МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Методические рекомендации к практическим занятиям для студентов специальности 6-05-0611-01 «Информационные системы и технологии» дневной и заочной форм обучения

Могилев 2026

УДК 004.65
ББК 32.973.26-0.18.2
С89

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Автоматизированные системы управления» «9» июня 2026 г., протокол № 11

Составитель ст. преподаватель Н. П. Скрылёв

Рецензент канд. техн. наук, доц. В. В. Кутузов

Методические рекомендации содержат описание хода проведения практических занятий по дисциплине «Системы управления базами данных» для специальности 6-05-0611-01 «Информационные системы и технологии».

Учебное издание

СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Ответственный за выпуск	А. И. Якимов
Корректор	И. В. Голубцова
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 21 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

© Белорусско-Российский
университет, 2026

Содержание

Введение 4
1 Практическое занятие № 1. Разработка технического задания на проектирование информационной системы 5
2 Практическое занятие № 2. CASE-средства концептуального проектирования функциональных моделей информационных систем 6
3 Практическое занятие № 3. Использование CASE-средств концептуального проектирования информационной модели системы в своем курсовом проекте 8
4 Практическое занятие № 4. Взаимодействие CASE-средств с системами управления базами данных (генерация схемы базы данных для своей БД) 10
5 Практическое занятие № 5. Создание баз данных на основе промышленной СУБД MS SQL Server для своего курсового проекта 13
6 Практическое занятие № 6. Создание связей между таблицами средствами T-SQL для своего курсового проекта 17
7 Практическое занятие № 7. Создание sql-скрипта заполнения базы данных в своем курсовом проекте 21
8 Практическое занятие № 8. Язык T-SQL. Добавление, изменение и удаление данных в таблицах средствами T-SQL в своей базе данных по курсовому проектированию 25
Список литературы 30

Введение

Целью учебной дисциплины «Системы управления базами данных» (СУБД) является формирование профессиональных компетенций для работы с современными системами управления базами данных, основами эксплуатации баз данных (БД) в составе автоматизированных систем обработки информации, внедряемых в различных областях науки, техники и экономики.

В результате освоения учебной дисциплины студент:

а) ознакомится:

– со способами работы с реляционными базами данных в однопользовательских и многопользовательских СУБД;

– с принципами моделирования и проектирования баз данных и перенесения в СУБД проекта;

– со способами реализации распределенной и параллельной обработки данных;

– с возможностями современных СУБД для построения систем поддержки принятия решений;

б) научится:

– практически создавать и администрировать базы данных;

– создавать интерфейс с базами данных;

– организовывать работу в многопользовательской базе данных;

– устанавливать и конфигурировать серверные и клиентские приложения баз данных;

в) овладеет:

– методами программирования систем на основе баз данных;

– навыками проектирования баз данных в СУБД SQL Server.

Методические рекомендации содержат описание восьми практических работ, выполняемых при изучении дисциплины «Системы управления базами данных», задания, контрольные вопросы, список литературы [1–11], которую необходимо использовать при подготовке к защите практических работ.

1 Практическое занятие № 1. Разработка технического задания на проектирование информационной системы

Цель работы: освоить методологию сбора и анализа требований к информационной системе; научиться формализовывать их в виде технического задания на проектирование базы данных.

Теоретические положения

Техническое задание (ТЗ) является основополагающим документом, определяющим цели, функции, структуру и этапы разработки информационной системы (ИС) или отдельной базы данных (БД). Качественно составленное ТЗ позволяет избежать недопонимания между заказчиком и разработчиком, четко определить границы проекта и служит основой для приемки выполненных работ. ТЗ разрабатывается на этапе концептуального проектирования и должно содержать следующие разделы.

1 Общие сведения: наименование системы, организации-заказчика и разработчика, плановые сроки начала и окончания работ.

2 Назначение и цели создания системы: описывается, для каких видов деятельности предназначена система и какие задачи она должна решать.

3 Характеристика объектов автоматизации: краткое описание предметной области и существующих бизнес-процессов, которые будут автоматизированы.

4 Требования к системе: включают требования к функциональным возможностям (перечень операций, которые должна выполнять система), к надежности, безопасности, эргономике, а также к техническому и программному обеспечению.

5 Состав и содержание работ по созданию системы: перечень этапов проектирования и разработки (например, сбор материалов, построение моделей, разработка БД, тестирование).

6 Требования к документированию: состав пояснительной записки, руководства пользователя и других документов.

7 Требования к информационному обеспечению: определяет, какие данные должны храниться в БД, источники данных и способы их ввода.

В контексте курсового проектирования по СУБД основной целью является описание предметной области и формализация требований к базе данных. Необходимо определить перечень сущностей (объектов), их атрибуты, связи между ними и основные бизнес-правила.

Например, для ИС «Автосервис» в ТЗ необходимо указать, что система должна хранить информацию о клиентах (ФИО, телефон, автомобиль), оказываемых услугах (название, стоимость, время выполнения), заказах (дата, статус, исполнитель), а также обеспечивать возможность регистрации заказов, поиска клиентов и формирования отчетов. Важным разделом является описание бизнес-правил: например, заказ не может существовать без клиента; удаление услуги не должно удалять историю заказов, если это запрещено.

Задание

Для заданной предметной области разработать техническое задание на проектирование базы данных. Техническое задание должно включать в себя назначение и цели разрабатываемой БД, описание предметной области, основные функции системы, перечень объектов (сущностей) и их атрибутов, перечень обязательных разделов и основные бизнес-правила (ограничения и связи между сущностями).

Содержание отчета: тема и цель работы; оформленное техническое задание в соответствии с перечисленными разделами; вывод.

Контрольные вопросы

1 Что такое техническое задание и какова его роль в разработке ИС?
2 Какие основные разделы включает ТЗ на проектирование БД?
3 Что такое бизнес-правила и почему их важно описывать в ТЗ?
4 Как цели создания системы влияют на выбор структуры базы данных?
5 Какие источники информации используются при составлении ТЗ для проектирования БД?

2 Практическое занятие № 2. CASE-средства концептуального проектирования функциональных моделей информационных систем

Цель работы: получить навыки определения индексируемых столбцов таблиц.

Теоретические положения

Функциональное моделирование – это подход к описанию системы с точки зрения выполняемых ею функций. Одним из современных стандартов моделирования бизнес-процессов является BPMN 2.0 (Business Process Model and Notation). BPMN предоставляет графическую нотацию, понятную как бизнес-аналитикам, так и техническим специалистам.

В данных практических работах будет использовано CASE-средство Modelio.

Основные элементы BPMN.

1 Пулы (Pools) и дорожки (Lanes) – контейнеры для организации процессов по участникам (например, «Клиент», «Автосервис», «Мастер»).

2 События (Events) – обозначают начало (Start Event), промежуточные состояния и завершение (End Event) процесса.

3 Задачи/действия (Activities/Tasks) – выполняемые работы (например, «Принять заявку», «Выполнить работу»).

4 Шлюзы (Gateways) – точки ветвления и слияния потоков.

4.1 Исключающий шлюз (XOR) – выбор одного из нескольких путей.

