERT INTO DocumentArchive (DocID, DocType, Content, CreatedDate)
    SELECT DocID, DocType, Content, CreatedDate
    FROM DocumentCollection
    WHERE CreatedDate < DATEADD(day, -@DaysOld, GETDATE());

    -- Удаляем из основной таблицы
    DELETE FROM DocumentCollection
    WHERE CreatedDate < DATEADD(day, -@DaysOld, GETDATE());

    COMMIT TRANSACTION;

    SELECT @@ROWCOUNT AS ArchivedCount;
END;
GO
```

```
-- Тестирование
EXEC sp_ArchiveOldDocuments @DaysOld = 365;
GO
```

Задание 8. Настройка безопасности для документной базы

sql

-- Создание ролей для разных типов доступа

```
CREATE ROLE DocumentReaders;  
CREATE ROLE DocumentWriters;  
CREATE ROLE DocumentAdmins;  
GO
```

-- Права на чтение

```
GRANT SELECT ON DocumentCollection TO DocumentReaders;  
GRANT SELECT ON vw_Books TO DocumentReaders;  
GRANT SELECT ON vw_Customers TO DocumentReaders;
```

-- Права на запись

```
GRANT INSERT, UPDATE, DELETE ON DocumentCollection TO DocumentWriters;  
GRANT EXECUTE ON sp_InsertDocument TO DocumentWriters;
```

-- Права администратора

```
GRANT CONTROL ON DATABASE::NoSQL_Admin TO DocumentAdmins;  
GO
```

-- Создание пользователей

```
CREATE LOGIN ReaderUser WITH PASSWORD = 'ReaderPass123!';  
CREATE USER ReaderUser FOR LOGIN ReaderUser;  
ALTER ROLE DocumentReaders ADD MEMBER ReaderUser;  
GO
```

Контрольные вопросы

1. Какие задачи администрирования актуальны для документ-ориентированных баз данных?
2. Как организовать хранение метаданных для документов?
3. Какие стратегии архивации данных существуют?
4. Как обеспечить безопасность доступа к документной базе?

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Северо-Кавказский федеральный университет»
Колледж СКФУ в г. Ставрополе

**Методические указания
к практическим занятиям**

ПМ.01 РАЗРАБОТКА, АДМИНИСТРИРОВАНИЕ И ЗАЩИТА БАЗ ДАННЫХ

МДК 01.02 Обслуживание и защита баз данных

Специальность 09.02.11 Разработка и управление программным обеспечением

Форма обучения очная

Ставрополь

Пояснительная записка

Методические указания по организации и проведению практических занятий составлены в соответствии с ФГОС СПО по специальности 09.02.11 Разработка и управление программным обеспечением и рабочей программой МДК 01.02 Обслуживание и защита баз данных.

Целью выполнения практических занятий является систематизация и закрепление теоретических знаний и формирование практических умений.

Особое значение для усвоения содержания МДК 01.02 Обслуживание и защита баз данных и привития практических навыков имеет правильная и четкая организация проведения и выполнения студентами практических работ под контролем преподавателя.

Перед началом выполнения каждой работы студенты должны ознакомиться с ее основными положениями, порядком выполнения работы.

По каждому практическому занятию предусматривается индивидуальный отчет перед преподавателями.

Выполнение студентами практических работ направленно на:

- обобщение, систематизацию, углубление, закрепление полученных теоретических знаний по конкретным темам;
- формирование умений применять полученные знания на практике и реализацию единства интеллектуальной и практической деятельности;
- развитие интеллектуальных умений у будущих специалистов: аналитических, проектировочных, конструктивных и др.;
- выработку при решении поставленных задач таких профессиональных качеств, как самостоятельность, ответственность, точность, творческая инициатива.

В результате изучения учебной дисциплины МДК 01.02 Обслуживание и защита баз данных обучающийся должен:

знать:

- модели данных, основные операции и ограничения;
- технологию установки и настройки сервера баз данных;
- требования к безопасности сервера базы данных;
- государственные стандарты и требования к обслуживанию баз данных.

уметь:

- проектировать и создавать базы данных;
- выполнять запросы по обработке данных на языке SQL;
- осуществлять основные функции по администрированию баз данных;
- разрабатывать политику безопасности SQL сервера, базы данных и отдельных объектов базы данных;

Иметь практический опыт в:

- участии в соадминистрировании серверов;
- разработке политики безопасности SQL сервера, базы данных и отдельных объектов базы данных;

Правила выполнения практических работ.

9. Студент должен прийти на практическое занятие подготовленным к выполнению практической работы.

10. Каждый студент после проведения работы должен представить отчет о проделанной работе с анализом полученных результатов и выводом по работе.

11. Таблицы и рисунки следует выполнять с помощью чертежных инструментов (линейки, циркуля, и т.д.) карандашом с соблюдением ЕСКД.

12. Расчет следует проводить с точностью до двух значащих цифр.

13. Исправления проводить на обратной стороне листа. При мелких исправлениях неправильное слово (буква, число и т.п.) аккуратно зачеркивается и над ним пишут правильное пропущенное слово (букву, число и т.п.).

14. Вспомогательные расчеты можно выполнять на отдельных листах, а при необходимости на листах отчета.

15. Если студент не выполнит практическую работу или часть работы, то он выполнит ее во внеурочное время, согласованное с преподавателем.

16. Оценку по практической работе студент получает с учетом срока выполнения работы, если:

- расчеты выполнены правильно и в полном объеме;
- сделан анализ проделанной работы и вывод по результатам работы;
- студент может пояснить выполнение любого этапа работы;
- отчет выполнен в соответствии с требованиями к выполнению работы.

Практическое занятие №1 Выбор оптимальной конфигурации сервера под определенные аппаратные платформы. Установка и настройка систем управления базами данных. Конфигурирование сервера в соответствии с техническим заданием.

Цель занятия

1. Научиться анализировать аппаратную платформу (характеристики компьютера) на предмет соответствия требованиям СУБД.
2. Освоить процесс установки (или верификации установки) Microsoft SQL Server и SQL Server Management Studio (SSMS).
3. Выполнить первоначальную настройку экземпляра SQL Server согласно техническому заданию (ТЗ) с использованием SSMS.

Необходимое оборудование и ПО

- Персональный компьютер (ПК) с ОС Windows 10/11 Pro или Enterprise.
- Установленный экземпляр **Microsoft SQL Server (Express Edition или выше)**.
- Установленная среда **SQL Server Management Studio (SSMS)**.
- Текстовый редактор (Блокнот) для фиксации параметров.

Ход работы:

Часть 1. Выбор оптимальной конфигурации (Анализ аппаратной платформы)

Прежде чем настраивать сервер БД, нужно понять, на каком «железе» мы работаем. От этого зависит, сможем ли мы выкрутить настройки на максимум или придется экономить ресурсы.

Задание 1.1: Сбор информации о системе

1. Нажмите сочетание клавиш Win + R, введите команду msinfo32 и нажмите Enter.
Это откроет окно «Сведения о системе».
2. В правой части окна найдите следующие параметры и запишите их в отчет (или блокнот):
 - **Процессор (Processor):** Модель и тактовая частота (например, *Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz*).
 - **Установленная память (ОЗУ) (Installed Physical Memory):** Объем в ГБ (например, 16,0 ГБ).
 - **Тип системы (System Type):** Убедитесь, что написано **x64**.
3. Откройте «Этот компьютер» (Мой компьютер) и посмотрите объем свободного места на диске **C:**. Запишите его.

Задание 1.2: Сравнение с минимальными требованиями SQL Server

SQL Server — требовательная система. Даже если у вас мало памяти, его можно установить, но работать будет медленно. Сравните свои характеристики с официальными требованиями для **SQL Server 2019/2022** (они одинаковы) .

Таблица 1 — Минимальные и рекомендуемые требования

Компонент	Минимальные требования	Рекомендуемые требования	Ваши показатели
Процессор	x64, 1.4 ГГц	2.0 ГГц и выше, желательно 4+ ядра	<i>Впишите сюда</i>
ОЗУ	1 ГБ (Express Edition — 512 МБ)	4 ГБ и более (минимум для нормальной работы)	<i>Впишите сюда</i>
Диск	6 ГБ свободно	6 ГБ + место под базы	<i>Впишите сюда</i>
ОС	Windows 10/11 (x64), Windows Server	-	*Win 10/11*

Анализ:

- *Вопрос:* Подходит ли ваш процессор? (Сравните частоту).
- *Вопрос:* Достаточно ли оперативной памяти для комфортной работы сервера? Если

ОЗУ меньше 4 ГБ, нужно быть готовым к тому, что сервер будет тормозить при активной работе.

Важно! Если частота процессора по документации 1.4 ГГц, а в системе написано 1.60 ГГц — это подходит. Однако на некоторых ноутбуках (как в примере с i7-1065G7) базовая частота может быть 1.3 ГГц, но в турбо-режиме подниматься выше. Установщик SQL Server может ругнуться, но это предупреждение, а не фатальная ошибка.

Часть 2. Установка (или проверка наличия) SQL Server и SSMS

Поскольку установка требует прав администратора, а в аудитории их может не быть, мы разберем два сценария.

Задание 2.1: Проверка наличия установленного SQL Server

1. Нажмите Win + R, введите services.msc и нажмите Enter.
2. В списке служб найдите службы, начинающиеся с **SQL Server**.
 - Обратите внимание на имя экземпляра. Обычно это:
 - SQL Server (MSSQLSERVER) — это **экземпляр по умолчанию**.
 - SQL Server (SQLEXPRESS) — это именованный экземпляр **Express**.
3. Посмотрите статус службы. Если она Running (Выполняется) или Started, то SQL Server уже установлен. Если нет — попробуйте нажать правой кнопкой -> "Запустить". (Может потребоваться права админа).

Задание 2.2: Проверка наличия SSMS (Management Studio)

1. Нажмите кнопку **Пуск**.
2. В поиске введите Microsoft SQL Server Management Studio или просто SSMS.
3. Если ярлык найден — запустите его. Если нет — возможно, потребуется установка. *Примечание: если прав на установку нет, обратитесь к преподавателю или системному администратору.*

4. Если SSMS нет, но есть доступ в интернет, можно скачать установщик (около 700 МБ) с официального сайта Microsoft. SSMS устанавливается отдельно от самого сервера.

5.

Часть 3. Конфигурирование сервера по ТЗ (Работа в SSMS)

Предположим, что сервер и SSMS у нас установлены и запущены. Теперь представьте, что вы получили **Техническое задание (ТЗ)**:

*«Для разрабатываемого интернет-магазина требуется развернуть экземпляр SQL Server. Для оптимизации производительности необходимо ограничить использование оперативной памяти сервером (чтобы ОС оставалось минимум 2 ГБ), а также настроить параметры файлов базы данных по умолчанию. Настроить процессорную привязку (affinity) для экземпляра в автоматическом режиме.»

Задание 3.1: Подключение к серверу

1. Запустите **SQL Server Management Studio**.
2. Откроется окно подключения (Connect to Server).
 - **Server type:** Database Engine
 - **Server name:** Введите имя вашего сервера.
 - Если экземпляр по умолчанию: просто . (точка) или localhost.
 - Если Express: .\SQLEXPRESS или ИМЯ_КОМПЬЮТЕРА\SQLEXPRESS.
 - **Authentication:** Windows Authentication.
 - Нажмите кнопку **Connect**.

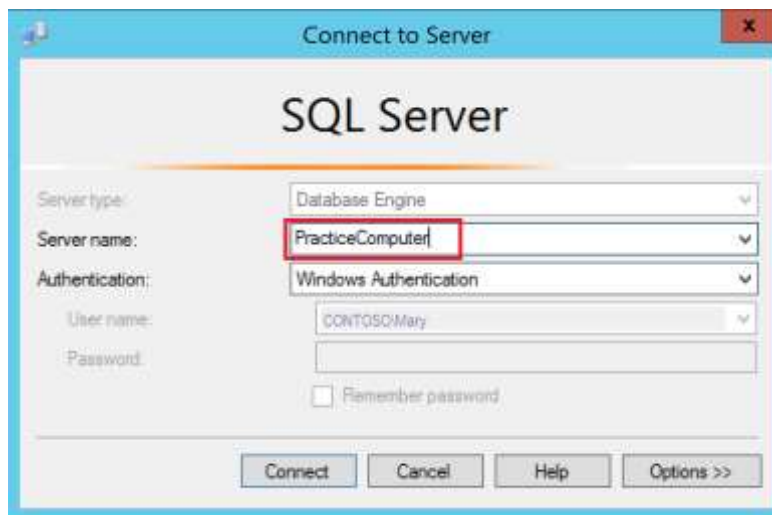


Рисунок 1 – Окно подключения к серверу

Задание 3.2: Настройка использования памяти (ОЗУ)

Суть настройки: Мы запретим SQL Server "съедать" всю память, оставив запас для Windows и других программ.

1. В обозревателе объектов (Object Explorer) слева найдите ваш сервер, нажмите на него **правой кнопкой мыши** и выберите **Properties** (Свойства).
2. В открывшемся окне слева выберите раздел **Memory** (Память).
3. Справа найдите параметр "**Maximum server memory (in MB)**" (Максимальная память сервера).
4. Включите логику: У вас всего, допустим, 8 ГБ ОЗУ (8192 МБ). Для Windows нужно оставить хотя бы 2 ГБ (2048 МБ) + немного запаса.
 - **Формула:** Max Memory = Общая ОЗУ — 2048 МБ (для ОС).
 - *Пример:* Если ОЗУ = 8192 МБ, то в поле вводим 6144 (8192 — 2048 = 6144). Если ОЗУ всего 4 ГБ (4096 МБ), то можно оставить 3072 МБ для SQL, но это мало.
5. Нажмите **ОК**.

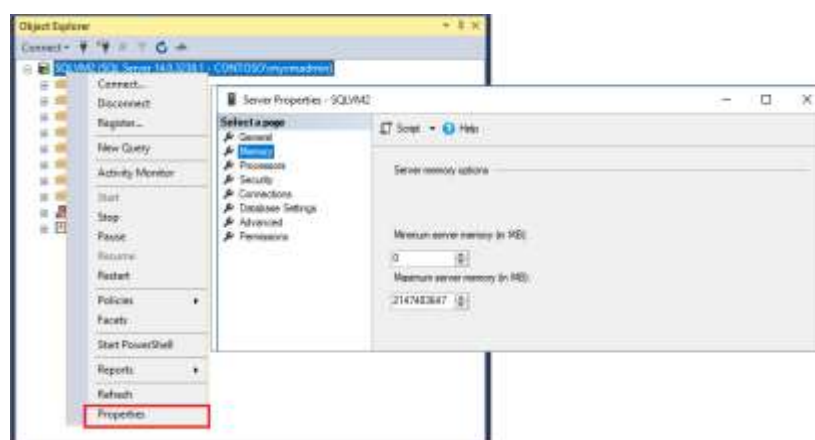


Рисунок 2 – Настройка максимального объема памяти сервера

Задание 3.3: Настройка файлов базы данных по умолчанию

Суть настройки: По умолчанию SQL Server хранит базы в системной папке. В ТЗ (или для производительности) их лучше выносить на отдельный диск (если бы он был). Мы посмотрим, где они лежат сейчас.

1. Снова нажмите правой кнопкой на сервер в Object Explorer -> **Properties**.
2. Выберите раздел **Database Settings** (Параметры базы данных).
3. Внизу окна найдите секцию "**Database default locations**" (Расположение файлов базы данных по умолчанию).
 - **Data:** Папка для файлов данных (.mdf).
 - **Log:** Папка для файлов журнала (.ldf).
4. *Задание:* Запишите эти пути в отчет.
5. *Для продвинутых:* Если бы у вас был диск D: и права администратора, вы бы создали там папку SQLData и изменили пути на D:\SQLData. Но в рамках лабораторной мы просто фиксируем значения.

Задание 3.4: Настройка файлов TempDB

TempDB - это "мусорная" база, куда SQL Server складывает временные данные. Она очень активно используется. Для ускорения работы ее файлы часто разносят на разные диски и создают несколько файлов данных.

1. В Object Explorer раскройте папку **System Databases**.
2. Найдите базу **tempdb**, нажмите на неё правой кнопкой -> **Properties**.
3. Перейдите в раздел **Files** (Файлы).
4. Обратите внимание на строки:
 - o tempdev — файл данных.
 - o templog — файл журнала.
5. Посмотрите столбец Initial Size (MB) и Path.
6. *Настройка по ТЗ:* Если на сервере несколько логических процессоров, можно добавить еще один файл данных, чтобы распределить нагрузку. Рекомендуется создавать столько файлов данных tempdb, сколько ядер у процессора (но не более 8) .
 - o Нажмите кнопку **Add**.
 - o В поле Logical Name введите, например, tempdev2.
 - o В поле Initial Size поставьте такое же значение, как у первого файла (например, 8 МБ для Express, но лучше сразу задать 64 МБ или 1024 МБ для продакшна).
 - o Остальное оставьте как есть.
 - o *Важно:* На слабых ПК добавление второго файла может не дать прироста, но мы выполняем задание.

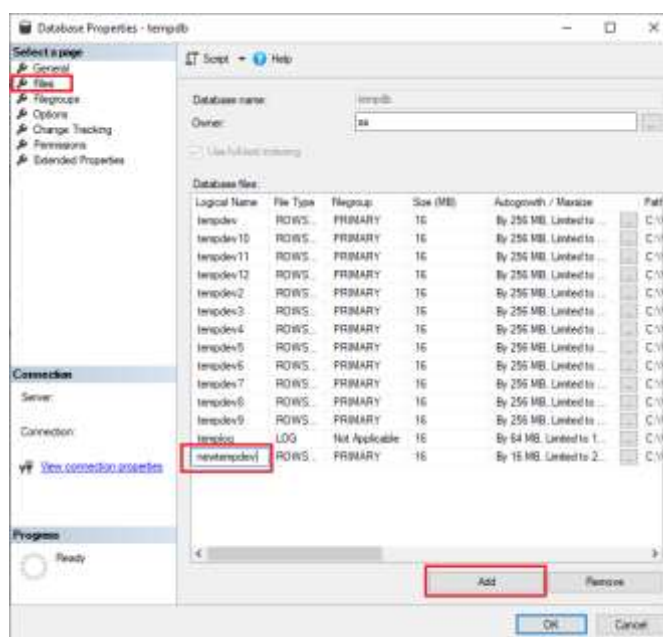


Рисунок 3. Добавление нового файла данных в TempDB

Задание 3.5: Настройка привязки к процессору (Affinity)

Суть: Если на сервере много программ, можно "прикрепить" SQL Server к конкретным ядрам процессора, чтобы другие программы не мешали ему своими вычислениями. Обычно оставляют AUTO.

1. Свойства сервера -> раздел **Processors** (Процессоры).
2. В секции "Processor affinity" (Связка процессоров) вы увидите сетку ядер (CPU 0, CPU 1, CPU 2...).
3. В ТЗ сказано: "Настроить в автоматическом режиме". Это значит, что нужно просто снять все галочки? Нет. Автоматический режим подразумевает, что SQL Server сам решит, на каких ядрах работать, если не ограничивать его принудительно.
 - Чтобы снять ручное ограничение, просто **снимите галочку** с чекбокса "**Automatically set processor affinity mask for all processors**" (Автоматически задавать маску связки для всех процессоров). Если галочка снята и все ядра серые — это и есть ручной режим.
 - Чтобы поставить AUTO, нужно **установить эту галочку**.
4. Установите галочку Automatically set.... Это позволит SQL Server использовать все доступные ядра.
5. Нажмите ОК.

Часть 4. Задания для самостоятельного выполнения (по вариантам)

Студенты делятся на 3 варианта (или выполняют все по очереди).

Вариант 1. Энергоэффективный сервер (Виртуалка / Ноутбук)

- **ТЗ:** Сервер работает на ноутбуке. Важно не перегревать процессор и экономить память для браузера и офисных программ. Всего ОЗУ 8 ГБ.

- **Задание:**

1. Рассчитайте и установите максимальный объем памяти для SQL Server (оставьте ОС не менее 2.5 ГБ).
2. Установите максимальную степень параллелизма (Max Degree of Parallelism) в 1. (Раздел Server Properties -> Advanced -> Max Degree of Parallelism). Это запретит запросам использовать несколько ядер, снижая нагрузку на ЦП.

Вариант 2. Выделенный сервер БД (Dedicated)

- **ТЗ:** Сервер мощный, 32 ГБ ОЗУ, 8 ядер. Кроме SQL Server, на нем ничего не работает.

- **Задание:**

1. Установите память так, чтобы максимально задействовать ресурсы, но оставить 4 ГБ для ОС.
2. В настройках TempDB создайте **4 файла данных** (tempdev2, tempdev3, tempdev4) одинакового размера.
3. Включите флаг "Optimize for Ad hoc Workloads" (Server Properties -> Advanced -> Optimize for Ad hoc Workloads -> True).

Вариант 3. Сервер для разработки (Development)

- **ТЗ:** На сервере часто создаются и удаляются тестовые базы. Важно, чтобы файлы БД не фрагментировались, а место освобождалось быстро.

- **Задание:**

1. Измените модель восстановления по умолчанию (Recovery model) для новых баз на **Simple**. (Server Properties -> Database Settings -> Default recovery model -> Simple).
2. Измените начальный размер файла по умолчанию для новых баз данных: Data = **50 MB**, Log = **25 MB**. (Там же, в разделе Database default locations, кнопка ... не меняет размер, это делается в model db. Откройте свойства системной базы **model** -> Files и измените размер для modeldev и modellog).

Контрольные вопросы:

1. Каковы минимальные требования к ОЗУ для SQL Server Express? А для Standard?
2. Почему важно ограничивать максимальный объем памяти для SQL Server?
3. Что такое TempDB и для чего она нужна?
4. В чем разница между файлом данных (.mdf) и файлом журнала (.ldf)?
5. Что такое "привязка к процессору" (processor affinity)?

Оформление отчета:

Отчет должен содержать:

1. Титульный лист: ФИО, группу, тему и цель работы.
2. Результаты анализа аппаратной платформы (скриншоты msinfo32).
3. Скриншоты основных этапов настройки (память, TempDB, affinity) с краткими пояснениями.
4. Вывод о том, подходит ли конфигурация вашего ПК для работы SQL Server и какие настройки были применены согласно вашему варианту.

Практическое занятие №2. Применение скриптов для инициализации баз данных, создания объектов внутри базы данных.

Цель занятия

1. Научиться создавать и выполнять скрипты T-SQL для создания объектов базы данных.
2. Освоить два подхода к созданию скриптов: написание "с нуля" и генерация на основе существующих объектов.
3. Сформировать понимание структуры скрипта инициализации (DDL) и наполнения (DML) базы данных.

Необходимое оборудование и ПО

- Персональный компьютер (ПК) с ОС Windows 10/11 Pro или Enterprise.
- Установленный экземпляр **Microsoft SQL Server (Express Edition или выше)**.
- Установленная среда **SQL Server Management Studio (SSMS)**.
- Текстовый редактор (Блокнот) для фиксации параметров.

Теоретическая часть (Краткий конспект)

Скрипт инициализации базы данных — это текстовый файл (обычно с расширением .sql), содержащий набор инструкций на языке T-SQL. При выполнении на сервере этот скрипт автоматически создает базу данных и все её объекты: таблицы, индексы, представления, хранимые процедуры и т.д. Это основа для развертывания базы данных на новом сервере или восстановления её структуры после сбоя.

Выделяют два типа команд в скриптах:

- **DDL (Data Definition Language — язык определения данных):** Команды, которые создают и изменяют структуру объектов (CREATE, ALTER, DROP).
- **DML (Data Manipulation Language — язык манипулирования данными):** Команды, которые работают с данными внутри объектов (INSERT, UPDATE, DELETE, SELECT).

Ход работы

Работа состоит из двух больших частей:

1. **Часть А (2 часа):** Ручное написание скрипта инициализации для учебной базы данных "Спортивная школа".
2. **Часть Б (2 часа):** Обратная задача — создание скрипта по готовой базе данных с помощью мастера **Generate Scripts**.

Часть А. Ручное создание скрипта инициализации

Представьте, что вы получили техническое задание разработать структуру базы данных для учета результатов в спортивной школе. Вам нужно создать базу данных SportSchool.

Задание А.1: Создание новой базы данных (DDL)

1. Запустите **SQL Server Management Studio** и подключитесь к вашему серверу.
2. На панели инструментов нажмите кнопку **New Query** (Новый запрос). Откроется окно редактора кода.
3. Скопируйте и вставьте в окно запроса следующий код. Это команда для создания базы данных. Мы сразу укажем логические имена файлов и пути (если есть права, можно изменить путь на D:\Data, но оставим значения по умолчанию).

```
sql
-- Скрипт инициализации базы данных SportSchool
-- Дата: [Сегодняшняя дата]
```

```

-- Отключаем все активные подключения к базе, если она есть, чтобы удалить её
-- (Этот блок используется для пересоздания базы во время разработки)
USE master;
GO

IF EXISTS (SELECT name FROM sys.databases WHERE name = N'SportSchool')
BEGIN
    ALTER DATABASE SportSchool SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
    DROP DATABASE SportSchool;
END
GO

-- Создаем базу данных SportSchool
CREATE DATABASE SportSchool
ON PRIMARY
( NAME = N'SportSchool', -- Логическое имя файла данных
  FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL15.MSSQLSERVER\MSSQL\DATA\SportSchool.mdf', -- Физический путь (может
отличаться)
  SIZE = 10MB,
  MAXSIZE = 100MB,
  FILEGROWTH = 10MB )
LOG ON
( NAME = N'SportSchool_Log', -- Логическое имя файла журнала
  FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL15.MSSQLSERVER\MSSQL\DATA\SportSchool_log.ldf',
  SIZE = 5MB,
  MAXSIZE = 50MB,
  FILEGROWTH = 10% );
GO

-- Переключаемся в контекст новой базы данных
USE SportSchool;
GO

PRINT 'База данных SportSchool успешно создана.';
GO

```

○ **Пояснение:**

- IF EXISTS... DROP DATABASE — полезная идиома для скриптов разработки. Она позволяет запускать скрипт многократно, каждый раз получая "чистую" базу.
 - USE SportSchool; — делает базу активной, чтобы все последующие команды выполнялись внутри неё .
4. Нажмите кнопку **! Execute** (Выполнить) или клавишу **F5**.
 5. В сообщениях вы должны увидеть Команды выполнены успешно. и База данных SportSchool успешно создана.
 6. В Object Explorer слева нажмите правой кнопкой на папку **Databases** и выберите **Refresh**. Новая база SportSchool должна появиться в списке.

Задание А.2: Создание таблиц (DDL)

Теперь создадим три основные таблицы: Sports (виды спорта), Coaches (тренеры), Athletes (спортсмены) и связующую таблицу Results (результаты соревнований) .

1. В том же окне запроса (или в новом, если закрыли) допишите следующий код **после** команды USE SportSchool; GO. Важно сохранять порядок: сначала база, потом таблицы.

sql

-- Создание таблицы Виды спорта

```
CREATE TABLE Sports (  
    SportID INT IDENTITY(1,1) PRIMARY KEY, -- Автоинкремент, первичный ключ  
    SportName NVARCHAR(100) NOT NULL UNIQUE, -- Уникальное название вида спорта  
    Description NVARCHAR(500) NULL  
);  
GO
```

-- Создание таблицы Тренеры

```
CREATE TABLE Coaches (  
    CoachID INT IDENTITY(1,1) PRIMARY KEY,  
    LastName NVARCHAR(50) NOT NULL,  
    FirstName NVARCHAR(50) NOT NULL,  
    Patronymic NVARCHAR(50) NULL,  
    BirthDate DATE NULL,  
    HireDate DATE NOT NULL DEFAULT GETDATE(), -- По умолчанию ставится  
сегодняшняя дата  
    Phone NVARCHAR(20) NULL,  
    Email NVARCHAR(100) NULL  
);  
GO
```

-- Создание таблицы Спортсмены

