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

Кафедра информатики и информационных систем

Составитель О. С. Семенова

Проектирование и разработка баз данных

Методические материалы

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

Кемерово 2026

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

Семенова Ольга Сергеевна
Проектирование и разработка баз 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 (1032 КБ). – Текст : электронный.

Методические материалы по дисциплине Проектирование и разработка баз данных: методические материалы для обучающихся специальности СПО 09.02.11 «Разработка и управление программным обеспечением» содержат указания для проведения лабораторных работ и самостоятельных работ.

Лабораторная работа №1
Модели данных

Цель работы: Изучить модели представления данных.

Теоретические положения

В классической теории баз данных, модель данных есть формальная теория представления и обработки данных в системе управления базами данных (СУБД), которая включает, по меньшей мере, три аспекта:

1) аспект структуры: методы описания типов и логических структур данных в базе данных;

2) аспект манипуляции: методы манипулирования данными;

3) аспект целостности: методы описания и поддержки целостности базы данных.

Аспект структуры определяет, что из себя логически представляет база данных, аспект манипуляции определяет способы перехода между состояниями базы данных (то есть способы модификации данных) и способы извлечения данных из базы данных, аспект целостности определяет средства описаний корректных состояний базы данных.

К классическим моделям данных относятся: иерархическая; сетевая; реляционная. Помимо этого, в последние годы стали появляться и активно внедряться на практике следующие модели данных: постреляционная; многомерная; объектно-ориентированная. Разрабатывают также всевозможные системы, которые основаны на других моделях данных, расширяющих известные модели. К ним относят; объектно-реляционные, дедуктивно-объектно-ориентированные, семантические, концептуальные и ориентированные модели.

Иерархическая модель. Первая версия СУБД появилась в 1968 г. Она содержала модель, представляющую собой упорядоченные наборы деревьев. Данная модель данных построена по принципу иерархии типов объектов (один тип объекта – главный, другие – подчиненные. Главный и подчиненные объекты связаны по типу «один ко многим». Для каждого подчиненного типа объекта имеется лишь один исходный тип объекта. Основной недостаток данной модели – достаточно длительный поиск необходимой информации.

Сетевая модель. В данной модели любой объект может быть как главным, так и подчиненным. Каждый объект имеет возможность участвовать в любом числе взаимодействий. Другими словами, любая информационная единица может иметь множество предков и множество потомков. В моделях подобного рода связи заложены внутри описаний объектов. Достоинством является гибкость модели, т.е. имеется возможность повышения быстродействия системы. Недостатком является нагрузка на информационные ресурсы.

Реляционная модель данных. Свое название получила от английского термина relation, что означает «отношение». При соблюдении определенных условий отношение можно представить в виде двумерной привычной для человека таблицы.

При табличной организации данных отсутствует иерархия элементов. Строки и столбцы могут быть просмотрены в любом порядке, поэтому высока гибкость выбора любого подмножества элементов в строках и столбцах. Любая таблица в реляционной базе состоит из строк, которые называют записями, и столбцов, которые называют полями. На пересечении строк и столбцов находятся конкретные значения данных. Для каждого поля определяется множество его значений.

В реляционной модели данных применяются разделы реляционной алгебры, откуда и была заимствована соответствующая терминология. В реляционной алгебре поименованный столбец отношения называется атрибутом, а множество всех возможных значений конкретного атрибута – доменом. Строки таблицы со значениями разных атрибутов называют кортежами. Атрибут, значение которого однозначно идентифицирует кортежи, называется ключевым (или просто ключом). Так ключевое поле – это такое поле, значения которого в данной таблице не повторяется. В отличие от иерархической и сетевой моделей данных в реляционной отсутствует понятие группового отношения. Для отражения ассоциаций между кортежами разных отношений используется дублирование их ключей. Сложный ключ выбирается в тех случаях, когда ни одно поле таблицы однозначно не определяет запись.

Записи в таблице хранятся упорядоченными по ключу. Ключ может быть простым, состоящим из одного поля, и сложным, состоящим из нескольких полей. Сложный ключ выбирается в тех случа-

ях, когда ни одно поле таблицы однозначно не определяет запись.

Кроме первичного ключа в таблице могут быть вторичные ключи, называемые еще внешними ключами, или индексами. Индекс – это поле или совокупность полей, чьи значения имеются в нескольких таблицах и которое является первичным ключом в одной из них. Значения индекса могут повторяться в некоторой таблице. Индекс обеспечивает логическую последовательность записей в таблице, а также прямой доступ к записи.

Основная масса современных БД для компьютеров являются реляционными. Достоинства реляционной модели данных – это простота, удобство реализации на ЭВМ, наличие теоретического обоснования и возможность формирования гибкой схемы БД, которая допускает настройку при формировании запросов. Подобная модель используется, как правило, в базах данных среднего размера. При увеличении количества таблиц в базе данных снижается скорость работы с ней. Возникают также проблемы при создании систем со сложными структурами данных (например, систем автоматизации проектирования).

Объектно-ориентированные БД включают в свой состав 2 модели данных: реляционную и сетевую, и применяются при создании крупных баз данных со сложными структурами. Объектно-ориентированная и объектно-реляционная модели данных появились в результате распространения объектно-ориентированного подхода в программировании. Объектная модель данных предлагает рассматривать БД как множество объектов, обладающих свойствами инкапсуляции, наследования и т.д.

Постреляционная модель данных представляет собой расширенную реляционную модель, снимающую ограничение неделимости данных, хранящихся в записях таблиц. Постреляционная модель данных допускает многозначные поля – поля, значения которых состоят из подзначений. Набор значений многозначных полей считается самостоятельной таблицей, встроенной в основную таблицу.

Многомерная модель. Многомерные СУБД являются узкоспециализированными СУБД, предназначенными для интерактивной аналитической обработки информации. Основные свойства, присущие к этим СУБД: агрегируемость, историчность и прогнозируемость данных.

Задания к лабораторной работе №1

1. Построить иерархическую, сетевую модель данных. Указать корневой тип, подтипы, узлы.

а) Многоуровневая файловая система;

б) Объектная модель Word.

2. Построить реляционную модель данных. Указать атрибуты, кортежи, степень отношения, кардинальное число, домен.

а) Студенты КузГТУ;

б) Книги.

3. Построить многомерную модель данных.

а) Интенсивность движения автомобилей на перекрестках по часам суток;

б) Добыча полезных ископаемых в РФ;

4. Построить объектно-ориентированную модель данных.

а) Вуз;

б) Школа.

Лабораторная работа №2
Связи: вид, степень, кардинальность, направленность, обязательность

Цель работы: Изучить ключевые характеристики связей в системах управления базами данных (СУБД) – вид, степень, кардинальность, направленность и обязательность.

Теоретические положения

Сущность (таблица, отношение) – это представление набора реальных или абстрактных объектов (людей, вещей, мест, событий, идей, комбинаций и т. д.), которые можно выделить в одну группу, потому что они имеют одинаковые характеристики и могут принимать участие в похожих связях. Каждая сущность должна иметь наименование, выраженное существительным в единственном числе. Каждая сущность в модели изображается в виде прямоугольника с наименованием.

Можно сказать, что Сущности представляют собой множество реальных или абстрактных вещей (людей, объектов, событий, идей и т. д.), которые имеют общие атрибуты или характеристики.

Экземпляр сущности – это конкретный представитель данной сущности. Атрибут сущности – это именованная характеристика, являющаяся некоторым свойством сущности.

Связь (отношение, relationship) – логическая ассоциация между сущностями, отражающая смысловые взаимосвязи объектов предметной области. Выделяют категориальные связи, основанные на принципе наследования общих характеристик от обобщённой (родовой) сущности к специализированным (дочерним) сущностям и связи «родитель‑потомок» (parent‑child relationship), основанные на подчинении сущностей-потомков сущности-родителю.

Кардинальность связи – это количественная характеристика, определяющая, сколько экземпляров одной сущности может быть связано с экземплярами другой сущности.