4.2 Параллельный шлюз (AND) – одновременное выполнение нескольких ветвей.

4.3 Инклюзивный шлюз (OR) – выполнение одного или нескольких путей.

5 Потоки последовательности (Sequence Flows) – показывают порядок выполнения действий.

6 Потоки сообщений (Message Flows) – показывают обмен информацией (данными, документами, запросами) между различными участниками процесса (пулами).

7 Артефакты – дополнительные объекты (данные, аннотации).

Преимущества BPMN:

– наглядность и понятность для всех участников проекта;

– возможность моделирования как простых, так и сложных процессов с ветвлениями и параллельными потоками;

– поддержка в современных CASE-средствах, включая Modelio;

– возможность генерации исполняемого кода (в некоторых реализациях).

Основные этапы построения функциональной модели с использованием BPMN в Modelio.

1 Создание проекта (File – Create a project).

2 Создание диаграммы (ПКМ по папке проекта – Create diagram – BPMN Collaboration diagram).

3 Создание процесса.

4 Добавление элементов процесса и связей между ними.

Пример – Для ИС «Автосервис» BPMN-диаграмма может включать:

– пул «Клиент»: события отправки заявки и получения результата;

– пул «Автосервис»: задачи «Принять заявку», «Назначить мастера», «Выполнить работы», «Сформировать чек»;

– шлюзы: проверка наличия материалов, выбор мастера, параллельное выполнение работ и оформление документов;

– потоки сообщений: обмен данными между клиентом и автосервисом.

Использование CASE-средства Modelio автоматизирует процесс построения диаграмм BPMN, обеспечивает проверку синтаксиса и генерацию отчетов.

Задание

Для предметной области, описанной в ТЗ, необходимо построить функциональную модель в Modelio, используя нотацию BPMN 2.0.

Содержание отчета: тема и цель работы; скриншоты BPMN-диаграмм.

Контрольные вопросы

1 Что такое функциональное моделирование информационных систем? Какова его основная цель?

2 Что такое BPMN и для решения каких задач он применяется? Назовите основные версии нотации.

3 В чем разница между пулом (Pool) и дорожкой (Lane) в BPMN? Приведите примеры использования.

4 Чем отличается поток последовательности (Sequence Flow) от потока сообщений (Message Flow)? В каких случаях какой используется?

5 Что такое подпроцесс (Sub-process) в BPMN? Какие преимущества дает использование подпроцессов?

3 Практическое занятие № 3. Использование CASE-средств концептуального проектирования информационной модели системы в своем курсовом проекте

Цель работы: на основе функциональной модели разработать логическую информационную модель с использованием CASE-средства Modelio для дальнейшего проектирования базы данных курсового проекта.

Теоретические положения

Информационная модель, или логическая модель данных, представляет собой абстрактное описание структуры базы данных, не зависящее от конкретной СУБД. Основным способом представления такой модели служит диаграмма «сущность – связь» (ER-диаграмма). В её составе выделяют три основных компонента.

Сущность – это объект реального мира, информация о котором хранится в базе данных, например, Студент, Заказ или Товар; на диаграмме она обозначается прямоугольником.

Атрибут представляет собой характеристику сущности, такую как фамилия и имя, дата рождения или стоимость; при этом первичный ключ принято подчеркивать.

Связь описывает взаимодействие между сущностями, к примеру, «Студент обучается на Факультете», и характеризуется двумя параметрами: мощностью, которая может быть «один-к-одному», «один-ко-многим» или «многие-ко-многим», а также степенью участия, которая бывает обязательной (обозначается двойной линией) или необязательной.

При работе в Modelio следует учитывать соответствие между классической ER-моделью и элементами UML. Сущность в ER-модели представляется как Класс на UML-диаграмме классов, а связь между сущностями любой мощности («один-к-одному», «один-ко-многим» или «многие-ко-многим») моделируется как Ассоциация между классами с обязательным указанием кратности. Первичный ключ может быть отмечен с помощью стереотипа «PK» либо в окне свойств соответствующего атрибута класса. Весь процесс проектирования в Modelio начинается с создания UML-диаграммы классов, которая впоследствии преобра-

зуется в физическую модель с использованием встроенного модуля SQL Designer.

Построение информационной модели в CASE-средстве выполняется в несколько последовательных шагов. Вначале необходимо создать сущности, задав их названия и определив атрибуты с указанием типов данных; на уровне логической модели используются универсальные типы, например, Integer, String, Date. Далее для каждой сущности следует выделить первичный ключ – атрибут или группу атрибутов, уникально идентифицирующих запись; при этом рекомендуется использовать суррогатные ключи, такие как идентификатор ID. Затем нужно построить связи между сущностями, соединив их линиями с указанием родительской и дочерней сторон, а для каждой связи определить мощность («один-ко-многим» или «многие-ко-многим») и тип обязательности; для связи «один-ко-многим» обязательно определяется внешний ключ в дочерней сущности. На заключительном этапе выполняется нормализация модели – проверка соответствия третьей нормальной форме с исключением транзитивных зависимостей и дублирования данных, а связи «многие-ко-многим» устраняются путем создания промежуточной ассоциативной сущности.

В качестве примера рассмотрим информационную систему «Автосервис», в которой выделяются следующие сущности: Клиент (Client), Автомобиль (Car), Заказ (Order), Услуга (Service) и Сотрудник (Employee). Между этими сущностями устанавливаются определенные связи. Клиент может владеть несколькими автомобилями, что соответствует отношению «один-ко-многим». Заказ всегда создается для конкретного автомобиля и конкретного клиента, причем эта связь может быть реализована либо через сущность Автомобиль, либо напрямую как отношение «один-ко-многим» от Клиента к Заказу. Более сложная связь существует между Заказом и Услугой: один заказ может включать множество услуг, и одна услуга может входить во множество заказов, что представляет собой отношение «многие-ко-многим». Для устранения этой связи вводится промежуточная сущность Order_Service (позиция заказа) с атрибутами Количество и Цена, которая связывается отношением «один-ко-многим» как с Заказом, так и с Услугой. Кроме того, каждый Сотрудник может выполнять множество заказов, что также является связью «один-ко-многим».

Задание

В рамках данной работы необходимо разработать информационную модель данных в Modelio с использованием UML и разработок прошлых практических работ.

Содержание отчета: тема и цель работы; список сущностей и их атрибутов (в виде таблицы); скриншот ER-диаграммы (логическая модель).

Контрольные вопросы

1 Что представляет собой логическая модель данных и чем она отличается от физической?

2 Что такое сущность, атрибут, первичный и внешний ключ?

3 Какие типы связей существуют в ER-модели? Приведите примеры.

4 Как обозначается обязательность участия сущности в связи?

5 Как разрешить связь «многие-ко-многим» в реляционной модели?

6 В чем заключается процесс нормализации базы данных? Назовите основные нормальные формы.

7 Какие преимущества дает использование CASE-средств при построении ER-диаграмм?

8 Как проверить правильность построенной модели данных с точки зрения бизнес-правил?

4 Практическое занятие № 4. Взаимодействие CASE-средств с системами управления базами данных (генерация схемы базы данных для своей БД)

Цель работы: изучить процесс автоматической генерации схемы базы данных на основе логической модели, построенной в CASE-средстве; освоить навыки экспорта и импорта метаданных между CASE-системой и промышленной СУБД.