```
CREATE TABLE Athletes (  
    AthleteID INT IDENTITY(1,1) PRIMARY KEY,  
    LastName NVARCHAR(50) NOT NULL,  
    FirstName NVARCHAR(50) NOT NULL,  
    Patronymic NVARCHAR(50) NULL,  
    BirthDate DATE NOT NULL,  
    CoachID INT NULL, -- Внешний ключ на тренера (может быть NULL, если спортсмен  
временно без тренера)  
    SportID INT NOT NULL, -- Внешний ключ на вид спорта  
    EnrollmentDate DATE NOT NULL DEFAULT GETDATE(),  
    CONSTRAINT FK_Athletes_Coaches FOREIGN KEY (CoachID) REFERENCES  
Coaches(CoachID)  
    ON DELETE SET NULL, -- Если удалили тренера, у спортсменов в этом поле станет  
NULL  
    CONSTRAINT FK_Athletes_Sports FOREIGN KEY (SportID) REFERENCES  
Sports(SportID)  
    -- Запрещаем удалять вид спорта, если есть спортсмены  
);  
GO
```

-- Создание таблицы Результаты

CREATE TABLE Results (

ResultID INT IDENTITY(1,1) PRIMARY KEY,

AthleteID INT NOT NULL,

CompetitionName NVARCHAR(200) NOT NULL,

CompetitionDate DATE NOT NULL,

ResultValue NVARCHAR(50) NOT NULL, -- Например, "2:35.4", "5.80м", "10 очков"

Place SMALLINT NULL,

Points INT NULL,

CONSTRAINT FK_Results_Athletes FOREIGN KEY (AthleteID) REFERENCES

Athletes(AthleteID)

ON DELETE CASCADE -- Если удалили спортсмена, все его результаты удалятся автоматически

);

GO

PRINT 'Таблицы успешно созданы.';

GO

2. Выделите только что написанный блок кода (от первого CREATE TABLE до последнего GO) и нажмите **F5**.

3. В Object Explorer найдите базу SportSchool, раскройте папку **Tables**. Если таблицы не видны, нажмите **Refresh** (Обновить). Вы должны увидеть созданные таблицы.

Задание А.3: Наполнение таблиц тестовыми данными (DML)

База есть, таблицы есть, но они пусты. Наполним их небольшим количеством данных, чтобы проверить связи.

1. Снова откройте окно нового запроса (или очистите старое).

2. Введите и выполните следующий код. Обратите внимание на порядок вставки данных: сначала в главные таблицы (Sports, Coaches), потом в зависимые (Athletes, Results).

sql

USE SportSchool;

GO

-- Добавляем виды спорта

INSERT INTO Sports (SportName, Description) VALUES

(N'Плавание', N'Соревнования на воде, различные стили'),

(N'Легкая атлетика', N'Бег, прыжки, метания'),

(N'Шахматы', N'Интеллектуальный вид спорта');

GO

-- Добавляем тренеров

INSERT INTO Coaches (LastName, FirstName, Patronymic, BirthDate, Phone, Email) VALUES

(N'Иванов', N'Иван', N'Иванович', '1980-05-15', '+7(123)456-78-90', 'i.ivanov@email.com'),

(N'Петрова', N'Мария', N'Сергеевна', '1975-10-20', '+7(123)456-78-91',

'm.petrova@email.com'),

(N'Сидоров', N'Петр', N'Алексеевич', '1990-01-10', NULL, NULL);

GO

-- Добавляем спортсменов (нужно знать ID видов спорта и тренеров)

-- Проще всего посмотреть ID, выполнив SELECT * FROM Sports или Coaches

-- Предположим, что Плавание (SportID=1), Легкая атлетика (SportID=2), Шахматы (SportID=3)

INSERT INTO Athletes (LastName, FirstName, Patronymic, BirthDate, CoachID, SportID)

VALUES

(N'Смирнов', N'Алексей', N'Дмитриевич', '2010-06-01', 1, 1), -- Плавание, тренер Иванов
(N'Кузнецова', N'Елена', N'Игоревна', '2011-03-15', 2, 2), -- Легкая атлетика, тренер Петрова
(N'Васильев', N'Дмитрий', N'Николаевич', '2009-11-20', 1, 1), -- Плавание, тренер Иванов
(N'Николаев', N'Артем', N'Павлович', '2012-08-09', 3, 3); -- Шахматы, тренер Сидоров

GO

-- Добавляем результаты

INSERT INTO Results (AthleteID, CompetitionName, CompetitionDate, ResultValue, Place, Points) VALUES

(1, N'Чемпионат города по плаванию', '2023-12-10', '50м в/с - 25.3с', 2, 10),

(1, N'Новогодний заплыв', '2024-01-05', '100м в/с - 60.1с', 1, 15),

(2, N'Первенство области по прыжкам в длину', '2023-09-20', '5.20м', 3, 8),

(4, N'Турнир по быстрым шахматам', '2024-02-01', '5 место', 5, 4);

GO

PRINT 'Тестовые данные добавлены.';

GO

-- Проверим, что получилось

SELECT * FROM Sports;

SELECT * FROM Coaches;

SELECT * FROM Athletes;

SELECT * FROM Results;

GO

3. После выполнения внизу, во вкладке **Messages**, вы увидите результаты команд PRINT, а во вкладке **Results** — содержимое всех таблиц. Это подтверждает, что данные добавлены и связи работают.

Часть Б. Генерация скрипта по готовой базе данных

Теперь мы выполним обратную задачу. Представьте, что база данных SportSchool — это "production" база, и нам нужно выгрузить скрипт для её развертывания на сервере преподавателя или коллеги. Для этого мы воспользуемся встроенным мастером **Generate Scripts**.

Задание Б.1: Запуск мастера Generate Scripts

1. В **Object Explorer** найдите и щелкните правой кнопкой мыши по базе данных SportSchool.

2. В контекстном меню выберите **Tasks** (Задачи) -> **Generate Scripts...** (Создать скрипты...).

3. Откроется окно приветствия **Introduction** (Введение). Нажмите кнопку **Next** > .

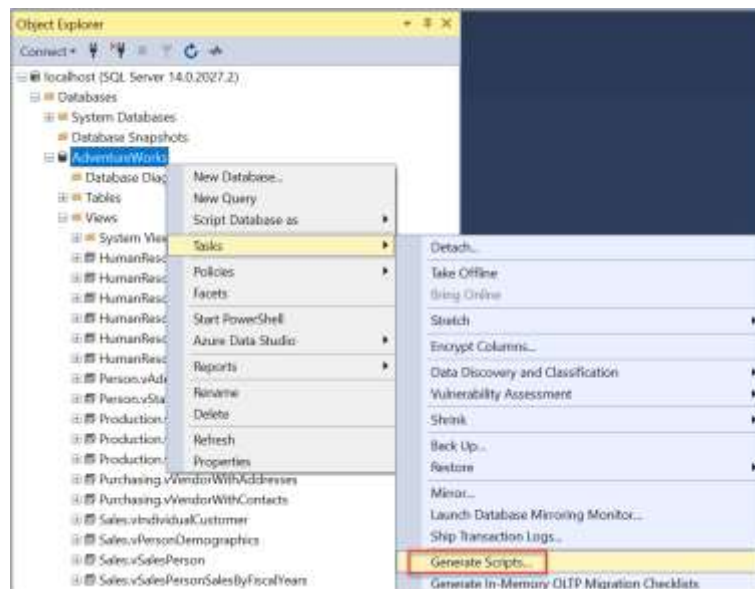


Рисунок 1 – Мастер Generate Scripts (Введение)

Задание Б.2: Выбор объектов для скриптования

1. На странице **Choose Objects** (Выбор объектов) у вас есть два варианта:
 - **Script entire database and all database objects** (Внести в скрипт всю базу данных целиком) — создаст скрипт для ВСЕГО, что есть в базе.
 - **Select specific database objects** (Выбрать конкретные объекты базы данных) — позволяет выбрать только нужные таблицы, процедуры и т.д.
2. Для нашей учебной цели выберите первый пункт: **"Script entire database and all database objects"**. Это гарантирует, что в скрипт попадет и структура, и настройки базы, и все объекты.
3. Нажмите **Next >**.

Задание Б.3: Настройка параметров скрипта (Критически важный шаг!)

Это самый важный этап. Здесь мы решаем, какой именно скрипт мы хотим получить: только структуру (DDL) или структуру + данные (DDL + DML).

1. На странице **Set Scripting Options** (Задание параметров скриптов) выберите место сохранения. Установите переключатель в положение **"Save to new query window"** (Сохранить в новом окне запроса), чтобы скрипт открылся прямо в SSMS.
2. Нажмите кнопку **Advanced** (Дополнительно) внизу страницы. Откроется окно **Advanced Scripting Options** (Дополнительные параметры скриптования).
3. В этом окне найдите и настройте следующие параметры:
 - **Types of data to script** (Типы данных для создания скриптов):
 - Schema only (Только схема) — если нужен только DDL (структура).
 - Data only (Только данные) — если нужен только DML (вставка данных).
 - Schema and data (Схема и данные) — если нужен полный скрипт инициализации со всеми данными.
 - Для нашего задания выберите **Schema and data**, чтобы скрипт полностью воссоздавал нашу базу "как есть".
 - **Script for Server Version** (Скрипт для версии сервера): Укажите вашу версию (например, SQL Server 2019), чтобы скрипт был совместим.
 - **Script USE DATABASE** (Скрипт USE DATABASE): Установите в **True**. Это добавит в начало скрипта команду **USE [SportSchool]**.
 - **Script DROP and CREATE** (Скрипт DROP и CREATE): Рекомендуется оставить **Script CREATE** (если вы хотите, чтобы при выполнении скрипта база просто создавалась)

или выбрать **Script DROP and CREATE** (если вы хотите, чтобы скрипт удалял существующую базу перед созданием, как мы делали вручную).

- **Table/View Options -> Script Indexes** (Индексы скриптов): Установите **True**, чтобы скрипт создавал и индексы (первичные ключи — это тоже индексы) .
- **Permissions** (Разрешения): Можно оставить **False**.
- 4. Нажмите **OK**, чтобы закрыть окно дополнительных параметров, и затем **Next >**.

Задание Б.4: Завершение работы мастера и анализ результата

1. На странице **Summary** (Сводка) проверьте выбранные настройки и нажмите **Next >**.
2. Мастер начнет генерацию. На странице **Save or Publish Scripts** (Сохранение или публикация скриптов) отобразится прогресс.
3. После завершения нажмите **Finish** (Готово).
4. **Перед вами откроется новое окно запроса, заполненное сгенерированным скриптом.** Он будет огромным (несколько сотен строк)!
5. **Задание:** Прокрутите скрипт и найдите в нем:
 - Секцию создания базы данных (в самом начале).
 - Команды CREATE TABLE для каждой из наших таблиц. Обратите внимание, как мастер описал ограничения (FOREIGN KEY).
 - **Самый конец скрипта** — там вы увидите множество команд INSERT INTO ... Это и есть наши тестовые данные, преобразованные в формат DML .

Часть 3. Задания для самостоятельного выполнения (по вариантам)

Студенты делятся на 3 варианта. Задание рассчитано на оставшееся время (около 1.5 часов) и выполняется в скрипте, аналогичном Части А.

Общее задание для всех вариантов:

Дополнить базу данных SportSchool новыми объектами, согласно вашему варианту. Все изменения должны быть оформлены в виде скрипта (новый запрос), который можно будет показать преподавателю.

Вариант 1. Расширение функционала "Спортивное оборудование"

- **ТЗ:** Для учета инвентаря необходимо добавить таблицу Equipment.
- **Задание:**
 1. Создать таблицу Equipment с полями: EquipmentID (INT, IDENTITY, PK), EquipmentName (NVARCHAR(100), NOT NULL), Quantity (INT, NOT NULL, DEFAULT 1), Condition (NVARCHAR(50), NULL), SportID (INT, внешний ключ к таблице Sports).
 2. Написать INSERT скрипт для добавления 3-5 единиц инвентаря (например, "Мяч футбольный", "Скакалка", "Гантели") с привязкой к существующим видам спорта (используйте подзапросы или известные ID).
 3. Написать SELECT запрос с JOIN, который покажет название инвентаря и название вида спорта, к которому он относится.

Вариант 2. Расширение функционала "Расписание тренировок"

- **ТЗ:** Для планирования занятий необходимо добавить таблицу Schedule.
- **Задание:**
 1. Создать таблицу Schedule с полями: ScheduleID (INT, IDENTITY, PK), WeekDay (NVARCHAR(20), NOT NULL), StartTime (TIME, NOT NULL), EndTime (TIME, NOT NULL), CoachID (INT, внешний ключ к Coaches), SportID (INT, внешний ключ к Sports).
 2. Добавить ограничение (CONSTRAINT) для проверки, что EndTime > StartTime.
 3. Написать INSERT скрипт для добавления 3-5 записей в расписание.
 4. Написать SELECT запрос, который выведет расписание с ФИО тренера и названием вида спорта.

Вариант 3. Расширение функционала "Категории спортсменов"

- **ТЗ:** Необходимо добавить справочник спортивных разрядов и связать его со спортсменами.
- **Задание:**
 1. Создать таблицу Ranks (Разряды) с полями: RankID (INT, IDENTITY, PK), RankName (NVARCHAR(50), NOT NULL UNIQUE), MinAge (INT, NULL) — минимальный возраст для получения разряда.
 2. Изменить таблицу Athletes, добавив в нее столбец RankID (INT, NULL), который будет внешним ключом к таблице Ranks. Используйте команду ALTER TABLE.
 3. Наполнить таблицу Ranks несколькими разрядами ("3-й юношеский", "2-й юношеский", "1-й взрослый" и т.д.).
 4. Обновить (UPDATE) данные в таблице Athletes, проставив нескольким спортсменам разряды.
 5. Написать SELECT запрос с JOIN, который выведет всех спортсменов и их разряды (если есть).

Контрольные вопросы

1. В чем разница между командами DDL и DML? Приведите по 2 примера на каждый тип.
2. Для чего в скрипте создания таблицы используется ключевое слово IDENTITY?
3. Что произойдет, если сначала попытаться вставить данные в таблицу Athletes, а только потом — в таблицу Sports? Почему?
4. В какой ситуации может пригодиться мастер **Generate Scripts** (Создать скрипты) в SSMS?
5. Какая настройка в мастере Generate Scripts отвечает за то, попадут ли в скрипт уже имеющиеся в таблицах данные?

Оформление отчета:

Отчет должен содержать:

1. Титульный лист: ФИО, группу, тему и цель работы.
2. Скриншоты или текстовые вставки кода для каждого из этапов Части А (создание БД, таблиц, вставка данных).
3. Скриншоты настроек мастера Generate Scripts (особенно окна "Advanced").
4. Финальный скрипт (или его фрагменты), сгенерированный мастером, демонстрирующий наличие данных (INSERT).
5. Код и результаты выполнения задания по варианту.
6. Вывод о практической пользе скриптов для инициализации и развертывания баз данных.

Практическое занятие №3. Создание и настройка балансировки подключений на сервер.

Цели работы:

1. Изучить теоретические основы балансировки подключений к SQL Server.
2. Научиться настраивать сетевые протоколы SQL Server для приема подключений.
3. Освоить создание нескольких точек входа (endpoints) на один сервер через настройку статических портов для разных экземпляров.
4. Научиться подключаться к разным экземплярам с клиентских машин, имитируя распределение нагрузки.
5. Закрепить навыки работы с Диспетчером конфигурации SQL Server и брандмауэром Windows.

Оборудование и ПО:

- **Серверная машина (одна из самых мощных в классе):**
 - ОС: Windows Server 2016/2019 или Windows 10/11 Pro.
 - Установленный SQL Server (например, SQL Server 2019 Developer Edition) с экземпляром по умолчанию (MSSQLSERVER) и одним именованным экземпляром (SQLBALANCE или INSTANCE2).
- **Клиентские машины (остальные ПК студентов):**
 - ОС: Windows 10/11.
 - Установленный SQL Server Management Studio (SSMS).
- **Сетевое окружение:** Рабочая группа или простой домен (если есть). Все ПК должны быть в одной локальной сети и "видеть" друг друга по IP-адресам или имени.

3. Теоретическая часть

Балансировка подключений — это метод распределения входящих сетевых запросов (клиентских подключений) между несколькими ресурсами. В контексте SQL Server это может означать:

- **Распределение только для чтения:** Направление отчетов и аналитических запросов на реплики (Always On Availability Groups).
- **Мультиплексирование:** Обработка большого количества коротких подключений через пул соединений (Connection Pooling).
- **Изоляция экземпляров:** Разные базы данных или приложения подключаются к разным экземплярам SQL Server на одном физическом сервере.

Поскольку мы не можем ставить сторонний балансировщик (как Loadbalancer.org или Azure Load Balancer), мы симулируем балансировку, создав несколько "входов" на сервер:

1. **Экземпляр по умолчанию** (порт 1433) — будет принимать подключения "приложения №1".
2. **Именованный экземпляр** (статический порт, например, 1500) — будет принимать подключения "приложения №2".

Клиентские ПК будут сознательно подключаться к разным портам, распределяя нагрузку на CPU и RAM сервера между разными экземплярами SQL Server.

Ход работы

Этап 1: Подготовка сервера. Настройка сетевых протоколов

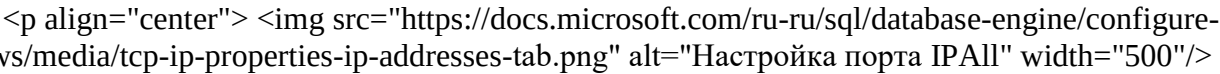
Задача: Включить протокол TCP/IP для экземпляров SQL Server и настроить статические порты.

1. **Запуск Диспетчера конфигурации SQL Server.**
 - На **серверной машине** нажмите Пуск -> Все программы -> Microsoft SQL Server [Версия] -> Средства настройки -> Диспетчер конфигурации SQL Server.
 - *Для быстрого доступа:* Нажмите Win + R, введите SQLServerManager[Версия].msc (например, SQLServerManager15.msc для 2019).

2. **Настройка экземпляра по умолчанию (MSSQLSERVER).**
 - В левом меню выберите "Сетевая конфигурация SQL Server" -> "Протоколы для MSSQLSERVER".
 - В правой части окна найдите протокол TCP/IP.
 - Если его статус "Отключено", нажмите на нем правой кнопкой мыши -> "Включить".
 - Появится предупреждение о том, что изменения вступят в силу после перезапуска службы. Нажмите **ОК**.
3. **Настройка именованного экземпляра (SQLBALANCE).**
 - Если второй экземпляр еще не создан, создайте его заранее или используйте существующий.
 - В левом меню выберите "Протоколы для ИМЯ_ВАШЕГО_ЭКЗЕМПЛЯРА" (например, SQLBALANCE).
 - Аналогично включите протокол TCP/IP.
4. **Проверка и назначение статических портов.**
 - В левом меню выберите "Протоколы для MSSQLSERVER".
 - Снова найдите протокол TCP/IP в списке, нажмите на нем правой кнопкой -> "Свойства".
 - Перейдите на вкладку "IP-адреса".
 - Прокрутите список вниз до раздела IPAll.
 - Очистите поле "TCP-порт", если там что-то есть.
 - В поле "TCP-динамические порты" убедитесь, что там пусто или стоит 0.

Если там есть значение (например, порт, назначенный динамически), удалите его. Мы настраиваем статику.

 - В поле "TCP-порт" для раздела IPAll введите 1433 (стандартный порт для дефолтного экземпляра).



* Нажмите **ОК**. * Повторите те же шаги для именованного экземпляра ('SQLBALANCE'), но в разделе 'IPAll' для "TCP-порт" укажите другое значение, например, '1500'. Это будет его статический порт.
5. **Перезапуск служб SQL Server.**
 - В левом меню Диспетчера конфигурации выберите "Службы SQL Server".
 - Найдите службы SQL Server (MSSQLSERVER) и SQL Server (SQLBALANCE).
 - Нажмите на каждую правой кнопкой мыши и выберите "Перезапустить".

Дождитесь окончания перезапуска обеих служб.

Этап 2: Настройка брандмауэра Windows

Задача: Разрешить входящие подключения к портам 1433 и 1500.

1. **Открытие оснастки брандмауэра.**
 - На серверной машине нажмите Win + R, введите wf.msc и нажмите Enter. Откроется окно "Брандмауэр Windows в режиме повышенной безопасности".
2. **Создание правила для порта 1433.**
 - В левом меню выберите "Правила для входящих подключений".
 - В правом меню (Действия) нажмите "Создать правило...".
 - **Тип правила:** Выберите "Для порта" -> Далее.
 - **Протокол и порты:**
 - Протокол: TCP.
 - Определенные локальные порты: 1433 -> Далее.
 - **Действие:** Выберите "Разрешить подключение" -> Далее.
 - **Профиль:** Оставьте все галки (Доменный, Частный, Публичный) -> Далее.
 - **Имя:** Введите имя, например, SQL Server Default Instance 1433. Можно добавить

описание. -> **Готово.**

3. **Создание правила для порта 1500.**

○ Повторите шаги из п.2, но на шаге "Протокол и порты" укажите порт 1500. Назовите правило, например, SQL Server Named Instance 1500.

Этап 3: Настройка ограничения подключений (User Connections)

Задача: Ограничить количество подключений к экземпляру, чтобы увидеть эффект "перегрузки".

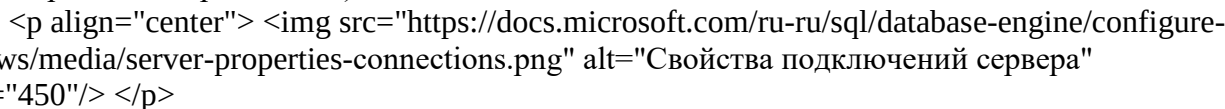
Выполняется на сервере через SSMS.

1. Подключитесь к экземпляру MSSQLSERVER.
2. В обозревателе объектов нажмите правой кнопкой мыши на самом сервере (вверху) и выберите **"Свойства"**.

3. Перейдите на страницу **"Подключения"**.

4. В разделе **"Параметры подключений"** найдите поле **"Максимальное число одновременных подключений (0 - не ограничено)"**.

5. Установите значение 3 (или небольшое число, например, 5, чтобы проще было достичь предела во время теста).


width="450"/>

6. Нажмите **ОК**. Обратите внимание, что для вступления в силу может потребоваться перезагрузка (хотя это динамическая настройка, но старые подключения могут сохраняться).

7. **Важно:** Верните ограничение обратно на 0 или большое значение после выполнения всех заданий, чтобы не парализовать работу сервера!

Этап 4: Клиентская часть. Подключение к разным экземплярам

Задача: С разных клиентских ПК подключиться к настроенным экземплярам, используя разные строки подключения, и пронаблюдать за распределением сессий.

1. **Определение имени или IP-адреса сервера.**

○ На серверной машине откройте командную строку (cmd) и введите команду ipconfig.
○ Найдите IPv4-адрес (например, 192.168.1.100). Запишите его. Также запишите сетевое имя сервера (на вкладке "Имя компьютера" в свойствах системы).

2. **Подключение с клиента №1 (группа А) к экземпляру по умолчанию.**

○ На клиентском ПК запустите SSMS.
○ В окне **"Подключение к серверу"**:
▪ **Имя сервера:** Введите tcp:192.168.1.100,1433 (или tcp:Имя_Сервера,1433).
▪ **Пояснение:** Префикс tcp: принудительно использует TCP/IP, а запятая и порт указывают конкретный адрес.

▪ **Проверка подлинности:** Выберите проверку подлинности Windows (или SQL Server, если настроено).

○ Нажмите **"Подключиться"**.

3. **Подключение с клиента №2 (группа Б) к именованному экземпляру.**

○ На другом клиентском ПК запустите SSMS.
○ **Имя сервера:** Введите tcp:192.168.1.100,1500.
○ Нажмите **"Подключиться"**.
4. **Мониторинг подключений на сервере.**
○ На серверной машине откройте новый экземпляр SSMS.
○ Подключитесь к любому экземпляру и выполните следующий SQL-запрос (он покажет все активные подключения со всех экземпляров, но лучше выполнять на каждом отдельно):

```
sql
SELECT
```

```

session_id AS [SPID],
login_name AS [Логин],
host_name AS [Имя_клиента],
program_name AS [Программа],
net_transport AS [Транспорт],
protocol_type AS [Протокол],
auth_scheme AS [Аутентиф],
client_net_address AS [IP_адрес],
local_net_address AS [Сервер_принял_на_IP],
local_tcp_port AS [Порт_сервера]
FROM sys.dm_exec_sessions
WHERE is_user_process = 1; -- Только пользовательские процессы
GO

```

○ **Анализ:** В результатах вы увидите, что клиенты, подключившиеся через tcp:... ,1433, имеют значение в колонке local_tcp_port = 1433, а те, кто подключался через порт 1500, имеют соответствующее значение. Это и есть наша симуляция балансировки — разные потоки данных пришли на разные "входные двери" сервера.

Этап 5: Имитация перегрузки и анализ

Задача: Достичь лимита подключений к экземпляру и пронаблюдать ошибку.

1. На клиенте №1 (группа А), который подключен к порту 1433, откройте **еще одно окно SSMS** и попробуйте подключиться к тому же экземпляру (tcp:192.168.1.100,1433). Повторите это несколько раз, пока не достигнете лимита в 3-5 подключений, установленного на Этапе 3.

2. Когда лимит будет исчерпан, при очередной попытке подключения вы увидите сообщение об ошибке:

"Сервер достиг своего максимального числа подключений. (Microsoft SQL Server, Ошибка: 1130)"

3. Теперь, не закрывая лишние окна на клиенте №1, попробуйте подключиться к **порту 1500** с клиента №1. Подключение должно пройти успешно, так как это другой экземпляр со своим счетчиком подключений. Это наглядно демонстрирует, как разделение на порты помогает изолировать нагрузку разных приложений.

5. Индивидуальные задания

Каждому студенту (или группе студентов) выдается свой вариант.

1. Настройте статический порт для вашего именованного экземпляра согласно варианту (см. таблицу ниже).

2. Настройте правило в брандмауэре для этого порта.

3. С клиентской машины подключитесь к серверу, используя ваш уникальный порт.

4. Выполните запрос к sys.dm_exec_sessions и сохраните скриншот, где видно ваше подключение и ваш порт.

5. *Варианты заданий (раздать студентам):*

Вариант	Имя экземпляра	Порт	База данных для проверки
1	INST_A	1601	AdventureWorks2017
2	INST_B	1602	AdventureWorksDW2017
3	INST_A	1610	AdventureWorks2017
4	INST_B	1620	AdventureWorksDW2017

Контрольные вопросы

1. Почему при изменении протоколов в Диспетчере конфигурации требуется

перезапуск службы SQL Server?

2. Для чего в строке подключения мы использовали префикс tcp:?
3. В чем разница между динамическим и статическим портом для SQL Server?
4. Какая системная динамическая view (DMV) позволяет увидеть активные сессии?
5. Что произойдет с подключениями, если в брандмауэре не открыть порт 1500, но служба SQL Server на нем запущена?

Содержание отчета:

Отчет должен содержать:

1. Титульный лист: ФИО, группу, тему и цель работы.
2. Схема сети лабораторного стенда (сервер, клиенты, IP-адреса, порты).
3. Скриншоты ключевых этапов:
 - Настройка статического порта в Диспетчере конфигурации.
 - Правило брандмауэра.
 - Результат запроса sys.dm_exec_sessions, демонстрирующий подключение с клиента на заданном порту.
4. Ответы на контрольные вопросы.
5. Вывод о проделанной работе (что нового узнали, с какими трудностями столкнулись).

Практическое занятие №4. Создание пользователей и назначение ролей. Управление правами доступа пользователей на уровне сервера, баз данных и данных.

Цели работы:

1. Изучить модель безопасности SQL Server: иерархия уровней (сервер, база данных, схема, объект).
2. Научиться создавать логины на уровне сервера и пользователей в базах данных.
3. Освоить назначение предопределенных серверных ролей и ролей баз данных.
4. Научиться создавать пользовательские роли и управлять правами на уровне объектов (таблицы, представления).
5. Закрепить навыки проверки прав доступа с помощью выполнения кода от имени другого пользователя.

Оборудование и ПО:

- **Сервер баз данных:**
 - Один из компьютеров в классе (лучше наиболее мощный) с установленным SQL Server (версия Developer или Express) и включенным смешанным режимом аутентификации (SQL Server и Windows Authentication).
 - На сервере должны быть созданы учебные базы данных (например, TestDB и ShopDB) с несколькими таблицами для тестирования прав.
- **Клиентские машины (ПК студентов):**
 - Установленный SQL Server Management Studio (SSMS).
 - Возможность подключения к серверу по сети (IP-адрес или имя компьютера).
- **Примечание:** Если развернуть отдельный сервер невозможно, каждый студент может работать со своим локальным экземпляром SQL Server Express (при условии, что он установлен и запущен). В этом случае роль «сервера» выполняет его собственная машина, а подключения выполняются через localhost или ..

Теоретическая часть

Безопасность SQL Server строится на двух основных уровнях:

- **Уровень сервера** — управление доступом к самому экземпляру SQL Server.
 - **Логин** — учётная запись для входа на сервер (Windows или SQL Server).
 - **Серверные роли** — предопределённые наборы прав на уровне всего сервера (например, sysadmin — полные права, securityadmin — управление логинами, dbcreator — создание БД).
- **Уровень базы данных** — управление доступом к конкретной базе данных.
 - **Пользователь** — сопоставляется с логином и действует внутри БД.
 - **Роли базы данных** — предопределённые (db_owner, db_datareader, db_datawriter и др.) и пользовательские роли, которым можно давать разрешения на объекты (таблицы, процедуры и т.д.).
- **Уровень объектов** — непосредственные права на таблицы, представления, хранимые процедуры (GRANT, DENY, REVOKE).

Схема взаимодействия:

text

Серверный логин → Пользователь БД → Членство в ролях БД → Права на объекты

Для проверки прав используется инструкция EXECUTE AS USER =

'имя_пользователя' или EXECUTE AS LOGIN = 'логин'.

Ход работы

Этап 1: Подготовка среды (30 минут)

1. Узнайте имя/адрес сервера SQL Server.

- Если работаете с сервером преподавателя, получите у него **IP-адрес** или **имя компьютера** и имя экземпляра (например, 192.168.1.10 или SERVERNAME\SQLEXPRESS).
 - Если работаете с локальным экземпляром, именем сервера будет localhost или . (точка).
 - 2. **Запустите SSMS и подключитесь к серверу.**
 - Нажмите Пуск → Microsoft SQL Server Tools → Microsoft SQL Server Management Studio.
 - В окне подключения укажите имя сервера, выберите **проверку подлинности Windows** (или SQL Server, если вам дали логин администратора).
 - Нажмите "**Подключиться**".
- Скриншот: окно «Connect to Server» с заполненным полем Server name и выбранной аутентификацией.*
- 3. **Создайте тестовую базу данных (если её нет).**
 - В обозревателе объектов щёлкните правой кнопкой мыши на папке "**Базы данных**" → "**Создать базу данных...**".
 - В поле "**Имя базы данных**" введите TestDB. Остальные параметры оставьте по умолчанию.
 - Нажмите "**ОК**".
 - 4. **Создайте в базе TestDB две таблицы для проверки прав.**
 - Нажмите на базу TestDB → "**Создать запрос**" (или нажмите Ctrl+N для нового запроса, предварительно выбрав TestDB в выпадающем списке баз).
 - Выполните следующий скрипт:

sql

```
CREATE TABLE Products (
    ProductID int PRIMARY KEY,
    ProductName nvarchar(100),
    Price decimal(10,2)
);
```

```
CREATE TABLE Orders (
    OrderID int PRIMARY KEY,
    CustomerName nvarchar(100),
    OrderDate datetime
);
```

```
INSERT INTO Products VALUES
(1, 'Ноутбук', 45000.00),
(2, 'Мышь', 1200.50),
(3, 'Коврик', 350.00);
```

```
INSERT INTO Orders VALUES
(101, 'Иванов', GETDATE()),
(102, 'Петров', GETDATE());
```

- Нажмите "**Выполнить**" (F5).

Скриншот: окно редактора запросов с выполненным скриптом и сообщением об успехе.

Этап 2: Создание логинов на уровне сервера

Мы создадим два SQL-логина: Manager (менеджер) и Clerk (клерк).

- 1. **Откройте окно нового запроса** (или используйте то же) и убедитесь, что подключение идёт от имени администратора (например, sa или ваша Windows-учётка с правами sysadmin).
- 2. **Создайте логин Manager:**

sql

```
CREATE LOGIN Manager WITH PASSWORD = 'Pa$$w0rd123';
```

Примечание: пароль должен соответствовать политике сложности Windows. Если политика отключена, можно использовать простой пароль, но в учебных целях лучше соблюдать сложность.

3. **Создайте логин Clerk:**

sql

```
CREATE LOGIN Clerk WITH PASSWORD = 'ClerkPass123';
```

4. **Проверьте созданные логины** через объектный обозреватель:

- В обозревателе объектов раскройте папку **"Безопасность"** → **"Имена входа"**. Там должны появиться Manager и Clerk.

Скриншот: папка Logins с двумя новыми логинами.

5. **Назначьте серверные роли (по желанию).**

- Например, дадим Manager право создавать базы данных (роль dbcreator), а Clerk оставим без дополнительных серверных прав.

- Выполните:

sql

```
ALTER SERVER ROLE dbcreator ADD MEMBER Manager;
```

- Либо сделайте через графический интерфейс: правый клик на логине Manager → **"Свойства"** → вкладка **"Серверные роли"** → отметьте dbcreator → ОК.

Скриншот: окно свойств логина, вкладка Server Roles с отмеченной ролью dbcreator.

Этап 3: Создание пользователей в базе данных и назначение ролей БД

Теперь для каждого логина создадим пользователя в базе TestDB и назначим права.

1. **Переключитесь в контекст базы TestDB.** Можно выполнить инструкцию USE TestDB; или выбрать TestDB в выпадающем списке баз.

2. **Создайте пользователя для логина Manager:**

sql

```
CREATE USER Manager FOR LOGIN Manager;
```

3. **Создайте пользователя для логина Clerk:**

sql

```
CREATE USER Clerk FOR LOGIN Clerk;
```

4. **Проверьте созданных пользователей:** в обозревателе объектов раскройте TestDB → **"Безопасность"** → **"Пользователи"**. Там должны появиться Manager и Clerk.

5. **Назначьте роли базы данных:**

- Дадим менеджеру право читать и изменять данные во всех таблицах (роль db_datareader + db_datawriter), а клерку — только чтение (db_datareader).

- Выполните:

sql

```
ALTER ROLE db_datareader ADD MEMBER Manager;
```

```
ALTER ROLE db_datawriter ADD MEMBER Manager;
```

```
ALTER ROLE db_datareader ADD MEMBER Clerk;
```

- Альтернативно можно сделать через графический интерфейс: свойства пользователя → вкладка **"Членство"** → отметить нужные роли.

Скриншот: окно свойств пользователя Manager с отмеченными ролями.

6. **Создайте пользовательскую роль и назначим ей специфические права.**

- Например, создадим роль SalesRole, которая сможет только просматривать таблицу Products и вставлять записи в Orders.

- Выполните:

sql

```
CREATE ROLE SalesRole;
```

```
GRANT SELECT ON Products TO SalesRole;
```

```
GRANT INSERT ON Orders TO SalesRole;
```

○ Добавим пользователя Clerk в эту роль (теперь у него будут права и от db_datareader, и от SalesRole — объединение).

```
sql
```

```
ALTER ROLE SalesRole ADD MEMBER Clerk;
```

Скриншот: сообщение о выполнении команд создания роли и назначения прав.

Этап 4: Проверка прав доступа

Мы проверим, какие действия могут выполнять пользователи Manager и Clerk.

Проверка 1: Подключение от имени Manager

1. В SSMS откройте **новое окно запроса** (можно через **Файл** → **Создать** → **Запрос ядра СУБД**).

2. В окне подключения укажите тот же сервер, но выберите **проверку подлинности SQL Server**.

○ **Имя входа:** Manager

○ **Пароль:** Pa\$\$w0rd123

3. Подключитесь.

4. Выполните следующие запросы и проанализируйте результаты:

```
sql
```

```
-- Попытка чтения из Products (должно быть успешно)
```

```
SELECT * FROM TestDB.dbo.Products;
```

```
-- Попытка вставки в Products (должно быть успешно, т.к. Manager имеет db_datawriter)
```

```
INSERT INTO TestDB.dbo.Products (ProductID, ProductName, Price)
```

```
VALUES (4, 'Клавиатура', 2500.00);
```

```
-- Проверка вставки
```

```
SELECT * FROM TestDB.dbo.Products;
```

```
-- Попытка создать новую таблицу (проверка серверной роли dbcreator)
```

```
CREATE TABLE TestDB.dbo.Test (id int); -- должно выполняться успешно
```

Скриншот: результаты выполнения запросов, видна новая запись и созданная таблица.

Проверка 2: Подключение от имени Clerk

1. Аналогично откройте ещё одно окно подключения от имени Clerk (пароль ClerkPass123).

2. Выполните:

```
sql
```

```
-- Чтение Products (должно быть успешно, т.к. Clerk в db_datareader)
```

```
SELECT * FROM TestDB.dbo.Products;
```

```
-- Попытка вставить запись в Products (должна быть ошибка, т.к. нет прав на запись)
```

```
INSERT INTO TestDB.dbo.Products (ProductID, ProductName, Price)
```

```
VALUES (5, 'Монитор', 12000.00); -- ОШИБКА!
```

```
-- Вставка в Orders (должна быть успешно, т.к. есть право INSERT через SalesRole)
```

