Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Кузбасский государственный технический университет
имени Т. Ф. Горбачева»

Институт профессионального образования

Кафедра информатики и информационных систем

Составитель О. С. Семенова

Управление базами данных

Методические материалы

Рекомендовано цикловой методической комиссией
Информационных систем и программирования
в качестве электронного издания
для использования в образовательном процессе

Кемерово 2026

Рецензент:
С. А. Асанов, старший преподаватель кафедры информационных и автоматизированных производственных систем

Семенова Ольга Сергеевна
Управление базами данных: методические материалы для обучающихся специальности СПО 09.02.11 «Разработка и управление программным обеспечением» / Кузбасский государственный технический университет имени Т. Ф. Горбачева ; кафедра информатики и информационных систем ; составитель О. С. Семенова. – Кемерово : КузГТУ, 2026. – 1 файл (1002 КБ). – Текст : электронный.

Методические материалы по дисциплине «Управление базами данных» содержат указания для проведения лабораторных работ и организации самостоятельной работ студентов.

Лабораторная работа №1
Установка и настройка систем управления базами данных

Цель работы: получение практических навыков установки и настройки сервера баз данных.

Теоретические положения

В современном информационном мире данные стали ключевым активом любой организации – от небольших стартапов до глобальных корпораций. Эффективное управление данными, обеспечение их целостности, безопасности и доступности являются критически важными задачами, от решения которых напрямую зависит успех бизнеса.

Microsoft SQL Server (MS SQL Server) представляет собой промышленную систему управления реляционными базами данных, которая уже несколько десятилетий остается одним из лидеров рынка корпоративных СУБД. Изучение этой платформы необходимо не только будущим администраторам баз данных, но и разработчикам, аналитикам, тестировщикам и IT-менеджерам, поскольку взаимодействие с базами данных составляет неотъемлемую часть практически любой IT-специальности.

MS SQL Server доступен в различных вариациях. Прежде всего, это MS SQL Server Enterprise – полный выпуск, нацеленный на использование в реальных проектах. Именно он используется на различных хостингах и серверах баз данных. Однако он доступен только в платной версии (не считая триального периода).

Для простых приложений может быть достаточно бесплатного выпуска Express. К тому же у него есть преимущество – его можно устанавливать в качестве реального сервера и использовать в реальных задачах несмотря на урезанный функционал по сравнению с полной версией.

Для установки MS SQL Server Express необходимо скачать и установить соответствующие программные компоненты.

Ссылка на скачивание: https://www.microsoft.com/ru-RU/download/details.aspx?id=101064 (рис. 1).

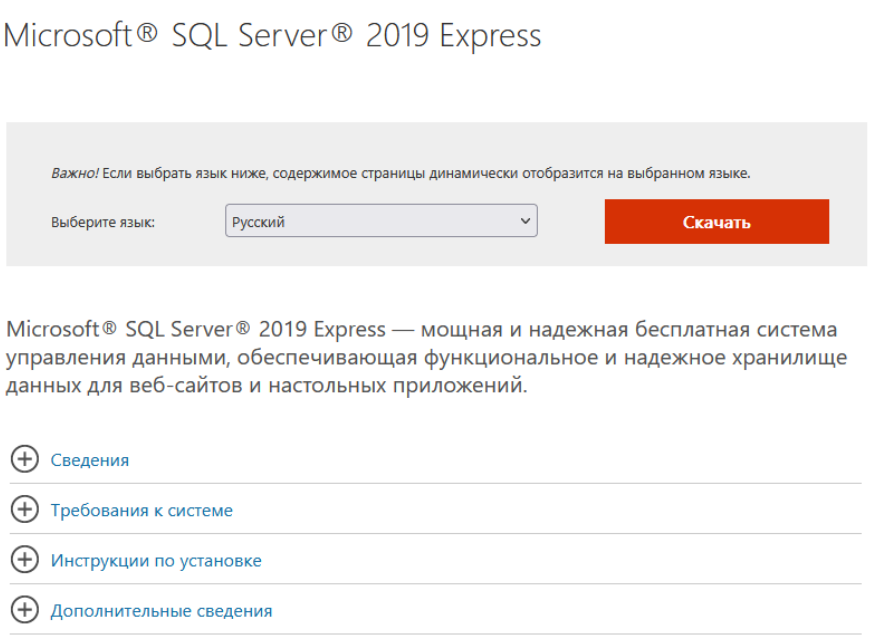

Рисунок 1 – Источник для скачивания версии

Установка выполняется в следующей последовательности:

1 шаг. Запуск скачанного файла установки.

2 шаг. Выбор типа установка (обновление или установка). Здесь выберем первый пункт «Новая установка изолированного экземпляра SQL…» (рис. 2).

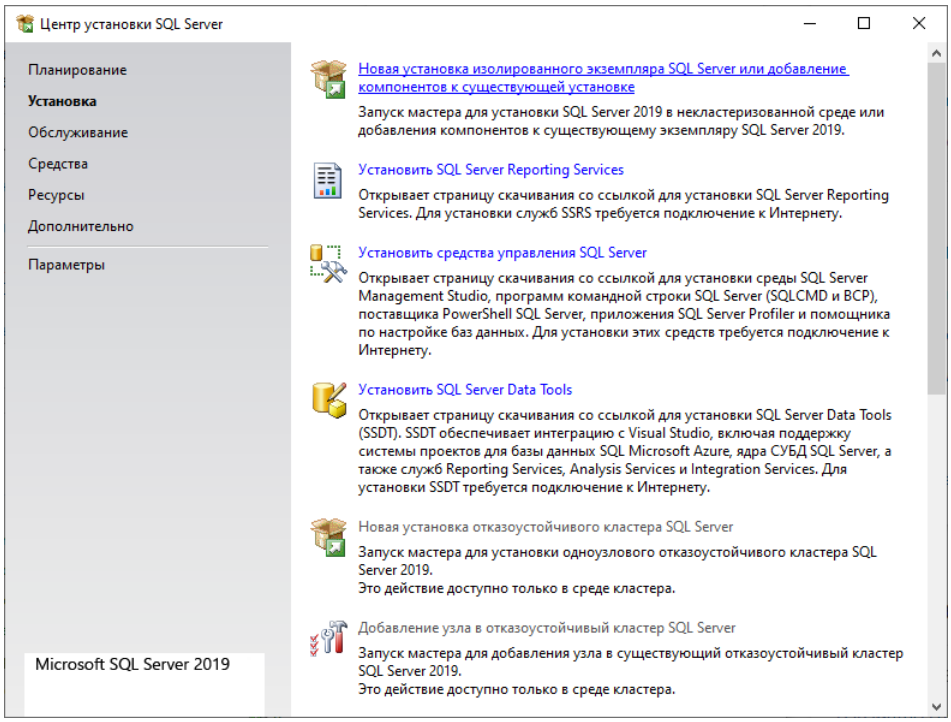

Рисунок 2 – Мастер установки MS SQL Server 2019

3 шаг. Подтверждение лицензионного соглашения.

4 шаг. Выбор компонентов (рис. 3). На этом этапе предлагается выбрать компоненты для установки. Здесь отметим только компонент «Службы ядра СУБД».

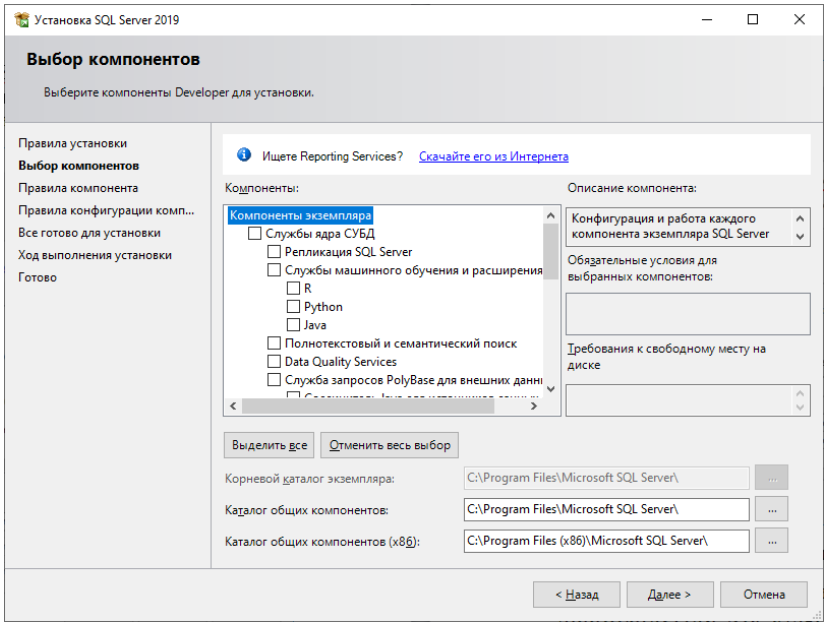

Рисунок 3 – Выбор компонентов

5 шаг. Настройки экземпляра. На этом этапе необходимо указать название и идентификатора (ID) запускаемого экземпляра SQL Server (рис. 4). Для имени указываем опцию «Экземпляр по умолчанию», а для ID устанавливаем MSSQLSERVER. Это будет то имя экземпляра, по которому мы сможем обращаться к серверу из внешних приложений.

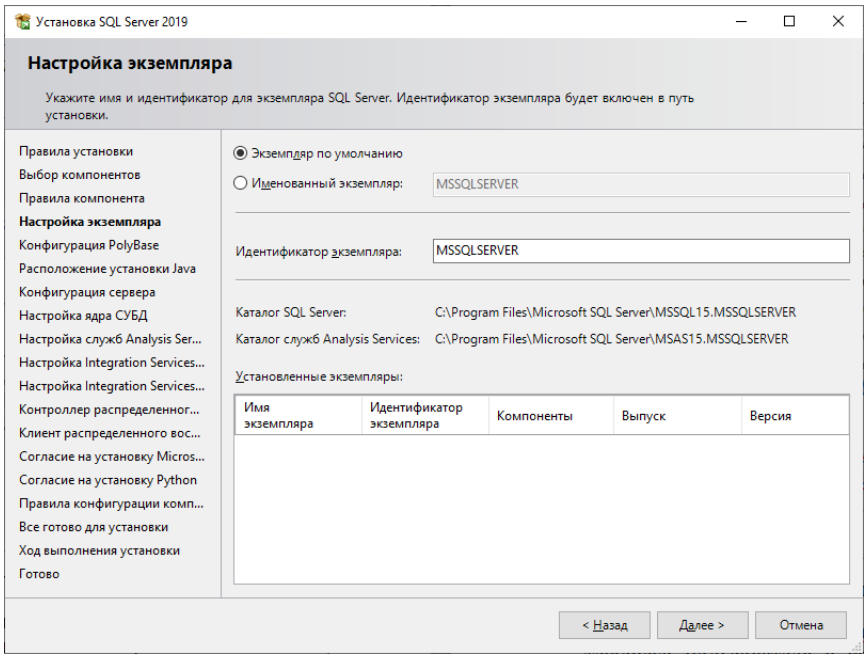

Рисунок 4 – Задание имени и идентификатора MS SQL Server

6 шаг. Настройка ядра СУБД. С помощью кнопки «Добавить текущего пользователя» добавим текущего пользователя в качестве администратора для сервера (рис. 5).

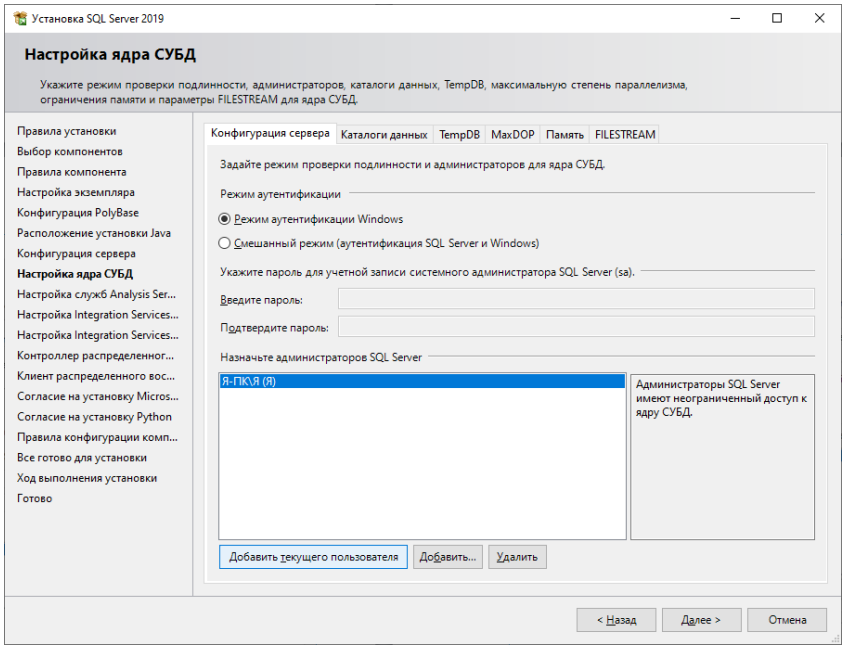

Рисунок 5 – Установка администратора для MS SQL Server 2019

На всех последующих шагах оставим настройки по умолчанию и на самом последнем экране для установки нажмем на кнопку «Установить».