Теоретические положения

Под CASE-средствами (Computer-Aided Software Engineering) понимают инструментальные программные средства, поддерживающие процессы создания и сопровождения информационных систем, включая проектирование баз данных. В контексте дисциплины СУБД наиболее важными являются CASE-средства, ориентированные на моделирование данных и генерацию схем баз данных.

Процесс проектирования базы данных с использованием CASE-средств обычно включает несколько последовательных этапов.

На первом этапе создаётся концептуальная модель предметной области, в рамках которой выявляются основные сущности, их атрибуты и связи между ними.

На втором этапе строится логическая модель, где определяются типы данных, первичные и внешние ключи, выполняется нормализация структуры.

На заключительном этапе выполняется генерация физической схемы – автоматическое создание SQL-скриптов для реализации базы данных в конкретной промышленной СУБД. Наиболее распространёнными CASE-средствами для проектирования БД являются ERwin Data Modeler, Microsoft Visio, dbForge Studio, MySQL Workbench, PowerDesigner и DBeaver. Каждое из этих средств обладает своими особенностями, но все они поддерживают основные функции моделирования и генерации.

Важнейшим понятием в работе CASE-средств является прямая инженерия

(Forward Engineering) – это процесс автоматической генерации SQL-скриптов и непосредственно самой базы данных на основе логической модели, построенной в CASE-средстве. При этом CASE-средство анализирует структуру модели, включая сущности, атрибуты, типы данных и связи между сущностями, после чего формирует инструкции CREATE TABLE, ALTER TABLE, CREATE INDEX, CREATE VIEW и другие. Особое внимание уделяется трансляции типов данных из универсального представления в типы, поддерживаемые конкретной СУБД: например, в MS SQL Server строковые данные могут отображаться как NVARCHAR, в Oracle – как VARCHAR2, в MySQL – как VARCHAR. Обратная инженерия (Reverse Engineering) представляет собой процесс восстановления логической модели по уже существующей физической базе данных. Она позволяет визуализировать структуру существующей БД в виде ER-диаграммы, документировать схему БД, а также анализировать и рефакторить структуру данных.

Использование CASE-средств при проектировании баз данных даёт ряд существенных преимуществ. Прежде всего, это наглядность – визуальное представление структуры данных значительно облегчает понимание модели и коммуникацию между разработчиками. Автоматизация генерации SQL-кода исключает рутинные ошибки, которые неизбежно возникают при ручном написании скриптов. Кроме того, обеспечивается согласованность – изменения в модели автоматически синхронизируются с генерируемым кодом. Важным преимуществом является также возможность документирования: CASE-средства позволяют автоматически создавать документацию по структуре базы данных. Наконец, поддержка различных СУБД означает, что одна и та же логическая модель может быть использована для генерации схем для разных платформ, что особенно ценно при разработке кроссплатформенных решений.

Процесс генерации схемы базы данных в CASE-средстве включает несколько обязательных шагов. Прежде всего выполняется проверка модели, т. е. валидация логической модели на наличие ошибок, таких как неопределённые первичные ключи, циклические зависимости или некорректные типы данных. Затем производится выбор целевой СУБД, для которой будет выполняться генерация, и настройка параметров генерации, включая правила именования таблиц и полей, способы создания внешних ключей и опции индексации. После этого CASE-средство формирует SQL-скрипт создания объектов базы данных, который может быть либо сохранён в файл для последующего выполнения, либо непосредственно выполнен на сервере БД, если выбрана соответствующая опция.

Настройки именования объектов являются важным аспектом генерации, поскольку согласованная система именования облегчает дальнейшую работу с базой данных. Рекомендуется давать имена таблицам во множественном числе (например, Employees, Orders), а полям – в единственном числе, отражающем суть данных. Для внешних ключей следует использовать формат FK_дочерняя_таблица_родительская_таблица, а для индексов – IX_таблица_поле. Категорически не рекомендуется использовать пробелы, специальные символы и зарезервированные слова SQL в именах объектов. Для

MS SQL Server, как правило, используется схема dbo, хотя при необходимости могут создаваться и собственные схемы в соответствии с требованиями технического задания.

Далее приведён пример сгенерированного CASE-средством SQL-скрипта для таблиц базы данных «Автосервис».

```
/* Создание таблицы Customers */
CREATE TABLE Customers (
   CustomerID INT NOT NULL IDENTITY(1,1),
   FullName NVARCHAR(100) NOT NULL,
   Phone NVARCHAR(20) NOT NULL,
   Email NVARCHAR(50) NULL,
   CONSTRAINT PK_Customers PRIMARY KEY (CustomerID)
);

/* Создание таблицы Orders */
CREATE TABLE Orders (
   OrderID INT NOT NULL IDENTITY(1,1),
   CustomerID INT NOT NULL,
   OrderDate DATETIME NOT NULL DEFAULT GETDATE(),
   TotalAmount DECIMAL(10,2) NOT NULL,
   CONSTRAINT PK_Orders PRIMARY KEY (OrderID)
);

/* Добавление внешнего ключа */
ALTER TABLE Orders
ADD CONSTRAINT FK_Orders_Customers
FOREIGN KEY (CustomerID)
REFERENCES Customers (CustomerID)
ON DELETE NO ACTION
ON UPDATE NO ACTION;
```

Как видно из примера, CASE-средство автоматически генерирует не только инструкции создания таблиц с указанием типов данных, но и инструкции для создания первичных и внешних ключей, сохраняя при этом заданные разработчиком правила именования объектов.

Задание

Для разрабатываемой в курсовом проекте базы данных необходимо выполнить генерацию схемы с использованием одного из CASE-средств. Сгенерированный скрипт следует выполнить на сервере MS SQL Server в среде SQL Server Management Studio и сравнить полученную физическую структуру с исходной логической моделью

Содержание отчета: тема и цель работы; скриншоты логической модели в CASE-средстве; сгенерированный SQL-скрипт создания БД (в виде приложения).

Контрольные вопросы

1 Какие этапы включает процесс генерации схемы БД в CASE-средстве?

2 Как выполняется трансляция типов данных из логической модели в физическую реализацию конкретной СУБД?

3 Какие преимущества даёт использование CASE-средств по сравнению с ручным написанием SQL-скриптов?

4 Какие настройки именования объектов необходимо учитывать при генерации схемы?

5 Практическое занятие № 5. Создание баз данных на основе промышленной СУБД MS SQL Server для своего курсового проекта

Цель работы: получить практические навыки создания базы данных в промышленной СУБД MS SQL Server с использованием средств языка T-SQL и графических средств SQL Server Management Studio.

Теоретические положения

Создание базы данных в MS SQL Server является первым и обязательным шагом перед разработкой информационной системы. База данных представляет собой контейнер для всех объектов, связанных с конкретным приложением: таблиц, представлений, хранимых процедур, триггеров, индексов и других компонентов.