```
INSERT INTO TestDB.dbo.Orders (OrderID, CustomerName, OrderDate)
```

```
VALUES (103, 'Сидоров', GETDATE()); -- успешно
```

```
-- Попытка создать таблицу (ошибка, т.к. нет серверной роли dbcreator)
```

```
CREATE TABLE TestDB.dbo.Test2 (id int); -- ОШИБКА!
```

Скриншот: окно с сообщением об ошибке INSERT (разрешение запрещено) и успешным

INSERT в Orders.

Проверка 3: Использование EXECUTE AS для тестирования прав без повторного входа

Можно переключить контекст выполнения внутри одного окна (требуется права IMPERSONATE).

1. Подключитесь под администратором.
2. Выполните:
sql
USE TestDB;
-- Переключение на пользователя Clerk
EXECUTE AS USER = 'Clerk';
-- Проверка прав (текущий пользователь теперь Clerk)
SELECT USER_NAME(); -- должно вернуть 'Clerk'
SELECT * FROM Products; -- успех
INSERT INTO Products VALUES (6, 'SSD', 8000); -- ошибка
-- Возврат к исходному пользователю
REVERT;
SELECT USER_NAME(); -- снова ваше имя

Скриншот: результаты запросов, показывающие USER_NAME() и ошибку.

Этап 5: Самостоятельное задание по вариантам (30 минут)

Каждому студенту (или паре) выдаётся вариант. Необходимо выполнить настройку прав согласно варианту и продемонстрировать результат.

Примерные варианты:

Вариант	Логин	Пароль	Серверная роль	Роль БД	Дополнительные права на объекты
1	User1	User1Pass	-	-	SELECT на Products, INSERT на Orders
2	User2	User2Pass	securityadmin	db_owner	все права
3	User3	User3Pass	-	db_datareader	-
4	User4	User4Pass	-	создать роль ReportReaders	GRANT SELECT ON Products TO ReportReaders; добавить пользователя в роль

Задание:

1. Создать логин по варианту.
2. Создать пользователя в TestDB.
3. Назначить указанные серверные роли и роли БД, а также индивидуальные права.
4. Проверить права, подключившись под этим пользователем (или используя EXECUTE AS).

Контрольные вопросы

1. В чем разница между логином и пользователем базы данных?
2. Какие predefined серверные роли вы знаете? Опишите назначение sysadmin, securityadmin, dbcreator.
3. Что такое роль базы данных? Как создать пользовательскую роль?
4. Каким образом можно дать пользователю право только на чтение данных из определённой таблицы?
5. Для чего используется команда EXECUTE AS?
6. Что произойдёт, если пользователь является членом двух ролей, одна из которых даёт право SELECT, а другая запрещает (DENY)?

Содержание отчета

Отчет должен содержать:

1. Титульный лист: ФИО, группу, тему и цель работы.
2. Описание выполненных действий по этапам (кратко).
3. Скриншоты ключевых моментов:
 - Создание логинов и пользователей (объектный обозреватель).
 - Назначение ролей (графически или текстом запроса).
 - Результаты проверки прав (успешные и ошибочные запросы).
 - Выполнение индивидуального задания (согласно варианту).
4. Ответы на контрольные вопросы.
5. Вывод: что нового узнали, какие навыки приобрели.

Дополнительные материалы

Для быстрого создания логинов и пользователей можно использовать графический интерфейс SSMS: папка Безопасность → Имена входа (правый клик → Создать имя входа). На вкладке "Пользователи" можно сразу сопоставить логин с пользователем в выбранных базах данных и назначить роли БД.

Шпаргалка по основным командам T-SQL для управления доступом:

CREATE LOGIN, ALTER LOGIN, DROP LOGIN

CREATE USER, ALTER USER, DROP USER

CREATE ROLE, ALTER ROLE, DROP ROLE

GRANT, DENY, REVOKE

ALTER SERVER ROLE ... ADD MEMBER ...

ALTER ROLE ... ADD MEMBER ...

Практическое занятие №5. Создание сложной структуры ролей. Использование методов шифрования паролей. Настройка аутентификации клиентского приложения. Применять предопределенные роли.

Цели работы:

1. Изучить иерархию ролей в SQL Server и научиться создавать сложные структуры с вложенностью ролей .
2. Освоить применение предопределенных (фиксированных) ролей сервера и баз данных для решения типовых задач администрирования .
3. Научиться реализовывать шифрование (хэширование) паролей при хранении в базе данных с использованием современных алгоритмов .
4. Настроить аутентификацию клиентского приложения через проверку хэшированных паролей.
5. Исследовать особенности наследования прав и приоритеты запретов (DENY) над разрешениями (GRANT).

Оборудование и ПО:

- **Сервер баз данных (или локальный экземпляр):**
 - SQL Server (Developer или Express Edition) со включенным смешанным режимом аутентификации (SQL Server and Windows Authentication Mode).
 - База данных AdventureWorks или любая учебная база (можно создать тестовую БД).
- **Клиентские машины (ПК студентов):**
 - SQL Server Management Studio (SSMS) установленный и настроенный для подключения к серверу.
- **Примечание:** Все действия выполняются встроенными средствами SQL Server и SSMS, установка дополнительного ПО не требуется.

Теоретическая часть

3.1. Иерархия ролей и вложенность

SQL Server поддерживает создание **пользовательских ролей** на уровне сервера и на уровне базы данных . Эти роли могут включать в себя другие роли, образуя иерархию. Это позволяет реализовать принцип минимальных привилегий: базовые роли наделяются правами на конкретные объекты, а затем включаются в состав более крупных ролей, соответствующих должностным обязанностям .

3.2. Предопределенные роли базы данных

В каждой базе данных существуют фиксированные роли, разрешения которых нельзя изменить :

- **db_owner** — полный контроль над базой данных.
- **db_datareader** — чтение данных из всех пользовательских таблиц.
- **db_datawriter** — добавление, изменение, удаление данных.
- **db_ddladmin** — выполнение любых команд DDL (создание/изменение объектов).
- **db_securityadmin** — управление разрешениями и членством в ролях.
- **db_denydatareader / db_denydatawriter** — запрет на чтение/запись.

3.3. Шифрование паролей (хэширование)

Хранение паролей в открытом виде недопустимо. Правильный подход:

1. **Соль (Salt)** — уникальная случайная строка для каждого пользователя .
2. **Хэширование** — применение необратимой функции к комбинации "пароль + соль".
3. **Проверка** — повторное вычисление хэша и сравнение с сохраненным значением.

SQL Server предоставляет функцию HASHBYTES для хэширования . Поддерживаемые алгоритмы: MD2 | MD4 | MD5 | SHA | SHA1 | SHA2_256 | SHA2_512 . Для паролей рекомендуется использовать SHA2_256 или SHA2_512 .

3.4. Приоритет разрешений

Важнейшее правило безопасности: **DENY отменяет GRANT** независимо от того, через какую роль получено разрешение. Даже если пользователь является членом роли sysadmin (или роли с разрешением CONTROL SERVER), явный запрет (DENY) на уровне объекта будет иметь приоритет .

Ход работы

Этап 1: Подготовка среды и создание структуры базы данных

1. **Запустите SSMS** и подключитесь к вашему экземпляру SQL Server под учетной записью с правами sysadmin (например, используйте проверку подлинности Windows).
2. **Создайте новую базу данных SecurityLab:**
 - В обозревателе объектов правый клик на папке "**Базы данных**" → "**Создать базу данных...**".
 - Имя базы данных: SecurityLab. Нажмите "**ОК**".
3. **Создайте таблицу пользователей с поддержкой хэширования паролей:**
 - Нажмите "**Создать запрос**".
 - Выполните следующий скрипт:

```
sql
```

```
USE SecurityLab;
```

```
GO
```

```
CREATE TABLE dbo.AppUsers (
    UserID INT IDENTITY(1,1) PRIMARY KEY,
    UserName NVARCHAR(50) NOT NULL UNIQUE,
    PasswordHash VARBINARY(64) NOT NULL, -- SHA2_256 дает 32 байта, SHA2_512 - 64
байта
    PasswordSalt VARBINARY(16) NOT NULL, -- 16 байт соли (128 бит)
    Email NVARCHAR(100),
    IsActive BIT DEFAULT 1,
    CreatedDate DATETIME DEFAULT GETDATE()
);
GO
```

```
CREATE TABLE dbo.EmployeeData (
    EmpID INT IDENTITY(1,1) PRIMARY KEY,
    FullName NVARCHAR(100),
    Position NVARCHAR(50),
    Salary DECIMAL(10,2),
    TaxInfo NVARCHAR(20), -- Конфиденциальные данные
    MedicalInfo NVARCHAR(MAX) -- Особо конфиденциальные данные
);
GO
```

-- Добавим тестовые данные

```
INSERT INTO dbo.EmployeeData (FullName, Position, Salary, TaxInfo, MedicalInfo)
VALUES
```

```
    ('Иванов Иван', 'Менеджер', 75000.00, 'ИНН 1234567890', 'Данные отсутствуют'),
    ('Петров Петр', 'Бухгалтер', 68000.00, 'ИНН 0987654321', 'Аллергия на пенициллин'),
    ('Сидорова Анна', 'Директор', 120000.00, 'ИНН 1122334455', 'Диабет 2 типа');
```

GO

Скриншот: окно с выполненным скриптом и сообщением "Команды выполнены успешно".

Этап 2: Создание сложной структуры ролей

Создадим иерархию ролей для корпоративного приложения:

- **Базовые роли:** SelectRole (только чтение), InsertRole (вставка), UpdateRole (обновление).
- **Составные роли:** DataEntryRole (ввод данных)

= InsertRole + UpdateRole. ReaderRole = SelectRole.

- **Прикладные**

роли: JuniorEmployee = DataEntryRole. SeniorEmployee = DataEntryRole + ReaderRole. HRManager — особая роль с доступом к конфиденциальным данным.

- **Роль приложения:** AppLoginRole — роль, от имени которой приложение подключается к БД.

1. **Создайте базовые роли и назначьте им разрешения:**

sql

USE SecurityLab;

GO

-- Создание ролей

CREATE ROLE SelectRole;

CREATE ROLE InsertRole;

CREATE ROLE UpdateRole;

CREATE ROLE DeleteRole;

GO

-- Назначение разрешений на уровне схемы

GRANT SELECT ON SCHEMA::dbo TO SelectRole;

GRANT INSERT ON SCHEMA::dbo TO InsertRole;

GRANT UPDATE ON SCHEMA::dbo TO UpdateRole;

GRANT DELETE ON SCHEMA::dbo TO DeleteRole;

GO

-- Однако! Для конфиденциальных данных (TaxInfo, MedicalInfo) запретим чтение всем, кроме HRManager

-- Пока просто создадим структуру, запреты добавим позже

2. **Создайте составные роли и включите в них базовые:**

sql

-- Роль для ввода данных (может вставлять и обновлять, но не читать и не удалять)

CREATE ROLE DataEntryRole;

GRANT INSERT, UPDATE ON SCHEMA::dbo TO DataEntryRole; -- Альтернативный

синтаксис

GO

```
-- Включим базовые роли в составные
EXEC sp_addrolemember 'DataEntryRole', 'InsertRole';
EXEC sp_addrolemember 'DataEntryRole', 'UpdateRole';
GO
```

```
-- Роль для чтения данных
CREATE ROLE ReaderRole;
EXEC sp_addrolemember 'ReaderRole', 'SelectRole';
GO
```

3. **Создайте прикладные роли (должностные позиции):**

sql

```
-- Младший сотрудник: только ввод данных
CREATE ROLE JuniorEmployee;
EXEC sp_addrolemember 'JuniorEmployee', 'DataEntryRole';
GO
```

```
-- Старший сотрудник: ввод + чтение
CREATE ROLE SeniorEmployee;
EXEC sp_addrolemember 'SeniorEmployee', 'DataEntryRole';
EXEC sp_addrolemember 'SeniorEmployee', 'ReaderRole';
GO
```

```
-- HR-менеджер: полный доступ ко всем данным (включая конфиденциальные)
CREATE ROLE HRManager;
GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA::dbo TO HRManager; --
```

Полные права

```
GO
```

4. **Проверьте созданную структуру** через обозревателя объектов:

- Раскройте **SecurityLab** → **Безопасность** → **Роли** → **Роли базы данных**.
- Вы должны увидеть все созданные роли. Откройте свойства роли JuniorEmployee →

вкладка "**Члены этой роли**" — там должны быть перечислены DataEntryRole (и далее по вложенности).

Скриншот: окно свойств роли с вкладкой Members и отображением вложенных ролей.

Этап 3: Реализация шифрования паролей (35 минут)

Реализуем хэширование паролей с использованием соли.

1. **Создайте хранимую процедуру для регистрации нового пользователя:**

sql

USE SecurityLab;

GO

CREATE PROCEDURE dbo.sp_RegisterUser

 @UserName NVARCHAR(50),

 @Password NVARCHAR(100),

 @email NVARCHAR(100)

AS

BEGIN

 SET NOCOUNT ON;

 DECLARE @Salt VARBINARY(16) = CRYPT_GEN_RANDOM(16); -- Генерация
случайной соли

 DECLARE @PasswordHash VARBINARY(64);

 -- Хэшируем пароль с солью (конкатенация строк)

 SET @PasswordHash = HASHBYTES('SHA2_512', @Password + CAST(@Salt AS
NVARCHAR(50)));

 INSERT INTO dbo.AppUsers (UserName, PasswordHash, PasswordSalt, Email)

 VALUES (@UserName, @PasswordHash, @Salt, @Email);

 SELECT SCOPE_IDENTITY() AS NewUserID;

END

GO

2. **Создайте хранимую процедуру для проверки пароля при входе:**

sql

CREATE PROCEDURE dbo.sp_ValidateUser

 @UserName NVARCHAR(50),

 @Password NVARCHAR(100)

AS

BEGIN

```
SET NOCOUNT ON;
```

```
DECLARE @Salt VARBINARY(16);
```

```
DECLARE @StoredHash VARBINARY(64);
```

```
DECLARE @ComputedHash VARBINARY(64);
```

```
DECLARE @UserID INT;
```

```
-- Получаем соль и хэш для пользователя
```

```
SELECT @UserID = UserID, @Salt = PasswordSalt, @StoredHash = PasswordHash
```

```
FROM dbo.AppUsers
```

```
WHERE UserName = @UserName AND IsActive = 1;
```

```
IF @UserID IS NOT NULL
```

```
BEGIN
```

```
-- Вычисляем хэш введенного пароля
```

```
SET @ComputedHash = HASHBYTES('SHA2_512', @Password + CAST(@Salt AS  
NVARCHAR(50)));
```

```
-- Сравниваем хэши
```

```
IF @ComputedHash = @StoredHash
```

```
BEGIN
```

```
-- Успешный вход
```

```
SELECT UserID, UserName, Email, 'Authentication Success' AS Status
```

```
FROM dbo.AppUsers
```

```
WHERE UserID = @UserID;
```

```
-- Можно также записать время последнего входа
```

```
UPDATE dbo.AppUsers SET CreatedDate = GETDATE() WHERE UserID = @UserID;
```

```
RETURN;
```

```
END;
```

```
END;
```

```
-- Неуспешный вход
```

```
SELECT 'Invalid username or password' AS Status;
```

```
END
```

GO

3. **Зарегистрируйте тестовых пользователей:**

sql

-- Создадим пользователей, соответствующих нашим ролям

EXEC dbo.sp_RegisterUser 'IvanovJunior', 'Pass123!', 'ivanov@company.local';

EXEC dbo.sp_RegisterUser 'PetrovSenior', 'Secure456!', 'petrov@company.local';

EXEC dbo.sp_RegisterUser 'HR_Anna', 'HRpass789!', 'anna.hr@company.local';

GO

-- Посмотрим, как сохранились данные

SELECT UserID, UserName, PasswordHash, PasswordSalt, Email FROM dbo.AppUsers;

Скриншот: результаты запроса с хэшированными паролями (бинарные данные).

Этап 4: Настройка аутентификации клиентского приложения

Создадим роль приложения и настроим сопоставление пользователей с ролями базы данных.

1. **Создайте логины SQL Server для приложения и пользователей:**

sql

USE master;

GO

-- Логин для приложения (сервисная учетная запись)

CREATE LOGIN AppService WITH PASSWORD = 'AppServicePass123!';

-- Логины для сотрудников (в реальном приложении они могут быть Windows-логинами)

CREATE LOGIN IvanovJunior WITH PASSWORD = 'Pass123!';

CREATE LOGIN PetrovSenior WITH PASSWORD = 'Secure456!';

CREATE LOGIN AnnaHR WITH PASSWORD = 'HRpass789!';

GO

2. **Сопоставьте логины с пользователями в базе данных и назначьте роли:**

sql

USE SecurityLab;

GO

-- Создаем пользователей для логинов

CREATE USER IvanovJunior FOR LOGIN IvanovJunior;

```
CREATE USER PetrovSenior FOR LOGIN PetrovSenior;
CREATE USER AnnaHR FOR LOGIN AnnaHR;
CREATE USER AppService FOR LOGIN AppService; -- для приложения
GO
```

```
-- Назначаем роли согласно должностным обязанностям
EXEC sp_addrolemember 'JuniorEmployee', 'IvanovJunior';
EXEC sp_addrolemember 'SeniorEmployee', 'PetrovSenior';
EXEC sp_addrolemember 'HRManager', 'AnnaHR';
```

-- Приложению дадим минимально необходимые права (например, только выполнение процедур аутентификации)

```
GRANT EXECUTE ON dbo.sp_ValidateUser TO AppService;
GRANT EXECUTE ON dbo.sp_RegisterUser TO AppService;
```

-- Приложению не нужен прямой доступ к таблицам!

```
GO
```

3. **Создайте процедуру для получения данных сотрудника с учетом прав**

пользователя:

```
sql
```

```
CREATE PROCEDURE dbo.sp_GetEmployeeData
```

```
@UserLogin NVARCHAR(50) -- Имя пользователя, от имени которого выполняется
```

запрос

```
AS
```

```
BEGIN
```

```
SET NOCOUNT ON;
```

-- Эта процедура будет выполняться от имени AppService, но должна проверять права пользователя

-- Упрощенный вариант: просто возвращаем данные, а права контролируются на уровне таблиц

-- В реальном приложении здесь может быть более сложная логика

```
SELECT EmpID, FullName, Position, Salary,
```

```
CASE
```

```
WHEN IS_MEMBER('HRManager') = 1 THEN TaxInfo
```

```

        ELSE '*** Restricted ***'
    END AS TaxInfo,
CASE
    WHEN IS_MEMBER('HRManager') = 1 THEN MedicalInfo
    ELSE '*** Restricted ***'
END AS MedicalInfo
FROM dbo.EmployeeData;
END
GO

```

```

-- Дадим право выполнения процедуры приложению
GRANT EXECUTE ON dbo.sp_GetEmployeeData TO AppService;
GO

```

Этап 5: Тестирование и анализ приоритета разрешений (50 минут)

Тест 1: Проверка прав младшего сотрудника (JuniorEmployee)

1. В SSMS откройте **новое окно запроса** и подключитесь под логином IvanovJunior (проверка подлинности SQL Server, пароль Pass123!).

```

2. Выполните:

sql
USE SecurityLab;
GO

```

```

-- Попытка чтения данных (должна быть ошибка - нет прав SELECT)
SELECT * FROM dbo.EmployeeData;
GO

```

```

-- Попытка вставки (должна быть успешна - есть права INSERT через DataEntryRole)
INSERT INTO dbo.EmployeeData (FullName, Position, Salary, TaxInfo, MedicalInfo)
VALUES ('Тестов Тест', 'Стажер', 30000.00, 'ИНН 0000000000', 'Нет данных');
GO

```

```

-- Проверим, что вставка сработала (опять ошибка чтения, но данные вставлены)
-- Попытка обновления (должна быть успешна)
UPDATE dbo.EmployeeData SET Salary = 35000.00 WHERE FullName = 'Тестов Тест';
GO

```

Скриншот: сообщения об ошибке SELECT и успешных INSERT/UPDATE.

Тест 2: Проверка прав старшего сотрудника (SeniorEmployee)

1. Откройте новое окно подключения под логином PetrovSenior (пароль Secure456!).
2. Выполните:

```
sql
```

```
USE SecurityLab;
```

```
GO
```

-- Чтение данных (должно быть успешно - есть права SELECT через ReaderRole)

```
SELECT * FROM dbo.EmployeeData;
```

```
GO
```

-- Обратите внимание на колонки TaxInfo и MedicalInfo - они отображаются как NULL или реальные данные?

-- (Зависти от того, есть ли явный запрет на эти колонки)

Тест 3: Проверка прав HR-менеджера (HRManager)

1. Откройте окно подключения под логином AnnaHR (пароль HRpass789!).
2. Выполните:

```
sql
```

```
USE SecurityLab;
```

```
GO
```

-- Должны отображаться все данные, включая конфиденциальные

```
SELECT * FROM dbo.EmployeeData;
```

```
GO
```

Тест 4: Демонстрация приоритета DENY над GRANT (ключевой эксперимент)

1. Вернитесь в окно подключения под **администратором** (или под любым логином с правами sysadmin).

2. Выполните:

```
sql
```

```
USE SecurityLab;
```

```
GO
```

-- Явно запретим чтение конфиденциальных данных для роли SeniorEmployee

```
DENY SELECT ON dbo.EmployeeData(TaxInfo, MedicalInfo) TO SeniorEmployee;
```

GO

-- Также запретим удаление данных для всех, кроме HRManager

DENY DELETE ON SCHEMA::dbo TO JuniorEmployee;

DENY DELETE ON SCHEMA::dbo TO SeniorEmployee;

GO

3. Вернитесь в окно PetrovSenior и выполните:

sql

USE SecurityLab;

-- Попытка прочитаты все колонки

SELECT * FROM dbo.EmployeeData; -- TaxInfo и MedicalInfo должны быть NULL или
ошибка?

GO

-- Явно запросим конфиденциальные колонки

SELECT FullName, TaxInfo, MedicalInfo FROM dbo.EmployeeData; -- ОШИБКА: The
SELECT permission was denied on the column 'TaxInfo'

GO

-- Попытка удалить запись

DELETE FROM dbo.EmployeeData WHERE FullName = 'Тестов Тест'; -- ОШИБКА: The
DELETE permission was denied

GO

Скриншот: сообщения об ошибках запрета доступа.

4. **Важный вывод:** Несмотря на то, что SeniorEmployee через роль ReaderRole имеет право SELECT на всю схему, явный DENY на уровне колонок переопределяет это разрешение. Аналогично, DENY DELETE переопределяет отсутствие явного запрета.

Этап 6: Проверка аутентификации через хранимые процедуры

Протестируем, как приложение может проверять пароли без прямого доступа к таблице AppUsers.

1. Подключитесь под логином AppService (пароль AppServicePass123!).

2. Выполните проверку пользователей:

sql

USE SecurityLab;

GO

-- Проверка корректного пользователя

EXEC dbo.sp_ValidateUser 'IvanovJunior', 'Pass123!';

GO

-- Проверка неверного пароля

EXEC dbo.sp_ValidateUser 'IvanovJunior', 'WrongPassword';

GO

-- Проверка несуществующего пользователя

EXEC dbo.sp_ValidateUser 'FakeUser', 'Pass123!';

GO

-- Регистрация нового пользователя (если приложению разрешена регистрация)

EXEC dbo.sp_RegisterUser 'NewUser', 'NewUserPass!', 'new@company.local';

GO

-- Попытка прямого доступа к таблице пользователей (должна быть ошибка)

SELECT * FROM dbo.AppUsers; -- ОШИБКА: The SELECT permission was denied

GO

Скриншоты: результаты успешной и неуспешной аутентификации, сообщение об ошибке доступа к таблице.

5. Индивидуальные задания по вариантам (40 минут)

Каждый студент получает вариант задания и выполняет его самостоятельно, фиксируя результаты скриншотами.

Вариант	Создать роль	Включить в состав	Добавить разрешения	Добавить запрет (DENY)
1	AuditorRole	SelectRole	Только чтение таблицы EmployeeData без конфиденциальных колонок	Запретить UPDATE
2	TempWorkerRole	InsertRole	INSERT во все таблицы	Запретить SELECT
3	ManagerAssistan	SeniorEmplo	UPDATE зарплаты (Salary)	Запретить

	t	yee		доступ к MedicalInfo
4	SecurityOfficer	SelectRole	Полный доступ ко всем данным, включая конфиденциальные	Запретить DELE TE
5	AppMonitoringRole	Нет	EXECUTE на sp_ValidateUser и SELECT из системных представлений (sys.dm_exec_sessions)	Запретить доступ к любым таблицам

Дополнительное задание для всех вариантов:

- Создайте тестового пользователя для своей роли.
- Проверьте права, подключившись под этим пользователем.
- Сделайте скриншот успешных и неуспешных операций.

Контрольные вопросы

1. В чем преимущество иерархической структуры ролей перед назначением прав напрямую пользователям?
2. Какие предопределенные роли базы данных вы использовали в работе? Для чего они предназначены ?
3. Почему для хранения паролей используется хэширование с солью, а не обратимые методы шифрования ?
4. Какой алгоритм хэширования рекомендуется использовать в SQL Server для паролей? Почему не рекомендуется MD5 или SHA1 ?
5. Что произойдет, если пользователь является членом двух ролей: одна дает разрешение GRANT SELECT, а другая — DENY SELECT?
6. Как проверить, какие роли назначены текущему пользователю? (Подсказка: функция IS_MEMBER() или IS_ROLEMEMBER()).
7. Почему приложению (AppService) не даны прямые права на таблицы, а только на выполнение хранимых процедур? Какой принцип безопасности здесь реализуется?
8. Можно ли создать роль, которая будет иметь права db_ddladmin, но только в рамках одной схемы? (Сложный вопрос)

Содержание отчета

Отчет должен содержать:

1. Титульный лист: ФИО, группу, тему и цель работы.
2. Схема созданной иерархии ролей (можно в виде рисунка или текстового описания).
3. Скриншоты ключевых этапов:

- Создание ролей и назначение членства (вложенность).
 - Регистрация пользователей с хэшированием паролей.
 - Результаты проверки прав для разных пользователей (включая ошибки).
 - Демонстрация приоритета DENY над GRANT.
 - Выполнение индивидуального задания.
4. Листинги (текст) основных SQL-запросов, разработанных в ходе работы.
 5. Ответы на контрольные вопросы.
 6. Вывод: какие механизмы безопасности SQL Server освоены, как они могут

применяться в реальных проектах.

// Методические указания для преподавателя

- Подготовка: Убедитесь, что на сервере включен смешанный режим аутентификации. Если нет — выполните:

sql

USE master;

GO

EXEC xp_instance_regwrite N'HKEY_LOCAL_MACHINE',

N'Software\Microsoft\MSSQLServer\MSSQLServer', N'LoginMode', REG_DWORD, 2;

GO

После изменения перезапустите службу SQL Server.

- Возможные проблемы:
 - Студенты могут забыть переключить контекст базы данных (USE SecurityLab).
 - При создании процедур с HASHBYTES обращайте внимание на типы данных: VARBINARY.
 - Функция CRYPT_GEN_RANDOM доступна в SQL Server 2008 и новее.
- Для слабых компьютеров: Все операции выполняются на стороне сервера, клиентские машины только отправляют запросы через SSMS. Нагрузка на клиента минимальна.
- Рекомендация: Разбейте работу на логические блоки и после каждого блока проводите мини-проверку (например, через опрос или просмотр скриншотов).

Практическое занятие №6. Мониторинг и регистрация действий пользователей в системе для анализа и выявления нарушений безопасности.

Цели работы:

1. Изучить встроенные средства аудита SQL Server для отслеживания действий пользователей.
2. Научиться создавать и настраивать объекты аудита на уровне сервера и базы данных.
3. Освоить методику сбора информации об успешных и неудачных попытках входа в систему.
4. Научиться анализировать журналы аудита для выявления подозрительной активности и нарушений политики безопасности.
5. Применять predefined группы действий аудита для мониторинга доступа к конфиденциальным данным.

Оборудование и ПО:

- **Сервер баз данных (или локальный экземпляр):**
 - SQL Server (Developer или Express Edition) со включенным смешанным режимом аутентификации.
 - Права администратора (sysadmin) для создания и включения аудита.
 - Наличие свободного места на диске для хранения журналов аудита (рекомендуется создать папку C:\SQLAudit).
- **Клиентские машины (ПК студентов):**
 - SQL Server Management Studio (SSMS) установленный и настроенный для подключения к серверу.
- **Примечание:** Все действия выполняются встроенными средствами SQL Server и SSMS, установка дополнительного ПО не требуется .

3. Теоретическая часть

3.1. Что такое аудит в SQL Server?

Аудит SQL Server — это встроенный механизм отслеживания и регистрации событий, происходящих в экземпляре SQL Server и его базах данных . Аудит позволяет фиксировать:

- Успешные и неудачные попытки входа в систему.
- Доступ к данным (операции SELECT, INSERT, UPDATE, DELETE).
- Изменения в структуре базы данных (DDL-операции).
- Изменения прав и разрешений.

3.2. Компоненты аудита SQL Server

Система аудита состоит из трех уровней:

1. **Объект аудита сервера (Server Audit):** Определяет куда будут сохраняться данные аудита. Варианты:
 - **Файл** — запись в бинарный файл с расширением .sqlaudit (рекомендуемый вариант для учебных целей).
 - **Журнал приложений Windows (Application Log)** — запись в Event Viewer.
 - **Журнал безопасности Windows (Security Log)** — требует дополнительных прав.
2. **Спецификация аудита сервера (Server Audit Specification):** Определяет, *какие события* на уровне всего сервера будут отслеживаться (например, вход в систему, изменения ролей сервера).
3. **Спецификация аудита базы данных (Database Audit Specification):** Определяет, *какие события* в конкретной базе данных будут отслеживаться (например, SELECT из таблицы, INSERT, UPDATE).

3.3. Группы действий аудита

Вместо настройки каждого отдельного действия SQL Server предоставляет predefined

группы действий . Важные для данной работы:

- FAILED_LOGIN_GROUP — отслеживает неудачные попытки входа.
- SUCCESSFUL_LOGIN_GROUP — отслеживает успешные попытки входа.
- DATABASE_OBJECT_ACCESS_GROUP — отслеживает доступ к объектам базы данных (SELECT, INSERT, UPDATE, DELETE).
- SCHEMA_OBJECT_ACCESS_GROUP — аналогично, но на уровне схемы.
- DATABASE_OBJECT_CHANGE_GROUP — отслеживает изменения объектов (CREATE, ALTER, DROP).
- SERVER_ROLE_MEMBER_CHANGE_GROUP — отслеживает изменения членства в ролях.

3.4. Правило приоритета

Как и в случае с разрешениями, при настройке аудита важно помнить, что:

- Аудит работает по принципу "разрешено все, что не запрещено явно".
- Можно фильтровать события, чтобы снизить нагрузку на сервер.
- **Важно:** Если аудит настроен на файл, необходимо следить за дисковым пространством.

Ход работы

Этап 1: Подготовка среды

1. **Запустите SSMS** и подключитесь к вашему экземпляру SQL Server под учетной записью с правами sysadmin (например, используйте проверку подлинности Windows).
2. **Создайте папку для хранения журналов аудита** на сервере (или локально, если работаете с локальным экземпляром):
 - Откройте **Проводник Windows**.
 - Создайте на диске C: папку с именем SQLAudit.
 - *Примечание:* Убедитесь, что у службы SQL Server есть права на запись в эту папку (обычно права есть у службы, если папка создана в корне диска).

3. **Создайте тестовую базу данных и таблицу с конфиденциальными данными:**

- Нажмите "**Создать запрос**".
- Выполните следующий скрипт:

```
sql
```

```
-- Создаем базу данных для лабораторной
```

```
CREATE DATABASE AuditLab;
```

```
GO
```

```
USE AuditLab;
```

```
GO
```

```
-- Создаем таблицу с конфиденциальными данными сотрудников
```

```
CREATE TABLE dbo.Employees (
```

```
    EmployeeID INT IDENTITY(1,1) PRIMARY KEY,
```

```
    FullName NVARCHAR(100),
```

```
    Position NVARCHAR(50),
```

```
    Salary DECIMAL(10,2),
```

```
    PassportNumber NVARCHAR(20),    -- Конфиденциально!
```

```
    HomeAddress NVARCHAR(200),    -- Конфиденциально!
```

```
    MedicalInfo NVARCHAR(MAX)    -- Особо конфиденциально!
```

```
);
```

```
GO
```

```
-- Заполняем тестовыми данными
```

```

INSERT INTO dbo.Employees (FullName, Position, Salary, PassportNumber, HomeAddress,
MedicalInfo)
VALUES
('Иванов Иван Иванович', 'Главный бухгалтер', 95000.00, '4510 123456', 'г. Москва, ул.
Ленина, д.1', 'Аллергия на аспирин'),
('Петрова Анна Сергеевна', 'HR-менеджер', 78000.00, '4511 789012', 'г. Москва, ул.
Гагарина, д.5', 'Здорова'),
('Сидоров Петр Николаевич', 'Разработчик', 120000.00, '4512 345678', 'г. Москва, ул.
Королева, д.10', 'Диабет 2 типа');
GO

```

```

-- Создадим тестовых пользователей для проверки аудита
CREATE LOGIN TestUser1 WITH PASSWORD = 'TestUserPass123!';
CREATE LOGIN TestUser2 WITH PASSWORD = 'TestUserPass456!';
GO

```

```

-- Создадим пользователей в базе данных
CREATE USER TestUser1 FOR LOGIN TestUser1;
CREATE USER TestUser2 FOR LOGIN TestUser2;
GO

```

```

-- Дадим минимальные права (только чтение)
ALTER ROLE db_datareader ADD MEMBER TestUser1;
ALTER ROLE db_datareader ADD MEMBER TestUser2;
GO

```

Скриншот: окно с выполненным скриптом и сообщением "Команды выполнены успешно".

Этап 2: Создание объекта аудита сервера (Server Audit)

1. Создание аудита через графический интерфейс SSMS:

- В **Обозревателе объектов** раскройте папку **"Безопасность"** (Security).
- Найдите папку **"Аудит"** (Audits), нажмите на ней правой кнопкой мыши

→ **"Создать аудит..."** (New Audit).

- В открывшемся окне **"Создание аудита"** (Create Audit) заполните:
 - **Имя аудита:** Audit_UserActivity
 - **Назначение аудита:** Файл (File)
 - **Путь к файлу:** C:\SQLAudit\ (нажмите кнопку "Обзор", если хотите

проверить путь)

- **Максимальный размер файла:** 10 МБ
- **Максимальное количество файлов:** 5
- **При сбое аудита:** Продолжить (Continue)

- Остальные параметры оставьте по умолчанию.

- Нажмите **"ОК"**.

Скриншот: окно Create Audit с заполненными полями.

2. Включение аудита:

- В папке **"Аудит"** найдите созданный объект Audit_UserActivity.
- Нажмите на нем правой кнопкой мыши → **"Включить аудит"** (Enable Audit).
- Дождитесь сообщения об успешном включении.

Скриншот: контекстное меню с выбранным пунктом Enable Audit.

3. **Альтернативный способ (T-SQL):** Выполните следующий скрипт (аналогичный тому, что сгенерировал SSMS):

```

sql
USE master;
GO

```

```

CREATE SERVER AUDIT [Audit_UserActivity]
TO FILE
( FILEPATH = N'C:\SQLAudit\
,MAXSIZE = 10 MB
,MAX_ROLLOVER_FILES = 5
)
WITH
( QUEUE_DELAY = 1000
,ON_FAILURE = CONTINUE
);
GO
ALTER SERVER AUDIT [Audit_UserActivity] WITH (STATE = ON);
GO

```

Этап 3: Создание спецификации аудита сервера (Server Audit Specification) (30 минут)

Настроим отслеживание попыток входа в систему.

1. Создание серверной спецификации через SSMS:

- В папке "Безопасность" найдите папку "Серверные спецификации аудита" (Server Audit Specifications).
- Нажмите правой кнопкой мыши → "Создать серверную спецификацию аудита...".

- Заполните:

- **Имя:** Spec_Logins
- **Аудит:** Audit_UserActivity (выберите из выпадающего списка)
- В таблице "Типы действий аудита" нажмите "Добавить" и выберите:
 - FAILED_LOGIN_GROUP — неудачные попытки входа.
 - SUCCESSFUL_LOGIN_GROUP — успешные попытки входа.

- Нажмите "ОК".

Скриншот: окно создания серверной спецификации с выбранными группами действий.

2. Включение серверной спецификации:

- Найдите созданную спецификацию Spec_Logins.
- Правый клик → "Включить серверную спецификацию аудита".

3. T-SQL вариант:

```

sql
USE master;
GO
CREATE SERVER AUDIT SPECIFICATION [Spec_Logins]
FOR SERVER AUDIT [Audit_UserActivity]
ADD (FAILED_LOGIN_GROUP),
ADD (SUCCESSFUL_LOGIN_GROUP)
WITH (STATE = ON);
GO

```

Этап 4: Создание спецификации аудита базы данных (Database Audit Specification) (40 минут)

Настроим отслеживание доступа к конфиденциальным данным в таблице Employees.

1. Создание спецификации БД через SSMS:

- В обозревателе объектов раскройте базу данных **AuditLab**.
- Раскройте папку "Безопасность".
- Найдите папку "Спецификации аудита базы данных" (Database Audit Specifications).

- Правый клик → "Создать спецификацию аудита базы данных...".

- Заполните:

- **Имя:** Spec_EmployeeDataAccess

- **Аудит: Audit_UserActivity**

- В таблице действий нажмите **"Добавить"** несколько раз, чтобы создать следующие записи:

Тип действия аудита	Класс объекта	Имя схемы	Имя объекта	Имя участника
SELECT	OBJECT	dbo	Employees	public
UPDATE	OBJECT	dbo	Employees	public
INSERT	OBJECT	dbo	Employees	public
DELETE	OBJECT	dbo	Employees	public

- *Пояснение:* Указание участника public означает, что мы отслеживаем действия всех пользователей.

- Нажмите **"ОК"**.

text

*Скриншот: окно создания спецификации БД с заполненными действиями. *

2. **Включение спецификации БД:**

- Правый клик на Spec_EmployeeDataAccess → **"Включить спецификацию аудита базы данных"**.

3. **T-SQL вариант:**

sql

USE AuditLab;

GO

CREATE DATABASE AUDIT SPECIFICATION [Spec_EmployeeDataAccess]

FOR SERVER AUDIT [Audit_UserActivity]

ADD (SELECT ON OBJECT::dbo.Employees BY public),

ADD (UPDATE ON OBJECT::dbo.Employees BY public),

ADD (INSERT ON OBJECT::dbo.Employees BY public),

ADD (DELETE ON OBJECT::dbo.Employees BY public)

WITH (STATE = ON);

GO

Этап 5: Генерация событий для аудита

Теперь выполним действия, которые должны быть зафиксированы в журнале аудита.

1. **Сымитируем неудачную попытку входа:**

- Откройте **новое окно запроса** (или попробуйте подключиться через другую вкладку).

- Попробуйте подключиться с неверным паролем для пользователя TestUser1.

- *Скриншот:* сообщение об ошибке входа.

2. **Выполним успешный вход и действия с данными:**

- Подключитесь под логином TestUser1 (пароль TestUserPass123!).

- Выполните следующие запросы:

sql

USE AuditLab;

GO

-- Чтение данных (попадет в аудит как SELECT)

SELECT * FROM dbo.Employees;

GO

-- Попытка обновить данные (попадет в аудит как UPDATE)

UPDATE dbo.Employees SET Salary = 100000 WHERE FullName LIKE '%Иванов%';

GO

```
-- Попытка вставить данные (попадет в аудит как INSERT)
INSERT INTO dbo.Employees (FullName, Position, Salary, PassportNumber, HomeAddress,
MedicalInfo)
VALUES ('Тестов Тест', 'Стажер', 30000, '1234 567890', 'г. Москва', 'Нет данных');
GO
```

3. **Повторите аналогичные действия от имени TestUser2.**

Этап 6: Анализ журналов аудита

Теперь самое важное — научиться читать и анализировать собранные данные.

Способ 1: Просмотр через Log File Viewer (графический интерфейс)

1. В SSMS перейдите в папку "Безопасность" → "Аудит".
2. Правый клик на Audit_UserActivity → "Просмотреть журналы аудита...".
3. Откроется окно "Просмотр файла журнала" (Log File Viewer). В правой части отобразятся все собранные события.
4. Изучите колонки:
 - event_time — время события.
 - action_id — код действия (SL — SELECT, UP — UPDATE, IN — INSERT, LGIF — LOGIN FAILED, LGIN — LOGIN SUCCEEDED).
 - succeeded — успешность операции (1 — успешно, 0 — неуспешно).
 - server_principal_name — имя логина, выполнившего действие.
 - database_name — имя базы данных.
 - schema_name и object_name — объект, к которому было обращение.
 - statement — текст выполненного запроса (если доступно).

Скриншот: окно Log File Viewer с отфильтрованными записями.

5. **Примените фильтр:** Нажмите кнопку "Фильтр" в верхней части. Отфильтруйте, например, по server_principal_name = 'TestUser1' или по action_id = 'LGIF' (неудачные логины).

Способ 2: Использование функции T-SQL fn_get_audit_file

1. Подключитесь под администратором и выполните следующий запрос:

```
sql
-- Получаем путь к файлам аудита
DECLARE @audit_path NVARCHAR(256);
SELECT @audit_path = log_file_path + log_file_name
FROM sys.server_file_audits
WHERE name = 'Audit_UserActivity';
```

-- Читаем данные аудита

```
SELECT
    event_time,
    action_id,
    succeeded,
    session_id,
    server_principal_name,
    database_name,
    schema_name,
    object_name,
    statement
FROM sys.fn_get_audit_file(@audit_path, DEFAULT, DEFAULT)
WHERE database_name = 'AuditLab' OR database_name IS NULL
ORDER BY event_time DESC;
```

2. Анализ результатов:

- Найдите все неудачные попытки входа (action_id = 'LGIF', succeeded = 0).
- Найдите все операции SELECT над таблицей Employees.

- Определите, кто и когда пытался изменить зарплаты сотрудников.

Этап 7: Выявление нарушений безопасности (кейс-стади)

Представьте, что вы администратор безопасности. Вам поступило сообщение о подозрительной активности. Проанализируйте журналы и ответьте на вопросы:

1. Были ли попытки несанкционированного доступа? (Неудачные логины).
2. Кто из пользователей обращался к конфиденциальным данным (PassportNumber, MedicalInfo)? (Подсказка: аудит фиксирует обращение к таблице в целом, но можно включить аудит на уровне колонок, см. дополнительное задание).
3. Есть ли подозрительные операции UPDATE в нерабочее время?

Практическое задание: Напишите запрос, который покажет все операции UPDATE, выполненные пользователем TestUser2.

sql

-- Ваш код здесь

Индивидуальные задания по вариантам

Каждый студент получает вариант задания и выполняет его самостоятельно, фиксируя результаты скриншотами.

Вариант	Создать аудит	Отслеживаемые события	Дополнительное условие
1	Audit_Var1	Все DML-операции (SELECT, INSERT, UPDATE, DELETE) над таблицей Employees	Добавить фильтр, чтобы аудит срабатывал только при доступе к колонке Salary
2	Audit_Var2	Только неудачные попытки входа и изменения в структуре БД (CREATE/ALTER/DROP)	Настроить аудит на уровне БД для отслеживания изменений схемы
3	Audit_Var3	Успешные логины и все операции, выполненные пользователем TestUser1	Использовать предложение WHERE в спецификации аудита
4	Audit_Var4	Доступ к таблице Employees в нерабочее время (после 18:00 и до 9:00)	Использовать предикат для фильтрации по времени
5	Audit_Var5	Изменения прав доступа (GRANT, DENY, REVOKE) и добавление пользователей в роли	Отслеживать группу DATABASE_PERMISSION_CHANGE_GROUP

Дополнительное задание (сложный уровень):

Настройте аудит на уровне колонок, чтобы фиксировать только обращения к колонкам PassportNumber и MedicalInfo. Для этого в спецификации аудита БД укажите:

text

ADD (SELECT ON OBJECT::dbo.Employees(PassportNumber) BY public)

Контрольные вопросы

1. Какие компоненты входят в систему аудита SQL Server? Опишите их назначение.
2. В чем разница между серверной спецификацией аудита и спецификацией аудита базы данных?

3. Какие группы действий аудита используются для отслеживания попыток входа в систему?
4. Как просмотреть собранные журналы аудита с помощью T-SQL?
5. Почему важно следить за размером файлов аудита и что произойдет при его переполнении?
6. Можно ли настроить аудит так, чтобы он срабатывал только при определенных условиях (например, только для конкретного пользователя или только в определенное время)?
7. Что означает параметр ON_FAILURE = CONTINUE в объекте аудита? Какие есть альтернативы?

Содержание отчета

1. Титульный лист: ФИО, группа, тема и цель работы.
2. Скриншоты ключевых этапов:
 - Создание объекта аудита сервера.
 - Создание серверной спецификации аудита.
 - Создание спецификации аудита базы данных.
 - Результаты просмотра журналов аудита (Log File Viewer) с отображением зафиксированных событий.
 - Выполнение индивидуального задания.
3. Листинги (текст) основных T-SQL запросов, использованных в работе.
4. Результаты анализа журналов (ответы на вопросы из кейс-стади).
5. Ответы на контрольные вопросы.
6. Вывод: какие возможности мониторинга SQL Server освоены, как они помогают в выявлении нарушений безопасности.

//Методические указания для преподавателя

- Подготовка:
 - Убедитесь, что на сервере создана папка C:\SQLAudit и у службы SQL Server есть права на запись в нее.
 - Проверьте, что аудит не занимает слишком много места (ограничьте максимальный размер файла).
 - Для слабых компьютеров: все операции по созданию аудита выполняются на сервере, клиенты только подключаются через SSMS.
- Возможные проблемы и их решение:
 - Ошибка "Access is denied" при записи аудита: Проверьте права на папку C:\SQLAudit. Служба SQL Server работает от имени учетной записи (например, NT SERVICE\MSSQLSERVER). Нужно дать этой учетной записи права на запись.
 - Аудит не включается: Проверьте, что объект аудита включен (STATE = ON), и что спецификации также включены.
 - Журналы не отображаются: После включения аудита выполните несколько действий, генерирующих события, затем подождите немного (аудит асинхронный) и обновите представление.
- Рекомендации:
 - Обратите внимание студентов на асинхронную природу аудита (параметр QUEUE_DELAY). События не пишутся мгновенно, а ставятся в очередь.
 - Продемонстрируйте, как можно архивировать старые файлы аудита и открывать их для анализа.
 - Обсудите сходство и различие между аудитом и другими инструментами мониторинга (например, профилировщиком SQL Server).

Практическое занятие №7. Применение триггеров в качестве дополнительного инструмента для управления правами доступа.

Цели работы:

1. Изучить типы триггеров в SQL Server (DML, DDL, LOGON) и их роль в обеспечении безопасности .
2. Научиться создавать DML-триггеры для аудита доступа к конфиденциальным данным и предотвращения несанкционированных изменений.
3. Освоить создание DDL-триггеров для защиты структуры базы данных от нежелательных изменений .
4. Научиться применять триггеры входа (LOGON) для ограничения доступа к серверу на основе различных условий .
5. Исследовать особенности выполнения триггеров в контексте прав пользователя и научиться использовать предложение EXECUTE AS для повышения безопасности .

Оборудование и ПО:

- **Сервер баз данных (или локальный экземпляр):**
 - SQL Server (Developer или Express Edition) со включенным смешанным режимом аутентификации (SQL Server and Windows Authentication Mode).
 - Права администратора (sysadmin) для создания триггеров и управления безопасностью.
- **Клиентские машины (ПК студентов):**
 - SQL Server Management Studio (SSMS) установленный и настроенный для подключения к серверу.
- **Примечание:** Все действия выполняются встроенными средствами SQL Server и SSMS, установка дополнительного ПО не требуется.

Теоретическая часть

3.1. Триггеры как инструмент безопасности

Триггеры в SQL Server — это особый тип хранимых процедур, которые автоматически выполняются при наступлении определенных событий . В контексте безопасности триггеры позволяют:

- Аудировать доступ к данным (кто, когда и какие данные изменял/просматривал).
- Блокировать нежелательные операции (например, удаление таблиц или массовое изменение зарплат).
- Ограничивать подключения к серверу (например, по времени суток или количеству сессий).

3.2. Типы триггеров для управления доступом

Тип триггера	Область действия	События	Применение в безопасности
DML триггеры	Таблицы и представления	INSERT, UPDATE, DELETE	Аудит изменений данных, запрет модификации в нерабочее время, проверка корректности изменений
DDL триггеры	База данных или сервер	CREATE_TABLE, DROP_TABLE, ALTER_TABLE и др.	Защита структуры БД, логирование изменений схемы, предотвращение удаления объектов
LOGON триггеры	Сервер	LOGON (подключение)	Ограничение количества сессий, блокировка подключений по времени, контроль доступа

3.3. Контекст выполнения и безопасность триггеров

Важная особенность: По умолчанию триггеры выполняются в контексте того пользователя, который вызвал их срабатывание. Например, если пользователь с ограниченными правами выполняет DELETE, а триггер пытается выполнить GRANT CONTROL SERVER, операция завершится ошибкой. Однако существует риск "повышения привилегий", когда злоумышленник создает триггер, который выполнит вредоносный код при срабатывании от имени администратора.

Рекомендации по безопасности триггеров:

- Всегда проверяйте существующие триггеры через системные представления (sys.triggers, sys.server_triggers).
- Используйте предложение WITH EXECUTE AS для выполнения триггера от имени конкретного пользователя с минимально необходимыми правами.
- Временное отключение подозрительных триггеров командой DISABLE TRIGGER.

3.4. Функция EVENTDATA()

Для DDL и LOGON триггеров доступна функция EVENTDATA(), возвращающая XML с информацией о событии: время, пользователь, тип события, текст команды и др. . Это основной инструмент для логирования действий.

Ход работы

Этап 1: Подготовка среды

1. **Запустите SSMS** и подключитесь к вашему экземпляру SQL Server под учетной записью с правами sysadmin (например, используйте проверку подлинности Windows).

2. **Создайте базу данных TriggerSecurityLab и тестовые объекты:**

○ Нажмите "Создать запрос".

○ Выполните следующий скрипт:

```
sql
```

```
-- Создаем базу данных
```

```
CREATE DATABASE TriggerSecurityLab;
```

```
GO
```

```
USE TriggerSecurityLab;
```

```
GO
```

```
-- Создаем таблицу сотрудников (конфиденциальные данные)
```

```
CREATE TABLE dbo.Employees (
```

```
    EmployeeID INT IDENTITY(1,1) PRIMARY KEY,
```

```
    FullName NVARCHAR(100) NOT NULL,
```

```
    Position NVARCHAR(50),
```

```
    Salary DECIMAL(10,2),
```

```
    PassportNumber NVARCHAR(20),
```

```
    Email NVARCHAR(100),
```

```
    LastModified DATETIME DEFAULT GETDATE(),
```

```
    ModifiedBy NVARCHAR(100) DEFAULT SUSER_NAME()
```

```
);
```

```
-- Создаем таблицу для логирования изменений данных (DML аудит)
```

```
CREATE TABLE dbo.DataChangeLog (
```

```
    LogID INT IDENTITY(1,1) PRIMARY KEY,
```

```
    TableName NVARCHAR(100),
```

```
    ActionType NVARCHAR(10), -- INSERT, UPDATE, DELETE
```

```
    RecordID INT,
```

```
    ChangedData XML, -- Для хранения старых и новых значений
```

```

        ChangeDate DATETIME DEFAULT GETDATE(),
        ChangedBy NVARCHAR(100),
        Application NVARCHAR(100)
    );

-- Создаем таблицу для логирования изменений структуры (DDL аудит)
CREATE TABLE dbo.SchemaChangeLog (
    LogID INT IDENTITY(1,1) PRIMARY KEY,
    EventType NVARCHAR(100),
    ObjectName NVARCHAR(255),
    ObjectType NVARCHAR(50),
    CommandText NVARCHAR(MAX),
    LoginName NVARCHAR(100),
    ChangeDate DATETIME DEFAULT GETDATE(),
    EventDataXML XML
);

-- Добавляем тестовые данные
INSERT INTO dbo.Employees (FullName, Position, Salary, PassportNumber, Email)
VALUES
    ('Иванов Иван', 'Главный бухгалтер', 95000, '4510 123456', 'ivanov@company.local'),
    ('Петрова Анна', 'HR-менеджер', 78000, '4511 789012', 'petrova@company.local'),
    ('Сидоров Петр', 'Разработчик', 120000, '4512 345678', 'sidorov@company.local');

-- Создаем тестовых пользователей
CREATE LOGIN Auditor WITH PASSWORD = 'AuditorPass123!';
CREATE LOGIN HR_Manager WITH PASSWORD = 'HRpass123!';
CREATE LOGIN AppUser WITH PASSWORD = 'AppPass123!';
GO

-- Создаем пользователей в БД
CREATE USER Auditor FOR LOGIN Auditor;
CREATE USER HR_Manager FOR LOGIN HR_Manager;
CREATE USER AppUser FOR LOGIN AppUser;
GO

-- Даем минимальные права
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Employees TO HR_Manager;
GRANT SELECT ON dbo.Employees TO Auditor;
GRANT SELECT ON dbo.Employees TO AppUser; -- Приложение только читает
GO
Скриншот: окно с выполненным скриптом и сообщением "Команды выполнены успешно".

```

Этап 2: Создание DML-триггера для аудита изменений данных

Создадим триггер, который будет автоматически логировать все изменения (INSERT, UPDATE, DELETE) в таблице Employees в таблицу DataChangeLog.

1. Создание триггера через графический интерфейс SSMS:

- В обозревателе объектов раскройте базу данных **TriggerSecurityLab** → **Таблицы** → **dbo.Employees**.
- Раскройте папку **"Триггеры"** (Triggers), нажмите правой кнопкой мыши → **"Создать триггер..."**.
- Откроется окно с шаблоном. Замените содержимое следующим кодом:

```

sql
CREATE TRIGGER trg_Employees_Audit
ON dbo.Employees
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ActionType NVARCHAR(10);
    DECLARE @RecordID INT;
    DECLARE @ChangedData XML;

    -- Обработка INSERT
    IF EXISTS (SELECT * FROM inserted) AND NOT EXISTS (SELECT * FROM deleted)
    BEGIN
        SET @ActionType = 'INSERT';

        DECLARE insert_cursor CURSOR FOR
            SELECT EmployeeID FROM inserted;

        OPEN insert_cursor;
        FETCH NEXT FROM insert_cursor INTO @RecordID;

        WHILE @@FETCH_STATUS = 0
        BEGIN
            -- Сохраняем новые данные как XML
            SELECT @ChangedData = (SELECT * FROM inserted WHERE EmployeeID =
@RecordID FOR XML AUTO);

            INSERT INTO dbo.DataChangeLog (TableName, ActionType, RecordID, ChangedData,
ChangedBy, Application)
            VALUES ('Employees', @ActionType, @RecordID, @ChangedData, SUSER_NAME(),
APP_NAME());

            FETCH NEXT FROM insert_cursor INTO @RecordID;
        END;

        CLOSE insert_cursor;
        DEALLOCATE insert_cursor;
    END

    -- Обработка DELETE
    IF EXISTS (SELECT * FROM deleted) AND NOT EXISTS (SELECT * FROM inserted)
    BEGIN
        SET @ActionType = 'DELETE';

        DECLARE delete_cursor CURSOR FOR
            SELECT EmployeeID FROM deleted;

        OPEN delete_cursor;
        FETCH NEXT FROM delete_cursor INTO @RecordID;
    END

```

```

WHILE @@FETCH_STATUS = 0
BEGIN
    -- Сохраняем удаленные данные
    SELECT @ChangedData = (SELECT * FROM deleted WHERE EmployeeID =
@RecordID FOR XML AUTO);

    INSERT INTO dbo.DataChangeLog (TableName, ActionType, RecordID, ChangedData,
ChangedBy, Application)
    VALUES ('Employees', @ActionType, @RecordID, @ChangedData, SUSER_NAME(),
APP_NAME());

    FETCH NEXT FROM delete_cursor INTO @RecordID;
END;

CLOSE delete_cursor;
DEALLOCATE delete_cursor;
END

-- Обработка UPDATE (есть и inserted, и deleted)
IF EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
BEGIN
    SET @ActionType = 'UPDATE';

    DECLARE update_cursor CURSOR FOR
    SELECT i.EmployeeID
    FROM inserted i
    INNER JOIN deleted d ON i.EmployeeID = d.EmployeeID;

    OPEN update_cursor;
    FETCH NEXT FROM update_cursor INTO @RecordID;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Сохраняем старые и новые данные
        SELECT @ChangedData = (
            SELECT
                (SELECT * FROM deleted d WHERE d.EmployeeID = @RecordID FOR XML
AUTO) AS OldValues,
                (SELECT * FROM inserted i WHERE i.EmployeeID = @RecordID FOR XML
AUTO) AS NewValues
            FOR XML PATH('Update')
        );

        INSERT INTO dbo.DataChangeLog (TableName, ActionType, RecordID, ChangedData,
ChangedBy, Application)
        VALUES ('Employees', @ActionType, @RecordID, @ChangedData, SUSER_NAME(),
APP_NAME());

        FETCH NEXT FROM update_cursor INTO @RecordID;
    END;

    CLOSE update_cursor;

```

```
DEALLOCATE update_cursor;
END
END;
GO
```

- Нажмите "**Выполнить**" (F5).

Скриншот: окно создания триггера с кодом и сообщение об успешном выполнении.

2. Проверка работы триггера:

- Откройте **новое окно запроса** и выполните тестовые операции:

```
sql
```

```
USE TriggerSecurityLab;
```

```
GO
```

-- Вставка новой записи

```
INSERT INTO dbo.Employees (FullName, Position, Salary, PassportNumber, Email)
VALUES ('Николаев Николай', 'Тестирующий', 65000, '4513 987654',
```

```
'nikolaev@company.local');
```

```
GO
```

-- Обновление записи

```
UPDATE dbo.Employees SET Salary = 70000 WHERE FullName = 'Николаев Николай';
```

```
GO
```

-- Удаление записи

```
DELETE FROM dbo.Employees WHERE FullName = 'Николаев Николай';
```

```
GO
```

-- Проверка лога изменений

```
SELECT * FROM dbo.DataChangeLog ORDER BY ChangeDate DESC;
```

```
GO
```

Скриншот: результаты запроса с логом изменений, показывающие все три операции.

Этап 3: Создание DDL-триггера для защиты структуры базы данных (35 минут)

Создадим триггер, который будет предотвращать удаление таблиц и логировать все изменения структуры.

1. Создание DDL-триггера уровня базы данных:

```
sql
```

```
USE TriggerSecurityLab;
```

```
GO
```

```
CREATE TRIGGER trg_SchemaProtection
```

```
ON DATABASE
```

```
FOR DROP_TABLE, ALTER_TABLE, CREATE_TABLE
```

```
AS
```

```
BEGIN
```

```
SET NOCOUNT ON;
```

```
DECLARE @EventData XML = EVENTDATA();
```

```
DECLARE @EventType NVARCHAR(100) =
```

```
@EventData.value('/EVENT_INSTANCE/EventType[1]', 'NVARCHAR(100)');
```

```
DECLARE @ObjectName NVARCHAR(255) =
```

```
@EventData.value('/EVENT_INSTANCE/ObjectName[1]', 'NVARCHAR(255)');
```

```
DECLARE @ObjectType NVARCHAR(50) =
```

```
@EventData.value('/EVENT_INSTANCE/ObjectType[1]', 'NVARCHAR(50)');
```

```

        DECLARE @CommandText NVARCHAR(MAX) =
@EventData.value('/EVENT_INSTANCE/TSQLCommand/CommandText)[1]', 'NVARCHAR(MAX)');
        DECLARE @LoginName NVARCHAR(100) =
@EventData.value('/EVENT_INSTANCE/LoginName)[1]', 'NVARCHAR(100)');

        -- Логируем событие
        INSERT INTO dbo.SchemaChangeLog (EventType, ObjectName, ObjectType, CommandText,
LoginName, EventDataXML)
        VALUES (@EventType, @ObjectName, @ObjectType, @CommandText, @LoginName,
@EventData);

        -- Запрещаем удаление таблиц (кроме определенных пользователей)
        IF @EventType = 'DROP_TABLE' AND @LoginName NOT IN ('sa',
'DOMAIN\Administrator')
        BEGIN
            RAISERROR('Удаление таблиц запрещено! Обратитесь к администратору.', 16, 1);
            ROLLBACK;
        END

        -- Для ALTER TABLE предупреждаем, но разрешаем
        IF @EventType = 'ALTER_TABLE'
        BEGIN
            PRINT 'Внимание! Изменение структуры таблицы залогировано.';
        END
    END;
GO

```

2. Проверка работы DDL-триггера:

sql

-- Попытка создать таблицу (должно сработать логирование)

```
CREATE TABLE dbo.TestTable (ID INT);
```

```
GO
```

-- Попытка удалить таблицу (должна быть ошибка)

```
DROP TABLE dbo.TestTable;
```

```
GO
```

-- Проверка лога изменений схемы

```
SELECT * FROM dbo.SchemaChangeLog ORDER BY ChangeDate DESC;
```

```
GO
```

Скриншот: сообщение об ошибке удаления таблицы и содержимое таблицы

SchemaChangeLog.

Этап 4: Создание триггера входа (LOGON) для ограничения подключений

Создадим триггер, который будет ограничивать количество одновременных сессий для пользователей и блокировать подключения в нерабочее время .

1. Создание триггера входа:

sql

```
USE master;
```

```
GO
```

```
CREATE TRIGGER trg_ConnectionLimit
ON ALL SERVER
```

```

FOR LOGON
AS
BEGIN
    DECLARE @LoginName NVARCHAR(100) = ORIGINAL_LOGIN();
    DECLARE @SessionCount INT;
    DECLARE @CurrentHour INT = DATEPART(HOUR, GETDATE());

    -- Ограничение для пользователя AppUser (не более 2 сессий)
    IF @LoginName = 'AppUser'
    BEGIN
        SELECT @SessionCount = COUNT(*)
        FROM sys.dm_exec_sessions
        WHERE is_user_process = 1
            AND original_login_name = @LoginName;

        IF @SessionCount > 2
        BEGIN
            RAISERROR('Превышен лимит одновременных подключений для AppUser
(максимум 2 сессии).', 16, 1);
            ROLLBACK;
        END
    END

    -- Блокировка подключений в нерабочее время для HR_Manager
    IF @LoginName = 'HR_Manager'
    BEGIN
        -- Блокируем с 20:00 до 8:00 и в выходные
        IF @CurrentHour < 8 OR @CurrentHour >= 20 OR DATEPART(WEEKDAY,
GETDATE()) IN (1, 7)
        BEGIN
            RAISERROR('Доступ для HR_Manager разрешен только в рабочее время (пн-пт,
8:00-20:00).', 16, 1);
            ROLLBACK;
        END
    END
END;
GO

```

2. Предоставление прав для просмотра состояния сервера:

- Чтобы триггер мог выполнять запрос к sys.dm_exec_sessions, необходимо дать соответствующие права:

```

sql
GRANT VIEW SERVER STATE TO AppUser;
GRANT VIEW SERVER STATE TO HR_Manager;
GO

```

3. Проверка работы триггера входа:

- Откройте **несколько окон подключения** под логином AppUser (пароль AppPass123!). Попробуйте открыть третье подключение — оно должно быть отклонено.

- Попробуйте подключиться под логином HR_Manager в нерабочее время или в выходной — подключение должно быть отклонено с сообщением об ошибке.

Скриншот: сообщение об ошибке подключения с текстом ограничения.

Этап 5: Исследование контекста выполнения и рисков безопасности

Важно! Этот этап демонстрирует потенциальные уязвимости и способы их предотвращения.

1. Создание "опасного" триггера (для демонстрации уязвимости):

sql

```
USE TriggerSecurityLab;
```

```
GO
```

```
-- Создаем таблицу-жертву
```

```
CREATE TABLE dbo.SensitiveData (
```

```
    ID INT PRIMARY KEY,
```

```
    SecretData NVARCHAR(MAX)
```

```
);
```

```
GO
```

```
-- Вставляем тестовые данные
```

```
INSERT INTO dbo.SensitiveData VALUES (1, 'Супер-секретные данные');
```

```
GO
```

```
-- Создаем пользователя с ограниченными правами
```

```
CREATE LOGIN MaliciousUser WITH PASSWORD = 'Malicious123!';
```

```
CREATE USER MaliciousUser FOR LOGIN MaliciousUser;
```

```
GRANT INSERT, UPDATE ON dbo.SensitiveData TO MaliciousUser; -- Только изменение, не чтение!
```

```
GO
```

-- Злоумышленник создает триггер, который попытается выполнить действие с повышенными правами

```
-- (этот код выполняется от имени MaliciousUser)
```

```
EXECUTE AS LOGIN = 'MaliciousUser';
```

```
GO
```

```
CREATE TRIGGER trg_MaliciousTrigger
```

```
ON dbo.SensitiveData
```

```
AFTER UPDATE
```

```
AS
```

```
BEGIN
```

-- Попытка прочитывать данные (не должно работать, т.к. MaliciousUser не имеет прав SELECT)

```
    DECLARE @data NVARCHAR(MAX);
```

```
    SELECT @data = SecretData FROM dbo.SensitiveData WHERE ID = 1;
```

```
-- Попытка выполнить привилегированную операцию
```

```
    EXEC('GRANT SELECT ON dbo.SensitiveData TO MaliciousUser');
```

```
END;
```

```
GO
```

```
REVERT; -- Возврат к администратору
```

```
GO
```

2. Проверка, что триггер НЕ сработает с повышенными правами:

- Подключитесь под администратором и выполните:

sql

```
USE TriggerSecurityLab;  
GO
```

```
-- Администратор обновляет данные (триггер сработает, но от имени администратора?)  
UPDATE dbo.SensitiveData SET SecretData = 'Измененные данные' WHERE ID = 1;  
GO
```

```
-- Проверка, получил ли MaliciousUser права SELECT?  
EXECUTE AS LOGIN = 'MaliciousUser';  
GO  
SELECT * FROM dbo.SensitiveData; -- Должно быть отказано!  
REVERT;  
GO
```

Скриншот: сообщение об ошибке доступа при попытке чтения от имени MaliciousUser.

3. **Безопасный способ: использование WITH EXECUTE AS**

- Создадим триггер, который выполняется от имени владельца схемы (с повышенными правами только для конкретных операций):

```
sql  
CREATE TRIGGER trg_SecureAudit  
ON dbo.SensitiveData  
AFTER UPDATE  
WITH EXECUTE AS OWNER -- Выполняется от имени владельца схемы  
AS  
BEGIN  
    -- Здесь можно выполнять операции, требующие повышенных прав  
    INSERT INTO dbo.DataChangeLog (TableName, ActionType, RecordID, ChangedBy)  
    SELECT 'SensitiveData', 'UPDATE', ID, SUSER_NAME()  
    FROM inserted;  
END;  
GO
```

Этап 6: Обнаружение и управление существующими триггерами (20 минут)

Научимся находить и отключать триггеры .

1. **Поиск всех триггеров в базе данных:**

```
sql  
-- DML и DDL триггеры уровня БД  
SELECT  
    name AS TriggerName,  
    type_desc AS TriggerType,  
    parent_class_desc AS Scope,  
    is_disabled AS IsDisabled,  
    create_date,  
    modify_date  
FROM sys.triggers  
ORDER BY type_desc;
```

2. **Поиск триггеров уровня сервера:**

```
sql  
SELECT  
    name,  
    type_desc,  
    is_disabled,  
    create_date  
FROM sys.server_triggers;
```

3. Отключение триггера:

sql

-- Отключение конкретного DML триггера

```
DISABLE TRIGGER trg_Employees_Audit ON dbo.Employees;  
GO
```

-- Отключение всех DDL триггеров в базе данных

```
DISABLE TRIGGER ALL ON DATABASE;  
GO
```

-- Отключение всех триггеров входа

```
DISABLE TRIGGER ALL ON ALL SERVER;  
GO
```

-- Включение обратно

```
ENABLE TRIGGER trg_Employees_Audit ON dbo.Employees;  
ENABLE TRIGGER ALL ON DATABASE;  
ENABLE TRIGGER ALL ON ALL SERVER;  
GO
```

Скриншот: результаты запросов к sys.triggers и sys.server_triggers.

Индивидуальные задания по вариантам

Каждый студент получает вариант задания и выполняет его самостоятельно, фиксируя результаты скриншотами.

Вариант	Тип триггера	Назначение	Дополнительные условия
1	DML (AFTER UPDATE)	Аудит изменений зарплаты: логировать старую и новую зарплату в отдельную таблицу SalaryAudit	Добавить проверку: запретить увеличение зарплаты более чем на 30% за один раз
2	DDL	Запретить изменение и удаление любых хранимых процедур в базе данных	Разрешить изменения только пользователю dbo
3	LOGON	Ограничить подключения для тестового пользователя TestUser временем с 9:00 до 18:00	Добавить исключение для подключений с локального компьютера (localhost)
4	DML (INSTEAD OF)	Создать представление vw_EmployeesSecure, скрывающее колонку PassportNumber. Через представление разрешить вставку и обновление, но при вставке автоматически шифровать паспортные данные	Использовать INSTEAD OF триггер на представлении
5	DDL	Логировать все попытки создания и удаления индексов в таблицу IndexChangeLog	Игнорировать автоматические операции по обслуживанию индексов (Rebuild, Reorganize)

Дополнительное задание для всех вариантов:

- Создайте тестового пользователя с минимальными правами и проверьте работу триггера от его имени.
- Сделайте скриншот успешных и неуспешных операций.
- Напишите запрос для просмотра собранных триггером данных.

Контрольные вопросы

1. В чем различие между AFTER и INSTEAD OF триггерами? В каких случаях применяется каждый тип ?
2. Какие виртуальные таблицы доступны в DML-триггере и что они содержат ?
3. Какую информацию можно получить с помощью функции EVENTDATA() в DDL-триггере ?
4. Почему триггеры могут представлять угрозу безопасности? Опишите сценарий "повышения привилегий" через триггер .
5. Как ограничить контекст выполнения триггера с помощью предложения WITH EXECUTE AS ?
6. Что произойдет, если триггер выполнит команду ROLLBACK? Как это повлияет на исходную операцию ?
7. Как временно отключить все триггеры в базе данных? Как найти все существующие триггеры ?
8. Можно ли с помощью триггера входа заблокировать подключение администратора? Как администратор может обойти блокировку ?

Содержание отчета:

1. Титульный лист, ФИО, группа, тема и цель работы.
2. Скриншоты ключевых этапов:
 - Создание DML-триггера для аудита изменений и результаты его работы (таблица DataChangeLog).
 - Создание DDL-триггера и попытка удаления таблицы (сообщение об ошибке).
 - Создание LOGON-триггера и результат попытки подключения при нарушении условий.
 - Демонстрация поиска существующих триггеров через системные представления.
 - Выполнение индивидуального задания (согласно варианту).
3. Листинги (текст) основных T-SQL запросов, разработанных в ходе работы.
4. Ответы на контрольные вопросы.
5. Вывод: какие возможности триггеров для обеспечения безопасности освоены, как они соотносятся с другими средствами защиты (аудит, разрешения).

// Методические указания для преподавателя

- **Подготовка:**
 - Убедитесь, что на сервере включен смешанный режим аутентификации.
 - Проверьте, что у студентов есть права на создание триггеров (минимум CREATE TRIGGER и ALTER на соответствующие объекты).
 - Для LOGON-триггеров требуется права CONTROL SERVER или ALTER ANY LOGON TRIGGER.
- **Возможные проблемы и их решение:**
 - **Триггер не срабатывает:** Проверьте, включен ли триггер (is_disabled = 0 в sys.triggers). Проверьте синтаксис и наличие ошибок в коде триггера.
 - **Ошибки с EVENTDATA():** Функция возвращает XML, требуется правильное извлечение значений с использованием .value() и пространств имен.
 - **LOGON-триггер блокирует все подключения:** Если триггер входа содержит ошибку и блокирует даже администратора, используйте **DAC (Dedicated Administrator Connection)** для подключения и отключения триггера. В SSMS строка подключения: admin:имя_сервера .
 - **Проблемы с правами:** Триггер может не иметь прав на запись в таблицу логирования. Решение: GRANT INSERT ON dbo.DataChangeLog TO public; или

использование WITH EXECUTE AS OWNER.

- **Рекомендации:**

- Акцентируйте внимание студентов на различии между аудитом через триггеры и официальным SQL Server Audit. Триггеры удобны для разработчиков и не требуют высоких привилегий для просмотра логов, но менее безопасны.

- Продемонстрируйте рекурсивное срабатывание триггеров и способы его предотвращения (триггеры не вызываются повторно по умолчанию, но можно включить RECURSIVE_TRIGGERS).

- Обсудите влияние триггеров на производительность (особенно курсоры внутри триггеров).

Практическое занятие №8. Документирование прав доступа и безопасность базы данных, включая учетные записи пользователей и их роли.

Цель работы: Научиться создавать учетные записи пользователей, управлять ролями и правами доступа к объектам базы данных, а также документировать политику безопасности.

Подготовительный этап (Проверка подключения)

1. Запустите **SQL Server Management Studio** от имени администратора (если доступно, если нет — просто запустите).
2. Подключитесь к вашему экземпляру SQL Server, используя учетную запись **sa** или учетную запись с правами **sysadmin** (вам выдает преподаватель).
 - **Сервер:** (обычно вводите название компьютера или (localdb)\MSSQLLocalDB, в зависимости от настройки вашего компьютера) .
 - **Аутентификация:** SQL Server Authentication.
 - **Логин:** sa (или ваш логин).
 - **Пароль:** (ваш пароль).

Краткие теоретические сведения

Безопасность SQL Server строится на иерархии вложенных друг в друга субъектов (securables):

1. **Имя входа (Login):** Создается на уровне сервера. Позволяет пользователю подключиться к SQL Server .
 - *Пример:* Создаем логин Ivanov_I с паролем.
2. **Пользователь (User):** Создается в конкретной базе данных. "Связывает" имя входа (Login) с этой базой данных. Без пользователя в базе, даже подключившись к серверу, вы не увидите данные .
 - *Пример:* В базе данных StudentDB создаем пользователя Ivanov_I, который привязан к логину Ivanov_I.
3. **Роль (Role):** Контейнер для прав. Права проще выдавать не конкретному пользователю, а роли, а затем добавлять пользователя в эту роль. Существуют **предопределенные роли** (например, db_datareader — чтение всех таблиц) и **пользовательские роли** (создаваемые вами) .
4. **Разрешение (Permission):** Право на выполнение конкретного действия (SELECT, INSERT, UPDATE, DELETE, CREATE TABLE, EXECUTE и т.д.) .
5. **Схема (Schema):** Контейнер для объектов (таблиц, представлений). Владелец схемы автоматически получает права на все объекты в ней.

Ход работы

Задание 1. Создание тестовой базы данных и таблиц

1. Нажмите **Новый запрос (New Query)**.
2. Скопируйте и выполните следующий скрипт для создания базы

данных Учебная_Безопасность и двух простых таблиц.