Задания к лабораторной работе №1

Подготовить вычислительную машину для установки. Освободить при необходимости дисковое пространство. Скачать необходимый файл установки. Выполнить установку MS SQL Server скачанной версии.

Лабораторная работа №2
Применение скриптов для инициализации баз данных, создания объектов внутри базы данных

Цель работы:

Теоретические положения

Для подключения к экземпляру SQL Server необходимо запустить SQL Server Management Studio (SSMS), в окне подключения указать имя сервера и параметры аутентификации, затем нажать «Connect».

После успешного подключения в Object Explorer необходимо найти узел «Databases», кликнуть по нему правой кнопкой мыши и выбрать «New Database…». В открывшемся окне вводится имя базы данных в поле «Database name», при необходимости настраиваются параметры файлов (путь хранения, начальный размер, автоматическое изменение размера) на вкладке «Files», а также параметры сортировки на вкладке «Options».

Для того чтобы создать БД с помощью скрипта необходимо использовать инструкцию CREATE DATABASE. Синтаксис инструкции:

```
CREATE DATABASE ИмяБазыДанных
ON
(
    NAME = логическое_имя_файла,
    FILENAME = 'путь_к_файлу.mdf',
    SIZE = размер,
    MAXSIZE = макс_размер,
    FILEGROWTH = прирост
)
LOG ON
(
    NAME = логическое_имя_журнала,
    FILENAME = 'путь_к_файлу.ldf',
    SIZE = размер,
    MAXSIZE = макс_размер,
    FILEGROWTH = прирост
);
```

Пример. Создать БД ShopDB:

```
CREATE DATABASE ShopDB
ON
(
    NAME = ShopDB_Data,
    FILENAME = 'C:\SQLData\ShopDB.mdf',
    SIZE = 100MB,
    MAXSIZE = 2GB,
    FILEGROWTH = 50MB
)
LOG ON
(
    NAME = ShopDB_Log,
    FILENAME = 'C:\SQLLog\ShopDB.ldf',
    SIZE = 50MB,
    MAXSIZE = 1GB,
    FILEGROWTH = 10%
);
```

Создать таблицу в существующей БД можно тремя способами:

• с помощью визуального интерфейса SSMS, выбрав в обозревателе объектов Tables и нажав в контекстном меню New,

• с помощью инструкции на языке T-SQL,

• с помощью графического конструктора SSMS в существующей диаграмме баз данных.

Для того чтобы создать таблицу в БД с помощью скрипта необходимо использовать инструкцию CREATE TABLE. Синтаксис инструкции:

```
CREATE TABLE ИмяТаблицы
(
    Столбец1 [ТипДанных] [ограничение],
    Столбец2 [ТипДанных] [ограничение]
);
```

Базы данных SQL Server работают со следующими типами данных (таблица 1).

Таблица 1 – Типы данных

Тип данных	**Описание**
binary(n)	двоичные данные фиксированной длины до 8000 байт

char(n)	строковый тип данных фиксированной длины без поддержки Unicode длиной до 8000 байтов; данные зависят от установленной кодовой страницы;
Varchar (n)	строковый тип с переменной длиной; если ANSI_PADDING=OFF, то будет выполняться удаление конечных пробелов, если ANSI_PADDING=ON, то удаление пробелов производиться не будет.
Nchar(n)	строковый тип, но с поддержкой Unicode; в этом случае для строковых констант надо задавать впереди букву N: N'ABC'.
Nvarchar (n)	строковый тип, как varchar(n), но с поддержкой Unicode.
Int	целый тип длиной в 8 байт и с диапазоном от -263 до 263-1.
Decimal[(p[,s])]	десятичный двоично-кодированный тип с p десятичными разрядами, из которых s – дробных;
Float[(n)]	плавающий (приблизительный) тип длиной в 4 байта
Datetime	тип данных для хранения даты (4 первых байта) и времени (4 последних байта)
Smalldatetime	тип данных для хранения даты (первых 2 байта) и времени (последние 2 байта)
Money	тип данных для хранения больших денежных величин с точностью до 4 знаков после запятой; для хранения данных отводится 8 байт.
Smallmoney	тип данных для хранения нормальных денежных величин с точностью до 4 знаков после запятой в диапазоне от -214 748.3648 до 214 748.3647; для хранения данных отводится 4 байта.
Bit	битовый (логический) тип со значениями 0 и 1; для хранения выделяется 1 разряд байта памяти.

Типы ограничений (Constraints) для столбцов таблицы в SQL Server:

- NOT NULL (запрещает столбцу содержать NULL-значения)
- UNIQUE (гарантирует уникальность всех значений в столбце)

• PRIMARY KEY (уникально идентифицирует каждую строку в таблице)

• FOREIGN KEY (внешний ключ, обеспечивает ссылочную целостность между таблицами)

• CHECK (определяет условие, которому должны удовлетворять значения столбца)

• DEFAULT (устанавливает значение по умолчанию при вставке новой строки)

• IDENTITY (автоматически генерирует уникальные последовательные числа, обычно используется для первичных ключей)

Пример. Создать таблицу Products в БД ShopDB:

```
CREATE TABLE Products
(
    ProductID INT IDENTITY(1,1)PRIMARY KEY,
    ProductName VARCHAR(100) NOT NULL,
    Description VARCHAR(100) NULL,
    Price SMALLMONEY(10,2) NOT NULL
);
```

Для создания ограничения внешнего ключа между таблицами можно использовать инструкцию ALTER TABLE … ADD CONSTRAINT:

```
ALTER TABLE ИмяДочернейТаблицы
ADD CONSTRAINT ИмяОграничения
    FOREIGN KEY (ВнешнийКлюч)
    REFERENCES ИмяРодительскойТаблицы (ПервичныйКлюч)
    ON DELETE стратегия
    ON UPDATE стратегия;
```

Пример. Создать ограничение внешнего ключа между таблицами Products и TypeProducts:

```
ALTER TABLE Products
ADD CONSTRAINT FK_Products_TypeProducts
    FOREIGN KEY (TypeID)
    REFERENCES TypeProducts(TypeID)
    ON DELETE RESTRICT
    ON UPDATE CASCADE;
```

Задания к лабораторной работе №2

1. Создать БД «Группа_ФИО» с помощью SQL-запроса.

2. Осуществить проектирование БД для информационной системы «Поликлиника».

Описание предметной области: Поликлиника оказывает различные медицинские услуги. Прием пациентов осуществляется врачами строго по талонам. Для врача каждой специальности определен набор талонов, используемый ежедневно. На каждого пациента заводится медицинская карта. Оплата услуги осуществляется после приема и постановки диагноза. Стоимость визита к врачу зависит от категории врача (1-я, 2-я, 3-я) и цели посещения: консультация, обследование, лечение и др. Некоторым пациентам предоставляется скидка на обслуживание. В БД должна храниться информация:

- о ВРАЧАХ: Ф.И.О. врача, специальность, категория;
- ПАЦИЕНТАХ: номер медкарты, Ф.И.О. пациента, дата рождения, адрес, пол, скидка на обслуживание (%);
- ежедневном ПРИЕМЕ пациентов: номер талона на прием к врачу, дата визита, цель посещения, стоимость визита (руб.);
- ДИАГНОЗАХ: код диагноза, наименование диагноза.

При проектировании БД необходимо учитывать следующее: врач осуществляет по талонам ежедневно несколько приемов. Каждый прием осуществляется одним врачом; пациент может приходить на прием к одному врачу несколько раз. На прием по талону приходит только один пациент; один и тот же диагноз может выставляться нескольким пациентам. На одном приеме выставляется один диагноз. Каждый врач обязательно принимает пациентов, которые взяли талон. Каждый прием обязательно осуществляется врачом; каждый пациент обязательно приходит на прием по талону. На каждый прием обязательно приходит пациент; возможный диагноз не обязательно выставляется на приеме (его может не быть у принятых врачом пациентов).

2. Заполнить таблицы тестовыми данными

3. Используя T-SQL

3.1 добавить к таблице Пациент поля, в которых будут храниться номер полиса пациента и название страховой компании;

3.2 добавить в БД новую таблицу с информацией о медицинских осмотрах, которые проходили врачи. При создании таблицы не

забыть создать ограничение первичного и внешнего ключа, задать стратегию обеспечения ссылочной целостности;

3.3 изменить тип данных любого поля в любой таблице, выбрав наиболее оптимальный.

Лабораторная работа №3
Создание пользовательских представлений

Цель работы: получить практические навыки создания и использования представлений для управления правами доступа к объектам БД и обеспечения логической независимости приложений от физической структуры базы данных.

Теоретические положения

Представление (View) – это виртуальная (логическая) таблица в базе данных, содержимое которой определяется результатом выполнения SQL-запроса (SELECT). Представление не хранит данные физически, как обычная таблица, а лишь сохраняет определение (запрос) в базе данных. При обращении к представлению СУБД выполняет сохраненный запрос и возвращает результат.

Представление является важным объектом базы данных и служит для решения следующих задач:

- сокрытие сложности данных (представление может объединять данные из нескольких таблиц с помощью операций соединения, подзапросов и агрегатных функций, предоставляя пользователю простой для восприятия интерфейс);
- обеспечение безопасности (с помощью представлений можно ограничить доступ пользователей только к определенным столбцам или строкам таблиц, скрыв конфиденциальные данные);
- сохранение часто используемых запросов (сложный запрос, который используется в нескольких отчетах или приложениях, можно сохранить как представление и обращаться к нему по имени, как к обычной таблице);
- обеспечение независимости логики приложений от структуры данных (если изменилась физическая структура БД, но представление оставлено прежним, то приложения, работающие через это представление, могут не потребовать изменений).

В большинстве современных СУБД (PostgreSQL, MySQL, Oracle, MS SQL Server) представления можно классифицировать следующим образом:

- простые (статические) представления, создаваемые на основе одного или нескольких запросов SELECT без использования агрегатных функций, GROUP BY, DISTINCT, INNER JOIN (или с ни-

ми, но без возможности модификации данных). Обычно допускают операции модификации данных (INSERT, UPDATE, DELETE) через само представление, если это не противоречит логике (например, все ключевые поля базовых таблиц присутствуют в представлении).

• Сложные (материализованные) представления, которые физически хранят результат запроса на диске. Данные в материализованном представлении необходимо периодически обновлять. Они используются для увеличения скорости выполнения сложных запросов, особенно в хранилищах данных, за счет использования предварительно вычисленных результатов.

Создать представление можно с помощью инструкции CREATE VIEW. Синтаксис инструкции:

```
CREATE [OR REPLACE] VIEW ИмяПредставления [(столбец1, столбец2, ...)]
AS
SELECT_запрос
[WITH [CASCADED | LOCAL] CHECK OPTION];
```

Параметр WITH CHECK OPTION гарантирует, что все команды INSERT и UPDATE, выполненные через представление, будут соответствовать условию WHERE, заданному при определении представления. Это предотвращает «исчезновение» данных из области видимости представления после внесения данных в таблицу, на основе выборки из которой создано данное представление.

Удаление представления осуществляется инструкцией DROP VIEW. Синтаксис инструкции:

```
DROP VIEW [IF EXISTS] ИмяПредставления;
```

Пример. Создание представления с добавлением вычисляемого столбца (например, цена с НДС 20%):

```
CREATE VIEW vw_Products_WithTax
AS
SELECT
    ProductID,
    ProductName,
    Description,
    Price AS BasePrice,
    Price * 1.20 AS PriceWithVAT
FROM Products;
```

Задания к лабораторной работе №3

1. Создать представление, содержащее информацию обо всех врачах, осуществляющих прием 20.09.21.