Физически база данных в MS SQL Server состоит из файлов данных и журнала транзакций. Файлы данных, имеющие расширение .mdf для основного файла и .ndf для дополнительных файлов, содержат сами данные таблиц и другие объекты базы данных. Журнал транзакций, имеющий расширение .ldf, фиксирует все изменения данных, что обеспечивает возможность восстановления базы данных в случае сбоев. Файлы данных могут быть логически объединены в файловые группы, что позволяет управлять распределением данных и повышать производительность за счёт размещения разных частей БД на разных дисковых устройствах.

Для создания базы данных используется инструкция CREATE DATABASE. В её синтаксисе можно выделить несколько ключевых элементов.

Параметр database_name задаёт имя создаваемой базы данных, которое должно быть уникальным в пределах сервера и иметь смысловую нагрузку, отражающую название разрабатываемой информационной системы.

Блок ON определяет файлы данных: при его отсутствии создаётся один файл данных с параметрами по умолчанию.

Ключевое слово PRIMARY указывает, что файл принадлежит первичной файловой группе, которая должна существовать в каждой базе данных.

Параметр NAME задаёт логическое имя файла, используемое в скриптах и сообщениях об ошибках, а FILENAME – полный путь к физическому файлу на диске.

Параметры SIZE, MAXSIZE и FILEGROWTH определяют соответственно начальный размер файла, максимальный размер (при этом UNLIMITED позволяет файлу расти до заполнения всего доступного дискового пространства) и шаг увеличения размера при автоматическом росте.

Блок LOG ON определяет файлы журнала транзакций, при этом рекомендуется размещать файлы данных и журнала на разных дисках для повышения производительности и надёжности.

Параметр COLLATE задаёт параметры сортировки для базы данных, определяющие правила сравнения строковых данных.

При создании базы данных следует придерживаться ряда рекомендаций. Имя базы данных должно быть кратким, но информативным, например AutomationSystemDB или UniversityDB. Начальный размер файлов данных должен составлять не менее 5...10 МБ для небольшой БД, чтобы минимизировать частоту автоматического роста, которая может снижать производительность.

Параметр MAXSIZE рекомендуется указывать явно, чтобы предотвратить неконтролируемый рост базы данных, способный заполнить всё дисковое пространство.

Параметр FILEGROWTH рекомендуется задавать в мегабайтах (например, 10 MB) вместо процентов, чтобы избежать слишком резкого роста при больших объёмах данных. Файлы данных и журнала должны размещаться на разных дисковых устройствах для повышения производительности операций ввода-вывода и отказоустойчивости.

Параметры сортировки должны соответствовать региональным настройкам и требованиям приложения: для белорусских и российских проектов обычно используется Cyrillic_General_CI_AS (регистронезависимое сравнение с учётом диакритических знаков) или Cyrillic_General_CS_AS (регистрозависимое сравнение).

Аббревиатура CI означает регистронезависимое сравнение (Case-Insensitive), а CS – регистрозависимое (Case-Sensitive). Аналогично, AS указывает на учёт диакритических знаков (Accent-Sensitive), а AI – на их игнорирование (Accent-Insensitive). Для большинства приложений на русском языке рекомендуется использовать Cyrillic_General_CI_AS, которая обеспечивает регистронезависимое сравнение и учитывает диакритические знаки, что соответствует типичным требованиям поиска и сортировки данных на русском языке.

После создания базы данных рекомендуется создать пользовательскую схему для группировки объектов. Схема представляет собой контейнер для объектов БД, принадлежащий конкретному пользователю или роли. Использование схем позволяет логически структурировать объекты БД, что

особенно важно в крупных проектах. Создание схемы выполняется с помощью инструкции CREATE SCHEMA, которая может также указывать владельца схемы через параметр AUTHORIZATION.

Пример создания базы данных «AutoServiceDB» с настройками файлов и параметрами сортировки приведён ниже.

```
CREATE DATABASE AutoServiceDB
ON PRIMARY
( NAME = 'AutoServiceDB_Data',
  FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\DATA\AutoServiceDB_Data.mdf',
  SIZE = 10 MB,
  MAXSIZE = 100 MB,
  FILEGROWTH = 10 MB )
LOG ON
( NAME = 'AutoServiceDB_Log',
  FILENAME = 'D:\Logs\AutoServiceDB_Log.ldf',
  SIZE = 5 MB,
  MAXSIZE = 50 MB,
  FILEGROWTH = 5 MB )
COLLATE Cyrillic_General_CI_AS;
GO

/* Создание пользовательской схемы */
CREATE SCHEMA AutoService;
GO
```

В данном примере файл данных и файл журнала размещены на разных дисках (C: и D: соответственно), что повышает отказоустойчивость системы. Параметры роста заданы в мегабайтах, а не в процентах, что обеспечивает предсказуемое поведение при автоматическом увеличении размера файлов.

После создания базы данных и схемы следующим этапом разработки информационной системы является создание таблиц – основных структурных единиц базы данных, в которых непосредственно хранятся данные. Таблица представляет собой совокупность строк и столбцов, где каждый столбец имеет строго определённый тип данных, а каждая строка содержит конкретные значения, соответствующие этим типам. Процесс создания таблиц в MS SQL Server выполняется с помощью инструкции CREATE TABLE, которая определяет имя таблицы, имена и типы данных столбцов, а также ограничения целостности, такие как первичные ключи, внешние ключи, ограничения на допустимые значения (CHECK) и ограничения уникальности (UNIQUE).

Рассмотрим пример создания таблиц для базы данных «Автосервис»:

```
CREATE TABLE AutoService.Customers (
CustomerID INT NOT NULL IDENTITY(1,1) CONSTRAINT PK_Customers
```

```
PRIMARY KEY,
        FullName NVARCHAR(100) NOT NULL,
       Phone NVARCHAR(20) NOT NULL CONSTRAINT
UQ_Customers_Phone UNIQUE,
        Email NVARCHAR(50) NULL,
        RegistrationDate DATETIME NOT NULL DEFAULT GETDATE(),
        IsActive BIT NOT NULL DEFAULT 1
    );
```

При создании таблиц следует учитывать ряд важных аспектов. Прежде всего, необходимо правильно выбрать типы данных для каждого столбца: для целочисленных идентификаторов используется INT или BIGINT, для строковых данных – NVARCHAR (если требуется поддержка Unicode, что актуально для кириллицы) или VARCHAR, для дат и времени – DATE, DATETIME или DATETIME2, для денежных значений – DECIMAL или MONEY. Каждому столбцу должно быть явно задано ограничение NULL или NOT NULL в зависимости от того, допускает ли столбец отсутствие значения. Для столбцов, которые должны автоматически генерировать уникальные значения (например, идентификаторы записей), используется свойство IDENTITY, которое указывает начальное значение и шаг приращения.

Ограничения целостности (CONSTRAINT) играют ключевую роль в поддержании корректности данных. Первичный ключ (PRIMARY KEY) обеспечивает уникальность идентификации каждой строки таблицы и не допускает NULL-значений. Внешний ключ (FOREIGN KEY) устанавливает связь между таблицами, обеспечивая ссылочную целостность. Ограничение UNIQUE гарантирует, что значения в указанном столбце или комбинации столбцов не повторяются, но при этом допускает одно значение NULL. Ограничение CHECK позволяет задать условие, которому должны удовлетворять значения в столбце, например, возраст должен быть больше 0 или статус заказа может принимать только определённые значения.