Выделяют следующие типы кардинальности:

- «Один-к-одному», 1:1. Каждый экземпляр первой сущности связан не более чем с одним экземпляром второй сущности, и наоборот. Пример: сотрудник и его рабочее место (если каждое место закреплено за одним сотрудником).

- «Один-ко-многим», 1:N. Один экземпляр первой сущности может быть связан с несколькими экземплярами второй, но каждый экземпляр второй сущности связан только с одним экземпляром первой. Пример: отдел и сотрудники (в одном отделе много сотрудников, но каждый сотрудник работает только в одном отделе).
- «Многие-ко-многим», M:N. Каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй, и каждый экземпляр второй – с несколькими экземплярами первой. Пример: студенты и дисциплины (студент изучает несколько дисциплин, каждую дисциплину изучает несколько студентов).

Степень связи – количество сущностей, участвующих в отношении. Различают:

- Унарные (рекурсивные) – связь между экземплярами одной сущности. Пример: иерархия сотрудников (менеджер – подчинённый);
- Бинарные – связь между двумя сущностями;
- N-арные – связь между N сущностями (N > 2). Пример тернарной связи: поставка (поставщик, товар, склад).

Направленность связи указывает, как интерпретируется отношение между сущностями. Ненаправленная связь симметрична, порядок сущностей не важен. Пример: «сотрудник работает с сотрудником» (коллеги). Направленная – имеет чётко определённый «источник» и «цель». Сущность, из которой отношение исходит, называется родительской сущностью. Сущность, в которой отношение заканчивается, называется подчиненной (дочерней) сущностью. Пример: «менеджер руководит сотрудником» (направление: менеджер → сотрудник).

Обязательность связи определяет, должны ли все экземпляры сущности участвовать в связи. Обязательная связь характеризуется тем, что каждый экземпляр сущности должен быть связан хотя бы с одним экземпляром другой сущности. Пример: каждый заказ должен быть связан с клиентом. Необязательная связь характеризуется тем, что некоторые экземпляры сущности могут не участвовать в связи. Пример: клиент может не иметь заказов.

Связи между сущностями представляют в виде диаграммы ER-экземпляров (рисунок 1).

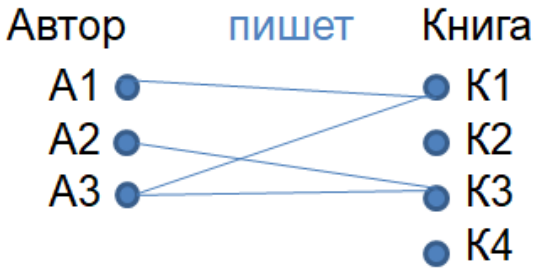


Рисунок 1 – Диаграмма ER-экземпляров

Здесь A1, A2, A3 – экземпляры сущности «Автор», K1, K2, K3, K4 – экземпляры сущности «Книга», линии показывают фактические связи «пишет» между конкретными экземплярами. Из диаграммы ER-экземпляров видно, что класс принадлежности сущности «Автор» обязательный, а сущности «Книга» – необязательный, кардинальность связи M:N.

Задания к лабораторной работе №2

1. Придумать примеры

- унарной (рекурсивной) связи
- бинарной связи (между 2-мя сущностями)
- тернарной связи (между 3-мя сущностями)

Нарисовать диаграммы ER-экземпляров.

2. Придумать примеры связей 1:1, 1:N, N:M. Нарисовать диаграммы ER-экземпляров.

3. Привести примеры обязательной и необязательной связи. Нарисовать диаграммы ER-экземпляров.

4. Сколько связей может быть между 2-мя сущностями. Например, между сущностями «Студент» и «Преподаватель»? Привести примеры связей.

Лабораторная работа №3
Проектирование реляционной БД. ER-метод

Цель работы: Изучить основные принципы проектирования реляционной базы данных ER-методом.

Теоретические положения

Проектирование базы данных (БД) – это процесс создания структуры БД, которая обеспечивает эффективное хранение, обработку и извлечение информации. Процесс обычно разделяют на три основных этапа:

1. Концептуальное (инфологическое) проектирование: создание абстрактной, не зависящей от СУБД модели данных на основе требований предметной области. Основной инструмент – ER-модель (Entity-Relationship model, модель «Сущность-Связь»).
2. Логическое проектирование: преобразование концептуальной модели в конкретную модель данных (например, реляционную). Определение таблиц, столбцов, доменов, первичных и внешних ключей.
3. Физическое проектирование: адаптация логической модели под конкретную СУБД.

Основными компонентами ER-модели являются: сущности, атрибуты и связи.

Сущность (Entity) – это реальный или абстрактный объект (человек, место, событие, концепция), информацию о котором необходимо хранить в БД. Каждая сущность является множеством подобных объектов – экземпляров сущности. Графическое обозначение сущности на ER-диаграмме – прямоугольник. Примеры: «Студент», «Группа», «Дисциплина», «Преподаватель».

Сущность обладает одним или несколькими атрибутами, которые однозначно идентифицируют каждый ее экземпляр (первичный ключ Primary Key, PK, альтернативный ключ Alternate Key, AK).

Сущность обладает одним или несколькими атрибутами, которые либо принадлежат ей, либо наследуются через отношение с другими сущностями (внешний ключ Foreign Key, FK). Атрибуты изображаются в виде списка их имен и меток внутри блока сущности. Каждый атрибут имеет уникальное имя. Именем атрибута является имя существительное. Атрибуты первичного ключа Primary

Key PK («табельный номер») располагаются в начале списка и отделяются от других атрибутов горизонтальной чертой (рис. 2).

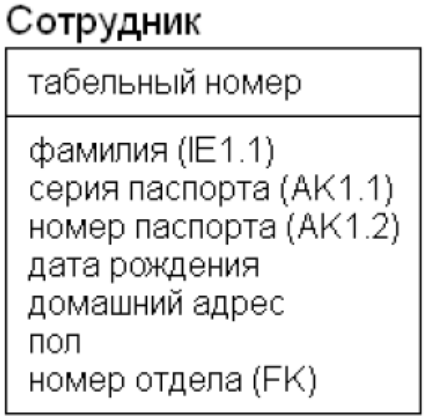


Рисунок 2 – Изображение сущности на ER-диаграмме

Связь (Relationship) – это ассоциация между двумя и более сущностями. Связь представляет собой бизнес-правило предметной области. Графическое обозначение связи – сплошная (идентифицирующая связь) или пунктирная (неидентифицирующая связь) линия.

Связи присваивается семантически значимое уникальное имя. Именем связи является глагольная форма. Связи «один ко многим» именуются со стороны родительской сущности. Связи «многие ко многим» – в обоих направлениях. Категориальные связи не именуются.

Связь между сущностями моделируется при помощи внешнего ключа. Внешний ключ – это первичный ключ родительской или обобщающей сущности, мигрировавший в сущность-потомок или категориальную сущность, соответственно.

Для связи может быть указана мощность (кардинальность), которая показывает, сколько экземпляров младшей сущности соответствуют одному экземпляру старшей сущности.

Выделяют идентифицирующие связи – связи типа «родитель-потомок», в которых потомок (зависимая сущность) однозначно определяется своей связью с родителем. Это означает, что экземпляр подчиненной сущности зависит от родительской сущности и не может существовать без экземпляра родительской сущности. В идентифицирующем отношении единственный экземпляр родительской сущности связан с множеством экземпляров подчиненной.

Неидентифицирующая связь связывает родительскую сущность с дочерней. Сущность-потомок в неидентифицирующей связи будет независимой от идентификатора родительской сущности.

Примером идентифицирующего отношения может служить связь сущностей «изготовитель – товар», а неидентифицирующего отношения – связь «книга – библиотека».

Выделяют следующие этапы проектирования БД с использованием ER-метода:

1. Выявление сущностей и их атрибутов на основе анализа требований заказчика к БД.

2. Определение связей между выделенными сущностями (можно использовать диаграммы ER-экземпляров).

3. Определение типов атрибутов (ключевые, составные и т.д.) и кардинальности связей.

4. Построение ER-диаграммы с указанием всех выявленных на предыдущем этапе связей.