2 .Создать представление, содержащее информацию обо всех посещениях врачей пациента Иванова А.С.

3. Создать представление, содержащее информацию обо всех услугах, стоимостью >500 руб.

4. Создать представление, содержащее информацию о стоимости приемов (обследований, диагностик и т.д.) конкретного врача.

5. Создать представление, содержащее информацию о выставленных диагнозах и их количестве.

6. Создать представление, содержащее информацию о сумме, заплаченной пациентами за лечение за весь период наблюдения.

7. Создать обновляемое представление, с выборкой информации из 1-й таблицы (любой) согласно заданному условию (условие придумать самостоятельно), без CHECK OPTION. Добавить новую запись в представление, не отвечающую условию. Добавить новую запись в представление, отвечающую условию. Просмотреть представление. Просмотреть таблицу, на основании которой оно было создано.

8. Создать обновляемое представление, с выборкой информации из 1-й таблицы (любой) согласно заданному условию (условие придумать самостоятельно), с CHECK OPTION. Добавить новую запись в представление, не отвечающую условию. Добавить новую запись в представление, отвечающую условию. Просмотреть представление. Просмотреть таблицу, на основании которой оно было создано.

Лабораторная работа №4
Работа с системными представлениями

Цель работы: Изучить архитектуру системных каталогов СУБД и их представлений, получить практические навыки написания запросов к системным представлениям для извлечения информации, научиться использовать динамические административные представления для мониторинга активности сервера.

Теоретические положения

Системные представления (System Views) – это специальные виртуальные таблицы в SQL Server, которые предоставляют доступ к метаданным (данным о данных) и информации о состоянии сервера. Они являются основным интерфейсом для получения служебной информации без необходимости прямого доступа к системным таблицам.

Метаданные – это информация о структуре базы данных (списки таблиц, столбцов, индексов, ограничений, хранимых процедур, пользователей, их прав и т.д.). Доступ к метаданным критически важен для администрирования БД, анализа и документирования структуры БД, отладки и оптимизации запросов, автоматизации задач развертывания и миграции.

SQL Server хранит метаданные в специальных системных таблицах, расположенных в системных базах данных master и msdb. Прямой доступ к ним не рекомендуется. Вместо этого используются три основных набора системных представлений:

• представления схемы sys, которые находятся в каждой базе данных и предоставляют наиболее полный и специфичный для SQL Server доступ к метаданным. Например, информацию обо всех таблицах в текущей базе данных можно узнать из системного представления sys.tables; сведения обо всех столбцах всех таблиц и представлений – sys.columns; информацию обо всех объектах БД – sys.objects; об индексах – sys.indexes; о внешних ключах – sys.foreign_keys.

• представления совместимости INFORMATION_SCHEMA, которые находятся в каждой базе данных и обеспечивают более простой, но менее полный доступ к метаданным. Например, INFORMATION_SCHEMA.TABLES предоставляет информацию о

таблицах; INFORMATION_SCHEMA.COLUMNS – о столбцах; INFORMATION_SCHEMA.VIEWS – о представлениях.

• динамические административные представления (DMV) и функции (DMF), которые находятся в схеме sys с префиксом sys.dm_ и предоставляют информацию о текущем состоянии сервера, его производительности, диагностические данные в режиме реального времени. Например, sys.dm_exec_sessions предоставляет доступ к информации обо всех активных пользовательских и системных сессиях; sys.dm_exec_requests – данные о выполняющихся в данный момент запросах; sys.dm_os_performance_counters – данные по счетчикам производительности Windows; sys.dm_db_index_usage_stats – данные по статистике использования индексов.

Задания к лабораторной работе №4

1. Просмотреть представления системного каталога sys.objects, sys.tables, sys.columns, sys.database_principals, sys.databases, sys.database_files, sys.events.

2. Вывести имя_объекта и дату_последнего_изменения для всех системных базовых таблиц.

3. Вывести имя_объекта и тип_объекта, если объект был создан в период с 01.09.2020 до 30.09.2020.

4. Вывести объекты, у которых владелец НЕ отличается от владельца схемы, т.е. principal_id =Null.

5. Вывести все столбцы пользовательских таблиц, которые не могут принимать значение Null и имя которых начинается на букву K.

6. Вывести идентификатор участника (principal_id) и время_последнего_изменения_участника, если он относится к типу «Пользователь SQL».

7. Вывести логическое имя файла в БД (name), если его размер превышает 5 страниц.

8. Вывести имя БД и статус, в котором она находится.

9. Вывести статус БД и количество БД, которые находятся в данном статусе (Использовать агрегатную функцию Count()).

10. Найти количество объектов БД, которые созданы в 2020 году.

11. Найти суммарный размер всех файлов БД.

Лабораторная работа №5
Язык программирования T-SQL

Цель работы: освоить синтаксис и ключевые возможности языка T-SQL для эффективного выполнения задач по извлечению, модификации, управлению и анализу данных, а также разработать комплекс SQL-скриптов, демонстрирующих применение операторов DML, DDL, процедурного программирования и встроенных функций.

Теоретические положения

Transact-SQL (T-SQL) – это процедурное расширение языка SQL (Structured Query Language), разработанное компанией Microsoft для взаимодействия с реляционными системами управления базами данных (СУБД) семейства Microsoft SQL Server и Azure SQL Database.

В T-SQL переменные используются для временного хранения данных во время выполнения пакета, хранимой процедуры, функции или скрипта. Синтаксис объявления переменной:

```
DECLARE @Имя_переменной Тип_данных
[= Начальное_значение];
```

Ключевые правила объявления переменной: имена переменных всегда начинаются с символа @, можно объявлять несколько переменных в одном операторе DECLARE, переменные являются локальными для пакета, хранимой процедуры или функции, область видимости переменной – с момента объявления до конца пакета или процедуры.

Пример:

```
DECLARE @EmployeeID INT;
DECLARE @EmployeeName NVARCHAR(50);
DECLARE @Counter INT = 0;
```

После объявления переменным можно присваивать значения с помощью операторов SET и SELECT.

Пример:

```
DECLARE @Price DECIMAL(10,2);
SET @Price = 99.99;
DECLARE @MaxSalary DECIMAL(10,2);
SELECT @MaxSalary = MAX(Salary) FROM Employees;
```

Глобальные системные переменные начинаются с @@ и предоставляют системную информацию:

Пример:

```
SELECT @@VERSION;     -- Версия SQL Server
SELECT  @@ROWCOUNT;  --  Количество  обработанных
строк последней инструкцией
SELECT @@ERROR;       -- Код последней ошибки
SELECT @@IDENTITY;    -- Последнее значение IDEN-
TITY
SELECT @@SERVERNAME;  -- Имя сервера
SELECT @@SPID;         -- Идентификатор процесса
```

Для вывода сообщений в окно отладки можно использовать оператор PRINT, а для задания условий – инструкцию IF...ELSE. Синтаксис:

```
IF условие
    оператор
ELSE
    оператор
```

Пример:

```
DECLARE @Salary DECIMAL(10,2) = 75000;
IF @Salary > 100000
    PRINT 'Высокая зарплата';
ELSE
    PRINT 'Средняя или низкая зарплата';
```

В T-SQL циклы являются процедурным расширением, противоречащим изначальной декларативной природе SQL. Однако они необходимы для обработки наборов данных, где нельзя использовать SELECT-операции, реализации сложной бизнес-логики, требующей итеративного подхода.

В T-SQL существует только один тип цикла WHILE, имеющий следующий синтаксис:

```
WHILE логическое_выражение
    { оператор | блок_операторов }
```

Пример цикла с условием:

```
DECLARE @Counter INT = 1;
WHILE @Counter <= 10
BEGIN
    PRINT 'Итерация: ' + CAST(@Counter AS VAR-
CHAR);
```

```
    SET @Counter = @Counter + 1;
END
```

Для управления выполнением циклов используются операторы BREAK (немедленный выход из цикла) и CONTINUE (переход к следующей итерации).

Пример. Обновление поля LastUpdated в таблице Employees для всех сотрудников, работающих в 1-м отделе с выводом соответствующего сообщения в окно отладки:

```
DECLARE @MinID INT, @MaxID INT;
-- Определение диапазона
SELECT @MinID = MIN(EmployeeID), @MaxID =
MAX(EmployeeID)
FROM Employees
WHERE DepartmentID = 1;
DECLARE @CurrentID INT = @MinID;
WHILE @CurrentID <= @MaxID
BEGIN
    -- Проверка существования записи
    IF EXISTS(SELECT 1 FROM Employees
              WHERE EmployeeID = @CurrentID AND
DepartmentID = 1)
    BEGIN
        -- Обработка записи
        UPDATE Employees
        SET LastUpdated = GETDATE()
        WHERE EmployeeID = @CurrentID;
        PRINT 'Обновлен сотрудник ID: ' +
CAST(@CurrentID AS VARCHAR);
    END

    SET @CurrentID = @CurrentID + 1;
END
```

Задание к лабораторной работе №5

1. Определить и вывести имя локального SQL сервера и номер версии.
2. Определить количество приемов в определенный день и вывести сообщение, превышает ли их количество 30 шт. или нет.
3. Определить услугу, продажи по которой самые маленькие. Снизить цену найденной услуги на 10%.

4. Приостановить процесс выполнения кода до указанного времени.

5. Определить количество врачей каждой категории и вывести категорию, количество врачей в которой максимально.

6. Если пациент в течение текущего месяца оплатил услуг, больше, чем 20000, то предоставить ему скидку 5%, если сумма оплат >10000, но <=20000, то предоставить скидку 3%, если <=10000, то скидку не предоставлять.

Примечание: При выполнении заданий можно использовать циклы и условия, курсоры использовать нельзя.

Лабораторная работа №6
Пользовательские функции

Цель работы: получить практические навыки проектирования и реализации пользовательских функций для инкапсуляции бизнес-логики, упрощения сложных вычислений и создания повторно используемых компонентов в запросах.

Теоретические положения

Пользовательская функция (User-Defined Function, UDF) – это программный объект базы данных, который инкапсулирует последовательность операторов SQL для выполнения определенной задачи и возвращает результат (скалярное значение или таблицу). Функции в СУБД аналогичны функциям в процедурных языках программирования: они принимают входные параметры, выполняют вычисления или операции с данными и возвращают результат.

Основные характеристики функций:

- могут иметь входные параметры (от 0 до 1024);
- всегда возвращают результат (в отличие от хранимых процедур);
- могут быть использованы в выражениях в инструкциях SELECT, WHERE, ORDER BY, JOIN и других;
- не могут изменять состояние базы данных (запрещены операции INSERT, UPDATE, DELETE, DDL-операции, кроме изменений временных таблиц);
- не могут вызывать хранимые процедуры (кроме расширенных хранимых процедур);
- не могут управлять транзакциями.

В SQL Server существуют следующие типы функций:

- Скалярные функции (Scalar-Valued Functions). Скалярные функции возвращают одно значение определенного типа данных (INT, VARCHAR, DATETIME и т.д.), могут содержать сложную логику с использованием переменных, операторов управления потоком (IF, WHILE), вызываются в любом месте, где допустимо выражение. Синтаксис скалярной функции:

```
CREATE FUNCTION [schema_name.]function_name
(
    @parameter1 data_type [ = default ],
    @parameter2 data_type [ = default ]
```

```
)
RETURNS return_data_type
[WITH { ENCRYPTION | SCHEMABINDING | RETURNS NULL
ON NULL INPUT | CALLED ON NULL INPUT }]
[AS]
BEGIN
    -- Тело функции
    DECLARE @result return_data_type;
    -- Логика вычислений
    RETURN @result;
END;
```

- Табличные функции (Table-Valued Functions)

✓ Встроенные табличные функции (Inline Table-Valued Functions) возвращают таблицу как результат выполнения одного оператора SELECT. Данные функции по сути являются параметризированными представлениями. Оптимизатор может «встраивать» их в основной запрос, что обеспечивает высокую производительность. Синтаксис:

```
CREATE FUNCTION [schema_name.]function_name
(
    @parameter1 data_type [ = default ],
    @parameter2 data_type [ = default ]
)
RETURNS TABLE
[WITH { ENCRYPTION | SCHEMABINDING }]
[AS]
RETURN
(
    SELECT ... -- Один SELECT-запрос
);
```

✓ Многооператорные табличные функции (Multi-Statement Table-Valued Functions) возвращают таблицу, структура которой объявляется явно. Могут содержать сложную процедурную логику с несколькими операторами. Менее производительны, чем встроенные функции, но более гибкие. Синтаксис:

```
CREATE FUNCTION [schema_name.]function_name
(
    @parameter1 data_type [ = default ],
    @parameter2 data_type [ = default ]
```

```
)
RETURNS @return_table TABLE
(
    column1 data_type,
    column2 data_type,
    ...
)
[WITH { ENCRYPTION | SCHEMABINDING }]
[AS]
BEGIN
    -- Логика с несколькими операторами
    INSERT INTO @return_table SELECT ...;
    -- Дополнительные операции
    RETURN;
END;
```

Функции можно разделить на детерминированные и недетерминированные. Детерминированная функция всегда возвращает одинаковый результат для одних и тех же входных параметров и состояния базы данных (например, функция для расчета площади круга по радиусу).