При проектировании таблиц необходимо также учитывать нормализацию данных. В соответствии с требованиями третьей нормальной формы (3NF) каждая таблица должна хранить данные только об одной сущности, каждый столбец должен зависеть только от первичного ключа, а не от других столбцов. Это позволяет избежать избыточности данных и аномалий при обновлении, вставке и удалении записей.

Задание

Для курсового проекта необходимо создать базу данных в MS SQL Server.

При создании БД следует выбрать параметры размещения файлов и журнала: файлу данных присвоить логическое имя [НазваниеБД]_Data, файлу журнала – [НазваниеБД]_Log. Начальные размеры должны быть установлены в 10 МБ для файла данных и 5 МБ для журнала. Параметры автоматического роста – 10 МБ для данных и 5 МБ для журнала, максимальный размер – 100 МБ для файла данных и 50 МБ для журнала (в зависимости от задания допускается

использование UNLIMITED).

Далее, на основе логической модели данных, созданной в рамках курсового проектирования, необходимо разработать структуру таблиц. После создания всех таблиц рекомендуется проверить структуру созданных таблиц с помощью системных представлений INFORMATION_SCHEMA.TABLES и INFORMATION_SCHEMA.COLUMNS. Убедиться в успешном создании БД и таблиц можно, проверив наличие новых объектов в обозревателе объектов SSMS.

Содержание отчета: тема и цель работы; SQL-код создания базы данных (инструкция CREATE DATABASE с полным набором параметров); SQL-код создания всех таблиц с указанием ограничений; скриншот обозревателя объектов с отображением созданной базы данных и таблиц.

Контрольные вопросы

1 Какова структура файлов базы данных в MS SQL Server? Для чего нужен каждый файл?

2 Какие параметры задаются при создании базы данных с помощью инструкции CREATE DATABASE?

3 Как выполнить проверку корректности создания базы данных?

4 Для чего нужно размещать файлы данных и журнала транзакций на разных физических дисках?

5 Какие команды T-SQL используются для изменения размера файлов базы данных?

6 Практическое занятие № 6. Создание связей между таблицами средствами T-SQL для своего курсового проекта

Цель работы: освоить практические навыки создания связей (внешних ключей) между таблицами в реляционной базе данных с использованием языка Transact-SQL и графических средств SQL Server Management Studio.

Теоретические положения

В реляционных базах данных связи между таблицами обеспечиваются посредством внешних ключей (Foreign Key). Внешний ключ представляет собой поле или совокупность полей в таблице, которые ссылаются на первичный ключ или уникальный ключ другой таблицы. Основное назначение внешних ключей – реализация ссылочной целостности данных, то есть гарантия того, что значения в дочерней таблице всегда соответствуют существующим значениям в родительской таблице. Помимо обеспечения целостности, внешние ключи позволяют организовать каскадное обновление и удаление записей, автоматически

поддерживая согласованность данных при изменениях в родительской таблице.

Для создания внешнего ключа используется либо инструкция CREATE TABLE (при создании таблицы), либо ALTER TABLE (для уже существующей таблицы). В синтаксисе можно выделить следующие ключевые элементы. CONSTRAINT задаёт имя ограничения внешнего ключа (рекомендуется использовать формат FK_дочерняя_таблица_родительская_таблица, например FK_Orders_Customers). FOREIGN KEY определяет столбец или столбцы дочерней таблицы, которые ссылаются на родительскую таблицу. REFERENCES указывает родительскую таблицу и её столбец (или столбцы), на которые ссылается внешний ключ; при этом родительский столбец обязательно должен быть первичным ключом или иметь ограничение уникальности. Параметры ON DELETE и ON UPDATE определяют действия, которые выполняются при удалении или обновлении записи в родительской таблице соответственно.

Рассмотрим подробнее варианты каскадных действий, доступные в MS SQL Server. Действие NO ACTION (используемое по умолчанию) запрещает удаление или обновление родительской записи, если существуют ссылающиеся на неё дочерние записи; при попытке выполнить такое изменение сервер генерирует ошибку. Действие CASCADE обеспечивает каскадное удаление или обновление дочерних записей при удалении или обновлении родительской записи. Действие SET NULL устанавливает значения внешнего ключа в NULL при удалении или обновлении родительской записи, однако для этого требуется, чтобы столбец внешнего ключа допускал хранение NULL-значений. Действие SET DEFAULT устанавливает значения внешнего ключа в значение по умолчанию при удалении или обновлении родительской записи, при этом столбец внешнего ключа должен иметь заданное значение по умолчанию.

Выбор конкретного каскадного действия определяется бизнес-логикой предметной области. Например, для таблиц, где удаление родительской записи логически влечёт удаление дочерних (удаление заказа влечёт удаление позиций заказа), используется ON DELETE CASCADE. Для таблиц, где удаление родительской записи должно быть запрещено при наличии дочерних (удаление клиента, у которого есть заказы), используется ON DELETE NO ACTION. Использование ON UPDATE CASCADE оправдано, если первичные ключи могут изменяться, хотя на практике первичные ключи чаще всего являются неизменяемыми искусственными идентификаторами.

При создании внешних ключей необходимо учитывать ряд технических требований. Во-первых, родительский столбец должен иметь ограничение PRIMARY KEY или UNIQUE. Во-вторых, столбцы внешнего и родительского ключей должны иметь одинаковые типы данных. В-третьих, при составном внешнем ключе порядок столбцов в FOREIGN KEY и REFERENCES должен совпадать. В одной таблице может существовать несколько внешних ключей, ссылающихся на разные таблицы. Внешние ключи могут быть созданы как непосредственно в инструкции CREATE TABLE, так и с помощью ALTER TABLE после создания обеих связанных таблиц; последний способ предпочтительнее, если таблицы уже содержат данные, поскольку позволяет контролировать процесс проверки существующих данных на соответствие

новому ограничению.

Рассмотрим пример создания связей для базы данных «Автосервис». В таблице 1 представлены сущности, между которыми необходимо установить связи.