5. Преобразование (маппинг) ER-диаграммы в реляционную схему выбранной СУБД. Каждая сущность преобразуется в таблицу (отношение). Атрибуты становятся столбцами таблицы. Ключевой атрибут становится первичным ключом (Primary Key, PK). Для моделирования связи «один-ко-многим» (1:M) в таблицу, соответствующую сущности на стороне «многих» (M), добавляется внешний ключ (Foreign Key, FK), ссылающийся на первичный ключ таблицы на стороне «один» (1). Для моделирования связи «многие-ко-многим» (M:N) создается новая таблица (ассоциативная сущность). Ее первичный ключ, как правило, составной и состоит из двух внешних ключей, ссылающихся на первичные ключи каждой из связанных сущностей. В эту таблицу могут быть добавлены собственные атрибуты, характеризующие связь (например, «Дата_сдачи», «Оценка» для связи между сущностями «Студент» и «Дисциплина»). Аналогичным способом моделируются N-арные связи. Для моделирования связи «один-к-одному» можно добавить внешний ключ в любую из таблиц, но обычно его размещают в таблице, которая является логически подчиненной.

Задания к лабораторной работе №3

1. Провести инфологическое проектирование предметной области «Студент». Предусмотреть хранение в БД следующей инфор-

мации: фамилия, имя, наименование дисциплины, преподаватель, оценка, год поступления в ВУЗ, группа, домашний адрес, школа, хобби, награды, дата рождения студента, E-mail студента, дата вручения награды, дата получения оценки, шифр дисциплины.

1.1. Сформировать набор сущностей

1.2. Сформировать перечень атрибутов каждой сущности

1.3. Выбрать первичный ключ каждой сущности

1.4. Установить связи между сущностями

2. Логическое проектирование в программе ERWin.

2.1. Отобразить полученную концептуально-инфологическую модель в виде реляционной модели.

2.3. Настроить кардинальность каждой связи. Написать глагол, характеризующий связь. Определить домен, на котором определен каждый атрибут отношения.

3. На физическом уровне проанализировать таблицы (Tables) и связи (Relationship). Для каждого атрибута выбрать оптимальный тип данных.

Лабораторная работа №4
Нормализация таблиц БД

Цель работы: Изучить основные принципы проектирования реляционных баз данных методом нормализации.

Теоретические положения

Классическая технология проектирования реляционных баз данных связана с теорией нормализации, основанной на анализе функциональных зависимостей между атрибутами отношений. Понятие функциональной зависимости является фундаментальным в теории нормализации реляционных баз данных. Функциональные зависимости определяют устойчивые отношения между объектами и их свойствами в рассматриваемой предметной области. Именно поэтому процесс поддержки функциональных зависимостей, характерных для данной предметной области, является базовым для процесса проектирования.

Процесс проектирования с использованием декомпозиции представляет собой процесс последовательной нормализации схем отношений, при этом каждая последующая итерация соответствует нормальной форме более высокого уровня и обладает лучшими свойствами по сравнению с предыдущей.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений.

В теории реляционных БД обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или форма проекции-соединения (5NF или PJNF).

Основные свойства нормальных форм:

- каждая следующая нормальная форма в некотором смысле улучшает свойства предыдущей;

• при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются.

Объект базы данных находится в первой нормальной форме тогда, когда каждый ее атрибут атомарен. Атрибут атомарен тогда, когда его значение теряет смысл при перестановке любой из его частей или при любом разбиении его на части. То есть, одно поле – одно значение.

Объект базы данных находится во второй нормальной форме тогда, когда он находится в первой нормальной форме и при этом любой его атрибут, не входящий в состав потенциального ключа, функционально полно зависит от каждого потенциального ключа. Это правило говорит об отделении функционально полных зависимостей на отдельные структуры.

Объект базы данных находится в третьей нормальной форме тогда, когда он находится во второй нормальной форме и отсутствуют транзитивные зависимости не ключевых объектов от ключевых. Транзитивная зависимость – это очевидная зависимость между полями. Если поле А равно x, то поле Б обязательно будет равно y. А если поле Б равно z, то тогда поле С будет равно m. Такой зависимости между объектами быть не должно.

Отношение находится в нормальной форме Бойса-Кодда, если оно находится в третьей нормальной форме, и каждый детерминант отношения является возможным ключом отношений.

В большинстве случаев, достижение третьей нормальной формы, или даже формы Бойса-Кодда, считается достаточным для реальных проектов баз данных. Однако в теории нормализации существуют нормальные формы высших порядков, которые уже связаны не с функциональными зависимостями между атрибутами отношений, а отражают более тонкие вопросы семантики предметной области, и связаны с другими видами зависимостей.

Задания к лабораторной работе №4

1. Разработайте структуру БД методом нормализации.

а) Библиотечная система;
б) Почтовое отделение;
в) Автовокзал.

2. Проверьте каждую таблицу из БД, спроектированной в предыдущей лабораторной работе, на соответствие 1,2,3 NF. Проведите нормализацию отношений при необходимости.

Лабораторная работа №5
Создание БД на сервере. Создание и модификация таблиц. Установление связей между таблицами

Цель работы: Научиться создавать базу данных, таблицы и связи между таблицами.

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».

Создать таблицу в Microsoft SQL Server можно тремя способами:

- с помощью визуального интерфейса SSMS, выбрав в обозревателе объектов Tables и нажав в контекстном меню New (рис. 3),
- с помощью инструкции на языке T-SQL,
- с помощью графического конструктора SSMS в существующей диаграмме баз данных.

Для создания таблиц с помощью визуального интерфейса SSMS необходимо развернуть узел базы данных в Object Explorer, найти папку «Tables», кликнуть по ней правой кнопкой мыши и выбрать «New Table». В конструкторе таблиц в первой колонке «Column Name» указываются имена столбцов, во второй колонке «Data Type» выбираются типы данных, при необходимости устанавливается флаг «Allow Nulls» для каждого столбца, в который разрешено вставлять нулевые значения.

Чтобы задать первичный ключ, необходимо выделить нужный столбец и нажать кнопку «Set Primary Key» на панели инструментов. Для настройки дополнительных свойств столбца (значение по умолчанию, уникальность и т. п.) используется панель «Column

Properties» внизу экрана. После завершения проектирования необходимо сохранить таблицу, нажав Ctrl+S или иконку сохранения.

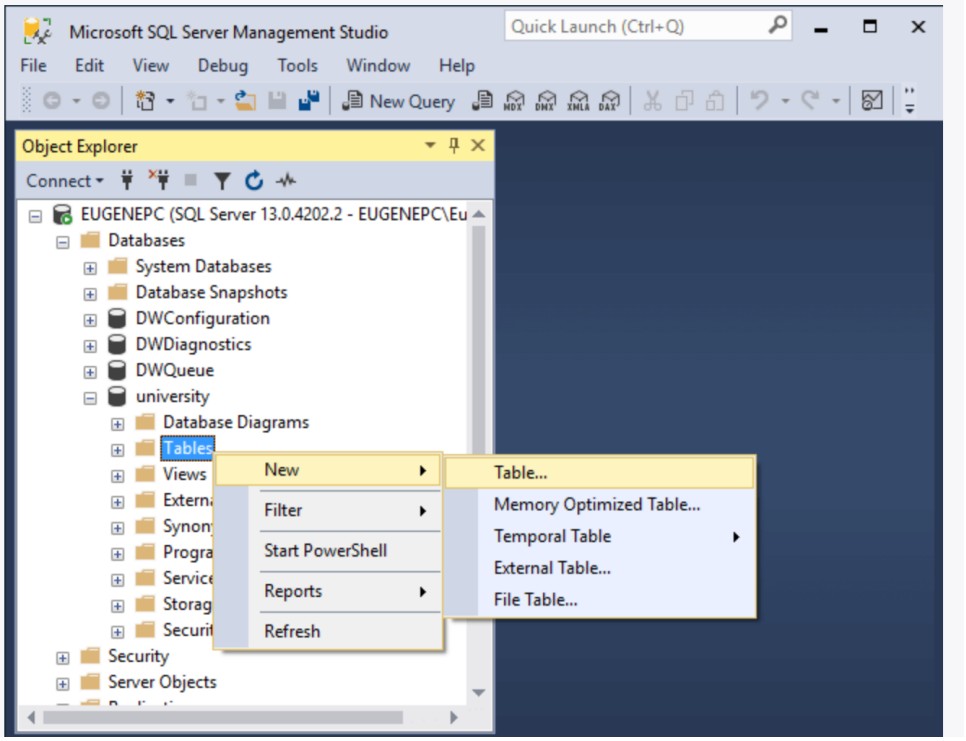


Рисунок 3 – Создание таблицы

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	строковый тип, как varchar(n), но с поддержкой

(n)	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.