```
sql
```

```
-- Создаем базу данных
```

```
CREATE DATABASE Учебная_Безопасность;
```

```
GO
```

```
USE Учебная_Безопасность;
```

```
GO
```

```
-- Создаем таблицу Студенты
```

```
CREATE TABLE Студенты (
```

```
    ID INT PRIMARY KEY IDENTITY(1,1),
```

```
    Фамилия NVARCHAR(50),
```

```
    Имя NVARCHAR(50),
```

```
    Группа NVARCHAR(10),
```

```
    Средний_балл DECIMAL(3,2)
```

```
);
```

```
GO
```

```
-- Создаем таблицу Преподаватели
```

```
CREATE TABLE Преподаватели (
```

```
    ID INT PRIMARY KEY IDENTITY(1,1),
```

```
    Фамилия NVARCHAR(50),
```

```
    Имя NVARCHAR(50),
```

```
    Должность NVARCHAR(50),
```

```
    Зарплата DECIMAL(10,2) -- Конфиденциальная информация
```

```
);
```

```
GO
```

```
-- Наполним таблицы тестовыми данными
```

```
INSERT INTO Студенты (Фамилия, Имя, Группа, Средний_балл) VALUES
```

```
('Иванов', 'Иван', '3-ИС-1', 4.2),
```

('Петрова', 'Мария', '3-ИС-1', 4.8),
('Сидоров', 'Петр', '3-ИС-2', 3.9);

INSERT INTO Преподаватели (Фамилия, Имя, Должность, Зарплата) VALUES
('Смирнова', 'Елена', 'Доцент', 75000.00),
('Кузнецов', 'Андрей', 'Профессор', 120000.00);

PRINT 'База данных и таблицы созданы.';
GO

3. В **Обозревателе объектов (Object Explorer)** найдите и обновите папку **Базы данных (Databases)**. Убедитесь, что появилась база Учебная_Безопасность.

Задание 2. Создание учетных записей (Логинов) на уровне сервера

Мы создадим два логина: для обычного пользователя и для администратора базы данных. Выполнять будем через графический интерфейс.

1. В **Обозревателе объектов** найдите папку **Безопасность (Security)** на уровне **сервера** (самая верхняя папка) .
2. Раскройте папку **Безопасность**, найдите папку **Имена входа (Logins)**.
3. Нажмите на папку **Имена входа** правой кнопкой мыши -> выберите **Создать имя входа... (New Login...)** .

(Рис. 1: Контекстное меню папки Logins)

4. В открывшемся окне заполните поля:
 - **Имя входа (Login name):** StudentUser
 - **Проверка подлинности (Authentication):** выберите **Проверка подлинности SQL Server (SQL Server Authentication)** .
 - **Пароль (Password):** введите StrongPass123! (запомните его!).
 - **Подтверждение пароля (Confirm Password):** StrongPass123!.
 - **Снимите галочку "Требовать смену пароля при следующем входе"** (Enforce password policy может остаться, но в учебных целях это не критично, но лучше оставить для понимания политик безопасности).

5. Нажмите **ОК**. Логин создан.

6. Аналогично создайте второй логин с именем AdminUser и паролем AdminPass456!.

Задание 3. Создание пользователей в базе данных и сопоставление с логинами

Теперь нужно разрешить этим логинам "видеть" нашу базу данных 'Учебная_Безопасность».

1. В **Обозревателе объектов** раскройте дерево до **Базы данных (Databases)** -

> Учебная_Безопасность -> Безопасность (Security) -> Пользователи (Users) .

2. Нажмите правой кнопкой на папку **Пользователи** -> выберите **Создать пользователя... (New User...)**.

3. В открывшемся окне:

- **Тип пользователя (User type):** оставьте Пользователь SQL с именем входа (SQL user with login) .
- **Имя пользователя (User name):** введите StudentUser (желательно, чтобы совпадало с логином для простоты).
- **Имя входа (Login name):** нажмите кнопку с многоточием (...), найдите в списке StudentUser и выберите его .
- **Схема по умолчанию (Default schema):** оставьте dbo (это стандартная схема, где лежат наши таблицы).
- Нажмите **ОК**.

(Рис. 2: Окно создания пользователя с выбором логина)

4. Аналогично создайте пользователя AdminUser для логина AdminUser в той же базе данных.

Задание 4. Назначение прав через предопределенные роли (Самый простой способ)

Сейчас оба пользователя могут подключиться к серверу и к базе данных, но ничего не увидят, так как у них нет прав на чтение.

1. **Назначим права "Чтеца" для StudentUser:**

- В той же базе данных Учебная_Безопасность найдите папку **Пользователи** и дважды кликните на пользователя StudentUser (или правой кнопкой -> **Свойства**).
- Перейдите на страницу **Членство (Membership)** .
- В списке "Членство в роли базы данных" найдите роль **db_datareader** и поставьте напротив нее галочку.
- **Пояснение:** Роль db_datareader — это предопределенная роль, которая дает право **чтения (SELECT)** всех данных из всех пользовательских таблиц .

○ Нажмите **ОК**.

2. **Назначим права "Владельца" для AdminUser:**

- Откройте свойства пользователя AdminUser.
- Перейдите на страницу **Членство**.
- Поставьте галочку напротив роли **db_owner**.
- **Внимание!** Роль db_owner дает **абсолютно все права** на базу данных (создание, удаление, изменение объектов, управление пользователями и т.д.). В реальной жизни такую роль дают только самым доверенным администраторам .

- Нажмите **ОК**.

Задание 5. Проверка прав доступа (Самый важный момент)

Сейчас мы проверим, как работают созданные нами пользователи. Для этого в SSMS нужно открыть **новое** окно подключения, используя учетные данные нового пользователя.

1. В SSMS нажмите **Файл (File) -> Создать (New) -> Соединение с ядром СУБД (Database Engine Query)** .

2. Откроется окно подключения. Введите:

- **Сервер:** (тот же, что и в начале работы).
- **Аутентификация:** Проверка подлинности SQL Server.
- **Имя входа:** StudentUser
- **Пароль:** StrongPass123!

3. Подключитесь. В новом окне запроса введите команду:

sql

SELECT * FROM Учебная_Безопасность.dbo.Студенты;

SELECT * FROM Учебная_Безопасность.dbo.Преподаватели;

4. **Результат:** Вы должны увидеть данные из **обеих** таблиц, так как роль db_datareader дает право читать все таблицы.

5. Теперь попробуйте изменить данные:

sql

UPDATE Учебная_Безопасность.dbo.Студенты SET Имя = 'Иванчик' WHERE ID = 1;

6. **Результат:** Вы должны получить **ошибку**. Это правильно, так как у StudentUser нет прав на изменение (роль db_datawriter не назначена).

7. Закройте это окно подключения (не закрывая основное окно администратора).

8. Аналогично создайте новое подключение для AdminUser (пароль AdminPass456!).

○ Выполните запрос на изменение данных. Он должен выполняться **успешно**, так как роль db_owner дает все права.

Задание 6. Тонкая настройка прав: создание пользовательской роли и выдача прав на уровне таблиц

Предположим, нам нужен пользователь "Кадровик", который должен видеть фамилии преподавателей, но не должен видеть их зарплату.

1. **Создайте новый логин HR_User** (как в Задании 2) с паролем HRPass000!.

2. **Создайте для него пользователя HR_User** в базе Учебная_Безопасность (как в Задании 3).

3. Теперь создадим **пользовательскую роль**.

- В базе Учебная_Безопасность раскройте папку **Роли (Roles) -> Роли базы данных**

(Database Roles).

- Нажмите правой кнопкой -> **Создать роль базы данных... (New Database Role...)** .
- **Имя роли (Role name):** Role_HR_ReadOnly.
- **Владелец (Owner):** оставьте dbo.
- Пока ничего не добавляйте. Нажмите **ОК**.
- 4. Добавим пользователя HR_User в нашу новую роль.
 - Откройте свойства роли Role_HR_ReadOnly (двойным кликом).
 - Перейдите на страницу **Членство**.
 - Нажмите **Добавить (Add)**, найдите пользователя HR_User и добавьте его.

Нажмите **ОК**.

- 5. Теперь самое сложное — выдадим разрешения **на уровне таблицы**.
 - Откройте свойства роли Role_HR_ReadOnly снова.
 - Перейдите на страницу **Защищаемые объекты (Securables)** .
 - Нажмите кнопку **Найти... (Search...)**.
 - Выберите **Все объекты типов... (All objects of the types...)** -> **ОК**.
 - В новом окне выберите типы объектов: поставьте галочку **Таблицы (Tables)** -> **ОК** .
 - Должно появиться окно выбора конкретных таблиц. Нажмите **Обзор... (Browse...)**.
 - В списке найдите нашу базу Учебная_Безопасность, раскройте ее, найдите **Таблицы**

(Tables) и поставьте галочки напротив dbo.Преподаватели и dbo.Студенты. Нажмите **ОК** .

- Теперь в нижней части окна ("Явные разрешения...") вы увидите список всех возможных действий для таблиц Преподаватели и Студенты.

- Для таблицы Студенты поставьте галочку **Разрешить (Grant)** напротив **SELECT**.
- Для таблицы Преподаватели нам нужно дать доступ не ко всем столбцам. Такой тонкой настройки через графику в учебных версиях может не быть (или это сложно), поэтому мы используем скрипт (см. шаг 6).

6. **Используем T-SQL для точной настройки (Выполните этот скрипт в окне запроса под администратором):**

```
sql
```

```
USE Учебная_Безопасность;
```

```
GO
```

```
-- Запрещаем все права на таблицу Преподаватели для нашей роли (на всякий случай)
```

```
REVOKE ALL ON Преподаватели TO Role_HR_ReadOnly;
```

```
GO
```

```
-- Разрешаем SELECT только на определенные столбцы (Фамилия, Имя, Должность)
GRANT SELECT (Фамилия, Имя, Должность) ON Преподаватели TO Role_HR_ReadOnly;
GO
```

-- Разрешаем SELECT на все столбцы таблицы Студенты (это дублирует графическую настройку)

```
GRANT SELECT ON Студенты TO Role_HR_ReadOnly;
GO
```

7. Проверка:

- Создайте новое подключение от имени HR_User.
- Выполните запрос `SELECT * FROM Учебная_Безопасность.dbo.Студенты`; — все должно работать.
- Выполните запрос `SELECT * FROM Учебная_Безопасность.dbo.Преподаватели`; — вы получите ошибку, так как `*` пытается выбрать все столбцы, включая защищенный столбец Зарплата.
- Выполните корректный запрос:
sql
`SELECT Фамилия, Имя, Должность FROM Учебная_Безопасность.dbo.Преподаватели`;
Этот запрос должен выполняться **успешно**.
- Попробуйте обновить данные: `UPDATE Учебная_Безопасность.dbo.Студенты SET Имя = 'Тест' WHERE ID = 1`; — должна быть ошибка, так как прав на изменение нет.

Задание 7. Документирование прав доступа

Составьте отчет о политике безопасности. Это можно сделать как в Word, так и в виде комментария в файле отчета.

Таблица 1. Учетные записи (логины)

Имя логина	Тип аутентификации	Пароль (учебный)	Назначение
StudentUser	SQL Server	StrongPass123!	Обычный студент для чтения
AdminUser	SQL Server	AdminPass456!	Администратор БД
HR_User	SQL Server	HRPass000!	Сотрудник отдела кадров

Таблица 2. Пользователи и их роли в базе данных Учебная_Безопасность

Имя пользователя	Связанный логин	Членство в ролях БД	Прямые разрешения
StudentUser	StudentUser	db_datareader	Нет
AdminUser	AdminUser	db_owner	Нет

HR_User	HR_User	Role_HR_ReadOnly	Нет
---------	---------	------------------	-----

Таблица 3. Разрешения для пользовательской роли Role_HR_ReadOnly

Тип объекта	Имя объекта	Тип разрешения	Разрешено/Запрещено	Примечание
Таблица	Студенты	SELECT	Grant	На все столбцы
Таблица	Преподаватели	SELECT	Grant	Только на столбцы Фамилия, Имя, Должность
Таблица	Студенты	INSERT/UPDATE/DELETE	- (не задано)	По умолчанию запрещено
Таблица	Преподаватели	INSERT/UPDATE/DELETE	- (не задано)	По умолчанию запрещено

Контрольные вопросы:

1. В чем разница между логином (Login) и пользователем (User)? Может ли быть логин без пользователя?
2. Что такое предопределенная роль базы данных? Какие еще предопределенные роли вы знаете (кроме db_datareader и db_owner)?
3. Зачем создавать пользовательские роли, если можно права давать напрямую пользователю?
4. Что такое схема (Schema) и как она связана с безопасностью?

Задание на дом (если не успели на паре):

Добавьте в базу еще одну таблицу "Успеваемость" и создайте роль "Куратор", которая может читать данные из таблиц "Студенты" и "Успеваемость", но не может их изменять. Также куратор не должен иметь доступа к таблице "Преподаватели". Отрадите изменения в документации (Таблице 3).

Практическое занятие №9. Создание пользователей и назначение ролей. Управление правами доступа пользователей на уровне сервера, баз данных и данных.

Цель работы: Научиться создавать учётные записи (логины) на уровне сервера, сопоставлять их с пользователями базы данных, назначать роли и индивидуальные разрешения на объекты, а также разграничивать доступ к данным на уровне столбцов и строк.

Необходимые знания

- Структура безопасности SQL Server: **Login** (сервер) → **User** (БД) → **Permissions**.
- Предопределённые серверные роли (например, sysadmin, securityadmin, dbcreator).
- Предопределённые роли базы данных (db_datareader, db_datawriter, db_owner и др.).
- Понятие схемы (schema).

Подготовительный этап

1. Запустите **SQL Server Management Studio**.
2. Подключитесь к экземпляру SQL Server с правами администратора (например, под учётной записью sa или доменной учётной записью, входящей в роль sysadmin). Эти учётные данные вам должен предоставить преподаватель.
3. Убедитесь, что в списке баз данных есть база **master** (она нужна для создания логинов).

Ход работы

Задание 1. Создание логинов на серверном уровне

Логин (login) – это учётная запись, которая позволяет подключиться к SQL Server. Логины хранятся в базе данных **master**.

1.1. Создание логина с помощью графического интерфейса

1. В **Object Explorer** раскройте папку вашего сервера.
2. Раскройте папку **Security** (Безопасность).
3. Щёлкните правой кнопкой мыши на папке **Logins** (Имена входа) и выберите **New Login...** (Создать имя входа).
4. В открывшемся окне:
 - **Login name:** введите Manager_Login.
 - Выберите **SQL Server authentication**.
 - **Password:** введите MgrPass123!.
 - **Confirm password:** повторите пароль.
 - Снимите флажок **User must change password at next login** (Пользователь должен сменить пароль при следующем входе) – для учебных целей это упростит работу.

5. В левой панели перейдите на страницу **Server Roles** (Серверные роли).

○ Здесь можно назначить логину глобальные права на весь сервер. Например, поставьте галочку напротив роли **dbcreator** – это позволит пользователю создавать свои базы данных.

6. Нажмите **ОК**.

1.2. Создание второго логина с помощью T-SQL (альтернативный способ)

Откройте новый запрос (New Query) и выполните следующий код. Он создаст логин Employee_Login с паролем и даст ему минимальные права (никаких серверных ролей).

```
sql
```

```
USE master;
```

```
GO
```

```
CREATE LOGIN Employee_Login WITH PASSWORD = 'EmpPass456!';
```

```
GO
```

```
-- Назначим этому логину роль securityadmin (может управлять другими логинами)
```

```
ALTER SERVER ROLE securityadmin ADD MEMBER Employee_Login;
```

```
GO
```

Примечание: Серверные роли securityadmin позволяет управлять логинами, а dbcreator – создавать базы данных. Будьте осторожны с ролью sysadmin – она даёт неограниченные права.

Задание 2. Создание тестовой базы данных и схем

Для отработки прав нам потребуется база данных с несколькими таблицами.

1. Создайте новую базу данных CompanyDB (можно через контекстное меню **Databases** → **New Database...** или выполните скрипт):

```
sql
```

```
CREATE DATABASE CompanyDB;
```

```
GO
```

```
USE CompanyDB;
```

```
GO
```

2. Создайте две схемы (схемы помогают группировать объекты и управлять правами):

```
sql
```

```
CREATE SCHEMA Sales;
```

```
GO
```

```
CREATE SCHEMA HR;
```

```
GO
```

3. Создайте таблицы в разных схемах:

sql

-- Таблица в схеме Sales

```
CREATE TABLE Sales.Orders (  
    OrderID INT PRIMARY KEY IDENTITY(1,1),  
    CustomerName NVARCHAR(100),  
    OrderDate DATE,  
    TotalAmount DECIMAL(10,2)  
);  
GO
```

-- Таблица в схеме HR

```
CREATE TABLE HR.Employees (  
    EmployeeID INT PRIMARY KEY IDENTITY(1,1),  
    FullName NVARCHAR(100),  
    Position NVARCHAR(50),  
    Salary DECIMAL(10,2) -- конфиденциальная информация  
);  
GO
```

-- Добавим немного тестовых данных

```
INSERT INTO Sales.Orders (CustomerName, OrderDate, TotalAmount)  
VALUES ('Иванов Иван', '2025-02-01', 1500.00),  
      ('Петров Петр', '2025-02-03', 2300.50);
```

```
INSERT INTO HR.Employees (FullName, Position, Salary)  
VALUES ('Сидорова Мария', 'Менеджер', 50000.00),  
      ('Козлов Дмитрий', 'Разработчик', 70000.00);  
GO
```

Задание 3. Создание пользователей базы данных

Чтобы логин мог работать с объектами внутри базы данных, ему необходимо сопоставить **пользователя** (user) в этой базе.

3.1. Создание пользователя для логина Manager_Login

1. Перейдите в базу данных **CompanyDB**.
2. Раскройте папку **Security** → **Users**.

3. Щёлкните правой кнопкой на **Users** → **New User....**

4. В окне создания пользователя:

- **User type:** оставьте SQL user with login.

- **User name:** введите Manager_User (можно и совпадающее с логином, но здесь мы покажем, что имена могут отличаться).

- **Login name:** нажмите кнопку с многоточием (...), найдите и выберите Manager_Login.

- **Default schema:** укажите Sales – теперь при обращении к таблицам без указания схемы будет подставляться Sales.

5. Нажмите **ОК**.

3.2. Создание пользователя для логина Employee_Login (через T-SQL)

Выполните следующий код в контексте базы CompanyDB:

```
sql
```

```
USE CompanyDB;
```

```
GO
```

```
CREATE USER Employee_User FOR LOGIN Employee_Login;
```

```
GO
```

```
-- По умолчанию схема – dbo, можно изменить:
```

```
ALTER USER Employee_User WITH DEFAULT_SCHEMA = HR;
```

```
GO
```

Задание 4. Назначение прав через роли базы данных

4.1. Фиксированные роли базы данных

Дадим пользователю Manager_User права на чтение и изменение данных во всех таблицах (но не на создание объектов). Для этого добавим его в роль **db_datareader** и **db_datawriter**.

1. Откройте свойства пользователя Manager_User (двойным щелчком).

2. Перейдите на страницу **Membership** (Членство).

3. В списке ролей отметьте:

- **db_datareader**

- **db_datawriter**

4. Нажмите **ОК**.

Объяснение: db_datareader разрешает SELECT из всех таблиц, db_datawriter разрешает INSERT, UPDATE, DELETE.

4.2. Создание пользовательской роли и выборочные права

Предположим, что сотрудник отдела кадров (Employee_User) должен видеть только

фамилии и должности сотрудников, но не зарплату. Ему также нельзя видеть заказы (таблица Sales.Orders).

Выполните следующие шаги:

1. Создайте пользовательскую роль в базе данных:

sql

```
USE CompanyDB;
```

```
GO
```

```
CREATE ROLE HR_ReadOnly;
```

```
GO
```

2. Добавьте пользователя Employee_User в эту роль:

sql

```
ALTER ROLE HR_ReadOnly ADD MEMBER Employee_User;
```

```
GO
```

3. Запретим доступ к таблице заказов (явный запрет, хотя по умолчанию доступа нет, но для надёжности):

sql

```
DENY SELECT ON Sales.Orders TO HR_ReadOnly;
```

```
GO
```

4. Разрешим доступ только к определённым столбцам таблицы HR.Employees:

sql

```
GRANT SELECT (EmployeeID, FullName, Position) ON HR.Employees TO HR_ReadOnly;
```

```
GO
```

Теперь пользователь Employee_User сможет видеть только три столбца из таблицы Employees и не увидит Orders.

Задание 5. Управление доступом на уровне строк (через представление)

Иногда требуется ограничить доступ не по столбцам, а по строкам. Например, пусть пользователь Manager_User видит только те заказы, у которых сумма меньше 2000. Достичь этого можно созданием представления (view) и выдачей прав на него, а не на базовую таблицу.

1. Создайте представление, которое фильтрует строки:

sql

```
CREATE VIEW Sales.OrdersUnder2000
```

```
AS
```

```
SELECT OrderID, CustomerName, OrderDate, TotalAmount
```

```
FROM Sales.Orders
```

```
WHERE TotalAmount < 2000;
```

```
GO
```

2. Запретим менеджеру прямой доступ к таблице Orders (если он его имел через роль db_datareader, то нужно отозвать):

```
sql
```

```
-- Отзываем право SELECT на таблицу Orders у роли db_datareader для этого пользователя?
```

```
-- Но db_datareader действует на все таблицы. Лучше вывести пользователя из db_datareader и дать права точно.
```

```
-- Поступим иначе: создадим новую роль и назначим Manager_User в неё, а из db_datareader удалим.
```

Чтобы не усложнять, давайте просто создадим отдельного пользователя для этого примера.

Но в рамках экономии времени предложите студентам следующий подход:

- Удалить пользователя Manager_User из роли db_datareader (через свойства пользователя, снять галочку).
- Выдать ему разрешение SELECT только на представление Sales.OrdersUnder2000 и на все столбцы таблицы HR.Employees (для примера).

Выполните:

```
sql
```

```
-- Удаляем из роли db_datareader
```

```
EXEC sp_droprolemember 'db_datareader', 'Manager_User';
```

```
GO
```

```
-- Даём права на представление
```

```
GRANT SELECT ON Sales.OrdersUnder2000 TO Manager_User;
```

```
GO
```

```
-- Даём права на всю таблицу Employees (для упрощения)
```

```
GRANT SELECT ON HR.Employees TO Manager_User;
```

```
GO
```

3. Проверка будет позже.

Задание 6. Проверка прав доступа (создание нескольких подключений)

Теперь проверим, как работают созданные нами пользователи. Для этого в SSMS нужно открыть **новые окна подключения** с разными учётными данными.

Проверка для пользователя Manager_User

1. Нажмите **Файл** → **Создать** → **Соединение с ядром СУБД**.

2. В окне подключения укажите:

- Сервер: (тот же, что и ранее)
- Аутентификация: **Проверка подлинности SQL Server**
- Имя входа: Manager_Login
- Пароль: MgrPass123!

3. Подключитесь.

4. В новом окне запроса выполните команды:

```
sql
```

```
USE CompanyDB;
```

```
GO
```

```
SELECT * FROM Sales.OrdersUnder2000; -- должно вернуть строки с суммой < 2000
```

```
GO
```

```
SELECT * FROM HR.Employees; -- должно вернуть все строки (право дано)
```

```
GO
```

```
SELECT * FROM Sales.Orders; -- ошибка, так как нет прямого доступа к таблице
```

```
GO
```

Первые два запроса должны выполняться успешно, третий – вернуть ошибку.

Проверка для пользователя Employee_User

1. Аналогично создайте новое подключение

для Employee_Login (пароль EmpPass456!).

2. Выполните:

```
sql
```

```
USE CompanyDB;
```

```
GO
```

```
SELECT * FROM HR.Employees; -- ошибка? Нет, но будут видны только
```

разрешённые столбцы

```
GO
```

```
SELECT EmployeeID, FullName, Position FROM HR.Employees; -- успех
```

```
GO
```

```
SELECT Salary FROM HR.Employees; -- ошибка: доступ запрещён к столбцу Salary
```

```
GO
```

```
SELECT * FROM Sales.Orders; -- ошибка: доступ запрещён (DENY)
```

```
GO
```

Убедитесь, что ошибки соответствуют ожиданиям.

Задание 7. Документирование настроек безопасности

Составьте в отчёте таблицы, описывающие созданные учётные записи и их права.

Таблица 1. Логины и серверные роли

Имя логина	Серверные роли	Пароль (учебный)
Manager_Login	dbcreator	MgrPass123!
Employee_Login	securityadmin	EmpPass456!

Таблица 2. Пользователи в базе CompanyDB

Имя пользователя	Связанный логин	Схема по умолчанию	Членство в ролях БД
Manager_User	Manager_Login	Sales	(выведен из db_datareader)
Employee_User	Employee_Login	HR	HR_ReadOnly

Таблица 3. Разрешения для пользователей / ролей

Роль / Пользователь	Тип объекта	Имя объекта	Разрешение	Примечание
Manager_User	VIEW	Sales.OrdersUnder2000	SELECT	Через прямое разрешение
Manager_User	TABLE	HR.Employees	SELECT	Дано напрямую
HR_ReadOnly	TABLE	HR.Employees	SELECT	Только столбцы ID, FullName, Position
HR_ReadOnly	TABLE	Sales.Orders	DENY SELECT	Явный запрет

Контрольные вопросы

1. В чём отличие логина от пользователя базы данных? Может ли существовать пользователь без логина?
2. Какие серверные роли вы знаете? Для чего используется роль securityadmin?
3. Как ограничить доступ пользователя к определённым столбцам таблицы?
4. Как ограничить доступ к определённым строкам таблицы?
5. Что произойдёт, если пользователь является членом двух ролей, одна из которых даёт разрешение, а другая запрещает?

Практическое занятие №10. Создание сложной структуры ролей. Использование методов шифрования паролей.

Цель работы: Научиться создавать иерархическую структуру ролей для эффективного управления доступом, а также освоить методы безопасного хранения паролей с использованием хеширования и «соли».

Теоретическая часть

1. Иерархия ролей в SQL Server

Управление доступом в SQL Server строится на принципах, аналогичных группам безопасности в Windows . Вместо выдачи прав каждому пользователю индивидуально, мы создаём **роли** (группы прав) и добавляем пользователей в эти роли.

Типы ролей:

- **Предопределенные роли сервера** — действуют на уровне всего экземпляра SQL Server (sysadmin, securityadmin, dbcreator и др.) .
- **Предопределенные роли базы данных** — встроены в каждую базу данных (db_owner, db_datareader, db_datawriter, db_ddladmin и др.) .
- **Определяемые пользователем роли** — создаются администратором для специфических задач.

Иерархия ролей означает, что одна роль может быть членом другой роли, наследуя все её права. Это позволяет создавать гибкие структуры: например, роль «Младший разработчик» может входить в роль «Разработчик» и получать её базовые права плюс дополнительные .

Важно! Не рекомендуется добавлять пользовательские роли в предопределенные роли, так как это может привести к непреднамеренному повышению прав .

2. Принципы безопасного хранения паролей

Хранение паролей в базе данных — ответственная задача. Категорически **запрещено** хранить пароли в открытом виде (**plain text**) .

Основные принципы :

1. **Необратимость** — пароль должен храниться в таком виде, из которого невозможно восстановить исходное значение.
2. **Уникальность** — одинаковые пароли разных пользователей должны храниться по-разному.
3. **Устойчивость к атакам** — защита от радужных таблиц и перебора.

Хеширование (Hashing) — процесс преобразования входных данных любой длины в выходную строку фиксированной длины. Хеш-функции необратимы: по хешу нельзя восстановить исходный пароль .

«Соль» (Salt) — случайная строка, которая добавляется к паролю перед хешированием. У каждого пользователя своя «соль». Это делает бесполезными радужные таблицы (precomputed hash tables), так как даже одинаковые пароли с разной «солью» дают разные хеши .

Рекомендуемые алгоритмы :

- SHA2_256, SHA2_512 (встроены в SQL Server через функцию HASHBYTES)
- Для промышленной разработки лучше использовать PBKDF2, bcrypt на стороне приложения

Функция HASHBYTES в SQL Server :

sql

HASHBYTES('SHA2_256', 'строка для хеширования')

Доступные алгоритмы: MD2, MD4, MD5, SHA, SHA1, SHA2_256, SHA2_512. **Важно:** MD5 и SHA1 считаются устаревшими и небезопасными .

Ход работы

Задание 1. Подготовка среды и создание базы данных

1. Запустите **SQL Server Management Studio**.
2. Подключитесь к экземпляру SQL Server с правами администратора (учётная запись sa или ваша доменная учётная запись) .
3. Нажмите **Новый запрос (New Query)** и выполните следующий скрипт для создания рабочей базы данных:

```
sql
-- Создаем базу данных для практической работы
CREATE DATABASE SecurityLab;
GO
```

```
USE SecurityLab;
GO
```

```
-- Создадим схему для пользовательских данных
CREATE SCHEMA Users;
GO
```

```
PRINT 'База данных SecurityLab создана';
GO
```

4. В **Object Explorer** найдите созданную базу данных SecurityLab, раскройте её и убедитесь, что папка Security → Schemas содержит схему Users.

Задание 2. Создание таблицы для хранения пользователей с паролями

Теперь создадим таблицу, в которой будут храниться учётные записи наших будущих пользователей. Таблица будет содержать **хеш пароля** и **соль** .

Выполните скрипт:

```
sql
USE SecurityLab;
GO
```

```
CREATE TABLE Users.Accounts (
    UserID INT PRIMARY KEY IDENTITY(1,1),
    Username NVARCHAR(50) NOT NULL UNIQUE,
    Email NVARCHAR(100),
    PasswordHash VARBINARY(64) NOT NULL, -- SHA2_256 дает 32 байта, SHA2_512 - 64
байта
    PasswordSalt VARBINARY(16) NOT NULL, -- 16 байт соли достаточно
    CreatedDate DATETIME DEFAULT GETDATE(),
    LastLoginDate DATETIME NULL
);
GO
```

```
PRINT 'Таблица Users.Accounts создана';
GO
```

Пояснение:

- PasswordHash — будет хранить результат хеширования пароля вместе с солью
- PasswordSalt — случайные 16 байт для каждого пользователя
- VARBINARY — двоичный тип данных, оптимальный для хешей

Задание 3. Создание логинов на уровне сервера

Для демонстрации работы ролей создадим несколько логинов (учётных записей для подключения к SQL Server) .

Способ 1: через графический интерфейс

1. В **Object Explorer** раскройте папку сервера → **Security**.
2. Правой кнопкой на **Logins** → **New Login....**
3. Создайте первый логин:
 - **Login name:** JuniorDev
 - **Authentication:** SQL Server authentication
 - **Password:** JuniorPass123! (снимите галочку "User must change password...")
4. Нажмите **ОК**.

Способ 2: через T-SQL

Выполните следующий скрипт для создания остальных логинов:

```
sql
```

```
USE master;
```

```
GO
```

```
CREATE LOGIN SeniorDev WITH PASSWORD = 'SeniorPass456!';
```

```
CREATE LOGIN ProjectManager WITH PASSWORD = 'ManagerPass789!';
```

```
CREATE LOGIN DBA WITH PASSWORD = 'DBAPass000!';
```

```
GO
```

```
PRINT 'Логины созданы';
```

```
GO
```

Задание 4. Создание пользователей в базе данных

Каждому логину необходимо сопоставить **пользователя** в нашей базе данных SecurityLab.

Только после этого логин сможет работать с объектами базы .