Таблица 1 – Сущности базы данных «Автосервис»

Имя таблицы	Описание	Первичный ключ
Customers	Клиенты автосервиса	CustomerID
Orders	Заказы на обслуживание	OrderID
OrderItems	Позиции заказа (услуги)	OrderItemID
Services	Справочник услуг	ServiceID
Employees	Сотрудники автосервиса	EmployeeID

Далее приведён SQL-код создания внешних ключей для указанных таблиц.

```
/* Связь между заказами и клиентами */
ALTER TABLE Orders
ADD CONSTRAINT FK_Orders_Customers
FOREIGN KEY (CustomerID)
REFERENCES Customers (CustomerID)
ON DELETE NO ACTION
ON UPDATE NO ACTION;

/* Связь между позициями заказа и заказами (каскадное удаление) */
ALTER TABLE OrderItems
ADD CONSTRAINT FK_OrderItems_Orders
FOREIGN KEY (OrderID)
REFERENCES Orders (OrderID)
ON DELETE CASCADE
ON UPDATE NO ACTION;

/* Связь между позициями заказа и услугами */
ALTER TABLE OrderItems
ADD CONSTRAINT FK_OrderItems_Services
FOREIGN KEY (ServiceID)
REFERENCES Services (ServiceID)
ON DELETE NO ACTION
ON UPDATE NO ACTION;

/* Связь между заказами и сотрудниками (исполнителями) */
ALTER TABLE Orders
ADD CONSTRAINT FK_Orders_Employees
FOREIGN KEY (EmployeeID)
```

```
REFERENCES Employees (EmployeeID)
ON DELETE SET NULL
ON UPDATE NO ACTION;
```

В данном примере для связи OrderItems-Orders выбрано каскадное удаление, поскольку при удалении заказа его позиции теряют смысл. Для связи Orders-Employees выбрано действие SET NULL, так как при увольнении сотрудника заказы, выполненные им, остаются в системе, но исполнитель становится неизвестным. Для остальных связей выбрано NO ACTION, поскольку удаление клиента, услуги или сотрудника, на которых ссылаются другие записи, должно быть запрещено для сохранения целостности данных.

Задание

Для разрабатываемой в курсовом проекте базы данных необходимо создать все связи между таблицами в соответствии с логической моделью данных.

Содержание отчета: тема и цель работы; полный SQL-код создания всех внешних ключей; таблица с описанием всех связей (родительская таблица, дочерняя таблица, столбцы, каскадные действия); обоснование выбора каскадных действий для каждой связи; скриншот диаграммы БД с отображением связей.

Контрольные вопросы

1 Что такое внешний ключ и для каких целей он используется?

2 Какие требования предъявляются к столбцам, участвующим во внешнем ключе?

3 Какие каскадные действия поддерживаются в MS SQL Server при создании внешних ключей?

4 В каких случаях целесообразно использовать ON DELETE CASCADE?

5 Чем отличается NO ACTION от CASCADE при удалении записи?

6 Что произойдёт, если попытаться удалить запись из родительской таблицы, имеющей дочерние записи, при ON DELETE NO ACTION?

7 Как добавить внешний ключ в уже существующую таблицу с данными? Какие могут возникнуть проблемы?

8 В чём разница между созданием внешнего ключа в CREATE TABLE и через ALTER TABLE?

7 Практическое занятие № 7. Создание sql-скрипта заполнения базы данных в своем курсовом проекте

Цель работы: научиться создавать SQL-скрипты для наполнения базы данных тестовыми данными с использованием языка T-SQL.

Теоретические положения

После создания структуры базы данных (таблиц, связей, ограничений целостности) возникает необходимость наполнить её данными для тестирования работоспособности, отладки запросов и демонстрации функциональности разрабатываемой информационной системы. Для этих целей используются SQL-скрипты заполнения (Data Population Scripts), формируемые путем ручного составления INSERT-запросов.

Скрипт заполнения базы данных представляет собой последовательность инструкций T-SQL, выполняющих вставку данных в таблицы. Основным инструментом для этого служит команда INSERT INTO.

Синтаксис команды INSERT INTO при ручном составлении запроса:

```
INSERT INTO [schema.]table_name (column1, column2, ...)
VALUES (value1, value2, ...)
```

При ручном составлении INSERT-запросов рекомендуется:

1) явно перечислять все столбцы таблицы, в которые производится вставка (это повышает читаемость скрипта и делает его устойчивым к возможным изменениям структуры таблицы);

2) для каждого значения использовать литералы соответствующего типа данных (строковые значения заключать в одинарные кавычки, числовые записывать без кавычек, для дат использовать формат 'ГГГГ-ММ-ДД');

3) при вставке нескольких записей в одну таблицу перечислять блоки VALUES через запятую;

4) для столбцов, имеющих значение по умолчанию (DEFAULT) или допускающих NULL, можно явно указывать ключевые слова DEFAULT или NULL.

При вставке данных необходимо учитывать ограничения, наложенные на таблицы.

1 Первичные ключи (PRIMARY KEY) – значения должны быть уникальными и не равными NULL. Если для столбца первичного ключа установлено свойство IDENTITY, значение вставлять не требуется – оно генерируется автоматически.

2 Внешние ключи (FOREIGN KEY) – значения должны существовать в родительской таблице, что определяет порядок заполнения: сначала заполняются справочные (родительские) таблицы, затем – дочерние.

3 Ограничения CHECK, NOT NULL – вставляемые значения должны им

соответствовать.

4 Ограничение UNIQUE – значения должны быть уникальными в пределах столбца или комбинации столбцов.

Правильный порядок заполнения таблиц является критически важным при ручном составлении скрипта. Нарушение порядка приведет к ошибкам выполнения из-за нарушения ссылочной целостности. Типовой порядок заполнения:

1) справочные (независимые) таблицы, не имеющие внешних ключей;

2) таблицы, имеющие внешние ключи только на уже заполненные таблицы;

3) таблицы-связи («многие-ко-многим») заполняются последними, после заполнения всех связываемых таблиц.

Важным аспектом при создании скрипта заполнения является транзакционная целостность. Все операции вставки рекомендуется выполнять в рамках одной транзакции с возможностью отката в случае ошибки. Для группировки операций используются конструкции BEGIN TRANSACTION, COMMIT и ROLLBACK.

Для предотвращения повторной вставки одних и тех же данных при многократном выполнении скрипта рекомендуется использовать конструкцию IF NOT EXISTS для проверки наличия записей в таблице перед вставкой:

```
IF NOT EXISTS (SELECT 1 FROM TableName WHERE condition)
BEGIN
   INSERT INTO TableName (column1, column2) VALUES (value1, value2)
END
```