Для создания связей (отношений) между таблицами сначала убедитесь, что в обеих таблицах есть подходящие столбцы (обычно первичный ключ в одной таблице и внешний ключ в другой). В Object Explorer кликните правой кнопкой мыши по папке «Keys» в целевой таблице и выберите «New Foreign Key…». В открывшемся редакторе нажмите кнопку с многоточием рядом с полем «Tables and Columns Specification», что откроет диалоговое окно настройки связи. В этом окне выберите родительскую таблицу (с первичным ключом) в выпадающем списке «Primary Key Table», затем сопоставьте столбцы: слева выберите столбец внешнего ключа из текущей таблицы, справа – соответствующий столбец первичного ключа из родительской таблицы. Нажмите «OK» для подтверждения настройки связи, затем сохраните изменения в таблице. После того, как связь будет создана её можно будет увидеть в папке «Keys» дочерней таблицы, а также визуально в диаграмме базы данных.

Задание к лабораторной работе №5

1. Создать новую БД.
2. Создать 2 таблицы, связанных связью с кардинальностью 1:n. Например, Студент и Группа, Сотрудник и Отдел, Товар и Категория и т.д.
3. В конструкторе таблиц добавить несколько полей, установить тип данных каждого поля.
4. Добавить к таблицам ограничения NOT NULL, PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, DEFAULT.
5. Выбрать стратегию обеспечения ссылочной целостности для операций удаления, обновления и вставки данных в созданные таблицы.
6. Заполнить таблицы данными (4 записи в родительской таблице, 10 записей в дочерней таблице).
7. Просмотреть содержимое таблиц.

Лабораторная работа №6
Генерация тестовых данных. Заполнение таблиц тестовыми данными

Цель работы: Научиться заполнять таблицы тестовыми данными.

Теоретические положения

Сгенерировать тестовые данные в таблицу SQL Server можно несколькими способами – от простых встроенных функций до сложных скриптов и сторонних инструментов.

Для генерации случайных чисел можно использовать функцию RAND():

```
SELECT ROUND((RAND() * (100 - 1)) + 1, 0);
```

Для генерации уникальных идентификаторов можно использовать функцию NEWID():

```
SELECT NEWID();
```

Генерацию тестовых (фейковых) данных различных типов предлагает бесплатный онлайн-сервис Mockaroo. На сервисе есть ограничение на количество генерируемых данных в бесплатном режиме. Сервис предоставляет возможность сохранения сгенерированных наборов данных в форматах JSON, CSV, SQL, TXT, XML. Для использования API сервиса, необходимо пройти регистрацию и получить в личном кабинете API Key.

Для работы с сервисом Mockaroo необходимо перейти на страницу https://www.mockaroo.com/, заполнить поля формы, выбрать формат, количество генерируемых строк данных и нажать на кнопку «Generate data» (рис. 4).

Полученные данные можно просмотреть, нажав на кнопку «Preview» (рис. 5), скачать в выбранном формате и затем импортировать в БД с помощью мастера импорта-экспорта неструктурированных файлов SQL Server.

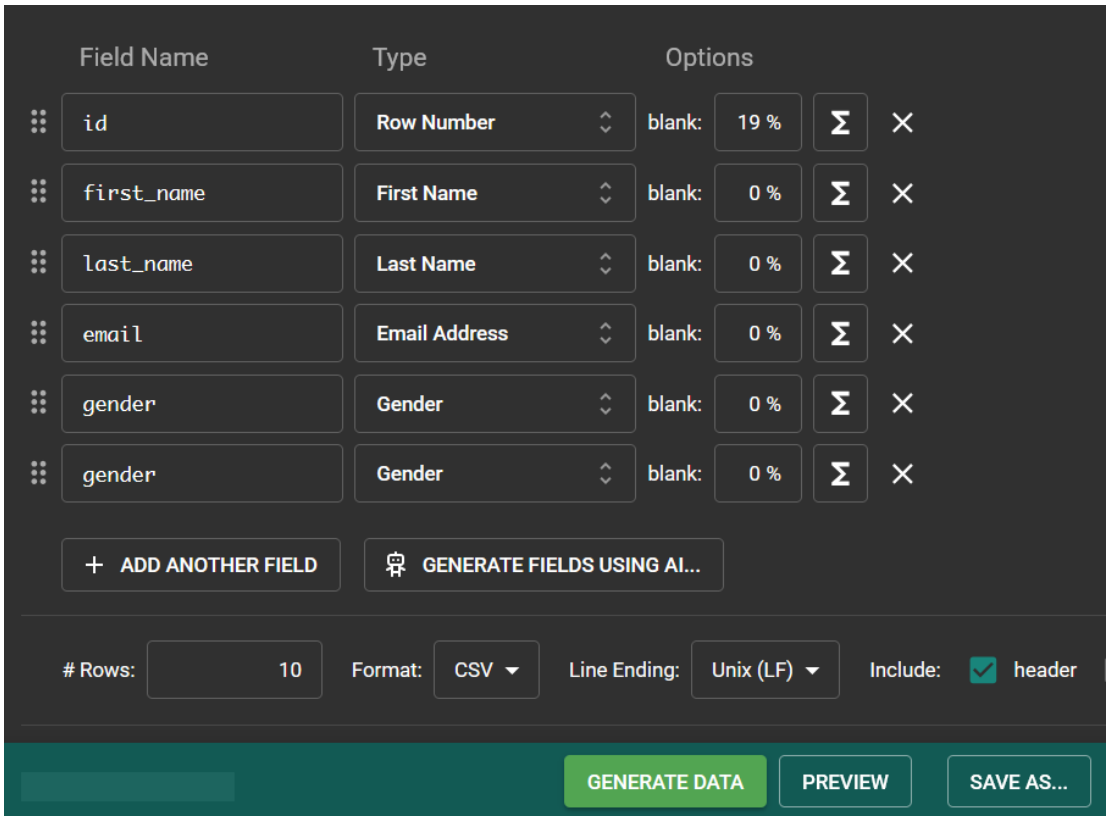
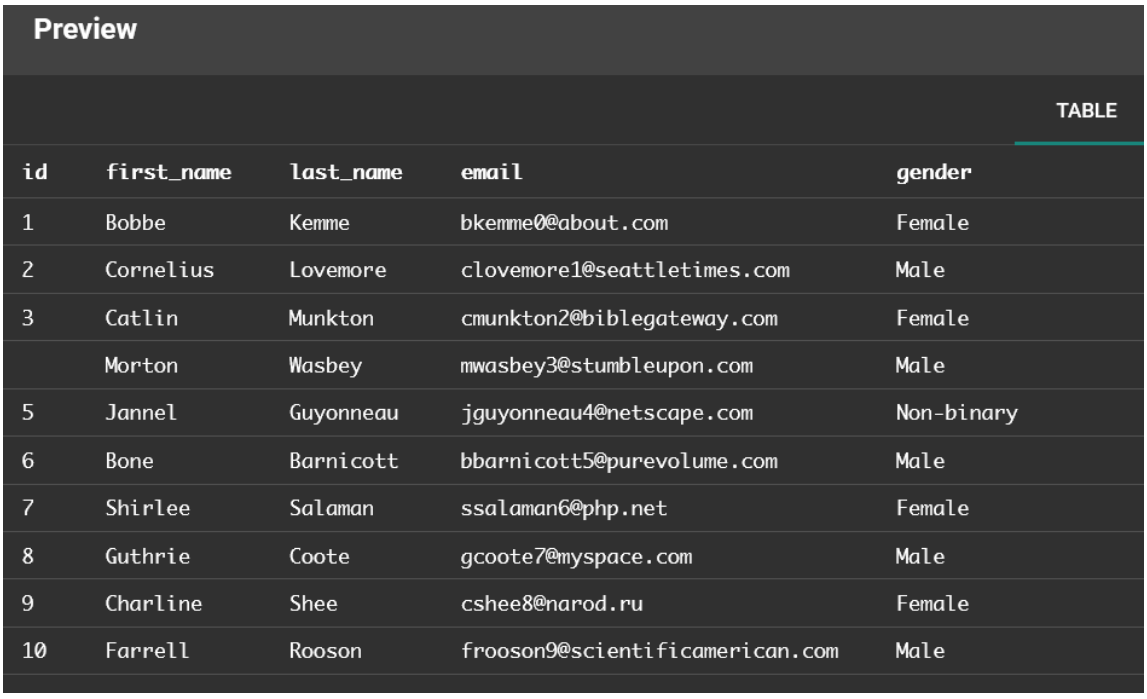