Недетерминированная функция может возвращать разные результаты при одинаковых параметрах (например, функция, использующая GETDATE() или RAND()).

Детерминированные функции могут быть использованы в индексированных вычисляемых столбцах и представлениях с индексами.

Вызов пользовательских функций так же, как и вызов встроенных функций.

Пример. Вызов скалярной функции:

```
SELECT dbo.CalculateDiscount(Price, Quantity) AS
Discount FROM Orders;
```

Пример. Вызов табличных функций:

```
SELECT * FROM
dbo.GetEmployeeSubordinates(@manager_id);
```

Задание к лабораторной работе №5

1. Создать функцию расчета стоимости визита, которая принимает на вход базовую стоимость услуги, категорию врача, скидку пациента в процентах. Категория врача увеличивает базовую стои-

мость приема: для 1-й категории базовая стоимость увеличивается на 20%, для 2-й категории – на 10%, для 3-й категории остается без изменений. Возвращаемое значение – окончательная стоимость визита.

2. Создать функцию определения возраста пациента на дату визита, которая принимает на вход дату рождения пациента и дату приема. Возвращаемое значение – возраст пациента в полных годах.

3. Создать функцию для проверки корректности талона, существует ли талон, не использован ли он ранее. Возвращаемое значение – TRUE, если талон действителен, FALSE в противном случае.

4. Создать функцию для проверки допустимости скидки, которая принимает на вход процент скидки и возвращает TRUE, если скидка в диапазоне 0-100%, FALSE, если значение некорректно.

5. Создать функцию, которая принимает на вход идентификатора врача и период времени. Функция возвращает в табличном виде количество приемов за период, суммарную стоимость всех приемов с учетом скидок, наиболее частый диагноз за период.

6. Создать функцию для формирования списка пациентов со скидкой.

Лабораторная работа №7
Пользовательские хранимые процедуры

Цель работы: Изучение теоретических основ, практических возможностей и преимуществ использования хранимых процедур для эффективной разработки и администрирования баз данных.

Теоретические положения

Хранимая процедура – это именованный программный модуль (набор предварительно скомпилированных инструкций T-SQL), хранящийся на сервере базы данных. Он предназначен для выполнения определенной логической последовательности действий с данными. Процедура инкапсулирует бизнес-логику, что позволяет обращаться к ней многократно из различных клиентских приложений. Это один из ключевых объектов программирования на стороне сервера.

Хранимые процедуры бывают системными, пользовательскими, временными, расширенными, CLR-процедурами. Пользовательские процедуры создаются разработчиками для инкапсуляции бизнес-логики на стороне сервера.

Базовый синтаксис создания пользовательской хранимой процедуры:

```
CREATE PROCEDURE [schema_name.]procedure_name
@parameter1 data_type [= default_value] [OUTPUT],
@parameter2 data_type [= default_value] [OUTPUT],
...
AS
BEGIN
    -- Тело процедуры
    SET NOCOUNT ON;
    -- SQL-операции
    -- Логика обработки
END
```

Параметры @parameter1, @parameter2 и т.д., передаваемые в хранимую процедуру, могут быть следующих видов:

• входным параметром (по умолчанию), например, `@CustomerID INT;`

• входным параметром со значением по умолчанию, напри-

мер, `@StatusID INT = 1;`

• выходным параметром, например, `@TotalAmount MONEY OUTPUT`.

Хранимые процедуры могут возвращать:

• статус завершения процедуры, например, `RETURN @@ERROR;`

• скалярные значения, вычисленные в процедуре, например, `SELECT @Result = COUNT(*) FROM Orders;`

• результирующие наборы данных, например, `SELECT * FROM Products WHERE CategoryID = @CategoryID;`

• сообщения, например, `RAISERROR ('Ошибка обработки', 16, 1).`

В хранимых процедурах можно осуществлять обработку ошибок с помощью блока инструкций TRY…CATCH. Например:

```
BEGIN TRY
    -- Код, который может вызвать ошибку
END TRY
BEGIN CATCH
    -- Обработка ошибки
    SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

Для прохождения в цикле по результирующему набору строк запроса можно использовать курсоры. Курсор – это особый временный объект SQL, позволяющий по отдельности считывать и обрабатывать каждую его строку. Упрощенный синтаксис объявления курсора:

```
DECLARE cursor_name CURSOR FOR select_statement;
```

Для использования курсора необходимо его открыть с помощью инструкции OPEN:

```
OPEN cursor_name;
```

Для чтения данных из курсора используется оператор FETCH. Упрощенный синтаксис оператора:

```
FETCH NEXT FROM cursor_name
INTO @variable1, @variable2, … ,@variableN
```

Закрытие открытого курсора и удаление ссылки курсора осуществляется с помощью операторов CLOSE и DEALLOCATE:

```
CLOSE cursor_name
DEALLOCATE cursor_name
```

Пример. Считать построчно записи из таблицы Person, осуществить обработку считанных данных:

```
DECLARE @LastName VARCHAR(50), @FirstName VARCHAR(50);
DECLARE my_cur CURSOR FOR SELECT LastName, FirstName FROM Person
OPEN my_cur
FETCH NEXT FROM my_cur INTO @LastName, @FirstName
WHILE @@FETCH_STATUS = 0
BEGIN
exec dbo.my_proc @LastName, @FirstName
PRINT 'Contact Name: ' + @FirstName + ' ' + @LastName
FETCH NEXT FROM my_cur INTO @LastName, @FirstName
END
CLOSE my_cur
DEALLOCATE my_cur
```

Задание к лабораторной работе №7

1. Создать хранимую процедуру, в которую в качестве значения параметра передаётся фамилия_врача. Найти все свободные талоны этого врача.
2. Создать хранимую процедуру, которой в качестве значения параметра передаётся месяц N. Для каждого врача рассчитать сумму услуг, оказанных в месяце N текущего года. Информацию записать в итоговую таблицу. Если врач в указанном месяце услуг не оказывал, то записать NULL.
3. Создать хранимую процедуру, которой в качестве значения параметра передаётся фамилия_врача, дата, время_начала_работы, количество_талонов, которые необходимо сгенерировать. Для заданного врача добавить указанное количество талонов. Время каждого приема – 15мин для врачей 1 категории, 10 – для врачей остальных категорий.

Лабораторная работа №8
Системные хранимые процедуры

Цель работы: изучить возможности системных хранимых процедур СУБД для эффективного администрирования базы данных, автоматизации рутинных задач диагностики, мониторинга производительности и управления объектами сервера.

Теоретические положения

Системные хранимые процедуры – это предопределенные процедуры, входящие в состав Microsoft SQL Server, которые инкапсулируют сложные операции администрирования и предоставляют интерфейс для доступа к системным каталогам и метаданным. Они устанавливаются вместе с СУБД и располагаются в системных базах данных, преимущественно в master и msdb.

Ключевые характеристики: начинаются с префикса sp_, физически хранятся в скрытой системной базе данных resource, логически отображаются в схеме sys каждой пользовательской и системной базы данных, выполняются в контексте текущей базы данных, но могут обращаться к системным объектам.

Перечень наиболее часто используемых процедур приведен в таблице 2.

Таблица 2 – Системные хранимые процедуры

Категория	Примеры процедур	Назначение
Управление безопасностью	sp_addlogin, sp_grantdbaccess, sp_addrole	Управление входами, пользователями, ролями и разрешениями
Администрирование баз данных	sp_attach_db, sp_detach_db, sp_renamedb	Создание, изменение, удаление баз данных
Мониторинг и диагностика	sp_who, sp_who2, sp_lock, sp_spaceused	Анализ активности, блокировок, использования пространства
Управление заданиями агента	sp_add_job, sp_add_jobstep, sp_start_job	Автоматизация задач через SQL Server Agent
Работа с метаданными	sp_tables, sp_columns, sp_help, sp_helpconstraint	Получение информации об объектах базы данных
Репликация	sp_addsubscription, sp_replcmds	Управление репликацией данных
Управление индексами	sp_helpindex, sp_statistics	Информация об индексах и статистике

Задание к лабораторной работе №8

1. Изучить раздел справочного руководства по системным хранимым процедурам SQL Server https://docs.microsoft.com/ru-ru/sql/relational-databases/system-stored-procedures/system-stored-procedures-transact-sql?view=sql-server-ver15
2. Получить информацию об указанной базе данных (sp_helpdb)
3. Получить информацию об указанном объекте БД (sp_help)
4. Получить информацию обо всех ограничениях объекта БД (sp_helpconstraint)
5. Удалить остаточные данные, оставленные на страницах базы данных процедурами изменения данных sp_clean_db_free_space
6. Удалить записи резервных наборов данных, которые старше указанной даты, дату задать самостоятельно (sp_delete_backuphistory)
7. Отобразить все глобальные параметры конфигурации текущего сервера (sp_configure). Отобразить выбранные параметры конфигурации сервера (default language, Database Mail XPs, query wait)
8. Отобразить статистику о Microsoft SQL Server (sp_monitor)
9. Используя выходные данные процедуры sp_server_diagnostics получить диагностические данные и сведения о работоспособности SQL Server
10. Определить место на диске, зарезервированное и используемое указанной таблицей (sp_spaceused)
11. Получить сведения обо всех текущих пользователях, об отдельном текущем пользователе по имени его входа, об активных процессах (sp_who)

Лабораторная работа №9
Триггеры

Цель работы: комплексное исследование триггеров как механизма автоматического реагирования на события в СУБД.

Теоретические положения

Триггер – это специальный тип хранимой процедуры в базе данных, которая автоматически выполняется в ответ на определенное событие, происходящее с объектами, размещенными на сервере, или с самим сервером.

Различают DML-триггеры (реагируют на изменение данных в таблицах и представлениях БД), DDL-триггер (реагируют на изменение схемы БД), LOGON-триггер (обработка событий входа на целевой сервер).

По времени срабатывания различают триггеры, срабатывающие после выполнения операции (AFTER/FOR) и триггеры, срабатывающие вместо операции (INSTEAD OF).

Синтаксис для создания триггера:

```
CREATE TRIGGER [schema_name.]trigger_name
ON { table | view }
{ FOR | AFTER | INSTEAD OF }
{ [INSERT] [,] [UPDATE] [,] [DELETE] }
AS
{ sql_statements }
```

При срабатывании триггера создаются виртуальные таблицы inserted и deleted. При выполнении операции INSERT создается таблица inserted, при выполнении операции DELETE создается таблица deleted, при выполнении операции UPDATE создаются обе таблицы – deleted, содержащая старые значения, и inserted, содержащая новые значения.

Пример. Создать триггер, срабатывающий при добавлении или изменении данных в таблице Employees. Триггер будет запрещать добавлять данные, если зарплата сотрудника превышает максимальное значение, заданное в таблице Positions:

```
CREATE TRIGGER trg_CheckSalary
ON Employees
AFTER INSERT, UPDATE
AS
```

```
BEGIN
    IF EXISTS (
        SELECT * FROM inserted i
        WHERE i.Salary > (SELECT MAX_Salary FROM
Positions WHERE PositionID = i.PositionID)
    )
    BEGIN
        RAISERROR('Зарплата превышает максималь-
ное значение', 16, 1)
        ROLLBACK TRANSACTION
    END
END
```

Задание к лабораторной работе №9

1. Создать триггер DML, проверяющий наличие диагоза при добавлении записи в таблицу Прием. При ошибке должно выводиться сообщение «Введен неверный номер диагноза». При создании триггера использовать конструкцию INSTEAD OF, иначе сработают ограничения таблиц.
2. Создать триггер DML: Зарплата врачей должна попадать в один из интервалов значений зарплаты, которые установлены для разных категорий сотрудников. Например, для 1 категории – не менее 30000, для 2 категории – не менее 20000, для 3-й категории – не менее 15000. Примечание: необходимо наличие таблицы Зарплата. Примерный перечень полей: Код(PK), НомерВрача(FK), ДатаНачисления, Сумма. При неверном вводе пользователю должно выводиться сообщение об ошибке.
3. Создать триггер DML: При операциях вставки и обновления А) присваивать полю значение по умолчанию в зависимости от значения других полей;
4. Б) переводить первую букву фамилии в верхний регистр.
5. Создать триггер, который будет разрешать изменение некоторого столбца таблицы всем, кроме владельца dbo.
6. Создать таблицу Аудит(Код, пользователь, время, операция), в которую записывать время доступа к конкретной таблице (на выбор) конкретного пользователя, название производимой им операции – U,D или I).
7. Создать таблицу Аудит_XML(Код, имяТаблицы, операция, стараяЗапись, новаяЗапись), в которую в формате XML записывать

все изменения, производимые в таблице с помощью операции Update.

Лабораторная работа №10
Транзакции

Цель работы: приобрести практические навыки работы с транзакциями в СУБД, изучить их свойства (ACID), механизмы управления и применение для обеспечения целостности данных в многопользовательской среде.

Теоретические положения

Транзакция – это логическая единица работы с базой данных, которая представляет собой последовательность операций, выполняемых как единое целое. Транзакция либо выполняется полностью, либо не выполняется вовсе.

Ключевые характеристики (свойства ACID): атомарность (все операции транзакции выполняются как одно целое), изолированность (транзакции не влияют друг на друга), согласованность (база данных переходит из одного согласованного состояния в другое) и долговечность (результаты успешно завершенной транзакции сохраняются в БД и становятся ее частью).

Основные команды для управления транзакциями:

`BEGIN TRANSACTION [transaction_name]` – начало транзакции, `COMMIT [TRANSACTION]` – подтверждение изменений, `ROLLBACK [TRANSACTION]` – откат изменений, `SAVE TRANSACTION savepoint_name` – точка сохранения транзакции. Пример:

```
BEGIN TRANSACTION
INSERT INTO Orders (CustomerID, OrderDate) VALUES
(1, GETDATE())
SAVE TRANSACTION AfterOrderInsert
UPDATE Inventory SET Quantity = Quantity - 1
WHERE ProductID = 100
IF @@ERROR <> 0
BEGIN
ROLLBACK TRANSACTION AfterOrderInsert  -- Откат к
точке сохранения
-- Заказ сохранен, но инвентарь не обновлен
END
ELSE
```

```
BEGIN
COMMIT  -- Подтверждение всех изменений
END
```

При параллельном доступе к данным возможны такие аномалии, как «грязное чтение» (чтение незафиксированных данных другой транзакцией), «неповторяемое чтение» (получение разных значений при повторном чтении данных), «фантомное чтение» (появление новых строк данных при повторном запросе).

Уровни изоляции транзакций в СУБД (таблица 3) – это настройка, определяющая, насколько транзакции изолированы друг от друга и какие аномалии параллельного доступа допускаются или предотвращаются.

Таблица 3 – Уровни изоляции транзакций

Уровень изоляции	**Грязное чтение**	**Неповторяемое чтение**	**Фантомное чтение**
READ UNCOMMITTED	Разрешено	Возможно	Возможно
READ COMMITTED (по умолчанию)	Запрещено	Возможно	Возможно
REPEATABLE READ	Запрещено	Запрещено	Возможно
SERIALIZABLE	Запрещено	Запрещено	Запрещено
SNAPSHOT	Запрещено	Запрещено	Запрещено

Установка уровня изоляции осуществляется с помощью инструкции `SET TRANSACTION ISOLATION LEVEL уровень`.

Пример:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
    -- Операции с данным уровнем изоляции
COMMIT
```

Задание к лабораторной работе №10

1. Создайте в новом окне любой запрос (Запрос1). Установите уровень изоляции транзакций в Запросе1 на 1. Начните транзакцию, в которой выполняется изменение данных и просмотр сделанных изменений.

Например:

```
BEGIN TRANSACTION
SELECT * FROM Employee
UPDATE Employee SET TITLE ='Тестовый заголовок'
```

```
WHERE EmployeeId=150
SELECT * FROM Employee
```

Данные были обновлены, но изменения видны только в текущем соединении.

2. Создайте в новом окне запрос (Запрос2). Установите уровень изоляции транзакций в Запросе2 на 1. Просмотрите данные объекта, изменяемого в запросе1:

Например: `SELECT * FROM Employee`

Запрос не вернет результатов, поскольку он ждет завершения транзакции, запущенной в Запросе1. Вернитесь в Запрос1, закомментируйте все строки (для того, чтобы не начать еще одну транзакцию), введите и выполните: COMMIT TRANSACTION . Перейдите в Запрос2. Запрос2 вернет результат.

Внимание! Количество BEGIN TRANSACTION должно быть равно количеству COMMIT+ROLLBACK. Каждое начало транзакции увеличивает значение переменной @@TRANCOUNT на 1.

3. Измените уровень изоляции транзакций в Запросе1 и Запросе2 на 0. Проделайте задания 1-2 повторно. В чем разница?

Создайте в новом окне любой запрос (Запрос3). Установите уровень изоляции транзакций в Запросе3 на 1. Начните транзакцию, в которой выполняется обновление данных и просмотр сделанных изменений.

Например:

```
BEGIN TRANSACTION
SELECT * FROM Employee
UPDATE Employee SET LastName='Иванов' WHERE Em-
ployeeId=12
SELECT * FROM Employee
```

Данные были обновлены, но изменения видны только в текущем соединении.

5. Создайте в новом окне запрос (Запрос4). Установите уровень изоляции транзакций в Запросе4 на 1. Обновите ту же запись, что и в задании 4:

Например: `UPDATE Employee SET LastName='Сидоров' WHERE EmployeeId=12`

Запрос не выполнится, поскольку он ждет завершения транзакции, запущенной в Запросе3. Вернитесь в Запрос3, закомментируйте все строки (для того, чтобы не начать еще одну транзакцию),

введите и выполните: ROLLBACK TRANSACTION. Перейдите в Запрос4. Запрос4 выполнится.

6. Измените уровень изоляции транзакций в Запросе3 и Запросе4 на 0. Проделайте задания 4-5 повторно. Есть разница? Почему?

7. Используя любые объекты БД, создайте транзакцию, произведите ее откат и фиксацию. Покажите, что данные существовали до отката (оператор SELECT), удалились после отката, снова были изменены, и затем были успешно зафиксированы.

8. Создайте транзакцию с обработкой ошибок.

9. Создайте транзакцию с точкой сохранения. Произведите откат до точки сохранения. Затем либо зафиксируйте внесенные изменения (COMMIT TRANSACTION), либо откатите их (ROLLBACK TRANSACTION).

Лабораторная работа №11
Работа с журналом транзакций

Цель работы: исследовать архитектуру и механизмы работы журнала транзакций в СУБД, изучить его роль в обеспечении целостности данных, восстановлении после сбоев, а также освоить практические навыки управления и мониторинга журнала транзакций.

Теоретические положения

Журнал транзакций – это специальный файл (или набор файлов) в СУБД, который хранит последовательную запись всех изменений, произведенных в базе данных (рис. 6). Это ключевой механизм обеспечения надежности и целостности данных.

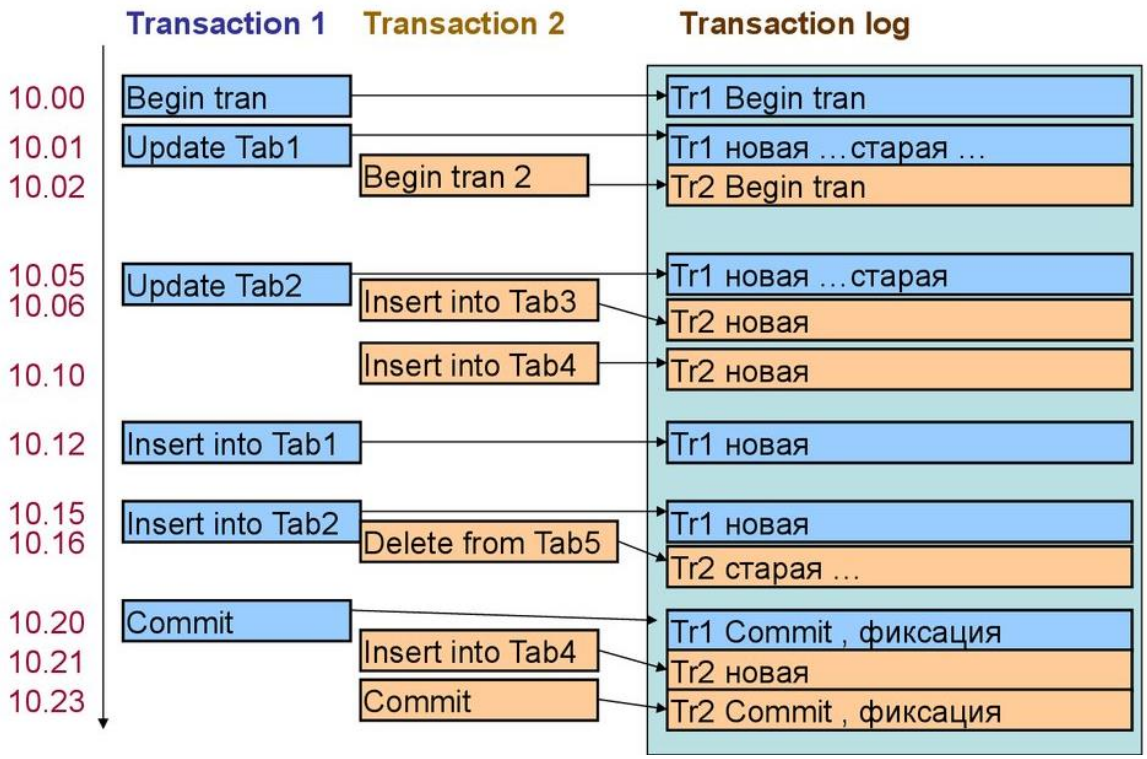

Рисунок 6 – Фиксация всех изменений в журнале транзакций

Основные функции журнала транзакций: обеспечение атомарности и долговечности транзакций, восстановление после сбоев, поддержка репликации и зеркалирования баз данных, возможность восстановления на указанный момент времени (Point-in-Time Recovery).

Логически журнал транзакций SQL Server работает так, как ес-

ли бы он являлся последовательностью записей в журнале. Каждая запись журнала идентифицируется регистрационным номером транзакции (log sequence number, LSN). Каждая новая запись добавляется в логический конец журнала с номером LSN, который больше номера LSN предыдущей записи. Каждая запись журнала содержит идентификатор транзакции, тип операции, измененный объект, его старое и новое состояния (или описание операции). Все записи журнала, связанные с определенной транзакцией, с помощью обратных указателей связаны в цепочку, которая предназначена для ускорения отката транзакции.

На физическом уровне журнал транзакций разбит на части, называемые виртуальными журналами (virtual log files, VLF), которые используются для оптимизации внутреннего управления журналом транзакций и не имеют фиксированных размеров. Когда файл VLF заполняется, процедура ведения журнала автоматически переходит к следующему VLF в журнале транзакций. В дальнейшем, ранее занятые VLF могут не понадобиться, т.к. описанные в них транзакции завершены. Поэтому, когда процедура ведения журнала достигает конца журнала транзакций, она снова заворачивает в начало и начинает писать в свободные VLF поверх того, что было там ранее (рис. 7).