Пример заполнения:

```
USE LibraryDB
GO

SET NOCOUNT ON
GO

BEGIN TRANSACTION

-- Заполнение таблицы "Издательства" (родительская, без внешних ключей)
INSERT INTO Publishers (publisher_name, publisher_address, publisher_phone)
VALUES
   ('Просвещение', 'г. Москва, ул. Тверская, д. 15', '+7 (495) 123-45-67'),
   ('Наука', 'г. Санкт-Петербург, Невский пр., д. 25', '+7 (812) 234-56-78'),
   ('Эксмо', 'г. Москва, ул. Ленинская, д. 7', '+7 (495) 345-67-89')

-- Заполнение таблицы "Авторы" (родительская, без внешних ключей)
INSERT INTO Authors (author_last_name, author_first_name, author_middle_name, author_birth_date)
VALUES
```

```
    ('Иванов', 'Иван', 'Иванович', '1965-03-15'),
    ('Петров', 'Петр', 'Петрович', '1972-07-22'),
    ('Сидорова', 'Анна', 'Михайловна', '1980-11-05')

-- Заполнение таблицы "Читатели" (родительская, без внешних ключей)
INSERT INTO Readers (reader_last_name, reader_first_name, reader_middle_name, reader_address, reader_phone, reader_registration_date)
VALUES
    ('Кузнецов', 'Алексей', 'Владимирович', 'г. Москва, ул. Строителей, д. 10, кв. 5', '+7 (915) 111-22-33', '2025-09-01'),
    ('Смирнова', 'Елена', 'Викторовна', 'г. Москва, ул. Мира, д. 20, кв. 15', '+7 (916) 222-33-44', '2025-09-15'),
    ('Попов', 'Дмитрий', 'Сергеевич', 'г. Москва, ул. Садовая, д. 30, кв. 8', '+7 (917) 333-44-55', '2025-10-01')

-- Заполнение таблицы "Книги" (дочерняя, имеет внешний ключ на издательство)
INSERT INTO Books (book_title, book_publication_year, book_pages_count, book_circulation, book_price, publisher_id)
VALUES
    ('Математика. 5 класс', 2023, 320, 10000, 450.00, 1),
    ('Физика. 7 класс', 2022, 280, 8000, 520.00, 1),
    ('История России', 2023, 400, 5000, 680.00, 2),
    ('Программирование на Python', 2024, 550, 3000, 890.00, 3),
    ('Английский язык', 2023, 350, 12000, 390.00, 2)
-- Заполнение таблицы-связи "Книги-Авторы" (дочерняя, имеет внешние ключи на книги и авторов)
INSERT INTO Book_Authors (book_id, author_id)
VALUES
    (1, 1),
    (2, 1),
    (3, 2),
    (4, 3),
    (5, 3)

-- Заполнение таблицы "Выдача книг" (дочерняя, имеет внешние ключи на читателей и книги)
INSERT INTO Book_Issuance (reader_id, book_id, issuance_date, return_deadline, actual_return_date, is_returned)
VALUES
    (1, 1, '2025-09-05', '2025-10-05', '2025-10-03', 1),
    (1, 2, '2025-09-05', '2025-10-05', NULL, 0),
    (2, 3, '2025-09-20', '2025-10-20', '2025-10-18', 1),
    (3, 5, '2025-10-05', '2025-11-05', NULL, 0)
```

```
COMMIT TRANSACTION
GO
```

Задание

В разрабатываемой базе данных необходимо создать SQL-скрипт заполнения, составленный вручную с использованием INSERT-запросов, содержащий:

1) минимальный набор тестовых данных для всех таблиц разработанной базы данных;

2)заполнение должно выполняться в правильном порядке (сначала родительские таблицы, затем дочерние с учетом иерархии внешних ключей);

3) для вставки данных в каждую таблицу использовать явное перечисление всех столбцов (за исключением столбцов с типом IDENTITY);

4) использовать транзакции для группировки операций вставки;

5) реализовать проверку на существование данных перед вставкой, используя конструкцию IF NOT EXISTS;

6) все значения должны быть согласованы между связанными таблицами (значения внешних ключей должны ссылаться на реально существующие записи в родительских таблицах).

Содержание отчета: тема и цель работы; ER-диаграмма БД; SQL-код скрипта заполнения; результат выполнения скрипта (скриншоты); вывод по работе.

Контрольные вопросы

1 Какова цель создания скрипта заполнения базы данных?

2 В каком порядке должны заполняться таблицы, имеющие связи по внешним ключам, и почему?

3 Что произойдет при попытке вставить запись в дочернюю таблицу, если значение внешнего ключа отсутствует в родительской таблице?

4 Почему при ручном составлении INSERT-запросов рекомендуется явно перечислять все столбцы?

5 Для чего используется конструкция SET NOCOUNT ON?

6 Как проверить, существуют ли уже данные в таблице, перед их вставкой, с помощью IF NOT EXISTS?

7 Какие ограничения целостности (PRIMARY KEY, FOREIGN KEY, CHECK, UNIQUE, DEFAULT, NOT NULL) могут препятствовать вставке данных и как это учитывать при формировании скрипта?

8 Что произойдет, если нарушить порядок очистки таблиц (сначала попытаться удалить данные из родительской таблицы, а затем из дочерней)?

8 Практическое занятие № 8. Язык T-SQL. Добавление, изменение и удаление данных в таблицах средствами T-SQL в своей базе данных по курсовому проектированию

Цель работы: научиться выполнять операции добавления, изменения и удаления данных в таблицах базы данных с использованием языка T-SQL.

Теоретические положения

Язык T-SQL (Transact-SQL) является расширением стандарта SQL, разработанным корпорацией Microsoft для работы с реляционными базами данных в среде SQL Server. Одной из ключевых задач при эксплуатации базы данных является управление данными, которое включает три основные операции: добавление новых записей (INSERT), изменение существующих (UPDATE) и удаление записей (DELETE). Эти операции относятся к группе DML (Data Manipulation Language – язык манипулирования данными).

1 Добавление данных (INSERT).

Команда INSERT предназначена для добавления новых строк в таблицу.

```
INSERT INTO [schema.]table_name (column1, column2, ..., columnN)
VALUES
   (value1_1, value1_2, ..., value1_N),
   (value2_1, value2_2, ..., value2_N),
   ...
```

При вставке данных необходимо соблюдать следующие правила:

1) значения должны соответствовать типам данных столбцов;

2) для столбцов, имеющих ограничение NOT NULL, значение обязательно должно быть указано;

3) для столбцов с ограничением UNIQUE значения должны быть уникальными;

4) значения внешних ключей должны существовать в родительской таблице;

5) если столбец имеет свойство IDENTITY, его значение генерируется автоматически, и указывать его в списке вставки не требуется (однако, при необходимости, можно явно включить вставку в столбец IDENTITY с помощью команды SET IDENTITY_INSERT table_name ON).

Пример

```
INSERT INTO Students (student_last_name, student_first_name, student_middle_name, student_group, student_birth_date)
VALUES ('Петров', 'Петр', 'Петрович', 'ИС-21', '2003-05-15')
```

2 Изменение данных (UPDATE).

Команда UPDATE предназначена для изменения существующих данных в таблице. Она позволяет обновить значения одного или нескольких столбцов в одной или нескольких строках.

Синтаксис:

```
UPDATE [schema.]table_name
SET column1 = value1, column2 = value2, ..., columnN = valueN
[WHERE condition]
```

Ключевые моменты при использовании UPDATE:

1) если условие WHERE не указано, изменения будут применены ко всем строкам таблицы;

2) при обновлении столбцов с внешними ключами новое значение должно существовать в родительской таблице;

3) можно обновлять несколько столбцов одновременно;

4) в выражении SET можно использовать арифметические операции и функции;

5) обновление первичного ключа допустимо, но должно обеспечивать уникальность и ссылочную целостность.

Примеры

```
-- Изменение адреса студента по условию
UPDATE Students
SET student_address = 'г. Москва, ул. Ленина, д. 10'
WHERE student_id = 5

-- Увеличение цены всех книг на 10%
UPDATE Books
SET book_price = book_price * 1.1

-- Изменение нескольких полей
UPDATE Books
SET book_circulation = 15000, book_price = 500.00
WHERE book_title LIKE '%Математика%'
```

3 Удаление данных (DELETE).

Команда DELETE предназначена для удаления строк из таблицы.

Синтаксис:

```
DELETE FROM [schema.]table_name
[WHERE condition]
```

Ключевые моменты при использовании DELETE:

1) если условие WHERE не указано, будут удалены *все* строки таблицы;

2) удаление записей, на которые ссылаются внешние ключи из других таблиц, может быть ограничено;

3) операция DELETE является транзакционной и может быть откачена.