```
sql
```

```
USE SecurityLab;
```

```
GO
```

```
CREATE USER JuniorDev FOR LOGIN JuniorDev;
```

```
CREATE USER SeniorDev FOR LOGIN SeniorDev;
```

```
CREATE USER ProjectManager FOR LOGIN ProjectManager;
```

```
CREATE USER DBA FOR LOGIN DBA;
```

```
GO
```

```
PRINT 'Пользователи созданы в базе SecurityLab';
```

```
GO
```

Задание 5. Создание иерархической структуры ролей

Теперь создадим пользовательские роли и выстроим иерархию .

Бизнес-сценарий:

- **Базовый уровень** — все сотрудники могут просматривать общую информацию (роль Employees)
- **Уровень разработчиков** — могут читать и изменять данные, но не структуру (роль Developers, входит в Employees)
- **Уровень старших разработчиков** — могут создавать и изменять структуру таблиц (роль SeniorDevelopers, входит в Developers)
- **Уровень администраторов** — полный доступ (роль Admins, входит в SeniorDevelopers)
- **Отдельная роль для проектных менеджеров** — только чтение отчётов (роль Managers, входит в Employees)

Выполните скрипт:

```
sql
```

```
USE SecurityLab;
```

```
GO
```

```
-- Создаем роли
CREATE ROLE Employees;
CREATE ROLE Developers;
CREATE ROLE SeniorDevelopers;
CREATE ROLE Admins;
CREATE ROLE Managers;
GO
```

```
-- Выстраиваем иерархию (роли входят в другие роли)
EXEC sp_addrolemember 'Employees', 'Developers';
EXEC sp_addrolemember 'Developers', 'SeniorDevelopers';
EXEC sp_addrolemember 'SeniorDevelopers', 'Admins';
EXEC sp_addrolemember 'Employees', 'Managers';
GO
```

```
PRINT 'Иерархия ролей создана';
GO
```

Задание 6. Назначение прав ролям

Теперь наделим каждую роль определёнными разрешениями .

```
sql
USE SecurityLab;
GO
```

```
-- 1. Роль Employees: только чтение определённых таблиц (создадим позже)
-- Пока дадим право на чтение схемы
GRANT SELECT ON SCHEMA::Users TO Employees;
GO
```

```
-- 2. Роль Developers: возможность изменять данные в таблицах
GRANT INSERT, UPDATE, DELETE ON SCHEMA::Users TO Developers;
GO
```

```
-- 3. Роль SeniorDevelopers: возможность создавать и изменять таблицы
-- (на уровне схемы Users)
GRANT ALTER, CREATE TABLE TO SeniorDevelopers; -- разрешения на уровне базы
GRANT ALTER ON SCHEMA::Users TO SeniorDevelopers; -- возможность изменять
```

объекты в схеме

```
GO
```

```
-- 4. Роль Admins: все права на базу данных
EXEC sp_addrolemember 'db_owner', 'Admins';
GO
```

```
-- 5. Роль Managers: только чтение (но не изменение)
GRANT SELECT ON SCHEMA::Users TO Managers;
-- Явно запретим изменение
DENY INSERT, UPDATE, DELETE ON SCHEMA::Users TO Managers;
GO
```

```
PRINT 'Права ролям назначены';
GO
```

Важное замечание: Некоторые разрешения (например, CREATE TABLE) выдаются на уровне базы данных, а не на уровне схемы. Поэтому роль SeniorDevelopers получила разрешение CREATE TABLE на всю базу, но ALTER ON SCHEMA::Users ограничивает её возможности только схемой Users.

Задание 7. Добавление пользователей в роли

Теперь распределим наших пользователей по ролям согласно их должностям :

sql

USE SecurityLab;

GO

-- JuniorDev входит только в роль Developers (через неё получает права Employees)

EXEC sp_addrolemember 'Developers', 'JuniorDev';

-- SeniorDev входит в SeniorDevelopers (включает права Developers и Employees)

EXEC sp_addrolemember 'SeniorDevelopers', 'SeniorDev';

-- ProjectManager входит в Managers (включает права Employees с запретом на изменение)

EXEC sp_addrolemember 'Managers', 'ProjectManager';

-- DBA входит в Admins (полный доступ)

EXEC sp_addrolemember 'Admins', 'DBA';

GO

-- Проверим членство в ролях

SELECT

u.name AS UserName,

r.name AS RoleName

FROM sys.database_role_members rm

JOIN sys.database_principals u ON rm.member_principal_id = u.principal_id

JOIN sys.database_principals r ON rm.role_principal_id = r.principal_id

WHERE u.type_desc = 'SQL_USER'

ORDER BY u.name, r.name;

GO

Задание 8. Создание тестовых данных и проверка прав

Создадим тестовую таблицу и проверим, как работают права у разных пользователей.

sql

USE SecurityLab;

GO

-- Создадим таблицу для тестирования

CREATE TABLE Users.EmployeeInfo (

EmpID INT PRIMARY KEY IDENTITY(1,1),

FullName NVARCHAR(100),

Department NVARCHAR(50),

Salary DECIMAL(10,2),

Email NVARCHAR(100)

);

GO

-- Добавим тестовые данные

INSERT INTO Users.EmployeeInfo (FullName, Department, Salary, Email)

VALUES

```
('Иванов Иван', 'IT', 75000, 'ivanov@company.com'),  
('Петрова Мария', 'HR', 65000, 'petrova@company.com'),  
('Сидоров Петр', 'IT', 80000, 'sidorov@company.com');
```

GO

Теперь проверим права, создавая новые подключения от имени разных пользователей.

Проверка 1: Пользователь JuniorDev

1. В SSMS нажмите **Файл** → **Создать** → **Соединение с ядром СУБД**.
2. Введите:
 - Имя входа: JuniorDev
 - Пароль: JuniorPass123!
3. Выполните запрос:

```
sql
```

```
USE SecurityLab;
```

GO

```
SELECT * FROM Users.EmployeeInfo; -- Должно работать (чтение через роль Employees)
```

GO

```
UPDATE Users.EmployeeInfo SET Email = 'new@mail.com' WHERE EmpID = 1; -- Должно  
работать (через роль Developers)
```

GO

```
-- Попробуйте создать таблицу
```

```
CREATE TABLE Users.Test (ID INT); -- Должна быть ОШИБКА (нет прав)
```

GO

Проверка 2: Пользователь SeniorDev

В новом окне подключения от имени SeniorDev (пароль SeniorPass456!) выполните:

```
sql
```

```
USE SecurityLab;
```

GO

```
-- Чтение и изменение работают (наследуется от Developers)
```

```
SELECT * FROM Users.EmployeeInfo;
```

```
UPDATE Users.EmployeeInfo SET Department = 'IT-Department' WHERE EmpID = 1;
```

GO

```
-- Попробуйте создать таблицу
```

```
CREATE TABLE Users.Projects (  
    ProjectID INT PRIMARY KEY,  
    ProjectName NVARCHAR(100)
```

```
); -- Должно работать (право CREATE TABLE у SeniorDevelopers)
```

GO

```
-- Попробуйте изменить структуру существующей таблицы
```

```
ALTER TABLE Users.EmployeeInfo ADD Phone NVARCHAR(20); -- Должно работать  
(ALTER ON SCHEMA)
```

GO

Проверка 3: Пользователь ProjectManager

В новом окне подключения от имени ProjectManager (пароль ManagerPass789!) выполните:

```
sql
```

```
USE SecurityLab;
```

GO

```
SELECT * FROM Users.EmployeeInfo; -- Должно работать (роль Managers имеет SELECT)
```

GO

```
UPDATE Users.EmployeeInfo SET Salary = 100000 WHERE EmpID = 1; -- Должна быть  
ОШИБКА (DENY)
```

GO

Проверка 4: Пользователь DBA

В новом окне от имени DBA (пароль DBAPass000!) проверьте, что доступны все операции, включая удаление объектов:

```
sql
USE SecurityLab;
GO
-- DBA может всё как член db_owner
DROP TABLE Users.Projects; -- Должно работать
GO
```

Задание 9. Реализация безопасного хранения паролей

Теперь вернёмся к нашей таблице Users.Accounts и реализуем процедуры регистрации и проверки пароля с использованием **хеширования и соли**.

9.1. Создание процедуры регистрации нового пользователя

Выполните скрипт для создания хранимой процедуры, которая будет добавлять нового пользователя с безопасным хранением пароля:

```
sql
USE SecurityLab;
GO

CREATE PROCEDURE Users.RegisterUser
    @Username NVARCHAR(50),
    @Email NVARCHAR(100),
    @PlainPassword NVARCHAR(100)
AS
BEGIN
    SET NOCOUNT ON;

    -- Генерируем случайную соль (16 байт)
    DECLARE @Salt VARBINARY(16) = CRYPT_GEN_RANDOM(16);

    -- Конкатенируем пароль и соль (преобразуем соль в NVARCHAR для конкатенации)
    DECLARE @PasswordWithSalt NVARCHAR(200) = @PlainPassword +
    CONVERT(NVARCHAR(32), @Salt, 2);

    -- Вычисляем хеш (используем SHA2_256 для 32-байтного результата)
    DECLARE @Hash VARBINARY(32) = HASHBYTES('SHA2_256', @PasswordWithSalt);

    -- Вставляем запись
    INSERT INTO Users.Accounts (Username, Email, PasswordHash, PasswordSalt)
    VALUES (@Username, @Email, @Hash, @Salt);

    SELECT SCOPE_IDENTITY() AS UserID;
END;
GO
```

Пояснение:

- CRYPT_GEN_RANDOM(16) — генерирует криптостойкие случайные 16 байт
- Мы добавляем соль к паролю **перед** хешированием
- Храним и хеш, и соль

9.2. Создание процедуры проверки пароля

```
sql
USE SecurityLab;
GO
```

```

CREATE PROCEDURE Users.ValidatePassword
    @Username NVARCHAR(50),
    @PlainPassword NVARCHAR(100)
AS
BEGIN
    SET NOCOUNT ON;

    -- Получаем соль и хеш для пользователя
    DECLARE @Salt VARBINARY(16), @StoredHash VARBINARY(32);

    SELECT @Salt = PasswordSalt, @StoredHash = PasswordHash
    FROM Users.Accounts
    WHERE Username = @Username;

    -- Если пользователь не найден
    IF @Salt IS NULL
    BEGIN
        SELECT 'Пользователь не найден' AS Result;
        RETURN;
    END

    -- Вычисляем хеш для введённого пароля с той же солью
    DECLARE @PasswordWithSalt NVARCHAR(200) = @PlainPassword +
    CONVERT(NVARCHAR(32), @Salt, 2);
    DECLARE @ComputedHash VARBINARY(32) = HASHBYTES('SHA2_256',
    @PasswordWithSalt);

    -- Сравниваем хеши
    IF @ComputedHash = @StoredHash
    BEGIN
        -- Обновляем время последнего входа
        UPDATE Users.Accounts SET LastLoginDate = GETDATE() WHERE Username =
    @Username;
        SELECT 'Успешный вход' AS Result;
    END
    ELSE
    BEGIN
        SELECT 'Неверный пароль' AS Result;
    END
END;
GO

```

9.3. Тестирование процедур

sql

USE SecurityLab;

GO

-- Регистрируем трёх пользователей

EXEC Users.RegisterUser 'alice', 'alice@company.com', 'AlicePass123';

EXEC Users.RegisterUser 'bob', 'bob@company.com', 'BobPass456';

EXEC Users.RegisterUser 'charlie', 'charlie@company.com', 'Charlie789';

GO

-- Посмотрим, как выглядят данные (обратите внимание: даже у одинаковых паролей хеши разные из-за соли)

```
SELECT Username, PasswordHash, PasswordSalt, CreatedDate FROM Users.Accounts;  
GO
```

-- Проверяем правильный пароль

```
EXEC Users.ValidatePassword 'alice', 'AlicePass123'; -- Должно быть "Успешный вход"
```

-- Проверяем неправильный пароль

```
EXEC Users.ValidatePassword 'alice', 'WrongPass'; -- Должно быть "Неверный пароль"
```

-- Проверяем несуществующего пользователя

```
EXEC Users.ValidatePassword 'unknown', 'pass'; -- Должно быть "Пользователь не найден"
```

```
GO
```

Обратите внимание, что даже если бы у двух пользователей был одинаковый пароль (например, bob и charlie мы специально задали похожие), их хеши будут различаться.

Задание 10. Документирование созданной структуры

Составьте в отчёте таблицы, описывающие созданную вами структуру.

Таблица 1. Роли и их иерархия

Роль	Входит в роль	Права
Employees	—	SELECT на схему Users
Developers	Employees	INSERT, UPDATE, DELETE на схему Users
SeniorDevelopers	Developers	CREATE TABLE (на БД), ALTER ON SCHEMA::Users
Admins	SeniorDevelopers	db_owner (все права)
Managers	Employees	SELECT на схему Users, DENY на изменение

Таблица 2. Пользователи и их членство в ролях

Имя входа (Login)	Пользователь в БД	Роли
JuniorDev	JuniorDev	Developers
SeniorDev	SeniorDev	SeniorDevelopers
ProjectManager	ProjectManager	Managers
DBA	DBA	Admins

Таблица 3. Тестирование прав доступа

Действие	JuniorDev	SeniorDev	ProjectManager	DBA
SELECT из Users.EmployeeInfo	✓	✓	✓	✓
UPDATE данных	✓	✓	X (DENY)	✓
CREATE TABLE в схеме Users	X	✓	X	✓
ALTER TABLE (добавление колонки)	X	✓	X	✓
DROP TABLE	X	X	X	✓

Контрольные вопросы

1. В чём преимущество иерархической структуры ролей перед назначением прав каждому пользователю индивидуально?
2. Какие типы ролей существуют в SQL Server? Чем отличаются роли уровня сервера от ролей уровня базы данных?

3. Почему нельзя хранить пароли в открытом виде? Какие атаки становятся возможными при таком хранении?
4. Что такое «соль» (salt) и зачем она нужна при хешировании паролей? Что произойдёт, если два пользователя выберут одинаковые пароли?
5. Какие алгоритмы хеширования рекомендуются для хранения паролей, а какие считаются устаревшими? Почему?
6. Как проверить, является ли пользователь членом определённой роли (например, через системные представления)?

Дополнительное задание (на 2 часа, для отличников)

Реализуйте систему аудита доступа: создайте таблицу Security.AccessLog и триггеры, которые будут записывать все неудачные попытки входа (вызов процедуры ValidatePassword с неверным паролем) в эту таблицу. Также добавьте блокировку учётной записи после 5 неудачных попыток (поле LockedUntil в таблице Accounts).

Критерии оценки

- Задания 1-4 – базовая часть (оценка «удовлетворительно»)
- Задания 5-7 – средний уровень (оценка «хорошо»)
- Задания 8-9 – высокий уровень (оценка «отлично»)
- Задание 10 (документирование) – обязательно для любой оценки
- Дополнительное задание – повышенная оценка

Возможные ошибки и их устранение

Ошибка	Вероятная причина	Решение
"Login failed for user"	Неверный пароль или логин не существует	Проверьте имя логина и пароль
"The SELECT permission was denied"	Нет прав на чтение	Проверьте членство в ролях, добавьте пользователя в нужную роль
"Cannot add role as member"	Попытка создать циклическую зависимость	Проверьте иерархию, нельзя добавить роль саму в себя
Не совпадают хеши при проверке пароля	Разная кодировка при конкатенации	Убедитесь, что соль преобразуется в строку одинаково при регистрации и проверке

Практическое занятие №11. Настройка аутентификации клиентского приложения. Применять предопределенные роли.

Цель работы: Научиться подключаться к экземпляру SQL Server, используя различные режимы аутентификации (Windows и SQL Server), создавать логины и пользователей, а также управлять их доступом к объектам базы данных с помощью предопределенных ролей.

Теоретическая часть

1. Аутентификация в SQL Server

Прежде чем пользователь сможет работать с данными, он должен доказать серверу, что он тот, за кого себя выдает. Этот процесс называется **аутентификацией**. SQL Server поддерживает два основных режима:

- **Режим проверки подлинности Windows (Windows Authentication):**
 - Пользователь подключается к SQL Server, используя свою учетную запись Windows (доменную или локальную).
 - SQL Server доверяет операционной системе и не запрашивает пароль повторно.
 - Это наиболее безопасный режим, так как пароли не передаются по сети, и применяются политики сложности паролей Windows.
 - **Рекомендация:** Использовать везде, где это возможно, особенно в корпоративной среде.
- **Режим проверки подлинности SQL Server (SQL Server Authentication):**
 - SQL Server сам управляет учетными записями (логинами) и паролями.
 - При подключении пользователь обязательно вводит логин и пароль.
 - Пароли передаются по сети (хотя и в зашифрованном виде), что делает их менее безопасными.
 - **Используется:** Когда нужно подключиться к SQL Server не из домена Windows (например, из внешнего приложения) или при работе с устаревшими приложениями.

2. Иерархия безопасности: Логины и Пользователи

Чтобы понять, как работают права доступа, нужно запомнить два ключевых понятия:

1. **Логин (Login):** Создается на уровне **сервера**. Это "пропуск", который позволяет подключиться к самому SQL Server.
2. **Пользователь (User):** Создается внутри конкретной **базы данных**. Он "привязывается" к определенному логину и определяет, что именно этот логин может *делать* внутри данной базы данных.

3. Предопределенные роли базы данных

Роли — это "группы прав". Вместо того чтобы выдавать права каждому пользователю по отдельности, мы выдаем их роли, а затем добавляем пользователя в эту роль. SQL Server предоставляет набор **предопределенных (фиксированных) ролей** на уровне базы данных. Их разрешения нельзя изменить, они созданы для самых типичных задач.

Имя роли	Описание	Типовые разрешения
db_owner	Владелец БД. Может делать всё: настраивать, обслуживать, удалять БД, управлять пользователями.	CONTROL DATABASE (полный контроль).
db_datareader	Читатель данных. Может читать (SELECT) данные из любых пользовательских таблиц.	GRANT SELECT на все таблицы.
db_datawriter	Писатель данных. Может добавлять (INSERT), изменять (UPDATE) и удалять (DELETE) данные в любых пользовательских таблицах.	GRANT INSERT, UPDATE, DELETE на все таблицы.
db_ddladmin	Администратор DDL. Может выполнять команды изменения структуры БД (CREATE, ALTER, DROP) для любых объектов.	ALTER, CREATE TABLE, CREATE PROCEDURE и др.

db_securityadmin	Администратор безопасности. Может управлять членством в пользовательских ролях и разрешениями.	ALTER ANY ROLE, CREATE SCHEMA .
db_accessadmin	Администратор доступа. Может добавлять или удалять пользователей в базе данных.	ALTER ANY USER, CREATE USER .
db_backupoperator	Оператор резервного копирования. Может создавать резервные копии базы данных.	BACKUP DATABASE, BACKUP LOG .
db_denydatareader	Запрет на чтение. Запрещает чтение данных из любых пользовательских таблиц.	DENY SELECT на все таблицы .
public	Есть у каждого пользователя БД. По умолчанию не имеет прав, но обладает разрешением CONNECT.	Минимальные права по умолчанию .

Ход работы

Задание 1. Подготовка тестовой среды и проверка режимов аутентификации

1. Запустите **SQL Server Management Studio**.
2. В диалоговом окне "**Соединение с сервером**" (Connect to Server) изучите выпадающий список "**Аутентификация**" (Authentication) :
 - о *Примечание для преподавателя:* Обратите внимание студентов на наличие пунктов Проверка подлинности Windows и Проверка подлинности SQL Server. Если версия SQL Server 2022 и выше, там также может быть Проверка подлинности Microsoft Entra .
3. Подключитесь к серверу, используя **Проверка подлинности Windows**.
4. После подключения, в **Обозревателе объектов (Object Explorer)** найдите и запишите, под каким именно именем пользователя Windows вы подключились (оно будет отображаться после имени сервера) .

Задание 2. Создание тестовой базы данных

Мы создадим базу данных UniversityDB, с которой будем работать на протяжении всей практики.

1. Нажмите кнопку "**Создать запрос**" (New Query) на панели инструментов.
2. Скопируйте и выполните следующий скрипт.

```
sql
```

```
-- Создаем базу данных
```

```
CREATE DATABASE UniversityDB;
```

```
GO
```

```
USE UniversityDB;
```

```
GO
```

```
-- Создаем таблицу Студенты
```

```
CREATE TABLE Студенты (
    ID INT PRIMARY KEY IDENTITY(1,1),
    Фамилия NVARCHAR(50) NOT NULL,
    Имя NVARCHAR(50) NOT NULL,
    Группа NVARCHAR(20),
    СреднийБалл DECIMAL(3,2)
```

```
);
```

```
GO
```

```
-- Наполняем таблицу тестовыми данными
```

```
INSERT INTO Студенты (Фамилия, Имя, Группа, СреднийБалл) VALUES
('Иванов', 'Иван', '3-ИС-1', 4.5),
('Петрова', 'Мария', '3-ИС-1', 4.8),
('Сидоров', 'Петр', '3-ИС-2', 3.9),
('Васечкин', 'Джон', '3-ИС-2', 4.1);
GO
```

```
PRINT 'База данных UniversityDB и таблица Студенты созданы.';
GO
```

3. В **Обозревателе объектов** обновите список баз данных, чтобы увидеть UniversityDB.

Задание 3. Создание логина и пользователя с доступом только для чтения

Создадим учетную запись для "Учебного отдела", которому нужно только просматривать список студентов, но нельзя ничего менять.

Шаг 1. Создание логина на сервере (SQL Server Authentication)

1. В **Обозревателе объектов** раскройте папку сервера → **Безопасность (Security)**.
2. Правой кнопкой мыши на папке **Имена входа (Logins)** → выберите **Создать имя входа... (New Login...)**.
3. В открывшемся окне:
 - **Имя входа (Login name):** UchebnyOtdel_Login
 - **Проверка подлинности (Authentication):** выберите **Проверка подлинности SQL Server**.
 - **Пароль (Password):** введите UchOtdelPass123!.
 - **Подтверждение пароля (Confirm Password):** UchOtdelPass123!.
 - **Снимите галочку** "Требовать смену пароля при следующем входе" (для учебных целей).
4. Нажмите **ОК**.

Шаг 2. Создание пользователя в базе данных

Теперь нужно "связать" созданный логин с базой UniversityDB, создав для него пользователя .

1. В **Обозревателе объектов** перейдите в **Базы данных (Databases)** → **UniversityDB** → **Безопасность** → **Пользователи**.
2. Правой кнопкой мыши на **Пользователи** → **Создать пользователя... (New User...)**.
3. В открывшемся окне:
 - **Тип пользователя (User type):** Пользователь SQL с именем входа.
 - **Имя пользователя (User name):** UchebnyOtdel_User (можно назвать иначе, но для простоты лучше логин).
 - **Имя входа (Login name):** нажмите кнопку с многоточием (...), найдите и выберите UchebnyOtdel_Login.
 - **Схема по умолчанию (Default schema):** оставьте dbo.
4. Нажмите **ОК**.

Шаг 3. Назначение прав через предопределенную роль db_datareader

Наш пользователь пока есть, но у него нет прав даже на чтение. Дадим ему их с помощью роли .

1. Откройте **свойства** пользователя UchebnyOtdel_User (двойным кликом).
2. Перейдите на страницу **Членство (Membership)**.
3. В списке "Членство в роли базы данных" **поставьте галочку** напротив роли db_datareader.
4. Нажмите **ОК**.

Задание 4. Создание логина и пользователя с полными правами

Создадим учетную запись для администратора деканата ("Сотрудник деканата"), который может не только смотреть, но и изменять данные (добавлять новых студентов, исправлять оценки).

Шаг 1. Создание логина (через T-SQL)

Вместо графического интерфейса, выполните этот код в новом окне запроса, чтобы увидеть альтернативный способ.

```
sql
USE master;
GO
CREATE LOGIN [Deanery_Login] WITH PASSWORD = 'DeanPass123!';
GO
PRINT 'Логин Deanery_Login создан';
GO
```

Шаг 2. Создание пользователя и назначение ролей (через T-SQL)

Выполните следующий код, убедившись, что вы находитесь в контексте базы UniversityDB.

```
sql
USE UniversityDB;
GO
```

```
-- Создаем пользователя для логина Deanery_Login
CREATE USER Deanery_User FOR LOGIN Deanery_Login;
GO
```

```
-- Добавляем пользователя в роль db_datareader (чтобы читать)
ALTER ROLE db_datareader ADD MEMBER Deanery_User;
-- Добавляем пользователя в роль db_datawriter (чтобы изменять)
ALTER ROLE db_datawriter ADD MEMBER Deanery_User;
GO
```

```
PRINT 'Пользователь Deanery_User создан и добавлен в роли чтения и записи.';
GO
```

Теперь пользователь Deanery_User может не только видеть таблицу Студенты, но и добавлять, изменять и удалять в ней записи .

Задание 5. Создание логина администратора сервера (sysadmin)

Создадим логин для главного администратора. Ему нужны не просто права на одну базу, а полный контроль над всем экземпляром SQL Server. Для этого мы добавим его в предопределенную **серверную роль** sysadmin (роль уровня сервера, а не базы данных).

1. Создайте логин Admin_Login с паролем любым удобным способом.
2. В **Обозревателе объектов** раскройте папку сервера **Безопасность** → **Роли сервера (Server Roles)**.
3. Найдите роль sysadmin, щелкните по ней правой кнопкой мыши и выберите **Свойства (Properties)**.
4. В открывшемся окне нажмите **Добавить (Add)**, найдите и добавьте логин Admin_Login .
5. Нажмите **ОК**.

Задание 6. Проверка прав доступа (самое важное!)

Теперь мы проверим, как работают созданные нами пользователи. Для этого нужно создать несколько новых подключений в SSMS .

1. В главном меню SSMS нажмите **Файл (File)** → **Создать (New)** → **Соединение с ядром СУБД (Database Engine Query)**.

Проверка 1: Пользователь "Учебный отдел" (только чтение)

- **Имя входа:** UchebnyOtdel_Login

- **Пароль:** UchOtdelPass123!
- **Аутентификация:** Проверка подлинности SQL Server

Подключитесь и выполните в новом окне запроса:

```
sql
USE UniversityDB;
GO
-- Попробуем прочитать данные
SELECT * FROM Студенты;      -- (1) Этот запрос должен выполняться УСПЕШНО
GO
-- Попробуем изменить данные
UPDATE Студенты SET Имя = 'Иванчик' WHERE ID = 1; -- (2) Этот запрос должен вернуть
```

ОШИБКУ

```
GO
```

Результат:

1. Вы увидите таблицу со студентами — **чтение разрешено.**
2. Вы получите ошибку: The UPDATE permission was denied on the object

'Студенты'... — **изменение запрещено**, что и требовалось.

Проверка 2: Пользователь "Сотрудник деканата" (чтение и запись)

Создайте новое подключение для логина Deanery_Login (пароль DeanPass123!).

Выполните:

```
sql
USE UniversityDB;
GO
-- Попробуем прочитать данные
SELECT * FROM Студенты;      -- (1) УСПЕШНО (роль db_datareader)
GO
-- Попробуем изменить данные
UPDATE Студенты SET Имя = 'Мария' WHERE ID = 2; -- (2) УСПЕШНО (роль
```

db_datawriter)

```
GO
```

```
-- Попробуем добавить новую запись
```

```
INSERT INTO Студенты (Фамилия, Имя, Группа, СреднийБалл) VALUES ('Новиков',
```

'Антон', '3-ИС-1', 4.6); -- (3) УСПЕШНО

```
GO
```

```
-- Посмотрим результат
```

```
SELECT * FROM Студенты;
```

```
GO
```

Результат: Все три операции должны выполняться без ошибок.

Проверка 3: Пользователь "Администратор" (полный контроль)

Создайте новое подключение для логина Admin_Login (ваш пароль).

Выполните:

```
sql
-- Администратор может выполнять команды не только в своей базе, но и на уровне сервера
USE master;
GO
-- Создадим новую тестовую базу данных (это требует прав)
CREATE DATABASE TestAdminDB;
GO
-- И сразу удалим её
DROP DATABASE TestAdminDB;
GO
USE UniversityDB;
```

GO

-- Администратор может делать с данными всё, что угодно

DELETE FROM Студенты WHERE ID = 5; -- Удалим добавленного ранее Новикова (если ID=5)

SELECT * FROM Студенты;

GO

Результат: Все команды выполняются успешно. Администратор (sysadmin) имеет неограниченные права.

Задание 7. Удаление пользователя из роли (обратная операция)

Предположим, сотрудник уволился, и нам нужно лишить его прав. Мы не будем удалять пользователя, а просто уберем его из роли.

Вернитесь в подключение от имени администратора (или sa) и выполните:

sql

USE UniversityDB;

GO

-- Удаляем пользователя Deanery_User из роли db_datawriter

ALTER ROLE db_datawriter DROP MEMBER Deanery_User;

GO

-- Теперь у него осталась только роль db_datareader (только чтение)

GO

Если вы проверите подключение от имени Deanery_Login снова, то обнаружите, что он все еще может читать, но попытка изменить данные вызовет ошибку.

Задание 8. Документирование

Составьте в отчете таблицы, описывающие созданную вами структуру.

Таблица 1. Учетные записи (логины)

Имя логина	Тип аутентификации	Уровень (Сервер/БД)	Предопределенная роль	Назначение
UchebnyOtdel_Login	SQL Server	База данных University DB	db_datareader	Только просмотр студентов
Deanery_Login	SQL Server	База данных University DB	db_datareader, db_datawriter	Просмотр и изменение данных о студентах
Admin_Login	SQL Server	Сервер	sysadmin	Полный административный доступ

Таблица 2. Результаты проверки доступа

Действие	UchebnyOtdel_Login	Deanery_Login	Admin_Login
SELECT из Студенты	✓	✓	✓
UPDATE данных в Студенты	✗ (ошибка)	✓	✓
INSERT в Студенты	✗ (ошибка)	✓	✓
CREATE DATABASE	✗ (ошибка)	✗ (ошибка)	✓

Контрольные вопросы

1. В чем основная разница между режимами аутентификации Windows и SQL Server? В каких случаях предпочтительнее использовать каждый из них?
2. Объясните разницу между логином (login) и пользователем (user) базы данных. Может ли существовать логин без пользователя, и наоборот?

3. Перечислите не менее трех predefined ролей базы данных и опишите, для каких задач они предназначены.
4. Что произойдет с доступом пользователя Deanery_User, если мы удалим его из роли db_datawriter, но оставим в роли db_datareader? Проверьте и опишите результат.
5. В чем ключевое отличие роли сервера (например, sysadmin) от роли базы данных (например, db_owner)?

Практическое занятие №12. Выполнение резервного копирования и восстановления. Настройка автоматического резервного копирования.

Цель работы: Научиться создавать резервные копии базы данных различными способами, выполнять восстановление базы данных из резервной копии, а также настраивать автоматическое выполнение резервного копирования по расписанию.

Теоретическая часть

Резервное копирование (Backup) — это процесс создания копии базы данных для последующего восстановления в случае сбоя, повреждения или потери данных .

Основные типы резервных копий в SQL Server:

- **Полное резервное копирование (Full Backup)** — содержит все данные базы данных на момент создания копии. Является основой для восстановления .
- **Разностное (дифференциальное) резервное копирование (Differential Backup)** — содержит только изменения, произошедшие с момента последнего полного резервного копирования .
- **Резервное копирование журнала транзакций (Transaction Log Backup)** — содержит записи обо всех изменениях в базе данных и позволяет восстановить состояние на конкретный момент времени .

Автоматизация резервного копирования достигается путем создания задания (Job) в **SQL Server Agent** — компоненте SQL Server, который позволяет запускать задачи по расписанию .

Ход работы

Задание 1. Подготовка тестовой базы данных

Для выполнения работы нам потребуется база данных, с которой мы будем работать.

1. Запустите **SQL Server Management Studio** и подключитесь к вашему экземпляру SQL Server (например, с использованием проверки подлинности Windows).
2. Нажмите **Новый запрос (New Query)**.
3. Скопируйте и выполните следующий скрипт для создания базы данных Практика_РезервноеКопирование и таблицы в ней.