Рисунок 4 – Форма для заполнения



id	first_name	last_name	email	gender
1	Bobbe	Kemme	bkemme0@about.com	Female
2	Cornelius	Lovemore	clovemore1@seattletimes.com	Male
3	Catlin	Munkton	cmunkton2@biblegateway.com	Female
	Morton	Wasbey	mwasbey3@stumbleupon.com	Male
5	Jannel	Guyonneau	jguyonneau4@netscape.com	Non-binary
6	Bone	Barnicott	bbarnicott5@purevolume.com	Male
7	Shirlee	Salaman	ssalaman6@php.net	Female
8	Guthrie	Coote	gcoote7@myspace.com	Male
9	Charline	Shee	cshee8@narod.ru	Female
10	Farrell	Rooson	frooson9@scientificamerican.com	Male

Рисунок 5 – Предварительный просмотр данных

Используя язык программирования Python и библиотеку Faker также можно сгенерировать тестовые данные, например, значения имен, электронных адресов, телефонных номеров.

Пример кода:

```
from faker import Faker
import psycopg2

fake = Faker()

for _ in range(1000):
    name = fake.name()
    email = fake.email()
    phone = fake.phone_number()
    # Вставка в БД
```

Задание к лабораторной работе №6

1.Заполнить 1-2 таблицы БД тестовыми данными с помощью кода, написанного на языке Python.

2. Заполнить остальные таблицы БД тестовыми данными с помощью сервиса Mockaroo.

Лабораторная работа №7
Создание скрипта БД. Проверка его работоспособности

Цель работы: сформировать практические навыки создания базы данных посредством SQL‑ скриптов, выполняемых на целевом сервере.

Теоретические положения

Скрипт БД – это текстовый файл с набором инструкций SQL, предназначенный для создания структуры базы данных (таблицы, представления, индексы), наполнения БД данными, модификации существующей структуры объектов БД, выполнения административных операций. Файл скрипта обычно сохраняется в формате sql, исполняется в среде целевой СУБД.

Типовой скрипт имеет следующую структуру:

• Директивы подключения и настройки:

```
USE [master];
GO
SET ANSI_NULLS ON;
SET QUOTED_IDENTIFIER ON;
```

• Проверка существования БД и её удаление (при необходимости):

```
IF EXISTS (SELECT name FROM sys.databases WHERE name = N'MyDB')
DROP DATABASE [MyDB];
```

• Создание базы данных

```
CREATE DATABASE [MyDB]
ON (NAME = N'MyDB_Data', FILENAME = N'C:\Data\MyDB.mdf', SIZE = 100MB)
LOG ON (NAME = N'MyDB_Log', FILENAME = N'C:\Data\MyDB.ldf', SIZE = 50MB);
```

• Переключение на созданную БД

```
USE [MyDB];
GO
```

• Создание таблиц и ограничений

`CREATE TABLE` с определением столбцов, типов, первичных ключей `PRIMARY KEY`, внешних ключей`FOREIGN KEY`, проверочных ограничений, значений по умолчанию.

• Заполнение данными

```
INSERT INTO [dbo].[Users] (Name, Email) VALUES
('Alice', 'alice@example.com');
```

Для проверки работоспособности скрипта необходимо провести синтаксический контроль посредством выполнения кода в SSMS с проверкой синтаксиса (Ctrl+F5) и поиск ошибок в окне «Сообщения».

Задание к лабораторной работе №7

Создать базу данных и объекты БД с помощью SQL‑ скрипта. Выполнить SQL‑ скрипт на целевом сервере, исправить при необходимости ошибки.

Лабораторная работа №8
Выбор и реализация стратегии обеспечения ссылочной целостности

Цель работы: освоить принципы и практические приёмы поддержания ссылочной целостности данных в реляционной базе данных.

Теоретические положения

Ссылочная целостность – фундаментальное свойство реляционной базы данных, гарантирующее согласованность и логическую непротиворечивость данных во всех связанных таблицах. Суть принципа заключается в том, что для каждого значения внешнего ключа в дочерней таблице должно существовать соответствующее значение первичного ключа в родительской таблице.

Ссылочная целостность позволяет предотвратить появление «битых» ссылок и исключить потерю данных при изменениях.

Ссылочная целостность может быть нарушена при следующих действиях, выполняемых в родительской таблице: UPDATE – изменение значения первичного ключа, DELETE – удаление записи с первичным ключом; в дочерней таблице: INSERT – вставка записи с несуществующим значением внешнего ключа, UPDATE – изменение внешнего ключа на некорректное значение. Следует заметить, что операция DELETE в дочерней таблице не нарушает целостность, так как удаляет лишь зависимую запись.

Выделяют следующие основные стратегии поддержания ссылочной целостности:

• RESTRICT (ограничить). Выбор данной стратегии позволяет запретить выполнение операции, если она ведёт к нарушению ссылочной целостности. СУБД проверяет наличие связанных записей в дочерней таблице и отклоняет команду при обнаружении их наличия.

• CASCADE (каскадное изменение). Выбор данной стратегии позволяет автоматически распространить выполненные изменения в родительской таблице на связанные таблицы. То есть при выполнении инструкции DELETE в родительской таблице удаляются все связанные записи в дочерних таблицах, при выполнении инструкции UPDATE в родительской таблице обновляются внешние ключи

во всех дочерних таблицах.

• SET NULL (установить NULL). Выбор данной стратегии позволяет разрешить выполняемую операцию, но заменить нарушенные внешние ключи в дочерней таблице на значение NULL. Следует заметить, что столбец внешнего ключа в дочерней таблице должен допускать вставлять значения NULL.

• SET DEFAULT (установить по умолчанию). Выбор данной стратегии позволяет заменить нарушенные значения в столбце внешнего ключа дочерней таблицы на значение по умолчанию. Следует заметить, что в родительской таблице должна существовать запись с первичным ключом, имеющим то значение, которое выбрано, как значение по умолчанию.

• IGNORE (игнорировать). Выбор данной стратегии позволяет не проверять целостность, выполнять операцию без контроля ссылочной целостности. При выборе данной стратегии возрастают риски появления некорректных ссылок и потери согласованности данных. Такую стратегию рекомендуется применять только в исключительных случаях (например, при массовом импорте с последующей ручной корректировкой).

Задание к лабораторной работе №8

Выбрать стратегию обеспечения ссылочной целостности для каждой связи в БД, проверить работоспособность выбранной стратегии.

Лабораторная работа №9
Создание запросов на выборку данных из одной таблицы согласно заданному условию

Цель работы: Научиться создавать запросы на выборку данных из одной таблицы.

Теоретические положения

Запрос – это запрограммированное на специальном языке (SQL) требование к системе на выполнение некоторых действий с объектами базы данных. Запросы создаются пользователем для выборки нужных сведений из одной или нескольких связанных таблиц. С помощью запросов можно также обновить, удалить или добавить данные в таблицы, создать, удалить, изменить объекты БД – таблицы, индексы, представления, триггеры и т.д.