Примеры

```
-- Удаление конкретной записи
DELETE FROM Students
WHERE student_id = 10

-- Удаление записей по условию
DELETE FROM Book_Issuance
WHERE is_returned = 1 AND actual_return_date < '2025-01-01'

-- Удаление всех записей таблицы
DELETE FROM Temporary_Data
```

4 Альтернатива DELETE – команда TRUNCATE.

Команда TRUNCATE TABLE удаляет все строки из таблицы, но в отличие от DELETE:

1) не регистрирует удаление каждой строки в журнале транзакций (только освобождение страниц данных);

2) выполняется быстрее, особенно для больших таблиц;

3) сбрасывает счетчик IDENTITY (если он есть) до начального значения;

4) не может использоваться, если на таблицу ссылаются внешние ключи;

5) не может использоваться с условием WHERE.

Синтаскис:

```
TRUNCATE TABLE [schema.]table_name
```

5 Использование транзакций при модификации данных.

Для обеспечения целостности данных при выполнении операций модификации рекомендуется использовать транзакции. Это позволяет либо применить все изменения, либо откатить их в случае ошибки.

Пример

```
BEGIN TRANSACTION

-- Выполнение операций модификации
UPDATE Accounts SET balance = balance - 1000 WHERE account_id = 1
UPDATE Accounts SET balance = balance + 1000 WHERE account_id = 2
```

```
-- Проверка условий
IF (SELECT balance FROM Accounts WHERE account_id = 1) >= 0
   COMMIT TRANSACTION
ELSE
   ROLLBACK TRANSACTION
```

6 Особенности работы с NULL-значениями.

При выполнении операций модификации данных необходимо учитывать особенности работы с NULL:

1) при вставке данных столбцу можно явно присвоить значение NULL (если это разрешено);

2) при обновлении можно установить значение NULL;

3) для проверки на NULL используется оператор IS NULL или IS NOT NULL, а не ‘=’.

Ниже приведены примеры скриптов для добавления, изменения и удаления данных в библиотечной системе:

```
USE LibraryDB
GO

-- Добавление нескольких записей в таблицу читателей
INSERT INTO Readers (reader_last_name, reader_first_name,
reader_middle_name, reader_address, reader_phone, reader_registration_date)
VALUES
   ('Соколов', 'Игорь', 'Александрович', 'г. Москва, ул. Лесная, д. 15, кв. 7',
'+7 (925) 111-22-33', GETDATE()),
   ('Михайлова', 'Ольга', 'Сергеевна', 'г. Москва, ул. Полевая, д. 3, кв. 12', '+7
(926) 222-33-44', GETDATE())

-- Добавление данных из другой таблицы (копирование)
INSERT INTO Books_Archive (book_title, book_publication_year, book_price)
SELECT book_title, book_publication_year, book_price
FROM Books
WHERE book_publication_year < 2020

-- Изменение контактных данных читателя
UPDATE Readers
SET reader_address = 'г. Москва, ул. Новая, д. 20, кв. 5',

_pages_count BETWEEN 300 AND 500 THEN book_price * 1.10
   ELSE book_price * 1.15
END
WHERE book_publication_year = 2024

-- Удаление читателя по идентификатору
DELETE FROM Readers
```

```
WHERE reader_id = 10

-- Удаление книг с тиражом менее 1000 экземпляров
DELETE FROM Books
WHERE book_circulation < 1000

-- Удаление записей о выдаче книг, возвращенных более года назад
DELETE FROM Book_Issuance
WHERE is_returned = 1 AND actual_return_date < DATEADD(year, -1,
GETDATE())

-- Удаление с использованием EXISTS (удаление книг без авторов)
DELETE FROM Books
WHERE NOT EXISTS (
   SELECT 1 FROM Book_Authors WHERE Book_Authors.book_id =
Books.book_id
)

-- Полная очистка временной таблицы (нельзя откатить)
TRUNCATE TABLE Temp_Books
```

Задание

Разработать SQL-скрипт, содержащий операции добавления (INSERT), изменения (UPDATE) и удаления (DELETE) данных в таблицах разработанной базы данных. Скрипт должен демонстрировать различные варианты каждой из операций и выполняться с использованием транзакций для обеспечения целостности данных.

Содержание отчета: тема и цель работы; ER-диаграмма БД; SQL-код скрипта; результаты выполнения скрипта (скриншоты); вывод по работе.

Контрольные вопросы

1 Какие существуют способы добавления данных с помощью команды INSERT? Чем отличается вставка с явным перечислением столбцов от вставки без него?

2 В чем разница между командами DELETE и TRUNCATE? В каких случаях применение TRUNCATE невозможно?

3 Какую роль играют транзакции при модификации данных? Приведите пример использования BEGIN TRANSACTION, COMMIT и ROLLBACK.

4 Что произойдет, если в командах UPDATE или DELETE не указать условие WHERE? Какие проверки необходимо выполнять перед массовыми операциями изменения или удаления данных?

5 Какие ограничения целостности (PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE, CHECK) могут препятствовать выполнению операций INSERT, UPDATE и DELETE?

Список литературы

1 **Кузин, А. В.** Базы данных : учеб. пособие / А. В. Кузин, С. В. Левонисова. – 6-е изд., стер. – М. : Академия, 2016. – 320 с.

2 **Аверченков, В. И.** Информационная безопасность сетей и систем : учеб. пособие / В. И. Аверченков, В. Т. Еременко, Е. А. Зайченко. – Могилев : Бел.-Рос. ун-т, 2020. – 212 с.

3 **Агальцов, В. П.** Базы данных: учебник: в 2 т. Т. 2: Распределенные и удаленные базы данных / В. П. Агальцов. – М. : Форум ; ИНФРА-М, 2012. – 272 с.: ил.

4 **Советов, Б. Я.** Базы данных: теория и практика : учебник / Б. Я. Советов. – 2-е изд. – М. : Юрайт, 2012. – 463 с.

5 **Куликов, С. С.** Работа с MySQL, MS SQL Server и Oracle в примерах : практ. пособие / С. С. Куликов. – Мн. : БОФФ, 2019. – 545 с.

6 **Полищук, Ю. В.** Базы данных и их безопасность : учеб. пособие / Ю. В. Полищук, А. С. Боровский. – М. : ИНФРА-М, 2022. – 210 с.

7 **Бен-Ган, И.** Microsoft SQL Server 2012. Создание запросов : учеб. курс Microsoft : пер. с англ. / И. Бен-Ган, Д. Сарка, Р. Талмейдж. – М. : Рус. редакция, 2015. – 720 с. : ил.

8 **Куликов, С. С.** Реляционные базы данных в примерах : практ. пособие / С. С. Куликов. – Мн. : Четыре четверти, 2020. – 424 с.

9 **Гвоздева, Т. В.** Проектирование информационных систем: технология автоматизированного проектирования. Лабораторный практикум : учеб.-справ. пособие / Т. В. Гвоздева, Б. А. Баллод. – СПб. ; М. ; Краснодар : Лань, 2018. – 156 с. : ил.