```
sql
```

```
-- Создаем базу данных
```

```
CREATE DATABASE Практика_РезервноеКопирование;
```

```
GO
```

```
USE Практика_РезервноеКопирование;
```

```
GO
```

```
-- Создаем таблицу Сотрудники
```

```
CREATE TABLE Сотрудники (
```

```
ID INT PRIMARY KEY IDENTITY(1,1),
Фамилия NVARCHAR(50) NOT NULL,
Имя NVARCHAR(50) NOT NULL,
Отдел NVARCHAR(50),
ДатаПриема DATE
);
GO
```

```
-- Наполняем таблицу тестовыми данными
INSERT INTO Сотрудники (Фамилия, Имя, Отдел, ДатаПриема) VALUES
('Иванов', 'Иван', 'IT', '2023-09-01'),
('Петрова', 'Мария', 'Бухгалтерия', '2022-05-15'),
('Сидоров', 'Петр', 'IT', '2024-01-10');
GO
```

```
-- Проверим, что данные добавлены
SELECT * FROM Сотрудники;
GO
```

4. В **Обозревателе объектов (Object Explorer)** обновите список баз данных, чтобы убедиться, что база Практика_РезервноеКопирование создана .

Задание 2. Создание полной резервной копии базы данных (графический способ)

1. В **Обозревателе объектов** раскройте папку **Базы данных (Databases)**.
2. Щелкните **правой кнопкой мыши** на базе данных Практика_РезервноеКопирование → наведите курсор на **Задачи (Tasks)** → выберите **Создать резервную копию... (Back Up...)** .

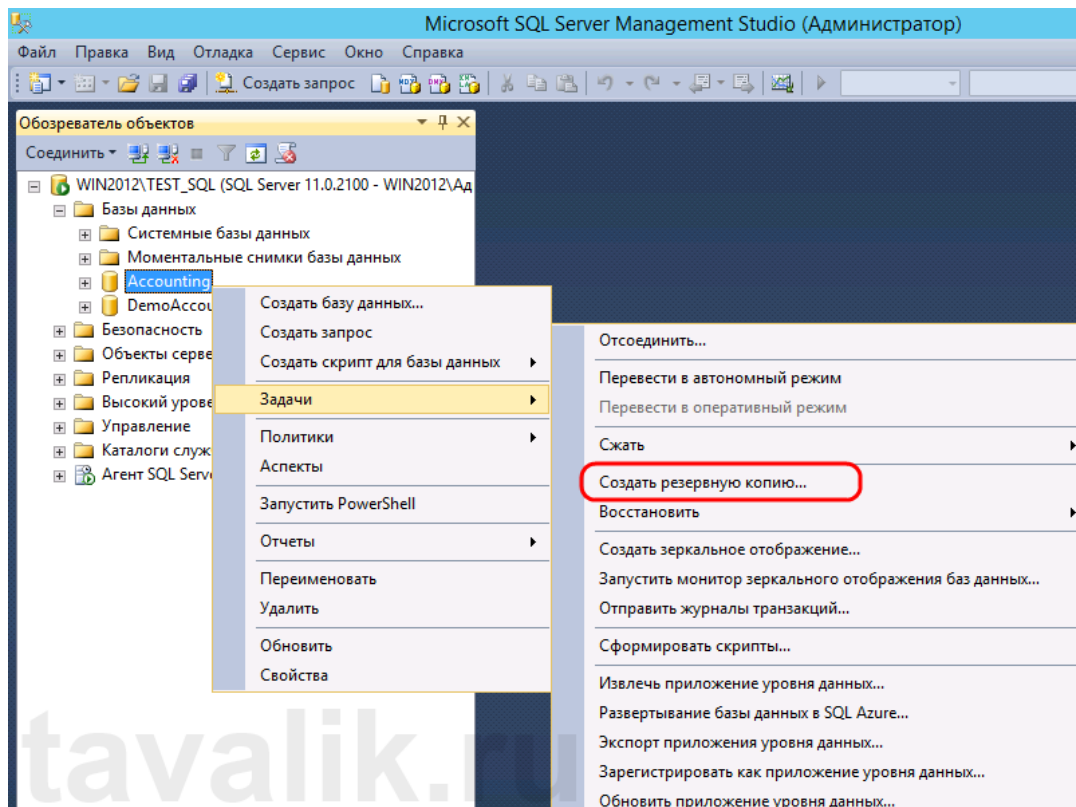


Рисунок 1: Контекстное меню базы данных -> Задачи -> Создать резервную копию

3. Откроется диалоговое окно "**Резервное копирование базы данных -**

Практика_РезервноеКопирование".

- Убедитесь, что в поле **База данных** выбрана Практика_РезервноеКопирование.
- В поле **Тип резервной копии (Backup type)** должно стоять значение "**Полная**" (**Full**) .
- В разделе **Компонент для резервного копирования (Backup component)** оставьте переключатель в положении **База данных (Database)** .

4. Обратите внимание на раздел **Назначение (Destination)**.

- Здесь указан путь к файлу резервной копии (по умолчанию — в папке Backup вашего экземпляра SQL Server). Файл будет иметь расширение .bak .
- Если вы хотите изменить путь или имя файла, можно нажать **Удалить (Remove)** для существующего пути, затем **Добавить (Add)** и указать свой путь. Для учебных целей можно оставить путь по умолчанию, но **запомните его** (или скопируйте).

5. Нажмите кнопку **ОК** для запуска процесса резервного копирования.

6. После успешного завершения должно появиться окно с сообщением: "**Резервное копирование базы данных "Практика_РезервноеКопирование" успешно завершено**" . Нажмите **ОК**.

Поздравляю! Вы создали свою первую резервную копию.

Задание 3. Внесение изменений в базу данных

Чтобы увидеть, как работает восстановление, внесем изменения в данные.

1. Выполните следующий скрипт, который удаляет одного сотрудника и добавляет нового.

sql

USE Практика_РезервноеКопирование;

GO

-- Удаляем сотрудника с ID=2 (Петрова Мария)

DELETE FROM Сотрудники WHERE ID = 2;

GO

-- Добавляем нового сотрудника

INSERT INTO Сотрудники (Фамилия, Имя, Отдел, ДатаПриема) VALUES ('Козлов', 'Николай',
'Отдел кадров', '2024-02-20');

GO

-- Посмотрим, что теперь в таблице

SELECT * FROM Сотрудники;

GO

2. Запишите или запомните, какие изменения произошли (например, пропала Петрова и появился Козлов). Текущее состояние базы данных отличается от того, которое было сохранено в резервной копии.

Задание 4. Восстановление базы данных из резервной копии

Важно! Для восстановления базы данных необходимо, чтобы никто другой не использовал её в этот момент. Мы переведем базу в режим восстановления.

1. В **Обозревателе объектов** щелкните **правой кнопкой мыши** на узел **Базы данных (Databases)** и выберите **Восстановить базу данных... (Restore Database...)**.

(Рисунок 3: Контекстное меню -> Восстановить базу данных)

2. В открывшемся окне **"Восстановление базы данных - Практика_РезервноеКопирование"**:

- В поле **База данных (Database)** выберите из выпадающего списка **Практика_РезервноеКопирование**.
- В разделе **Источник для восстановления (Source for restore)** выберите переключатель **Устройство (Device)**.
 - Нажмите на кнопку с многоточием (...) справа от поля **Устройство**.
- 3. Откроется диалоговое окно **"Выбор устройств резервного копирования"**.
 - В выпадающем списке **"Носитель резервной копии"** выберите **"Файл"**.
 - Нажмите кнопку **Добавить (Add)**.
 - Найдите и выберите файл резервной копии (с расширением .bak), который вы создавали

в **Задании 2**.

- Нажмите **ОК**, чтобы закрыть окно выбора файла, и еще раз **ОК**, чтобы закрыть окно **"Выбор устройств резервного копирования"**.

Вы вернетесь в окно **"Восстановление базы данных"**. Убедитесь, что в списке **"Наборы резервных копий для восстановления"** отображается ваша резервная копия (с типом **"Полная"**).

4. В левой части окна перейдите на страницу **Параметры (Options)**.

5. В разделе **Параметры восстановления (Restore options)** обратите внимание на пункт **"Перезапись существующей базы данных (WITH REPLACE)"** . Это позволит нам восстановить базу поверх существующей. Для учебных целей его можно не включать, если вы не меняли имя базы.

○ Также важно проверить параметр **"Состояние восстановления"**.

Выберите **"RECOVERY"** (оставляет базу данных в рабочем состоянии) .

6. Нажмите кнопку **ОК** для начала процесса восстановления.

7. После успешного завершения вы увидите сообщение: **"Восстановление базы данных "Практика_РезервноеКопирование" успешно завершено"**.

8. **Проверка:** Теперь снова выполните запрос **SELECT * FROM Сотрудники;**. Вы должны увидеть данные **на момент создания резервной копии** (т.е. Иванова, Петрову, Сидорова). Изменения, внесенные в Задании 3, пропали — база данных успешно откатилась к состоянию, сохраненному в резервной копии .

Задание 5. Настройка автоматического резервного копирования с помощью SQL Server Agent

Теперь настроим задание, которое будет автоматически создавать полную резервную копию нашей базы данных, например, каждый понедельник в 22:00.

Важное условие: Для выполнения этого задания на вашем компьютере должна быть запущена служба **SQL Server Agent (MSSQLSERVER)**. Обычно она запускается автоматически, но если нет — можно проверить это в диспетчере задач (вкладка Службы) или в оснастке "Службы" Windows .

Шаг 1: Создаем резервную копию и сохраняем ее как задание

1. Повторите шаги 1-3 из **Задания 2**, чтобы дойти до заполненного окна "Резервное копирование базы данных".

2. **Вместо нажатия кнопки ОК**, найдите на панели инструментов в верхней части окна кнопку **"Скрипт" (Script)** .

3. В выпадающем меню выберите **"Действие скрипта в задание..." (Script Action to Job...)** .

Шаг 2: Настройка нового задания

Откроется диалоговое окно **"Создание задания" (New Job)** .

1. На странице **"Общие" (General)**:

○ В поле **"Имя" (Name)** введите: Резервное копирование - Практика_РезервноеКопирование.

○ В поле **"Владелец" (Owner)** можно оставить sa или вашу учетную запись.

○ В поле **"Категория" (Category)** можно выбрать [Незадокументировано (локально)] или Database Maintenance.

○ В поле **"Описание" (Description)** можно написать: Автоматическая полная резервная копия базы данных каждую неделю.

(Рисунок 6: Заполнение страницы "Общие" нового задания)

2. Перейдите на страницу **"Шаги" (Steps)** .

○ Вы увидите, что один шаг (Step 1) уже создан автоматически. Это и есть команда резервного

копирования, которую мы настроили.

- Вы можете нажать кнопку **"Изменить..." (Edit...)**, чтобы посмотреть, какая именно T-SQL команда будет выполняться. Там будет что-то вроде:

```
sql
```

```
BACKUP DATABASE [Практика_РезервноеКопирование] TO DISK = N'...путь...' WITH  
NOFORMAT, NOINIT, NAME = N'...', SKIP, NOREWIND, NOUNLOAD, STATS = 10
```

Обратите внимание, что это полный аналог действий, выполненных вами в графическом интерфейсе .

- Если все в порядке, закройте окно свойств шага, нажав **"Отмена"** или **"ОК"** (если ничего не меняли).

3. Перейдите на страницу **"Расписания" (Schedules)** .

- Нажмите кнопку **"Создать..." (New...)**.

- В окне **"Создание расписания задания" (New Job Schedule)**:

- **Имя (Name):** Еженедельное - ПН 22:00.
- В разделе **"Тип расписания" (Schedule type)** оставьте Повторяющееся (Recurring).
- В разделе **"Частота" (Frequency)**:
 - **Выполняется (Occurs):** Еженедельно.
 - **Повторять каждую (Recurs every):** 1 неделя в.
 - Отметьте галочкой день недели: **понедельник**.
- В разделе **"Ежедневная частота" (Daily frequency)**:
 - **Выполняется однократно в (Occurs once at):** 22:00:00.
- В разделе **"Продолжительность" (Duration)**:
 - **Дата начала (Start date):** укажите сегодняшнюю дату.
 - **Дата окончания (End date):** можно оставить Без даты окончания (No end date).

- Нажмите **ОК**, чтобы сохранить расписание.

4. Нажмите **ОК** еще раз в окне "Создание задания", чтобы завершить его создание .

Шаг 3: Проверка задания

1. В **Обозревателе объектов** найдите папку **"Агент SQL Server" (SQL Server Agent)** и раскройте её .

2. Раскройте папку **"Задания" (Jobs)**. Вы должны увидеть там ваше только что созданное задание.

3. Чтобы проверить, правильно ли оно работает, можно запустить его вручную.

- Щелкните **правой кнопкой мыши** на задании Резервное копирование - Практика_РезервноеКопирование.

- Выберите **"Запустить задание на шаге..." (Start Job at Step...)**.

- Выберите шаг Step 1 и нажмите **"Запустить" (Start)**.

- Дождитесь сообщения об успешном выполнении.

Поздравляю! Вы настроили автоматическое резервное копирование. Теперь каждый понедельник в 22:00 SQL Server будет самостоятельно создавать резервную копию вашей базы данных.

Контрольные вопросы и задания

1. Какие типы резервных копий поддерживает SQL Server? Опишите разницу между полной и разностной копией .
2. Для чего нужна служба SQL Server Agent? Может ли она работать, если остановлен сам SQL Server? .
3. Что произойдет с данными, если во время восстановления базы данных не использовать опцию WITH REPLACE, а восстанавливаемая база данных уже существует?
4. В чем преимущество автоматического резервного копирования по расписанию перед ручным созданием копий? .
5. *Дополнительно:* Изучите папку, куда была сохранена ваша резервная копия (из Задания 2). Найдите файл .bak. Попробуйте изменить его свойства (например, сделать "только для чтения") и повторите процесс восстановления из Задания 4. Что изменилось? Удалось ли восстановить базу?

Практическое занятие №13. Настройка репликации. Конфигурация мастера и слейва. Синхронизация данных между узлами. Решение проблем с репликацией.

Цель работы: Научиться настраивать репликацию транзакций между двумя экземплярами SQL Server (издателем и подписчиком), выполнять синхронизацию данных и диагностировать типовые проблемы, возникающие в процессе репликации.

Теоретическая часть

Что такое репликация?

Репликация — это набор технологий для копирования и распространения данных и объектов баз данных из одной базы данных в другую с последующей синхронизацией для обеспечения согласованности.

В SQL Server репликация использует архитектуру "издатель-подписчик" (publisher-subscriber) :

- **Издатель (Publisher)** — источник данных, экземпляр базы данных, который делает данные доступными для репликации.
- **Распространитель (Distributor)** — экземпляр (может совпадать с издателем), который хранит историю репликации и транзакции.

- **Подписчик (Subscriber)** — целевой экземпляр, получающий данные .

Типы репликации

- **Репликация транзакций (Transactional replication)** — идеально подходит для сценариев "мастер-слейв". Издатель отправляет транзакции подписчику практически в реальном времени .
- **Репликация моментальными снимками (Snapshot replication)** — периодически отправляет полную копию данных .
- **Репликация слиянием (Merge replication)** — двусторонняя синхронизация с возможностью разрешения конфликтов .

Компоненты репликации

- **Публикация (Publication)** — набор статей (таблиц, процедур), которые реплицируются как единое целое .
- **Статья (Article)** — отдельный объект базы данных (таблица, представление), включённый в публикацию .
- **Подписка (Subscription)** — запрос на получение публикации подписчиком .

Проблемы репликации

Основные проблемы включают: остановку агентов, конфликты данных, рассинхронизацию, проблемы с задержкой . Для мониторинга используется **Replication Monitor** .

Подготовительный этап

Требования к окружению

Для выполнения работы необходимо:

1. **Два экземпляра SQL Server** (можно запустить два экземпляра на одном компьютере, например, MSSQLSERVER (основной) и SQLSERVER02 (дополнительный)). Если второй экземпляр отсутствует, можно использовать локальную базу данных как подписчик (подписка на тот же экземпляр, но другую базу).
2. **Запущенный SQL Server Agent** на обоих экземплярах (или хотя бы на экземпляре издателя).
3. **Учётная запись с правами sysadmin** (учётная запись Windows или SQL Server).
4. **Общая папка** для моментальных снимков (snapshot folder). В учебных целях можно использовать локальную папку с правами на запись для всех.

Важно! На компьютерах с ограниченными правами может не быть доступа к сетевым ресурсам. В этом случае мы будем использовать **локальную папку** и агенты, запущенные под учётной записью SQL Server Agent.

Создание тестовой базы данных на издателе

1. Запустите **SQL Server Management Studio** и подключитесь к **первому экземпляру** (издатель).

2. Выполните скрипт для создания базы данных ShopDB_Replica:

```
sql
```

```
-- Создаём базу данных на издателе
```

```
CREATE DATABASE ShopDB_Replica;
```

```
GO
```

```
USE ShopDB_Replica;
```

```
GO
```

```
-- Таблица товаров
```

```
CREATE TABLE Products (
```

```
    ProductID INT PRIMARY KEY IDENTITY(1,1),
```

```
    ProductName NVARCHAR(100) NOT NULL,
```

```
    Price DECIMAL(10,2),
```

```
    Quantity INT
```

```
);
```

```
GO
```

```
-- Таблица заказов
```

```
CREATE TABLE Orders (
```

```
    OrderID INT PRIMARY KEY IDENTITY(1,1),
```

```
    ProductID INT FOREIGN KEY REFERENCES Products(ProductID),
```

```
    OrderDate DATETIME DEFAULT GETDATE(),
```

```
    Quantity INT
```

```
);
```

```
GO
```

```
-- Наполняем тестовыми данными
```

```
INSERT INTO Products (ProductName, Price, Quantity) VALUES
```

```
('Ноутбук', 45000.00, 10),
```

```
('Мышь', 800.00, 50),
```

```
('Клавиатура', 1500.00, 30);
```

```
GO
```

```
INSERT INTO Orders (ProductID, Quantity) VALUES
```

```
(1, 1),
```

```
(2, 2),
```

```
(1, 1);
```

```
GO
```

```
PRINT 'База данных издателя создана';
```

```
GO
```

3. Создайте на **втором экземпляре** (подписчик) пустую базу данных ShopDB_Subscriber (без таблиц):

```
sql
```

```
-- Выполнить на подписчике
```

```
CREATE DATABASE ShopDB_Subscriber;
```

```
GO
```

Ход работы

Задание 1. Настройка распространителя (Distributor)

Распространитель будет настроен на том же экземпляре, что и издатель (локальный распространитель) .

1. В **Object Explorer** (подключение к издателю) раскройте папку **Репликация (Replication)**.
2. Щёлкните правой кнопкой мыши на папке **Репликация** и выберите **Настроить распространителя (Configure Distribution)**.
(Рисунок 1: Контекстное меню репликации -> Настроить распространителя)
3. Запустится **Мастер настройки распространителя**.
 - Нажмите **Далее**.
 - На странице **Распространитель** выберите "Имя_сервера" будет действовать как собственный распространитель... (первый вариант) .
 - Нажмите **Далее**.
 - На странице **Папка моментальных снимков** укажите локальную папку, например, C:\ReplicationSnapshot. Если папка не существует, мастер предложит её создать .
 - **Важно:** Убедитесь, что у службы SQL Server Agent есть права на запись в эту папку.
 - На странице **База данных распространителя** оставьте имя distribution и пути к файлам по умолчанию .
 - На странице **Издатели** убедитесь, что ваш сервер отмечен галочкой.
 - Нажмите **Далее**, затем **Готово**.
4. После завершения настройки в папке **Репликация** появятся подпапки **Локальные публикации** и **Локальные подписки**.

Задание 2. Создание публикации (Publication)

1. В папке **Репликация** щёлкните правой кнопкой мыши на **Локальные публикации** - > **Создать публикацию...** .
2. В мастере создания публикации:
 - Нажмите **Далее**.
 - Выберите базу данных ShopDB_Replica и нажмите **Далее**.
 - Выберите тип публикации: **Публикация транзакций (Transactional publication)** .
 - Нажмите **Далее**.
3. На странице **Статьи (Articles)**:
 - Разверните папку **Таблицы**.
 - Отметьте галочками таблицы Products и Orders.
 - Нажмите **Далее** .
4. На странице **Фильтрация строк таблицы** оставьте всё как есть (без фильтров) и нажмите **Далее**.
5. На странице **Агент моментальных снимков**:
 - Отметьте галочкой **Создать моментальный снимок немедленно** .
 - Нажмите **Далее**.
6. На странице **Безопасность агентов**:
 - Нажмите кнопку **Параметры безопасности...** для агента моментальных снимков.
 - В открывшемся окне выберите **Запуск от имени учетной записи агента SQL Server** .
 - Убедитесь, что в разделе подключения выбрано **Олицетворять учетную запись процесса**.
 - Нажмите **ОК**.
 - Повторите те же действия для агента чтения журнала.
 - Нажмите **Далее**.
7. На странице **Действия мастера** оставьте галочку **Создать публикацию** и нажмите **Далее**.

8. Введите имя публикации: ShopDB_TransactionPublication .
9. Нажмите **Готово** и дождитесь завершения создания публикации.

Задание 3. Создание подписки (Subscription)

1. В папке **Локальные публикации** щёлкните правой кнопкой мыши на созданной публикации ShopDB_TransactionPublication и выберите **Создать подписку...** .
2. В мастере создания подписки:
 - Нажмите **Далее**.
 - Убедитесь, что выбрана правильная публикация.
 - Нажмите **Далее**.
 - На странице **Расположение агента распространения** выберите **Выполнять все агенты в распространителе (push-подписка)** .
 - Нажмите **Далее**.
3. На странице **Подписчики**:
 - Нажмите кнопку **Добавить подписчик** -> **Добавить подписчик SQL Server** .
 - В диалоговом окне подключитесь ко **второму экземпляру (подписчику)**, где создана база ShopDB_Subscriber.
 - В списке подписчиков появится ваш второй сервер.
 - В строке этого сервера выберите из выпадающего списка **База данных** значение ShopDB_Subscriber.
 - Нажмите **Далее**.
4. На странице **Безопасность агента распространения**:
 - Нажмите кнопку с многоточием (...) в строке подписки.
 - В открывшемся окне выберите **Запуск от имени учетной записи агента SQL Server** .
 - Выберите способ подключения: **Олицетворять учетную запись процесса** для издателя и распространителя.
 - Для подключения к подписчику также выберите **Олицетворять учетную запись процесса**.
 - Нажмите **ОК**.
 - Нажмите **Далее**.
5. На странице **Расписание синхронизации**:
 - Выберите **Выполнять непрерывно** (для минимальной задержки) .
 - Нажмите **Далее**.
6. На странице **Инициализация подписки**:
 - Убедитесь, что отмечена галочка **Инициализировать**.
 - Выберите **Немедленно**.
 - Нажмите **Далее**.
7. На странице **Действия мастера** оставьте галочку **Создать подписку(и)** и нажмите **Далее**.
8. Нажмите **Готово** и дождитесь завершения.

Задание 4. Проверка синхронизации данных

1. Подключитесь к **подписчику** (второй экземпляр) и выполните запрос:

```
sql
USE ShopDB_Subscriber;
GO
SELECT * FROM Products;
SELECT * FROM Orders;
GO
```

Вы должны увидеть те же данные, что и в базе издателя. Это означает, что моментальный снимок был успешно применён.

2. Теперь проверьте репликацию транзакций в реальном времени.

На **издате**ле выполните:

```
sql
USE ShopDB_Replica;
GO
-- Добавим новый продукт
INSERT INTO Products (ProductName, Price, Quantity) VALUES ('Монитор', 12000.00, 15);
GO
-- Создадим заказ
INSERT INTO Orders (ProductID, Quantity) VALUES (4, 2);
GO
```

3. Подождите несколько секунд и выполните на **подписчике**:

```
sql
USE ShopDB_Subscriber;
GO
SELECT * FROM Products;
SELECT * FROM Orders;
GO
```

Новые данные должны появиться на подписчике. Репликация работает!

Задание 5. Мониторинг состояния репликации

Для отслеживания состояния и выявления проблем используется **Replication Monitor** .

1. В **Object Explorer** (на издате) щёлкните правой кнопкой мыши на папке **Репликация** и выберите **Запустить монитор репликации (Launch Replication Monitor)** .
2. В левой панели разверните своего издателя и выберите публикацию ShopDB_TransactionPublication.
3. В правой панели изучите вкладки:
 - **Все подписки (All Subscriptions)** — статус подписки, задержка .
 - **Маркеры трассировки (Tracer Tokens)** — для измерения задержки на каждом этапе .

4. Чтобы создать маркер трассировки, на вкладке **Маркеры трассировки** нажмите **Вставить маркер (Insert Tracer)** . Через некоторое время вы увидите время прохождения маркера от издателя до подписчика.

Задание 6. Моделирование и устранение проблем репликации

Проблема 1: Остановка агента распространения

1. В **Object Explorer** найдите папку **Агент SQL Server -> Задания (Jobs)**.
2. Найдите задание с именем, содержащим распространитель (например, имя_сервера-ShopDB_TransactionPublication-имя_подписчика-номер).
3. Щёлкните правой кнопкой мыши и выберите **Отключить задание**.
4. На **издате** выполните ещё одну вставку данных:

```
sql
USE ShopDB_Replica;
GO
INSERT INTO Products (ProductName, Price, Quantity) VALUES ('Принтер', 8500.00, 5);
GO
```

5. Проверьте, что на **подписчике** новые данные не появились.
6. В **Replication Monitor** вы увидите, что статус подписки изменился, агент не работает .
7. Включите задание обратно (щёлкните правой кнопкой -> **Включить задание**).
8. Агент запустится и доставит накопившиеся изменения. Проверьте подписчика.

Проблема 2: Конфликт данных (ручное изменение на подписчике)

Репликация транзакций обычно предполагает, что изменения происходят только на издате. Если изменить данные на подписчике, это может привести к конфликтам .

1. На **подписчике** выполните изменение (хотя это не рекомендуется для этого типа репликации):

```
sql
USE ShopDB_Subscriber;
GO
UPDATE Products SET Price = 1000 WHERE ProductID = 2; -- Мышь
GO
```

2. На **издателе** измените ту же запись:

```
sql
USE ShopDB_Replica;
GO
UPDATE Products SET Price = 950 WHERE ProductID = 2; -- Мышь
GO
```

3. Проверьте, какое значение в итоге оказалось на подписчике. (Обычно побеждает изменение издателя, так как оно приходит через репликацию и перезаписывает локальное изменение).

Проблема 3: Просмотр нераспространённых команд

Если подписчик долго недоступен или агент остановлен, команды накапливаются в базе распространителя.

1. В **Replication Monitor** выберите подписку и откройте окно агента распространения двойным щелчком.

2. Перейдите на вкладку **Нераспространенные команды (Undistributed Commands)**. Здесь отображается количество команд, ожидающих применения, и примерное время их выполнения .

Задание 7. Документирование настроек репликации

Составьте в отчёте таблицу с параметрами вашей репликации.

Параметр	Значение
Топология	Издатель + локальный распространитель
Экземпляр издателя	Ваш_сервер
Экземпляр подписчика	Ваш_сервер\Имя_экземпляра
База данных издателя	ShopDB_Replica
База данных подписчика	ShopDB_Subscriber
Тип репликации	Транзакционная (Transactional)
Публикация	ShopDB_TransactionPublication
Статьи (таблицы)	Products, Orders
Тип подписки	Push-подписка
Папка моментальных снимков	C:\ReplicationSnapshot
Статус синхронизации	Работает / Не работает
Средняя задержка (из маркеров)	... секунд

Контрольные вопросы

1. Объясните разницу между ролями издателя, распространителя и подписчика в репликации SQL Server .

2. Какие типы репликации существуют в SQL Server? В каких случаях целесообразно использовать каждый из них?

3. Что такое "push-подписка" и "pull-подписка"? Какая из них была использована в работе?

4. Для чего нужна папка моментальных снимков (snapshot folder)?

5. Какие инструменты можно использовать для мониторинга состояния репликации и диагностики проблем?

6. Что произойдёт с данными на подписчике, если на подписчике вручную изменить

данные, которые затем обновятся на издатель?

7. Как измерить фактическую задержку репликации между издателем и подписчиком?

Практическое занятие №14. Обслуживание и мониторинг базы данных. Регулярное обслуживание (вакуумирование, дефрагментация). Сбор метрик производительности. Диагностика и устранение неполадок.

Цель работы: Научиться выполнять регулярные операции по обслуживанию базы данных (дефрагментация индексов, обновление статистики), освоить методы сбора метрик производительности с помощью динамических представлений (DMV), а также приобрести навыки диагностики типовых проблем и их устранения.

Теоретическая часть

1. Регулярное обслуживание базы данных

Даже самая хорошо спроектированная база данных со временем теряет производительность из-за фрагментации и устаревшей статистики.

1.1. Фрагментация индексов

Когда данные добавляются, изменяются или удаляются, индексы фрагментируются. Это приводит к тому, что для чтения данных требуется больше операций ввода-вывода.

- **Внутренняя фрагментация** — страницы индекса заполнены не полностью, что приводит к нерациональному использованию места и увеличению количества страниц для чтения.
- **Внешняя фрагментация (логическая фрагментация)** — логический порядок страниц не соответствует физическому, что замедляет последовательное сканирование.

Операции обслуживания:

- **ALTER INDEX ... REORGANIZE** — дефрагментация на уровне листовых страниц. Выполняется онлайн (не блокирует таблицу).
- **ALTER INDEX ... REBUILD** — полное перестроение индекса. Может выполняться офлайн (блокирует таблицу) или онлайн (в редакциях Enterprise).

1.2. Статистика

Оптимизатор запросов использует статистику для выбора наиболее эффективного плана выполнения. Устаревшая статистика может привести к выбору неоптимального плана.

- **UPDATE STATISTICS** — обновляет статистику для таблицы или индекса.

1.3. Управление файлами

- **Автоматическое сжатие (AUTO_SHRINK)** должно быть отключено, так как оно вызывает сильную фрагментацию и потребляет ресурсы.
- **Рост файлов** лучше настроить на фиксированное значение (в мегабайтах), а не в процентах, чтобы избежать чрезмерной фрагментации файлов.

2. Мониторинг производительности

SQL Server предоставляет мощный инструментарий для мониторинга — **динамические административные представления (DMV)**. Они хранят информацию о состоянии сервера, запросах, сессиях и использовании ресурсов.

Ключевые DMV для мониторинга:

- **sys.dm_db_index_physical_stats** — информация о фрагментации индексов.
- **sys.dm_exec_requests** — текущие запросы, их статус, ожидания.
- **sys.dm_exec_sessions** — информация о сессиях пользователей.
- **sys.dm_exec_query_stats** — статистика выполнения запросов (количество запусков, суммарное время CPU, чтения и т.д.).
- **sys.dm_db_resource_stats** — использование ресурсов (CPU, IO, память) за последний час.

Ход работы

Задание 1. Подготовка тестовой среды

Для того чтобы увидеть эффект от обслуживания, нам нужна база данных с таблицами и индексами, которые со временем фрагментируются.

1. Запустите **SQL Server Management Studio** и подключитесь к вашему экземпляру.

2. Создайте базу данных для практической работы:

sql

```
CREATE DATABASE MaintenanceLab;
```

```
GO
```

```
USE MaintenanceLab;
```

```
GO
```

3. Создайте таблицу с кластеризованным и некластеризованным индексами, а затем наполните её большим объёмом данных, чтобы индексы стали фрагментированными.

sql

```
-- Таблица для журнала событий
```

```
CREATE TABLE EventLog (
```

```
    EventID INT IDENTITY(1,1) PRIMARY KEY, -- кластеризованный индекс по умолчанию
```

```
    EventDate DATETIME NOT NULL,
```

```
    EventType NVARCHAR(50),
```

```
    EventMessage NVARCHAR(MAX),
```

```
    UserName NVARCHAR(100),
```

```
    ProcessID INT
```

```
);
```

```
GO
```

```
-- Создадим некластеризованный индекс по дате
```

```
CREATE INDEX IX_EventDate ON EventLog(EventDate);
```

```
GO
```

```
-- Создадим некластеризованный индекс по типу события
```

```
CREATE INDEX IX_EventType ON EventLog(EventType);
```

```
GO
```

```
-- Наполним таблицу большим количеством записей (10 000 строк)
```

```
-- Используем цикл для генерации данных
```

```
DECLARE @i INT = 1;
```

```
WHILE @i <= 10000
```

```
BEGIN
```

```
    INSERT INTO EventLog (EventDate, EventType, EventMessage, UserName, ProcessID)
```

```
    VALUES (
```

```
        DATEADD(day, -(@i % 365), GETDATE()), -- даты за последний год
```

```
        CASE (@i % 5)
```

```
            WHEN 0 THEN 'INFO'
```

```
            WHEN 1 THEN 'WARNING'
```

```
            WHEN 2 THEN 'ERROR'
```

```
            WHEN 3 THEN 'DEBUG'
```

```
            ELSE 'CRITICAL'
```

```
        END,
```

```
        'Сообщение события номер ' + CAST(@i AS NVARCHAR(10)),
```

```
        CASE (@i % 10)
```

```
            WHEN 0 THEN 'User1'
```

```
            WHEN 1 THEN 'User2'
```

```
            WHEN 2 THEN 'User3'
```

```
            WHEN 3 THEN 'Admin'
```

```
            WHEN 4 THEN 'System'
```

```
            ELSE 'Guest'
```

```

        END,
        @i % 100
    );
    SET @i = @i + 1;
END;
GO

PRINT 'Тестовая таблица создана и наполнена данными.';
GO

```

Задание 2. Оценка текущей фрагментации индексов

Прежде чем что-то делать, нужно измерить текущий уровень фрагментации.

1. Выполните следующий запрос для анализа фрагментации индексов таблицы EventLog:

```

sql
USE MaintenanceLab;
GO

SELECT
    OBJECT_NAME(ips.object_id) AS TableName,
    i.name AS IndexName,
    ips.index_type_desc AS IndexType,
    ips.avg_fragmentation_in_percent AS AvgFragmentationPercent,
    ips.fragment_count AS FragmentCount,
    ips.avg_fragment_size_in_pages AS AvgFragmentSizePages,
    ips.page_count AS PageCount
FROM sys.dm_db_index_physical_stats(
    DB_ID('MaintenanceLab'),    -- ID базы данных
    OBJECT_ID('EventLog'),      -- ID объекта (таблицы)
    NULL,                       -- ID индекса (все индексы)
    NULL,                       -- ID раздела (все)
    'LIMITED'                   -- Режим сканирования
) ips
JOIN sys.indexes i ON ips.object_id = i.object_id AND ips.index_id = i.index_id
ORDER BY ips.avg_fragmentation_in_percent DESC;
GO

```

Анализ результатов:

- **0–10%** — отлично, ничего делать не нужно.
- **10–30%** — рекомендуется реорганизация (REORGANIZE).
- **>30%** — рекомендуется перестроение (REBUILD) .

Запишите текущие значения фрагментации для каждого индекса. Они понадобятся для сравнения после выполнения обслуживания.

Задание 3. Выполнение дефрагментации индексов

Теперь выполним операции по дефрагментации.

Шаг 1. Реорганизация индекса (при фрагментации 10-30%)

Если какой-либо из индексов имеет фрагментацию в диапазоне 10-30%, выполните реорганизацию:

```

sql
ALTER INDEX IX_EventDate ON EventLog REORGANIZE;
GO

```

Шаг 2. Перестроение индекса (при фрагментации >30%)

Если фрагментация превышает 30%, выполните перестроение:

```
sql
ALTER INDEX IX_EventType ON EventLog REBUILD;
GO
```

Примечание: Перестроение индекса может занять некоторое время и заблокировать таблицу. В реальной среде это планируют на время наименьшей нагрузки.

Шаг 3. Повторная проверка фрагментации

Выполните запрос из **Задания 2** ещё раз и сравните новые значения фрагментации с исходными. Убедитесь, что процент фрагментации снизился (обычно до 0% после перестроения).

Задание 4. Обновление статистики

После изменения данных и дефрагментации индексов важно обновить статистику, чтобы оптимизатор запросов имел актуальную информацию.

1. Выполните команду для обновления статистики по таблице:

```
sql
UPDATE STATISTICS EventLog;
GO
```

2. Можно обновить статистику для конкретного индекса:

```
sql
UPDATE STATISTICS EventLog IX_EventDate;
GO
```

3. Для просмотра текущей статистики (количество строк, плотность и т.д.) используйте:

```
sql
DBCC SHOW_STATISTICS ('EventLog', IX_EventDate);
GO
```

Задание 5. Мониторинг текущей активности

Теперь перейдём к мониторингу. Представьте, что пользователи жалуются на замедление работы. Нужно выяснить, какие запросы выполняются прямо сейчас.

Шаг 1. Просмотр текущих запросов

Выполните запрос, который покажет все выполняющиеся в данный момент запросы, их статус, время выполнения и тип ожидания:

```
sql
SELECT
    r.session_id,
    r.status,
    r.command,
    r.cpu_time AS CPUTime_ms,
    r.total_elapsed_time AS ElapsedTime_ms,
    r.reads,
    r.writes,
    r.logical_reads,
    r.wait_type,
    r.wait_time,
    r.wait_resource,
    t.text AS QueryText,
    SUBSTRING(t.text, r.statement_start_offset/2 + 1,
        (CASE WHEN r.statement_end_offset = -1 THEN LEN(t.text)
            ELSE r.statement_end_offset/2 - r.statement_start_offset/2 + 1
        END)) AS CurrentStatement
```

```
FROM sys.dm_exec_requests r
CROSS APPLY sys.dm_exec_sql_text(r.sql_handle) t
WHERE r.session_id > 50 -- исключаем системные сессии
ORDER BY r.cpu_time DESC;
GO
```

Этот запрос показывает :

- **wait_type** — на что именно "ждёт" запрос (например, PAGEIOLATCH_EX — ожидание чтения страницы с диска).
- **status** — running, runnable, suspended.
- **QueryText** — текст выполняющегося запроса.

Шаг 2. Моделирование проблемы

Создадим искусственную нагрузку, чтобы увидеть, как она отражается в DMV.

Откройте **новое окно запроса** (под тем же пользователем) и выполните "тяжёлый" запрос:

```
sql
USE MaintenanceLab;
GO
```

-- Тяжёлый запрос без индекса (вызовет сканирование таблицы)

```
SELECT * FROM EventLog
WHERE EventMessage LIKE '%9999%'
ORDER BY EventDate DESC;
GO
```

Пока этот запрос выполняется, вернитесь в первое окно и повторно выполните мониторинговый запрос из Шага 1. Вы должны увидеть выполняющийся запрос, его высокое значение reads и, возможно, статус ожидания.

Шаг 3. Поиск "горячих" запросов по статистике выполнения

Для поиска запросов, которые потребляют больше всего ресурсов в целом (а не только текущих), используется sys.dm_exec_query_stats .

Найдите топ-5 запросов по времени CPU:

```
sql
SELECT TOP 5
    qs.total_worker_time/1000 AS TotalCPUTime_ms,
    qs.execution_count,
    qs.total_elapsed_time/1000 AS TotalElapsedTime_ms,
    qs.total_logical_reads,
    qs.total_logical_writes,
    SUBSTRING(st.text, qs.statement_start_offset/2 + 1,
        (CASE WHEN qs.statement_end_offset = -1 THEN LEN(st.text)
            ELSE qs.statement_end_offset/2 - qs.statement_start_offset/2 + 1
        END)) AS QueryText
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) st
ORDER BY qs.total_worker_time DESC;
GO
```

Этот запрос покажет, какие запросы в целом наиболее ресурсоёмки.

Задание 6. Мониторинг использования ресурсов

Для понимания, хватает ли базе данных выделенных ресурсов (CPU, I/O, память), можно использовать следующие DMV.

Шаг 1. Мониторинг ресурсов за последний час

```
sql
SELECT
    AVG(avg_cpu_percent) AS AvgCPUPercent,
```

```

MAX(avg_cpu_percent) AS MaxCPUPercent,
AVG(avg_data_io_percent) AS AvgDataIOPercent,
MAX(avg_data_io_percent) AS MaxDataIOPercent,
AVG(avg_memory_usage_percent) AS AvgMemoryPercent,
MAX(avg_memory_usage_percent) AS MaxMemoryPercent
FROM sys.dm_db_resource_stats;
GO

```

Этот запрос показывает среднее и максимальное использование ресурсов за последний час .
Если значения регулярно приближаются к 100%, это сигнал о необходимости оптимизации или увеличения ресурсов.

Шаг 2. Размер базы данных и свободное место

```

sql
SELECT
    name AS FileName,
    type_desc AS FileType,
    size/128.0 AS CurrentSizeMB, -- size указан в страницах по 8KB, делим на 128 для MB
    FILEPROPERTY(name, 'SpaceUsed')/128.0 AS SpaceUsedMB,
    (size - FILEPROPERTY(name, 'SpaceUsed'))/128.0 AS FreeSpaceMB
FROM sys.database_files;
GO

```

Задание 7. Диагностика и устранение типовых неполадок

Проблема 1: Блокировки (Locking) и взаимоблокировки (Deadlocks)

Смоделируем ситуацию, когда одна транзакция блокирует другую.

1. В окне №1 начните транзакцию, но не завершайте её:

```

sql
USE MaintenanceLab;
GO

```

```

BEGIN TRAN

```

```

    UPDATE EventLog SET EventMessage = 'Изменение в транзакции 1' WHERE EventID = 1;
    -- Транзакция не завершена COMMIT или ROLLBACK

```

```

GO

```

2. В окне №2 попробуйте обновить ту же запись:

```

sql
USE MaintenanceLab;
GO

```

```

UPDATE EventLog SET EventMessage = 'Попытка изменения 2' WHERE EventID = 1;

```

```

GO

```

Этот запрос "зависнет", ожидая завершения первой транзакции.

3. В окне мониторинга (окно №3) найдите блокировки:

```

sql
SELECT
    blocking.session_id AS BlockingSession,
    blocked.session_id AS BlockedSession,
    blocked.wait_type,
    blocked.wait_time,
    blocking_text.text AS BlockingQuery,
    blocked_text.text AS BlockedQuery
FROM sys.dm_exec_requests blocked
JOIN sys.dm_exec_requests blocking ON blocked.blocking_session_id = blocking.session_id
CROSS APPLY sys.dm_exec_sql_text(blocking.sql_handle) blocking_text

```

```
CROSS APPLY sys.dm_exec_sql_text(blocked.sql_handle) blocked_text;
GO
```

4. **Устранение:** Завершите блокирующую транзакцию (в окне №1 выполните ROLLBACK или COMMIT). Либо, в крайнем случае, можно принудительно завершить блокирующую сессию:

```
sql
KILL <номер_блокирующей_сессии>;
```

Проблема 2: Неоптимальный запрос из-за устаревшей статистики

1. Обновите статистику, чтобы "сбросить" её актуальность (в реальности она устаревает со временем):

```
sql
UPDATE STATISTICS EventLog WITH ROWCOUNT = 100, PAGECOUNT = 10; --
```

"Обманываем" статистику

2. Выполните запрос, который будет искать редкие данные:

```
sql
SELECT * FROM EventLog WHERE EventType = 'CRITICAL';
```

3. Посмотрите план выполнения запроса. Для этого нажмите **Включить фактический план выполнения (Include Actual Execution Plan)** перед выполнением запроса (Ctrl+M).

Вы увидите, что используется сканирование (Index Scan), а не поиск (Index Seek), хотя по типу события есть индекс.

4. **Устранение:** Обновите статистику корректно:

```
sql
UPDATE STATISTICS EventLog IX_EventType WITH FULLSCAN;
GO
```

5. Повторите запрос. Теперь план должен использовать Index Seek, что гораздо эффективнее.

Задание 8. Документирование

Составьте в отчёте таблицы, описывающие результаты мониторинга и обслуживания.

Таблица 1. Анализ фрагментации индексов

Имя индекса	Тип индекса	Фрагментация ДО (%)	Фрагментация ПОСЛЕ (%)	Выполненное действие
PK__EventLog__2E3BD7D3 (кластеризованный)	CLUSTERED	...	...	REBUILD / REORGANIZE
IX_EventDate	NONCLUSTERED	...	...	REBUILD / REORGANIZE
IX_EventType	NONCLUSTERED	...	...	REBUILD / REORGANIZE

Таблица 2. Мониторинг ресурсов (за последний час)

Метрика	Среднее значение	Максимальное значение
CPU (%)	...	...
Data IO (%)	...	...
Memory (%)	...	...

Таблица 3. Топ-3 самых ресурсоёмких запроса (по CPU)

Запрос	Время CPU (мс)	Количество чтений	Количество выполнений
SELECT * FROM EventLog ...	...	...	...
...	...	...	...

Контрольные вопросы

1. В чем разница между реорганизацией (REORGANIZE) и перестроением (REBUILD) индекса? В каких случаях что применять?
2. Почему устаревшая статистика может привести к выбору неоптимального плана запроса?
3. Для чего нужны динамические административные представления (DMV)?
Приведите примеры трёх DMV и опишите, какую информацию они предоставляют.
4. Что означает поле wait_type в sys.dm_exec_requests? Какие типы ожиданий вы знаете и на что они указывают?
5. Как обнаружить блокировку и определить, какая сессия является блокирующей?
6. Почему рекомендуется отключать автоматическое сжатие (AUTO_SHRINK) базы данных?

Практическое занятие №15. Аудит безопасности баз данных. Создание и управление защищенными соединениями с сервером.

Цель работы: Научиться настраивать аудит действий пользователей на уровне сервера и базы данных, анализировать журналы аудита, а также освоить методы создания защищенных соединений с SQL Server (принудительное шифрование, расширенная защита).

Теоретическая часть

1. Аудит безопасности в SQL Server

Аудит (Audit) — это процесс отслеживания и записи событий, происходящих в системе, с целью последующего анализа и выявления подозрительных действий или нарушений политики безопасности. SQL Server предоставляет встроенный механизм аудита, который позволяет фиксировать:

- Успешные и неудачные попытки входа в систему
- Выполнение DDL-операций (CREATE, ALTER, DROP)
- Выполнение DML-операций (SELECT, INSERT, UPDATE, DELETE) над важными таблицами
- Изменения прав доступа
- Действия с ролями и пользователями

Компоненты аудита в SQL Server:

1. **Серверный аудит (Server Audit)** — определяет, куда будут сохраняться данные аудита (файл, журнал приложений Windows, журнал безопасности) .
2. **Спецификация серверного аудита (Server Audit Specification)** — определяет, какие действия уровня сервера отслеживать (например, LOGIN_CHANGE_GROUP, FAILED_LOGIN_GROUP).
3. **Спецификация аудита базы данных (Database Audit Specification)** — определяет, какие действия уровня базы данных отслеживать (например, SELECT, INSERT на конкретной таблице) .

2. Защищенные соединения

При подключении клиента к SQL Server данные передаются по сети. Для защиты от перехвата и атак "человек посередине" используются следующие механизмы:

2.1. Принудительное шифрование (Force Encryption)

Требует, чтобы все подключения к SQL Server были зашифрованы с использованием TLS (протокол защиты транспортного уровня). Если шифрование не поддерживается клиентом, соединение отклоняется.

2.2. Расширенная защита (Extended Protection)

Функция, помогающая предотвратить атаки ретрансляции аутентификации (authentication relay attacks). Она обеспечивает, что клиент знает, к какому именно сервису он подключается.

- **Привязка службы (Service Binding)** — проверка имени участника службы (SPN) для атак с заманиванием (luring attacks).
- **Привязка канала (Channel Binding)** — проверка маркера привязки канала (CBT) для защиты от спуфинговых атак (spoofing attacks). Требует шифрования.

Уровни расширенной защиты:

- **Off** — отключена (менее безопасно, совместимо со старыми ОС)
- **Allowed** — разрешена (требуется для ОС, поддерживающих защиту, игнорируется для остальных)
- **Required** — обязательна (только защищенные соединения, наиболее безопасно)

Ход работы

Задание 1. Подготовка тестовой среды

1. Запустите **SQL Server Management Studio** и подключитесь к вашему экземпляру SQL Server (желательно с правами администратора, например, sa).

2. Создайте тестовую базу данных AuditLab и таблицу с конфиденциальными данными.

sql

```
CREATE DATABASE AuditLab;  
GO
```

```
USE AuditLab;  
GO
```

```
CREATE TABLE Employees (  
    EmployeeID INT PRIMARY KEY IDENTITY(1,1),  
    FullName NVARCHAR(100),  
    Position NVARCHAR(50),  
    Salary DECIMAL(10,2),  
    PassportNumber NVARCHAR(20)  
);  
GO
```

```
INSERT INTO Employees (FullName, Position, Salary, PassportNumber) VALUES  
(Иванов Иван', 'Менеджер', 50000, '1234 567890'),  
(Петрова Мария', 'Бухгалтер', 60000, '2345 678901'),  
(Сидоров Петр', 'Разработчик', 70000, '3456 789012');  
GO
```

```
SELECT * FROM Employees;  
GO
```

3. Создайте тестового пользователя, от имени которого мы будем проверять аудит:

sql

```
USE master;  
GO  
CREATE LOGIN TestUser WITH PASSWORD = 'TestPass123!';  
GO  
USE AuditLab;  
GO  
CREATE USER TestUser FOR LOGIN TestUser;  
GO  
GRANT SELECT ON Employees TO TestUser;  
GO
```

Задание 2. Настройка аудита входа в систему (логирование попыток подключения)

Прежде чем создавать сложные спецификации аудита, настроим базовое логирование попыток входа — это самый простой и эффективный способ отслеживать подозрительную активность .

1. В **Обозревателе объектов** щёлкните **правой кнопкой мыши** на имени сервера и выберите **Свойства**.

2. Перейдите на страницу **Безопасность (Security)**.

3. В разделе **Аудит входа (Login auditing)** выберите:

- **Неудачные и успешные попытки входа (Both failed and successful logins)**
- Это позволит фиксировать все попытки подключения к серверу.

Нажмите **ОК**. Появится предупреждение, что изменение вступит в силу только после перезапуска службы SQL Server.

4. В **Обозревателе объектов** снова щёлкните правой кнопкой на сервере и выберите **Перезапустить (Restart)**. Подтвердите перезапуск.

Примечание: Если у студентов нет прав на перезапуск сервера, преподаватель может выполнить этот шаг демонстрационно, либо можно использовать виртуальную машину, где у студентов есть права администратора.

5. После перезапуска создайте несколько попыток входа:
 - Успешную (подключитесь от имени TestUser и сразу отключитесь)
 - Неудачную (попробуйте подключиться с неправильным паролем)
6. Просмотрите журнал аудита. Аудит входа записывается в **журнал приложений**

Windows.

- Нажмите **Пуск** → выполните команду eventvwr.msc (или найдите "Просмотр событий").
- Перейдите в **Журналы Windows** → **Приложение**.
- Найдите события с источником **MSSQLSERVER** (или имя вашего экземпляра).
- Успешные входы имеют код события **18454**, неудачные — **18456**.

Задание 3. Создание серверного аудита (Server Audit)

Теперь создадим более гибкую систему аудита с сохранением результатов в файл.

Шаг 1. Создание аудита через графический интерфейс

1. В **Обозревателе объектов** раскройте папку **Безопасность (Security)**.
2. Щёлкните правой кнопкой мыши на папке **Аудит (Audits)** → выберите **Создать аудит... (New Audit...)**.
3. В открывшемся окне заполните:
 - **Имя аудита:** Audit_EmployeeData
 - **Назначение журнала (Audit destination):** **Файл**
 - **Путь к файлу (File path):** C:\SQLAudit\ (если папка не существует, её нужно создать заранее)
 - **Максимальный размер файла:** 100 МБ
 - **Максимальное количество файлов:** 5 (для ротации)
 - **Задержка очереди (Queue delay):** 1000 миллисекунд
4. Нажмите **ОК**.
5. Аудит создаётся в отключённом состоянии. Чтобы включить его, щёлкните правой кнопкой на Audit_EmployeeData → **Включить аудит (Enable Audit)**.

Шаг 2. (Альтернатива) Создание аудита с помощью T-SQL

```
sql
USE master;
GO
CREATE SERVER AUDIT Audit_EmployeeData
TO FILE (FILEPATH = 'C:\SQLAudit\', MAXSIZE = 100 MB, MAX_ROLLOVER_FILES = 5)
WITH (QUEUE_DELAY = 1000);
GO
ALTER SERVER AUDIT Audit_EmployeeData WITH (STATE = ON);
GO
```

Задание 4. Создание спецификации аудита базы данных

Теперь укажем, какие именно действия в нашей базе AuditLab нужно отслеживать.

Например, мы хотим фиксировать все операции SELECT над таблицей Employees, а также любые изменения структуры таблицы.

Шаг 1. Создание спецификации через GUI

1. В **Обозревателе объектов** перейдите в базу данных AuditLab → **Безопасность**.
2. Щёлкните правой кнопкой мыши на папке **Спецификации аудита базы данных (Database Audit Specifications)** → выберите **Создать спецификацию аудита базы данных....**
3. В открывшемся окне:
 - **Имя:** AuditSpec_Employees
 - **Аудит:** выберите из списка Audit_EmployeeData
 - В сетке **Действия аудита (Audit actions)** добавьте строки:

Тип действия аудита	Схема объекта	Имя объекта	Имя участника
SELECT	dbo	Empl oyees	public
INSERT	dbo	Empl oyees	public
UPDATE	dbo	Empl oyees	public
DELETE	dbo	Empl oyees	public

4. Нажмите **ОК**.

5. Включите спецификацию аудита: правой кнопкой на AuditSpec_Employees → **Включить спецификацию аудита базы данных**.

Шаг 2. (Альтернатива) Создание спецификации с помощью T-SQL

```
sql
USE AuditLab;
GO
CREATE DATABASE AUDIT SPECIFICATION AuditSpec_Employees
FOR SERVER AUDIT Audit_EmployeeData
ADD (SELECT ON dbo.Employees BY public),
ADD (INSERT ON dbo.Employees BY public),
ADD (UPDATE ON dbo.Employees BY public),
ADD (DELETE ON dbo.Employees BY public)
WITH (STATE = ON);
GO
```

Задание 5. Тестирование аудита

1. Создайте новое подключение от имени пользователя TestUser (пароль TestPass123!).
2. В этом подключении выполните несколько операций с таблицей Employees:

```
sql
USE AuditLab;
GO
SELECT * FROM Employees;          -- Чтение данных
GO
UPDATE Employees SET Salary = 75000 WHERE EmployeeID = 1; -- Изменение данных
GO
INSERT INTO Employees (FullName, Position, Salary, PassportNumber)
VALUES ('Новиков Антон', 'Стажёр', 30000, '4567 890123'); -- Вставка
GO
```

3. Вернитесь в основное подключение (с правами администратора) и просмотрите результаты аудита.

Способ 1: Просмотр через файл

- Откройте папку C:\SQLAudit\. Там должен быть файл с расширением .sqlaudit.
- Дважды щёлкните по нему — он откроется в SSMS как таблица с детальной

информацией о каждом событии.

(Рисунок 5: Просмотр файла аудита в SSMS)

Способ 2: Просмотр с помощью функции fn_get_audit_file

```
sql
SELECT
    event_time,
    session_id,
    database_name,
    schema_name,
```

```

object_name,
statement,
session_server_principal_name AS UserName,
action_id,
succeeded
FROM sys.fn_get_audit_file('C:\SQLAudit\*.sqlaudit', DEFAULT, DEFAULT)
WHERE database_name = 'AuditLab'
ORDER BY event_time DESC;
GO

```

Вы увидите все выполненные операции SELECT, INSERT, UPDATE с указанием времени, пользователя и текста команды .

Задание 6. Настройка защищенных соединений (принудительное шифрование)

Для настройки защищенных соединений используется оснастка **SQL Server Configuration Manager**. Это внешняя программа, которая обычно доступна в меню Пуск → Microsoft SQL Server.

Примечание: Если у студентов нет прав на запуск Configuration Manager или изменение конфигурации сервера, это задание можно выполнить демонстрационно или пропустить, ограничившись объяснением теории.

Шаг 1. Включение принудительного шифрования

1. Запустите **SQL Server Configuration Manager** (от имени администратора).
 2. В левой панели выберите **SQL Server Network Configuration → Protocols for MSSQLSERVER** (или имя вашего экземпляра).
 3. Щёлкните правой кнопкой на **Protocols for MSSQLSERVER** → выберите **Properties**.
 4. Перейдите на вкладку **Flags** (Флаги):
 - Найдите параметр **Force Encryption** (Принудительное шифрование).
 - Установите значение **Yes**.
- (Рисунок 6: Вкладка Flags с параметром Force Encryption)*
5. Перейдите на вкладку **Certificate** (Сертификат):
 - Здесь можно выбрать сертификат для шифрования. Если сертификат не установлен, SQL Server использует самозаверяющий сертификат (самоподписанный). В учебных целях это допустимо.
 - Обычно сертификат выбирается автоматически.
 6. Нажмите **ОК**.

Шаг 2. Настройка расширенной защиты (Extended Protection)

1. В **SQL Server Configuration Manager** снова откройте свойства **Protocols for MSSQLSERVER**.
2. Перейдите на вкладку **Advanced** (Дополнительно).
3. Найдите параметр **Extended Protection** (Расширенная защита).
4. Установите значение **Allowed** (Разрешено) или **Required** (Обязательно) — для учебных целей выберите **Allowed**.
5. Параметр **Accepted NTLM SPNs** можно оставить пустым (используется, когда несколько SPN указывают на один сервер).

Шаг 3. Перезапуск службы

1. В **SQL Server Configuration Manager** выберите **SQL Server Services**.
2. Щёлкните правой кнопкой на **SQL Server (MSSQLSERVER)** → выберите **Restart**.

Шаг 4. Проверка защищенного соединения

1. Попробуйте подключиться к серверу из SSMS. Если всё настроено правильно, подключение должно быть успешным, но теперь оно будет зашифрованным.

2. Чтобы убедиться, что шифрование используется, можно выполнить запрос:

```

sql
SELECT

```

```

session_id,
encrypt_option,
net_transport,
auth_scheme
FROM sys.dm_exec_connections
WHERE session_id = @@SPID;
GO

```

Значение encrypt_option = TRUE указывает, что текущее соединение зашифровано .

Задание 7. Документирование

Составьте в отчёте таблицы, описывающие настроенную систему аудита и параметры защищенных соединений.

Таблица 1. Настройки аудита

Параметр	Значение
Тип аудита входа (Login auditing)	Неудачные и успешные попытки
Имя серверного аудита	Audit_EmployeeData
Назначение журнала	Файл, C:\SQLAudit\
Имя спецификации аудита БД	AuditSpec_Employees
Отслеживаемые действия	SELECT, INSERT, UPDATE, DELETE на dbo.Employees
Код события успешного входа	18454
Код события неудачного входа	18456

Таблица 2. Пример событий аудита (из fn_get_audit_file)

Время	Пользователь	Действие	Объект	Успешно
2026-02-18 10:15:23	TestUser	SELECT	Employees	1
2026-02-18 10:15:35	TestUser	UPDATE	Employees	1
2026-02-18 10:15:42	TestUser	INSERT	Employees	1

Таблица 3. Настройки защищенных соединений

Параметр	Значение
Force Encryption (принудительное шифрование)	Yes
Extended Protection (расширенная защита)	Allowed
Используемый сертификат	Самозаверяющий (или указать)
encrypt_option для текущего соединения	TRUE

Контрольные вопросы

1. В чём разница между серверным аудитом (Server Audit) и спецификацией аудита базы данных (Database Audit Specification)?
2. Какие коды событий в журнале Windows соответствуют успешным и неудачным попыткам входа в SQL Server?
3. Почему недостаточно хранить пароли в открытом виде? Объясните необходимость аудита доступа.
4. Что такое расширенная защита (Extended Protection) и какие типы атак она предотвращает?
5. Какие два метода привязки используются в расширенной защите и в чём их различие?
6. Как проверить, зашифровано ли текущее соединение с SQL Server?

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Северо-Кавказский федеральный университет»

Колледж СКФУ в г. Ставрополе

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к самостоятельной работе**

ПМ.01 РАЗРАБОТКА, АДМИНИСТРИРОВАНИЕ И ЗАЩИТА БАЗ ДАННЫХ

**МДК 01.01 Технология разработки баз данных
МДК 01.02 Обслуживание и защита баз данных**

Специальность 09.02.11 Разработка и управление программным обеспечением
Форма обучения очная

Ставрополь

1. Пояснительная записка

Методические указания для самостоятельной работы студентов при изучении дисциплин МДК. 01.01 Технология разработки баз данных, МДК. 01.02 Обслуживание и защита баз данных предназначены для обучающихся по специальности 09.02.11 Разработка и управление программным обеспечением.

При организации СР важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности. Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

По учебному плану специальности 09.02.11 Разработка и управление программным обеспечением в дисциплине МДК. 01.01 Технология разработки баз данных на внеаудиторную самостоятельную работу обучающихся отводится 36 часов, при изучении дисциплины МДК. 01.02 Обслуживание и защита баз данных 36 часов

План-график выполнения СРС

№ п/п	Наименование разделов и тем дисциплины, их краткое содержание; вид самостоятельной работы	Форма контроля	Объем часов
МДК 07.01. Управление и автоматизация баз данных			
1	Тема 1.1. Язык структурированных запросов Подготовка рефератов на темы: «Курсорные операции в хранимых процедурах. Обработка ошибок внутри хранимых процедур.» «Генерация исключений и сообщений об ошибках. Защита от SQL-инъекций с помощью хранимых процедур. Использование параметризованных запросов.» «Проверка корректности данных с помощью триггеров. Автоматическое заполнение полей с помощью триггера. Создание триггера, запрещающий удаление записей из таблицы, если они связаны с другими таблицами.» «Создание триггера, который реализует каскадное обновление связанной информации. Создание триггера со сложной логикой, включающей обработку ошибок. Оптимизация триггера с использованием временных таблиц.»	Подготовка рефератов	18
2	Тема 1.2. NoSQL базы данных Работа с учебной литературой по темам: 1. Основные понятия и история развития NoSQL технологий. Преимущества и недостатки NoSQL технологий по сравнению с реляционными базами данных. Типы NoSQL баз данных. 2. Ключ-значение базы данных. Основные принципы работы ключ-значение баз данных. Пример использования Redis: установка, основные команды, типы данных. Применение и сценарии использования ключ-значение баз данных. 3. Документо-ориентированные базы данных. Популярные системы: MongoDB, Couchbase, Firebase.	Работа с учебной литературой	18

	Структура документов и схемы данных. Запросы и индексация в document-oriented базах. Реальные примеры использования. 4. Колоночные базы данных. Архитектура колоночных баз данных. Области применения. Концепции колонок ориентированного подхода. Системы типа Cassandra, HBase. 5. Графовые базы данных. Основные понятия графов: узлы, ребра, свойства. Примеры запросов к графам: язык запросов Cypher. Сценарии использования графовых баз данных. 6. Проектирование схем данных в NoSQL. CAP-теорема и её значение. Подходы к денормализации данных. Паттерны проектирования для разных типов NoSQL баз данных. Управление консистентностью и доступностью данных. Методы оптимизации производительности NoSQL систем управления базами данных. Основные принципы управления данными и обслуживания NoSQL систем управления базами данных		
МДК 07.02. Сертификация информационных систем			
1	Тема 2.2. Управление доступом к базам данных Работа с литературой по темам: Хранимые процедуры и триггеры, Механизмы доступа к базам данных, Банк данных: состав, схема	Работа с учебной литературой	8
2	Тема 2.2. Управление доступом к базам данных Работа с литературой по темам: Технологии создания базы данных с применением языка SQL, Особенности обработки данных в объектно-ориентированных базах данных	Работа с учебной литературой	2
3	Тема 2.3. Резервное копирование и восстановление данных в штатном режиме Работа с литературой по темам: Резервное копирование и восстановление данных в штатном режиме. Синхронизация данных между узлами. Решение проблем с репликацией.	Работа с учебной литературой	10
4	Тема 2.4. Мониторинг и журналирование событий, возникающих в процессе функционирования баз данных Работа с литературой по темам: Мониторинг и журналирование событий, возникающих в процессе функционирования баз данных	Работа с учебной литературой	10
5	Тема 2.5. Обеспечение безопасной работы сервера системы управления базами данных Работа с литературой по темам: Структуры памяти. Однопроцессорные и многопроцессорные базы данных, Транзакции, блокировки и согласованность данных	Работа с учебной литературой	6
	Итого		72

3. Методические рекомендации по выполнению самостоятельной работы

3.1. Методические рекомендации по работе с учебной литературой

При работе с книгой необходимо подобрать литературу, научиться правильно ее читать, вести записи. Для подбора литературы в библиотеке используются алфавитный и систематический каталоги.

Важно помнить, что рациональные навыки работы с книгой - это всегда большая экономия времени и сил.

Правильный подбор учебников рекомендуется преподавателем, читающим лекционный курс. Необходимая литература может быть также указана в методических разработках по данному курсу.

Изучая материал по учебнику, следует переходить к следующему вопросу только после правильного уяснения предыдущего, описывая на бумаге все выкладки и вычисления (в том числе те, которые в учебнике опущены или на лекции даны для самостоятельного вывода).

При изучении любой дисциплины большую и важную роль играет самостоятельная индивидуальная работа.

Особое внимание следует обратить на определение основных понятий курса. Студент должен подробно разбирать примеры, которые поясняют такие определения, и уметь строить аналогичные примеры самостоятельно. Нужно добиваться точного представления о том, что изучаешь. Полезно составлять опорные конспекты. При изучении материала по учебнику полезно в тетради (на специально отведенных полях) дополнять конспект лекций. Там же следует отмечать вопросы, выделенные студентом для консультации с преподавателем.

Выводы, полученные в результате изучения, рекомендуется в конспекте выделять, чтобы они при перечитывании записей лучше запоминались.

Многим студентам помогает составление листа опорных сигналов, содержащего важнейшие и наиболее часто употребляемые формулы и понятия. Такой лист помогает запомнить формулы, основные положения лекции, а также может служить постоянным справочником для студента.

Основные виды систематизированной записи прочитанного:

Аннотирование – предельно краткое связное описание просмотренной или прочитанной книги (статьи), ее содержания, источников, характера и назначения;

Планирование – краткая логическая организация текста, раскрывающая содержание и структуру изучаемого материала;

Тезирование – лаконичное воспроизведение основных утверждений автора без привлечения фактического материала;

Цитирование – дословное выписывание из текста выдержек, извлечений, наиболее существенно отражающих ту или иную мысль автора;

Конспектирование – краткое и последовательное изложение содержания прочитанного.

Конспект – сложный способ изложения содержания книги или статьи в логической последовательности. Конспект аккумулирует в себе предыдущие виды записи, позволяет всесторонне охватить содержание книги, статьи. Поэтому умение составлять план, тезисы, делать выписки и другие записи определяет и технологию составления конспекта.

Методические рекомендации по составлению конспекта:

1. Внимательно прочитайте текст. Уточните в справочной литературе непонятные слова. При записи не забудьте вынести справочные данные на поля конспекта;
2. Выделите главное, составьте план;
3. Кратко сформулируйте основные положения текста, отметьте аргументацию автора;
4. Законспектируйте материал, четко следуя пунктам плана. При конспектировании старайтесь выразить мысль своими словами. Записи следует вести четко, ясно.
5. Грамотно записывайте цитаты. Цитируя, учитывайте лаконичность, значимость мысли.

В тексте конспекта желательно приводить не только тезисные положения, но и их доказательства. При оформлении конспекта необходимо стремиться к емкости каждого предложения. Мысли автора книги следует излагать кратко, заботясь о стиле и выразительности написанного. Число дополнительных элементов конспекта должно быть логически обоснованным, записи должны распределяться в определенной последовательности, отвечающей логической структуре произведения. Для уточнения и дополнения необходимо оставлять поля.

Овладение навыками конспектирования требует от студента целеустремленности, повседневной самостоятельной работы.

Информационное обеспечение обучения

Перечень рекомендуемой основной и дополнительной литературы, Интернет-ресурсов, необходимых для освоения профессионального модуля

Основная литература:

1. Волк В. К. Базы данных. Проектирование, программирование, управление и администрирование учебник для СПО / В. К. Волк - 3-е изд., стер. - Санкт-Петербург: Лань, 2024 - 340 с. - ISBN 978-5-507-47482-0
2. Домбровская Г., Новиков Б., Бейликова А. Оптимизация запросов в PostgreSQL/ пер. с англ. Д. А. Бейликова. - М.: ДМК Пресс, 2022 - 278 с. - ISBN 978-5-97060-963-7

Дополнительная литература:

1. Мамедли Р. Э. Системы управления базами данных: учебник для СПО / Р. Э. Мамедли - Санкт-Петербург: Лань, 2024 - 228 с. - ISBN 978-5-507-48730-1
2. Молдованова, О. В. Информационные системы и базы данных: учебное пособие для СПО / О. В. Молдованова. — 2-е изд. — Саратов: Профобразование, 2024. — 177 с. — ISBN 978-5-4488-1177-7.

Интернет-ресурсы:

1. <http://www.citforum.ru/> - Центр информационных технологий.
2. <http://www.5ballov.ru/> - Образовательный портал.
3. <http://www.fio.ru/> - Федерация Интернет – образования.
4. <http://tests.academy.ru/> - Тесты из области информационных технологий.
5. <http://www.codenet.ru/> - Все для программиста.
6. <http://public.tsu.ru/~wawlasov/start.htm> - В помощь учителю информатики.
7. <http://sciedu.city.ru/> - Наука и образование в России.