Для написания запросов используется язык SQL. Язык SQL состоит из ограниченного числа команд, специально предназначенных для управления данными:

- Команды для определения данных. Подъязык для определения данных (Data Definition Language, DDL), используется для создания (полного определения) базы данных, изменения ее структуры и удаления БД.
- Команды для обработки данных. Подъязык манипулирования данными (Data Manipulation Language, DML) предназначен для поддержки базы данных (ввод данных, их изменение, выборка).
- Команды для администрирования данных. Подъязык управления данными (Data Control Language, DCL), предназначен для защиты базы данных от различных вариантов повреждения.

Оператор SELECT является наиболее часто используемым оператором в T-SQL и используется для извлечения данных из одной или нескольких таблиц. Чтобы выбрать данные из одной таблицы, необходимо использовать следующий синтаксис:

```
SELECT столбец1, столбец2, столбец3,...
FROM имя_таблицы;
```

Оператор SELECT начинается с ключевого слова SELECT, за которым следует список столбцов, которые необходимо извлечь из указанной таблицы. Список столбцов разделяется запятыми. После указания столбцов, которые нужно выбрать, необходимо использо-

вать ключевое слово FROM, за которым следует имя таблицы, из которой нужно выбрать данные.

Рассмотрим практический пример: предположим, есть таблица под названием «Клиент», которая содержит следующие столбцы: «Код клиента», «Имя клиента», «Имя контактного лица», «Страна» и «Город». Мы можем использовать оператор SELECT для извлечения данных из этой таблицы следующим образом:

```
SELECT [Имя клиента], Страна, Город
FROM Клиент;
```

Этот оператор будет извлекать данные из таблицы «Клиент» и возвращать столбцы Имя клиента, Страна, Город. Следует заметить, что имена столбцов, состоящие из нескольких слов, необходимо заключить в квадратные скобки.

Для фильтрации данных необходимо использовать предложение WHERE, после которого необходимо записать условие, которое должно быть выполнено для извлечения данных.

Например, если необходимо получить клиентов только из «России», мы можем изменить запрос следующим образом:

```
SELECT [Имя клиента], Страна, Город
FROM Клиент
WHERE Страна = 'Россия';
```

При извлечении данных из одной таблицы можно использовать условие BETWEEN, чтобы указать диапазон возвращаемых значений.

Например, предположим, что есть таблица с именем «Продажа», которая содержит столбцы для названия продукта, даты продажи и суммы. Чтобы получить все данные о продажах за январь можно использовать условие BETWEEN следующим образом:

```
SELECT [Название продукта], [Дата продажи], сумма
FROM Продажа
WHERE [Дата продажи] BETWEEN ‘2019-01-01’ AND
‘2019-01-31’;
```

Оператор LIKE используется для поиска строк, которые соответствуют заданному шаблону. Шаблон может содержать специальные символы, которые представляют любой символ или группу символов. Например, если мы хотим найти все продукты, содержащие слово «Яблоки», мы можем использовать следующий запрос:

```
SELECT  FROM Продажа
WHERE [Название продукта] LIKE '%Яблоки%'
```

Здесь знак процента (%) обозначает, что перед и после слова «Яблоки» могут быть любые символы. Таким образом, запрос найдет все строки, содержащие слово «Яблоки», вне зависимости от того, где это слово находится в строке. Кроме знака процента, есть еще два специальных символа, которые можно использовать в операторе LIKE – знак подчеркивания (означает любой одиночный символ) и квадратные скобки (означают любой одиночный символ из заданного диапазона, например, [a-z] означает любой одиночный символ от a до z).

При написании запросов можно использовать встроенные функции (см. таблицу 2).

Например, чтобы получить все данные о продажах за январь можно использовать функцию Month():

```
SELECT [Название продукта], [Дата продажи], сумма
FROM Продажа
WHERE Month([Дата продажи])=1;
```

Таблица 2 – Встроенные функции

Функция	**Описание**
Day()	Возвращает целое число, представляющее дату (день месяца) указанного значения типа *date*.
GetDate()	Возвращает значение системной даты
Int()	Целая часть числа
DatePart (интервал, дата)	Возвращает значение типа Variant of Integer, представляющее собой указанную часть даты. Например, DatePart (m, Дата_рождения)
DateDiff (интервал, дата1, дата2)	Возвращает значение типа Variant of Long, указывающее число интервалов времени (лет, дней, минут, секунд и др.) между датами. Например, DateDiff (year, '2005-12-31', '2006-01-01'))
Format (выражение; формат)	Форматирование выражения
iif (выражение; верно; ложно)	Возвращает одну из частей, в зависимости от результата вычисления выражения

Year(дата)	Возвращает год из указанной даты
Month(дата)	Возвращает месяц из указанной даты
CAST (переменная AS тип_данных)	Преобразует выражение одного типа данных в другой. Например, CAST(Цена AS int)
SUBSTRING (переменная, нач_позиция , количество)	Возвращает заданное количество символов, начиная с начальной позиции. Например, SUBSTRING (Фамилия, 1,1)

Задание к лабораторной работе №9

1. Вывести имена и фамилии студентов, фамилия которых содержит слово ‘ова’.

2. Вывести на экран ФИО и E-mail студентов, родившихся ’12.12.2000’

3. Добавить к таблице с оценками поле “Характеристика оценки” (возможные значения этого поля: «Ответ на уроке», «Контрольная работа», «Экзамен»). Вывести номер студента, оценку, если характеристика оценки = ‘ответ на уроке’. Повторяющиеся записи исключить.

4. Вывести на экран преподавателей, фамилия которых начинается с буквы “Т”.

5. Выбрать всех студентов, родившихся в 1980-1995гг. Данные вывести в алфавитном порядке.

6. Вывести всех студентов, проживающих в заданном городе.

7. Вывести оценки всех студентов по предмету, номер/ID которого равен 2.

8. Вывести список наград, которые вручались сегодня (см. функцию GetDate()).

9. Вывести 10 первых по списку дисциплин, шифр которых начинается на 1970.

10. Вывести тех студентов, которые родились в мае. В качестве заголовка столбца использовать псевдоним.

11. Проверить любое поле на равенство NULL.

Лабораторная работа №10
Групповые операции. Агрегатные функции

Цель работы: Приобрести навыки создания запросов с агрегатными функциями.

Теоретические положения

Агрегатные функции – это функции, которые работают с группами данных в запросе. Они используются для выполнения различных операций с данными, таких как суммирование, подсчет количества записей в наборе, вычисление среднего, минимума и максимума и т.д.

В T-SQL существует следующие агрегатные функции:

• SUM – функция суммирования значений, принимающая на вход столбец или выражение и возвращающая сумму значений этого столбца/выражения.

Пример: `SELECT SUM(price) as total_price FROM products;`

• COUNT – функция подсчета количества записей, принимающая на вход столбец или выражение и возвращающая количество записей.

Пример: `SELECT COUNT() FROM products;`

• AVG – функция вычисления среднего значения. Она может принимать столбец или выражение и возвращает среднее значение.

Пример: `SELECT AVG(price) FROM products;`

• MIN – функция вычисления минимального значения. Она может принимать столбец или выражение и возвращает минимальное значение.

Пример: `SELECT MIN(price) FROM products;`

• MAX – функция вычисления максимального значения. Она может принимать столбец или выражение и возвращает максимальное значение.

Пример: `SELECT MAX(price) FROM products;`

При использовании агрегатных функций обычно необходимо создать предложение GROUP BY. В предложении GROUP BY указываются все поля, к которым не применяется агрегатная функция. Если агрегатные функции применяются ко всем полям в запросе, предложение GROUP BY создавать не нужно.

Предложение GROUP BY должно следовать сразу же за предложением WHERE или FROM, если предложение WHERE отсутствует. В предложении GROUP BY поля указываются в том же порядке, что и в предложении SELECT.