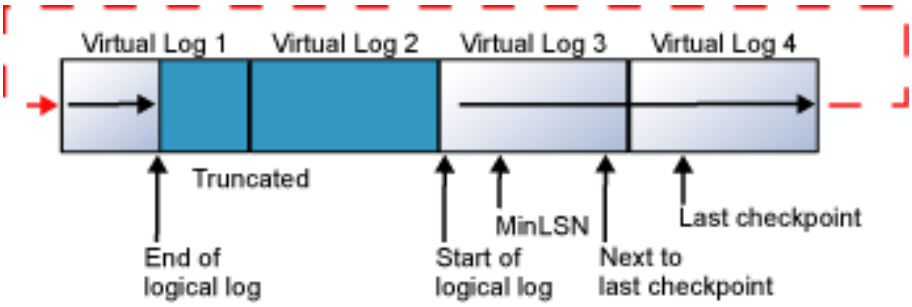

Рисунок 7 – Пример физической архитектуры журнала транзакций

Для просмотра журнала транзакций можно использовать функцию fn_dump_dblog, при запуске которой необходимо указать все ее 64 параметра, в том числе абсолютный путь к файлу журнала транзакций:

```
SELECT * FROM fn_dump_dblog
(NULL,NULL,N'DISK',1,N'E:\ApexSQL\backups\Adventu
reWorks2012_05222013.trn', DEFAULT, DEFAULT, DE-
FAULT, DEFAULT, DEFAULT, DEFAULT, DEFAULT, DE-
```

```
FAULT,DEFAULT,      DEFAULT,      DEFAULT,      DE-
FAULT,DEFAULT,DEFAULT,DEFAULT,DEFAULT,DEFAULT,DEF
AULT,   DEFAULT,   DEFAULT,   DEFAULT,   DEFAULT,
DEFAULT,DEFAULT,DEFAULT,DEFAULT,          DEFAULT,
DEFAULT, DEFAULT, DEFAULT, DEFAULT, DEFAULT, DE-
FAULT,    DEFAULT,    DEFAULT,DEFAULT,    DEFAULT,
DEFAULT,DEFAULT, DEFAULT, DEFAULT, DEFAULT, DE-
FAULT,    DEFAULT,    DEFAULT,    DEFAULT,    DE-
DE-
FAULT,DEFAULT,DEFAULT,DEFAULT,DEFAULT,DEFAULT,DEF
AULT,DEFAULT,  DEFAULT,  DEFAULT,  DEFAULT,  DE-
FAULT,DEFAULT,DEFAULT,      DEFAULT,      DEFAULT,
DEFAULT);
```

Данная функция выводит все записи из журнала транзакций (рис. 8).

	Current LSN	Operation	Context	Transaction ID	transaction name	Description
1	0000003c:00000082:0001	LOP_BEGIN_XACT	LCX_NULL	0000:00001abb	AutoCreateQPStats	AutoCreateQPStats;0x0105000000000005150000036c005a231a6cc79...
2	0000003c:00000085:0001	LOP_BEGIN_XACT	LCX_NULL	0000:00001abc	AutoCreateQPStats	AutoCreateQPStats;0x0105000000000005150000036c005a231a6cc79...
3	0000003c:0000007f:0001	LOP_BEGIN_XACT	LCX_NULL	0000:00001aba	AutoCreateQPStats	AutoCreateQPStats;0x0105000000000005150000036c005a231a6cc79...
4	0000003c:0000007c:0001	LOP_BEGIN_XACT	LCX_NULL	0000:00001ab9	AutoCreateQPStats	AutoCreateQPStats;0x0105000000000005150000036c005a231a6cc79...
5	0000003c:0000006e:0001	LOP_BEGIN_XACT	LCX_NULL	0000:00001ab8	DELETE	DELETE;0x0105000000000005150000036c005a231a6cc79bb4acc6d...
6	0000003c:00000025:0087	LOP_BEGIN_XACT	LCX_NULL	0000:00001ab7	SplitPage	SplitPage;0x0105000000000005150000036c005a231a6cc79bb4acc6...
7	0000003c:00000025:006e	LOP_BEGIN_XACT	LCX_NULL	0000:00001ab6	SplitPage	SplitPage;0x0105000000000005150000036c005a231a6cc79bb4acc6...
8	0000003c:00000025:0057	LOP_BEGIN_XACT	LCX_NULL	0000:00001ab5	SplitPage	SplitPage;0x0105000000000005150000036c005a231a6cc79bb4acc6...
9	0000003c:00000025:0036	LOP_BEGIN_XACT	LCX_NULL	0000:00001ab4	SplitPage	SplitPage;0x0105000000000005150000036c005a231a6cc79bb4acc6...
10	0000003c:00000025:0021	LOP_BEGIN_XACT	LCX_NULL	0000:00001ab3	BTree Split/Shrink	BTree Split/Shrink;0x0105000000000005150000036c005a231a6cc79...

Рисунок 8 – Пример вывода функции fn_dump_dblog

Задание к лабораторной работе №11

1. Ознакомиться с теоретическим материалом.
2. Открыть и просмотреть файл журнала транзакций *.LDF или файл резервной копии журнала *.TRN в любом двоичном редакторе.
3. При наличии прав просмотреть активную часть журнала транзакций с помощью функции sys.fn_dblog.
4. При наличии прав просмотреть журнал транзакций из резервной копии журнала транзакций с помощью функции fn_dump_dblog.
5. Используя sys.dm_db_log_space_usage получить сведения об используемом сейчас журналом объеме пространства и о необходимости его усечения.
6. Используя sys.database_files получить сведения о текущем размере файла журнала, его максимальном размере и параметре автоматического увеличения файла (столбцы size, max_size и growth).

7. Произвести усечение и сжатие самой БД и ее журнала транзакций с помощью SSMS.

Лабораторная работа №12
Создание пользователей и назначение ролей

Цель работы: приобрести практические навыки управления безопасностью в СУБД, изучить механизмы аутентификации и авторизации, освоить создание пользователей, назначение ролей и управление разрешениями для обеспечения безопасного доступа к данным.

Теоретические положения

В Microsoft SQL Server, как и в других современных СУБД, реализована многоуровневая модель безопасности, которая работает по принципу иерархии субъектов безопасности (security principals). На верхнем уровне находятся имена входа (логины, logins) – это учетные записи на уровне экземпляра SQL Server, которые позволяют подключиться к серверу. Логины бывают двух основных типов: логины с аутентификацией Windows (когда SQL Server доверяет проверке подлинности, выполненной операционной системой) и логины с аутентификацией SQL Server (когда проверка выполняется самим SQL Server с использованием имени пользователя и пароля).

Логины с аутентификацией SQL Server создаются с помощью инструкции CREATE LOGIN:

```
CREATE LOGIN login_name { WITH <option_list1> |
FROM <sources> }
```

Пример:

```
CREATE LOGIN [TestLogin]
WITH PASSWORD=N'Pa$$w0rd',
DEFAULT_DATABASE=[Test],
DEFAULT_LANGUAGE=[русский],
CHECK_EXPIRATION=OFF,  CHECK_POLICY=ON
```

После успешной аутентификации на уровне сервера пользователь получает доступ к экземпляру SQL Server, но это еще не означает доступ к конкретной базе данных. Для работы с базой данных необходимо иметь соответствующего пользователя (user) внутри этой базы. Пользователь базы данных – это второй уровень безопасности, который связывается с логином через специальное сопоставление. Один логин может быть сопоставлен с пользователями в разных базах данных, но в каждой базе может быть только один пользователь, связанный с конкретным логином. Это разделение

логинов и пользователей позволяет гибко управлять правами: один и тот же человек (логин) может иметь разные права в разных базах данных в зависимости от того, каким пользователем он в них представлен.

Пользователи базы данных создаются с помощью инструкции CREATE USER:

```
CREATE USER user_name
[{FOR | FROM } LOGIN login_name]
    [ WITH <limited_options_list> [ ,... ] ]
[ ; ]
```

Пример:

```
CREATE USER [TestLogin] FOR LOGIN [TestLogin]
WITH DEFAULT_SCHEMA=[dbo]
```

Ключевым механизмом управления правами в SQL Server является система ролей. Роли – это именованные группы разрешений, которые можно назначать пользователям. Существуют роли уровня сервера (server roles) и роли уровня базы данных (database roles).

Предопределенные серверные роли, такие как sysadmin, serveradmin, securityadmin, предоставляют административные привилегии на уровне всего экземпляра SQL Server. Например, член роли sysadmin имеет неограниченные права и может выполнять любые действия на сервере.

Определяемые пользователем роли сервера впервые стали применяться в SQL Server 2012. Для создания и удаления этих ролей используются инструкции языка Transact-SQL CREATE SERVER ROLE и DROP SERVER ROLE соответственно, например:

```
CREATE SERVER ROLE program_admin;
```

Для добавления или удаления членов роли сервера используется инструкция ALTER SERVER ROLE, например:

```
ALTER SERVER ROLE program_admin ADD MEMBER Anna;
```

Роли базы данных, в свою очередь, делятся на предопределенные (fixed database roles) и пользовательские (user-defined database roles). Предопределенные роли базы данных включают в себя роль db_owner (полный контроль над базой данных), db_datareader (чтение всех данных), db_datawriter (изменение всех данных), db_ddladmin (выполнение DDL-операций) и другие. Эти роли охватывают наиболее распространенные сценарии доступа к БД.

Для создания новой определяемой пользователем роли базы данных в текущей базе данных применяется инструкция CREATE

ROLE. Синтаксис:

```
CREATE ROLE role_name [AUTHORIZATION owner_name]
```

Для изменения имени определяемой пользователем роли базы данных применяется инструкция ALTER ROLE, а для удаления роли из базы данных – инструкция DROP ROLE. Например:

```
CREATE ROLE marketing AUTHORIZATION Vasya;
ALTER ROLE marketing ADD MEMBER 'Vasya';
ALTER ROLE marketing ADD MEMBER 'Anna';
```

Управление разрешениями в SQL Server осуществляется с помощью трех основных команд: GRANT, DENY и REVOKE. Команда GRANT предоставляет разрешение на выполнение определенного действия.

Пример. Выдача разрешения для роли ManagerRole право читать данные из таблицы Customers:

```
GRANT SELECT ON Customers TO ManagerRole
```

Команда DENY явно запрещает действие, даже если оно было предоставлено через другие механизмы (например, через членство в роли). DENY имеет высший приоритет и переопределяет любые разрешения, выданные посредством инструкции GRANT.

Команда REVOKE отменяет ранее выданное разрешение GRANT или запрет DENY, возвращая разрешение в нейтральное состояние. Важно понимать иерархию разрешений: разрешения могут наследоваться через членство в ролях, при этом явные запреты (DENY) имеют приоритет над явными разрешениями (GRANT), а разрешения на уровне объекта имеют приоритет над разрешениями на уровне базы данных.

Принцип разделения обязанностей является важнейшим аспектом корпоративной безопасности. В контексте СУБД это означает, что административные задачи должны быть распределены между несколькими людьми или ролями, чтобы минимизировать риски злоупотребления полномочиями. Например, пользователь, отвечающий за резервное копирование (предопределенная роль db_backupoperator), не должен иметь прав на удаление данных, а пользователь, имеющий доступ к конфиденциальным финансовым данным, не должен иметь прав на изменение схемы базы данных. SQL Server поддерживает этот принцип через детализированную систему ролей и разрешений, позволяющую точно настраивать права доступа для каждой должности.

Особого внимания заслуживают специальные пользователи и роли, такие как dbo (database owner), guest и PUBLIC. Пользователь dbo является владельцем базы данных и имеет все права в ней. Любой член предопределенной роли сервера sysadmin подключается к любой базе данных как dbo. Пользователь guest – это особая учетная запись, которая позволяет подключаться к базе данных логинам, не имеющим сопоставленного пользователя в этой базе. По умолчанию учетная запись guest отключена, и ее включение требует осторожности, так как это может создать брешь в безопасности. Роль PUBLIC – это специальная роль, в которую автоматически входят все пользователи базы данных. Любое разрешение, предоставленное роли PUBLIC, получают все пользователи, поэтому необходимо тщательно контролировать права, назначаемые этой роли.

Задание к лабораторной работе №12

1. Создать роль сервера с именем «группа_ФИО»
2. Предоставить созданной роли сервера разрешения:
ALTER ANY DATABASE
CREATE ANY DATABASE
VIEW SERVER STATE
3. Создать имя входа «ФИО», указать пароль.
4. Добавить имя входа «ФИО» в роль сервера «группа_ФИО».
5. Создать БД с именем «БД_ФИО».
6. Создать в БД 2 любые таблицы.
7. Создать пользователя БД. Сопоставить пользователя БД с именем входа «ФИО».
8. Создать новую роль БД «РольФИО».
9. Включить созданного пользователя БД в новую роль БД «РольФИО».
10. Предоставить созданной роли БД разрешения на объекты БД:
На таблицу 1 – SELECT
На таблицу 2 – INSERT, UPDATE, DELETE

Лабораторная работа №13
Резервное копирование и восстановление данных

Цель работы: освоить теоретические основы и практические навыки резервного копирования и восстановления данных в СУБД, научиться разрабатывать планы восстановления для различных сценариев сбоев.