Пример. Вывести для каждой категории товара среднюю цену:

```
SELECT category, AVG(price) FROM products
GROUP BY category;
```

Если необходимо указать условия для ограничения результатов, но поле, к которому их требуется применить, используется в агрегированной функции, предложение WHERE использовать нельзя. Вместо него следует использовать предложение HAVING. Предложение HAVING работает так же, как и WHERE, но используется для агрегированных данных.

Запрос может включать и предложение WHERE, и предложение HAVING, при этом условия отбора для полей, которые не используются в статистических функциях, указываются в предложении WHERE, а условия для полей, которые используются в статистических функциях, – в предложении HAVING.

Пример. Вывести только те категории, для которых средняя цена больше 100:

```
SELECT category, AVG(price) FROM products
GROUP BY category
HAVING AVG(price)>100;
```

Задание к лабораторной работе №10

1. Вывести количество студентов, родившихся в 2000 году.
2. Вывести количество мероприятий, проведенных сегодня.
3. Вывести количество оценок по каждому предмету (группировка по коду предмета).
4. Вывести количество дисциплин, шифр которых начинается на 1970.
5. Найти количество студентов, родившихся в 1995-1999гг. и проживающих в г. Кемерово.
6. Найти количество студентов, поступивших в ВУЗ в каждом году. Вывести только те года, в которые поступило менее 1000 студентов (можно взять любое другое число).
7. Вывести количество преподавателей, фамилия которых начинается с буквы “В”.

8. Поместить в БД информацию о стипендии студентов. Найти суммарную стипендию, получаемую всеми студентами.

9. Вывести количество мероприятий, проведенных в каждый день. Отобразить только те, которых больше 2.

10. Найти дату самой «старой» оценки.

Лабораторная работа №11
Вычисляемые поля. Псевдонимы

Цель работы: Приобрести навыки создания запросов с вычисляемыми полями.

Теоретические положения

Вычисляемые поля – это поля, которые не хранятся в таблице БД, а вычисляются при каждом обращении к ней. Например, можно использовать вычисляемые поля для создания отчета, который показывает общую выручку, вычисленную как произведение количества проданных товаров и цены за единицу товара:

```
SELECT [Название товара], Цена, Количество,
Цена*Количество AS Выручка
FROM Продажи
```

В этом запросе используется оператор AS, чтобы назвать вычисляемое поле «Выручка» и оператор умножения (*) для умножения цены на количество, чтобы вычислить общую выручку.

Рассмотрим еще один пример использования вычисляемых полей. Предположим, таблица Клиенты содержит следующие поля: ID (идентификатор клиента), FirstName (имя клиента), LastName (фамилия клиента), Email (адрес электронной почты клиента). Необходимо создать вычисляемое поле, которое будет содержать полное имя клиента, объединяя имя и фамилию. Для этого можно использовать следующий код:

```
SELECT ID, FirstName, LastName,
CONCAT(FirstName, ' ', LastName) AS FullName,
Email
FROM Customers
```

В этом запросе используется функция CONCAT, чтобы объединить имя и фамилию клиента в одну строку, разделяя их пробелом, затем используется оператор AS, чтобы назвать вычисляемое поле «FullName».

Задание к лабораторной работе №11

1. Найти количество дней, которое проучились студенты группы ИСт-221.

2. Вычислить возраст тех студентов, которые родились в 1995-1999гг.

3. Вычислить возраст всех студентов группы, код которой равен 2.

4. Найти ФИО самого молодого студента (См. вложенные запросы).

5. Вычислить, какая стипендия будет у каждого студента, если её увеличить на 10%.

6. Вычислить, какая стипендия будет у студентов, если из неё вычесть плату за общежитие. Размер платы фиксированный.

7. Вычислить, какая стипендия будет у студентов, если её увеличить на 3% для тех, кто поступил в ВУЗ в 2022 году и на 5% - для тех, кто поступил в ВУЗ в 2023 году. Для остальных студентов стипендия останется прежней (См. объединение UNION).

8. Вычислить дату самого «старого» мероприятия.

Лабораторная работа №12
Создание запросов на выборку данных из нескольких таблиц согласно заданному условию

Цель работы: Отработать навыки создания запросов на выборку данных из нескольких таблиц согласно заданному условию.

Теоретические положения

Соединение – операция над двумя отношениями, имеющими общие атрибуты, в результате которой получается новое отношение, состоящее из всех атрибутов исходных отношений и объединяющее только те кортежи исходных отношений, в которых значения общих атрибутов совпадают.

В T-SQL для соединения таблиц используется оператор INNER JOIN. JOIN позволяет выбирать данные из двух или более таблиц с помощью определения связей между ними.

Упрощенный синтаксис соединения:

```
FROM таблица1 <join_type> таблица2
[ ON ( join_condition ) ]
```

Типы соединений (join_type):

- INNER JOIN (внутреннее)
- LEFT [OUTER] JOIN (внешнее)
- RIGHT [OUTER] JOIN (внешнее)
- FULL [OUTER] JOIN (внешнее)
- CROSS JOIN (перекрестное соединение/декартово произведение)

Обычно соединение таблиц осуществляется с помощью первичных и внешних ключей:

```
SELECT список_полей
FROM PARENT_таблица INNER JOIN CHILD_таблица
ON PARENT_таблица.полеPK = CHILD_таблица.полеFK
```

Рассмотрим пример: есть две таблицы – «Orders» (дочерняя таблица) и «Customers» (родительская таблица). Таблица «Orders» содержит информацию о заказах, а таблица «Customers» – информацию о клиентах. Обе таблицы имеют общее поле «customer_id».

Выберем все заказы, которые были сделаны клиентом с id, равным 1. Для этого необходимо использовать оператор INNER JOIN, который объединяет две таблицы по условию «customer_id =

1».

```
SELECT orders.order_id, orders.order_date, cus-
tomers.customer_name
FROM orders INNER JOIN customers
ON orders.customer_id = customers.customer_id
WHERE customers.customer_id = 1;
```

Сначала выбираются необходимые поля из таблиц – «order_id», «order_date» и «customer_name». Затем таблицы соединяются с помощью оператора INNER JOIN. Ключевое слово ON указывает, по какому полю соединяются таблицы. В данном случае это поле «customer_id». Затем добавляется фильтр WHERE, который выбирает только те строки, для которых значение поля «customer_id» равно 1.

Задание к лабораторной работе №12

1. Вывести ФИО студентов группы Ист-211 в виде: ФИО|Группа

2. Вывести ФИО студентов, занимающихся борьбой или дзюдо (т.е. прописать 2 условия).

3. Вывести оценки всех студентов по предмету «История» в виде: ФИО|Оценка

4. Вывести ФИО и адрес всех студентов, получивших награды сегодня.

5. Вывести количество оценок “5” у каждого студента в виде: ФИО|Количество5

6. Найти количество наград у тех студентов, которые занимаются плаваньем. Результат оформить в виде: ФИО |Количество наград

7. Найти суммарную стипендию, получаемую студентами каждой группы. Результат оформить в виде: Группа |СуммСтипендия

8. Вывести количество студентов в каждой группе в виде: Группа|КоличествоСтудентов

9. Вывести средний балл студентов по каждому предмету. Отобразить только те предметы, по которым средний балл <4. Результат оформить в виде: ФИО |Предмет|СрБалл

10. Найти студента с наименьшим средним баллом по всем предметам.

Лабораторная работа №13
Создание запросов на добавление данных

Цель работы: Освоить синтаксис и применение оператора INSERT языка SQL для добавления новых записей в таблицы реляционной базы данных. Изучить различные формы записи оператора INSERT и научиться добавлять как отдельные строки, так и множества данных.

Теоретические положения

Оператор INSERT – один из основных операторов языка манипулирования данными. Его основная задача – добавление новых данных в существующую таблицу, при этом данные добавляются строками (кортежами). Каждый оператор INSERT добавляет одну или несколько полных строк.

Синтаксис оператора:

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

Обратите внимание, что в скобках после имени таблицы указываются названия столбцов, в которые будут вставляться данные. Если не указывать названия столбцов, то данные будут вставлены в столбцы в том порядке, в котором они указаны в таблице.