Теоретические положения

Резервное копирование и восстановление данных представляют собой критически важные процессы в жизненном цикле любой СУБД, обеспечивающие сохранность информации и возможность ее восстановления после различных инцидентов.

Резервное копирование – это процесс создания копий данных, которые могут быть использованы для восстановления исходной информации после ее потери или повреждения. Восстановление – процесс применения резервных копий для возвращения системы в работоспособное состояние. Эти два процесса неразрывно связаны и должны рассматриваться как единый комплекс мер по обеспечению отказоустойчивости и непрерывности бизнес-процессов.

В большинстве современных СУБД существуют три основных типа резервных копий: полные, дифференциальные и резервные копии журнала транзакций.

Полное резервное копирование (Full Backup) создает копию всей базы данных на момент времени выполнения операции. Это фундаментальный тип бэкапа, который содержит все данные, необходимые для восстановления базы в состояние на момент создания резервной копии. Полные бэкапы служат отправной точкой для восстановления и обычно выполняются регулярно (ежедневно или еженедельно).

Команда BACKUP в SQL Server имеет сложный синтаксис с множеством параметров, позволяющих точно контролировать процесс создания резервных копий. Базовый синтаксис для создания полной резервной копии базы данных выглядит следующим образом:

```
BACKUP DATABASE database_name TO DISK = 'path\backup_file.bak' WITH options
```

Дифференциальное резервное копирование (Differential

Backup) захватывает только те данные, которые изменились с момента последнего полного бэкапа. Это достигается за счет отслеживания изменений через битовую карту измененных экстентов. Дифференциальные бэкапы обычно меньше по размеру и быстрее создаются по сравнению с полными, но при восстановлении требуют наличия соответствующего полного бэкапа, на который они ссылаются. Они особенно полезны для крупных баз данных, где ежедневное полное резервное копирование занимает слишком много времени или ресурсов.

Дифференциальное резервное копирование использует аналогичный синтаксис с указанием ключевого слова DIFFERENTIAL, например: `BACKUP DATABASE AdventureWorks TO DISK = 'D:\Backups\AdventureWorks_Diff.bak' WITH DIFFERENTIAL, INIT, NAME = 'AdventureWorks Differential Backup', CHECKSUM.`

Резервное копирование журнала транзакций (Transaction Log Backup) является уникальной особенностью СУБД, использующих механизм журналирования. Этот тип бэкапа захватывает все записи журнала транзакций, созданные с момента последнего резервного копирования журнала. В отличие от полных и дифференциальных бэкапов, резервные копии журнала транзакций не содержат самих данных, а лишь информацию об изменениях, что делает их значительно меньше по размеру. Однако их ключевое преимущество – возможность восстановления базы данных на любой момент времени в пределах сохраненной цепочки журналов. Это достигается за счет последовательного применения журналов транзакций к полной резервной копии до нужного момента времени. Важно отметить, что резервное копирование журнала транзакций доступно только при использовании моделей восстановления FULL или BULK_LOGGED. В модели SIMPLE журнал транзакций автоматически усекается и не может быть использован для восстановления на момент времени. Резервное копирование журнала транзакций выполняется с помощью команды BACKUP LOG, например:
`BACKUP LOG adb TO tailLogBackup WITH NORECOVERY, NO_TRUNCATE`

Модели восстановления в SQL Server определяют, как система управляет журналом транзакций и какие типы резервного копирования доступны.

Модель полного восстановления (FULL Recovery Model) обеспечивает максимальную защиту данных, позволяя восстанавливать базу на любой момент времени. В этой модели все операции полностью журналируются, и журнал транзакций не усекается автоматически – для его очистки необходимо выполнять резервное копирование журнала.

Модель неполного протоколирования (BULK_LOGGED Recovery Model) представляет собой компромисс между производительностью и возможностями восстановления. В этой модели массовые операции (такие как BULK INSERT, CREATE INDEX, SELECT INTO) выполняются с минимальным журналированием, что значительно уменьшает размер журнала транзакций и улучшает производительность операций загрузки данных. Однако это накладывает ограничения на восстановление: нельзя восстановить базу на момент времени внутри массовой операции, и при восстановлении требуется доступ ко всем файлам данных, участвовавших в минимально журналируемых операциях.

Простая модель восстановления (SIMPLE Recovery Model) является наиболее ограниченной, но и самой простой в управлении. В этой модели журнал транзакций автоматически усекается при каждой контрольной точке, что предотвращает его неконтролируемый рост, но делает невозможным резервное копирование журнала и восстановление на указанный момент времени. База данных может быть восстановлена только до состояния последней полной или дифференциальной резервной копии, что означает потенциальную потерю всех изменений, сделанных после последнего бэкапа. Простая модель подходит для тестовых проектов, некритичных баз данных, где потеря данных между бэкапами приемлема.

Изменение модели восстановления базы данных осуществляется инструкцией ALTER, например:

```
USE master;
ALTER DATABASE database_name SET RECOVERY FULL;
```

Восстановление данных в SQL Server – это многоэтапный процесс, который может варьироваться в зависимости от типа сбоя и выбранной стратегии восстановления. Восстановление данных в SQL Server выполняется с помощью команды RESTORE, которая имеет еще более сложный синтаксис, чем команда BACKUP, из-за необходимости поддержки различных сценариев восстановления.

Базовый синтаксис восстановления из полной резервной копии: `RESTORE DATABASE database_name FROM DISK = 'path\backup_file.bak' WITH options.`

Стандартный процесс восстановления начинается с применения полной резервной копии с параметром NORECOVERY, который оставляет базу данных в нерабочем состоянии, позволяя применять дополнительные резервные копии. Затем, если имеются, применяются дифференциальные бэкапы (также с NORECOVERY), после чего последовательно применяются резервные копии журнала транзакций в хронологическом порядке. Последняя применяемая резервная копия (обычно последний журнал транзакций) применяется с параметром RECOVERY, который переводит базу данных в согласованное состояние и делает ее доступной для пользователей. Такой подход позволяет восстановить базу данных до момента, непосредственно предшествующего сбою, при условии наличия непрерывной цепочки бэкапов.

Пример восстановления БД:

```
RESTORE DATABASE adb FILEGROUP='C' FROM backup2a
WITH NORECOVERY
RESTORE LOG adb FROM log_backup3 WITH NORECOVERY
RESTORE LOG adb FROM log_backup4 WITH NORECOVERY
RESTORE LOG adb FROM log_backup5 WITH NORECOVERY
RESTORE LOG adb FROM tailLogBackup WITH RECOVERY
```

Задание к лабораторной работе №13

1. Задать простую модель восстановления БД. Создать резервную копию БД. В БД изменить некоторые объекты БД и данные. Восстановить БД из резервной копии.

2. Задать полную модель восстановления БД. Создать резервную копию БД. В БД изменить некоторые объекты БД и данные. Создать резервную копию журнала транзакций. Восстановить БД из резервной копии (использовать параметр WITH NORECOVERY). Восстановить журнал транзакций из резервной копии (использовать параметр WITH RECOVERY).

3. Создать моментальный снимок БД. Просмотреть моментальный снимок БД. В БД изменить некоторые объекты БД и данные. Восстановить базу данных до состояния, сохраненного в моментальном снимке.

Лабораторная работа №14
Импорт и экспорт данных в таблицы БД

Цель работы: освоить методы и инструменты импорта и экспорта данных в СУБД.

Теоретические положения

Импорт и экспорт данных представляют собой фундаментальные процессы в жизненном цикле любой системы управления базами данных, обеспечивающие взаимодействие между различными информационными системами.

Импорт данных – это процесс загрузки информации из внешних источников в таблицы базы данных. Экспорт данных – процесс извлечения информации из базы данных и сохранения ее во внешних форматах для дальнейшего использования.

Современные СУБД поддерживают работу с широким спектром форматов данных, каждый из которых имеет свои преимущества, недостатки и оптимальные сценарии использования.

Формат CSV (Comma-Separated Values) остается одним из самых популярных для обмена табличными данными благодаря своей простоте, читаемости как машинами, так и людьми, и широкой поддержке в различных программных продуктах. CSV-файлы представляют собой обычный текст, где каждая строка соответствует записи в таблице, а значения разделяются специальными символами-разделителями, обычно запятыми или точками с запятой. Несмотря на кажущуюся простоту, работа с CSV имеет свои нюансы: необходимость обработки специальных символов в данных, различия в кодировках текста, проблемы с представлением многострочных значений и сложных структур данных.

Формат XML (eXtensible Markup Language) предлагает более структурированный подход, позволяющий представлять иерархические данные с помощью тегов, атрибутов и вложенных элементов. XML хорошо подходит для данных со сложной структурой, где важны отношения между элементами и метаинформация о данных. Однако XML файлы обычно более объемны и требуют больше ресурсов для разбора и обработки по сравнению с CSV.

Формат JSON (JavaScript Object Notation) представляет данные в виде пар ключ-значение и массивов, что делает его удобным для

работы с полуструктурированными данными.

Файлы Microsoft Excel остаются чрезвычайно популярными в бизнес-среде, и возможность импорта данных из Excel и экспорта в Excel является обязательным требованием для многих корпоративных систем. Excel-файлы могут содержать несколько листов, форматирование, формулы и другие элементы, что делает работу с ними сложнее, но и более функциональной по сравнению с простыми текстовыми форматами.

Microsoft SQL Server предлагает богатый набор инструментов и методов для импорта данных: SQL Server Import and Export Wizard, утилита командной строки bcp, команда T-SQL BULK INSERT, функции OPENROWSET и OPENDATASOURCE.

SQL Server Import and Export Wizard представляет собой графический инструмент, встроенный в SQL Server Management Studio, который позволяет выполнять импорт и экспорт данных через интуитивно понятный интерфейс с пошаговыми инструкциями. Мастер поддерживает широкий спектр источников данных, включая другие СУБД (Oracle, MySQL, PostgreSQL), файлы различных форматов (Excel, CSV, XML), а также OLE DB и ODBC источники.

Утилита командной строки bcp (bulk copy program) является одним из старейших и наиболее эффективных инструментов для массового копирования данных между экземпляром SQL Server и файлом данных в пользовательском формате. Синтаксис команды bcp включает множество параметров для тонкой настройки процесса: указание кодовых страниц, управление размерами пакетов, обработка ошибок, контроль формата данных.

Например, команда `bcp AdventureWorks.Sales.Customer out customers.dat -n -S SERVERNAME -T` экспортирует данные из таблицы Customer в бинарный файл, а команда `bcp AdventureWorks.Sales.Customer in customers.dat -n -S SERVERNAME -T -h "TABLOCK"` выполняет обратный импорт с установкой блокировки таблицы для увеличения производительности.

Команда T-SQL BULK INSERT предоставляет серверный механизм массовой загрузки данных, который можно использовать непосредственно в скриптах и хранимых процедурах. В отличие от bcp, BULK INSERT выполняет только импорт данных. Синтаксис BULK INSERT позволяет задавать множество параметров, анало-

гичных bcp: кодовую страницу, размер пакета, максимальное количество ошибок, режим обработки пустых значений.

Пример использования: `BULK INSERT AdventureWorks.Sales.Customer FROM 'D:\Data\customers.csv' WITH (FIELDTERMINATOR = ',', ROWTERMINATOR = '\n', FIRSTROW = 2, MAXERRORS = 10)`. Эта команда загружает данные из CSV файла, используя запятую как разделитель полей, символ новой строки как разделитель строк, пропуская первую строку (заголовок) и допуская до 10 ошибок перед прерыванием операции. BULK INSERT особенно эффективен при работе с большими файлами данных, так как использует минимальное журналирование и оптимизированные алгоритмы загрузки.

Функции OPENROWSET и OPENDATASOURCE предоставляют возможность доступа к удаленным данным как к обычным таблицам, что позволяет выполнять сложные операции импорта с использованием стандартных SQL-запросов. OPENROWSET может читать данные из файлов различных форматов или подключаться к другим базам данных через OLE DB провайдеров.

Например, запрос `SELECT * FROM OPENROWSET ('Microsoft.ACE.OLEDB.12.0', 'Excel 12.0; Database=D:\Data\customers.xlsx;', 'SELECT * FROM [Sheet1$]')` позволяет работать с данными из Excel-файла как с обычной таблицей. Аналогично, OPENDATASOURCE обеспечивает доступ к удаленным экземплярам сервера без предварительной настройки связанных серверов: `SELECT * FROM OPENDATASOURCE('SQLNCLI', 'Server=RemoteServer; Trusted_Connection=yes;').AdventureWorks.Sales.Customer`.

Экспорт данных из SQL Server может выполняться с использованием тех же инструментов, что и импорт, но в обратном направлении, а также с помощью некоторых специализированных методов. SQL Server Management Studio предоставляет простой способ экспорта результатов запросов через контекстное меню «Сохранить результаты как…»: любой запрос можно экспортировать в CSV, Excel или XML формат с помощью нескольких кликов мыши. Для регулярного экспорта или работы с большими объемами данных более предпочтительными являются скриптовые методы. Команда SELECT ... INTO OUTFILE в сочетании с различными форматами

вывода позволяет создавать файлы данных непосредственно из SQL-запросов. Например, запрос `SELECT * FROM Customers FOR XML PATH('Customer'), ROOT('Customers')` создает XML документ с заданной структурой, а использование предложения FOR JSON AUTO или FOR JSON PATH генерирует JSON-представление результатов запроса.

Службы SQL Server Reporting Services (SSRS) предоставляют мощные возможности для экспорта данных в различные форматы, включая PDF, Excel, Word, CSV и другие, с поддержкой сложного форматирования, разбивки на страницы и параметризации. Отчеты SSRS могут использовать данные из различных источников, выполнять сложные вычисления и преобразования, а затем экспортироваться в требуемые форматы как по расписанию, так и по запросу.

Задание к лабораторной работе №14

1. В файле Excel создайте таблицу из 6 столбцов. В каждый столбец занесите данные.
2. Осуществите импорт данных из файла Excel с помощью мастера импорта и экспорта SQL Server.
3. Осуществите экспорт данных из запроса в таблицу Excel.
4. Создайте текстовый файл с разделителями. Занесите в него данные (10 строк, 5 столбцов). Осуществите импорт данных с помощью импорта неструктурированных файлов в SQL.
5. Экспортируйте данные из любой таблицы в текстовый файл.
6. Скопируйте данные из одной таблицы в другую.

Лабораторная работа №15
Аутентификация и авторизация пользователей клиентского приложения

Цель работы: исследовать механизмы аутентификации и авторизации пользователей клиентских приложений в Microsoft SQL Server, изучить различные подходы к управлению доступом и разработать безопасную архитектуру взаимодействия приложения с базой данных.

Теоретические положения

Аутентификация и авторизация представляют собой фундаментальные механизмы безопасности, обеспечивающие защиту данных и функциональности в клиент-серверных приложениях, взаимодействующих с системами управления базами данных. В контексте Microsoft SQL Server эти процессы образуют многоуровневую систему безопасности, которая начинается с момента установления соединения приложения с сервером базы данных и заканчивается контролем доступа к конкретным объектам данных внутри базы. Аутентификация – это процесс подтверждения идентичности пользователя или приложения, пытающегося установить соединение с SQL Server. Это первый барьер безопасности, который проверяет, действительно ли субъект является тем, за кого себя выдает, используя для этого различные механизмы проверки учетных данных. Авторизация, следующая за успешной аутентификацией, определяет, какие действия разрешено выполнять прошедшему проверку субъекту, к каким объектам базы данных он имеет доступ и с какими ограничениями. Эти два процесса тесно взаимосвязаны и образуют единую систему контроля доступа, от корректности реализации которой зависит безопасность всей информационной системы.

Для клиентского приложения ключевым элементом взаимодействия с SQL Server является строка подключения – специальная строка параметров, которая определяет, как приложение устанавливает соединение с базой данных. Строка подключения содержит такую информацию, как имя сервера, имя базы данных, метод аутентификации и учетные данные.

При использовании аутентификации Windows строка подклю-

чения обычно выглядит как: `string connectionString = "Server=myServerAddress; Database=myDataBase; Trusted_Connection=True;"`. В этом случае приложение использует учетные данные пользователя Windows, под которым оно запущено, что исключает необходимость хранения паролей в приложении.

Для аутентификации SQL Server строка подключения включает логин и пароль: `string connectionString = "Server=myServerAddress;Database=myDataBase;UserId=myUsername;Password=myPassword;"`.

Важно понимать, что хранение такой строки подключения в открытом виде в конфигурационном файле (например, app.config или web.config) представляет серьезную угрозу безопасности.

Существует несколько подходов к безопасному хранению учетных данных в клиентских приложениях. Для приложений .NET одним из наиболее безопасных методов является использование защищенной конфигурации, которая позволяет шифровать разделы конфигурационных файлов с помощью DPAPI (Data Protection API) или RSA. В этом случае строка подключения шифруется и может быть расшифрована только на том же компьютере и под той же учетной записью пользователя, под которой она была зашифрована. Альтернативным подходом является хранение только имени сервера и базы данных в конфигурации, а учетных данных – в безопасном хранилище, таком как Windows Credential Manager или Azure Key Vault. Для веб-приложений рекомендуется использовать Managed Identity в Azure или Service Accounts в Windows, что позволяет полностью избежать хранения паролей в конфигурации приложения.

Разработка системы аутентификации в клиентском приложении, взаимодействующем с SQL Server, требует решения нескольких ключевых задач: проверки учетных данных пользователя, управления сессиями, обеспечения безопасного хранения токенов аутентификации и обработки сценариев истечения срока действия или обновления учетных данных. Простейший подход предполагает прямое использование логина и пароля SQL Server для каждого запроса к базе данных, но такой метод имеет серьезные недостатки: необходимость постоянной передачи учетных данных, отсутствие централизованного управления сессиями, сложность реализации

безопасного хранения паролей на клиентской стороне. Более продвинутые подходы включают использование промежуточного слоя аутентификации, такого как веб-сервис или API, который проверяет учетные данные и возвращает токен доступа, используемый приложением для последующих запросов к базе данных.

Для веб-приложений типичным подходом является использование формовой аутентификации (Forms Authentication), когда пользователь вводит учетные данные в веб-форму, приложение проверяет их в базе данных (используя безопасное сравнение хешей паролей, а не самих паролей) и создает сессию с токеном аутентификации, который хранится в cookie или localStorage. При последующих запросах приложение использует этот токен для идентификации пользователя и определения его прав доступа.

Пример авторизации на C#:

```
using System;
using System.Linq;
using Microsoft.EntityFrameworkCore;

public class User
{
    public int Id { get; set; }
    public string Login { get; set; }
    public string Pass { get; set; }
}
public class Context : DbContext
{
    public DbSet<User> Users { get; set; }
    protected override void OnConfiguring (DbCon-
textOptionsBuilder      options)      =>      op-
tions.UseSqlServer ("Server=.;Database=DB; Trust-
ed_Connection=True;");
}

class Program
{
    static void Main()
    {
        using var db = new Context();
        Console.Write("Логин: ");
```

```
        string login = Console.ReadLine();
        Console.Write("Пароль: ");
        string pass = Console.ReadLine();

        // LINQ-запрос для авторизации
        var user = db.Users
.FirstOrDefault(u => u.Login == login && u.Pass
== pass);
        Console.WriteLine(user != null ? "OK" :
"NO");
    }
}
```

Данный код представляет собой консольное приложение для аутентификации пользователей, которое проверяет соответствие введенных учетных данных данным, хранящимся в базе данных SQL Server. В начале программы определяются две основные сущности: класс User, который моделирует таблицу пользователей в базе данных с тремя свойствами (уникальный идентификатор Id, логин Login и пароль Pass), и класс Context, наследующий от DbContext, который настраивает подключение к базе данных через Entity Framework Core. В классе Context определяется свойство Users типа DbSet<User>, представляющее коллекцию пользователей в базе данных, и в методе OnConfiguring задается строка подключения к локальному экземпляру SQL Server с использованием аутентификации Windows, указывающая на базу данных с именем DB.

Основная логика программы находится в методе Main класса Program. При запуске приложение создает экземпляр контекста базы данных, используя блок using. Затем программа последовательно запрашивает у пользователя ввод логина и пароля через консоль, сохраняя введенные значения в переменные login и pass. Для проверки аутентификационных данных выполняется LINQ-запрос к базе данных: с помощью метода FirstOrDefault происходит поиск первого пользователя в таблице Users, у которого значения полей Login и Pass точно совпадают с введенными пользователем значениями. Если такой пользователь найден, переменная user получит ссылку на соответствующий объект, в противном случае ее значение будет равно null.

Результат проверки выводится на консоль с помощью тернарного оператора: если user не равен null, программа печатает «OK»,

что означает успешную аутентификацию, если же пользователь не найден или введенные данные не совпали ни с одной записью в базе, выводится «NO», сигнализируя об отказе в доступе.

Приведенный пример кода реализует базовый механизм аутентификации, однако в реальных приложениях такой подход считается небезопасным из-за хранения паролей в открытом виде в базе данных, отсутствия хеширования, шифрования передачи данных и механизмов защиты от SQL-инъекций, которые обеспечиваются использованием параметризованных запросов через Entity Framework.

Задание к лабораторной работе №15

1. Создайте приложение на C# для аутентификации пользователей, которое будет запрашивать логин и пароль, выполнять LINQ-запрос к базе данных SQL Server для поиска пользователя с соответствующими учетными данными в таблице Users, и выводить сообщение об успешной или неуспешной авторизации, при этом реализуйте безопасное хранение паролей путем хеширования и добавьте проверку на количество неудачных попыток входа, блокируя пользователя после пяти неуспешных попыток в течение часа, а также обеспечьте ведение лога всех попыток входа в отдельную таблицу LoginAttempts с фиксацией времени, имени пользователя и результата попытки.

Содержание самостоятельной работы

Цель самостоятельной работы обучающихся – формирование у студентов профессиональных компетенций в области проектирования, разработки, администрирования и оптимизации баз данных через выполнение практико-ориентированных заданий.

Самостоятельная работа необходима для формирования у обучающихся способности самостоятельно решать задачи профессиональной деятельности, формирования умения и навыков планирования времени, формирования стремления развиваться и совершенствоваться.

Виды самостоятельной работы обучающихся указаны в таблице 4.

Таблица 4 – Виды самостоятельной работы

№ п/п	Вид СРС
1	Изучение литературы
2	Проведение сравнительного анализа механизмов работы с данными в различных СУБД: оценка синтаксиса и производительности хранимых процедур/функций, исследование триггеров, изучение уровней изоляции и управление блокировками транзакций
3	Изучение архитектурных особенностей различных СУБД: способы хранения метаданных, кэширование планов выполнения запросов/процедур, использование системных таблиц и представлений
4	Исследование ограничений СУБД на размеры объектов БД, максимальную глубину вложенности транзакций, количество параметров процедур
5	Изучение встроенных инструментов мониторинга и диагностики сервера СУБД
6	Изучение стандартов безопасности (ISO/IEC 27001)

Обучающиеся должны изучить интернет-ресурсы и литературу по вопросам, представленным в таблице 4.

Список литературы

1. Стружкин, Н. П. Базы данных: проектирование.: учебник для СПО / Н. П. Стружкин, В. В. Годин. – Москва : Юрайт, 2025. – 477 с.
2. Советов, Б. Я. Базы данных: учебник для СПО / Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской. – 4-е изд., пер. и доп. – Москва : Юрайт, 2025. – 403 с.
3. Нестеров, С. А. Базы данных: учебник и практикум для СПО / С. А. Нестеров. – 2-е изд. – Москва : Юрайт, 2025. – 258 с.
4. 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.
5. Илюшечкин, В. М. Основы использования и проектирования баз данных : учебник для СПО / В. М. Илюшечкин. – Москва : Юрайт, 2025. – 213 с.
6. Мамедли, Р. Э. Системы управления базами данных : учебник для СПО / Р. Э. Мамедли. – Санкт-Петербург : Лань, 2024. – 228 с.
7. Полтавцева, М. А. Безопасность баз данных : учебник для СПО / М. А. Полтавцева. – Санкт-Петербург : Лань, 2024. – 356 с.
8. Федорова, Г. Н. Разработка, администрирование и защита баз данных : учебник для студентов среднего профессионального образования / Г. Н. Федорова ; Г. Н. Федорова. – 6-е изд., перераб. – Москва : Академия, 2024. – 288 с.

СОДЕРЖАНИЕ