Пример. Дана таблица «Students» с полями «id», «name», «age». Необходимо добавить нового студента с именем «Иван» и возрастом 20 лет:

```
INSERT INTO students (name, age) VALUES ('Иван',
20);
```

Также есть возможность добавлять несколько записей одновременно с помощью оператора INSERT INTO. Для этого нужно указать в скобках значения для следующих строк, разделяя их запятой.

Пример. Добавить в таблицу «Students» новые записи:

```
INSERT INTO students (name, age)
VALUES ('Иван', 20), ('Мария', 22), ('Петр', 19);
```

Для добавления данных, выбранных из одной или нескольких таблиц, можно использовать следующую инструкцию:

```
INSERT INTO имя_целевой_таблицы [(столбец1, стол-
бец2, ...)] SELECT столбец1, столбец2, ...
FROM имя_источника_данных [WHERE условие];
```

Пример. Скопировать в таблицу Retired_Employees всех сотрудников старше 60 лет из таблицы Employees:

```
INSERT INTO Retired_Employees (emp_id, full_name, position, birth_date)
SELECT id, CONCAT(last_name, ' ', first_name), position, birth_date
FROM Employees
WHERE birth_date <= DATEADD(YEAR, -60, GETDATE());
```

Задание к лабораторной работе №13

1. С помощью инструкции SQL INSERT… добавить в таблицу расписание работы секции.

2. С помощью инструкции SQL INSERT INTO / SELECT … INTO создать резервные копии всех таблиц БД.

3. Добавить в таблицу поле, в котором будет храниться фотография студента. С помощью функции OPENROWSET в режиме BULK добавить фотографии 2 студентов.

Лабораторная работа №14
Создание запросов на удаление данных

Цель работы: Освоить синтаксис и применение оператора DELETE языка SQL для удаления записей из таблиц реляционной базы данных. Изучить различные формы записи оператора DELETE, особенности работы с ограничениями целостности и научиться безопасно удалять данные.

Теоретические положения

Каждая база данных содержит множество таблиц, которые хранят информацию о различных объектах. Иногда возникает необходимость удалить данные из таблицы, например, если объект перестал существовать или изменил свое состояние.

В SQL запросах для удаления данных используется команда DELETE:

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

Здесь имя_таблицы – это имя таблицы, из которой нужно удалить данные, а условие – это условие, которое определяет, какие именно строки нужно удалить.

Например, чтобы удалить все данные из таблицы «Employees», можно использовать следующий запрос:

```
DELETE FROM Employees;
```

Данный подход может быть опасен, так запрос удалит все данные, включая те, которые еще могут быть необходимы. Поэтому, в большинстве случаев, для удаления данных используется условие.

Например, чтобы удалить только тех сотрудников, которые работают на должности «менеджер», можно использовать следующий запрос:

```
DELETE FROM Employees
WHERE position = 'менеджер';
```

Этот запрос удалит только те строки, которые удовлетворяют условию «position = 'менеджер'«. В результате, из таблицы будут удалены только данные о сотрудниках, занимающих эту должность.

Для удаления данных из целевой таблицы с проверкой данных, находящихся в связанной таблице, осуществляется с помощью подзапроса:

```
DELETE FROM Employees
```

```
WHERE department_id IN (
SELECT department_id
FROM Departments
WHERE location = 'Москва'
);
```

В некоторых СУБД (MySQL, SQL Server) можно использовать соединения для более сложных условий удаления данных:

```
DELETE FROM t1
FROM table1 t1
INNER JOIN table2 t2 ON t1.id = t2.table1_id
WHERE t2.status = 'не активный';
```

Задание к лабораторной работе №14

1. Удалить все записи об оценках, значение которой равно 3.
2. Удалить все оценки студента, фамилия которого начинается на «А».
3. Удалить самого «старого» студента из БД.
4. Удалить всю информацию о любом предмете.
5. Удалить всех студентов из БД, которые родились в 1995-1996гг.
6. Всех студентов, у которого значение поля «Отчислен» = «да», удалить из БД.

Лабораторная работа №15
Создание запросов на обновление данных

Цель работы: Освоить синтаксис и применение оператора UPDATE языка SQL для модификации существующих записей в таблицах реляционной базы данных.

Теоретические положения

В SQL для создания запроса на обновление данных используется оператор UPDATE. Синтаксис запроса выглядит следующим образом:

```
UPDATE имя_таблицы SET
имя_столбца1=значение1,    имя_столбца2=значение2
WHERE условие;
```

Обратите внимание, что обновление данных возможно только в тех строках, которые удовлетворяют заданному условию.

Пример. Обновление одной строки по уникальному идентификатору:

```
UPDATE Students
SET email = 'ivanov.new@mail.ru',
phone = '+7-999-123-45-67'
WHERE student_id = 101;
```

Пример. Дана таблица «Employees» с полями «id», «name», «salary». Необходимо увеличить зарплату сотрудника с id=1 на 10%:

```
UPDATE Employees SET salary=salary*1.1 WHERE
id=1;
```

Пример. Привести все email к нижнему регистру:

```
UPDATE Users
SET email = LOWER(email);
```

В операторе UPDATE можно использовать подзапросы.

Пример. Установить менеджера для сотрудника на основе данных из таблицы «Departments»:

```
UPDATE Employees
SET manager_id = (
SELECT manager_id
FROM Departments
WHERE department_id = Employees.department_id)
WHERE manager_id IS NULL;
```

Задание к лабораторной работе №15

1. Увеличить все оценки по физкультуре на 1 балл у тех студентов, которые занимаются борьбой.

2. Добавить логическое поле «Отчислен» в таблицу. Для каждого студента установить значение этого поля “да”, если он проучился 5 лет.

3. Изменить дату сдачи экзамена по математике на текущую дату у тех студентов, которые учатся в группе N.

4. Для тех студентов, которые родились в 1995-1996гг., к полю Характеристика добавить слово «Старшекурсник».

5. Изменить поле Характеристика у заданного студента на «Учащийся проявлял активность на занятиях, старательно выполнял домашние задания».

6. Для самого «старого» студента увеличить стипендию в текущем месяце на 10%.

7. Для всех студентов, занимающихся плаванием и имеющих любые награды, добавить к стипендии в текущем месяце 1000 руб.

8. Для всех Татьян в Татьянин день добавить к стипендии в текущем месяце 500 руб.

Лабораторная работа №16
Создание кластеризованных и некластеризованных индексов. Оптимизация запросов

Цель работы: Освоить создание кластеризованных и некластеризованных индексов, изучить их влияние на производительность запросов, научиться анализировать планы выполнения запросов и применять индексы для оптимизации запросов.

Теоретические положения

Индекс – это специальная структура данных, которая ускоряет операции поиска и извлечения данных из таблицы. Аналогия индекса – содержание книги, которое позволяет быстро найти нужную информацию, не просматривая всю книгу.

Индекс создает упорядоченную структуру (чаще B-дерево), которая содержит ключевые значения и ссылки на соответствующие строки в таблице.

Кластеризованный индекс определяет физический порядок хранения данных в таблице. Данные в таблице упорядочены в соответствии со структурой кластеризованного индекса. Кластеризованный индекс создается только один на таблицу (так как данные могут быть физически упорядочены только одним способом), физически переупорядочивает строки таблицы, листья B-дерева содержат сами данные таблицы. Кластеризованный индекс создается автоматически при определении первичного ключа.

Пример создания таблицы и кластеризованного индекса:

```
CREATE TABLE Students (
student_id INT PRIMARY KEY CLUSTERED,
name VARCHAR(100),
birth_date DATE
);
```

Пример создания кластеризованного индекса на существующей таблице:

```
CREATE CLUSTERED INDEX IX_Students_birth_date
ON Students(birth_date DESC);
```

Некластеризованный индекс – это отдельная структура, которая содержит ключевые значения и указатели на соответствующие строки в таблице.

На одну таблицу можно большое количество некластеризованных индексов (до 999 в SQL Server). Некластеризованный индекс не изменяет физический порядок данных, листья B-дерева содержат указатели на данные (либо ключ кластеризованного индекса, либо уникальный физический адрес строки в таблице базы данных, RID). Некластеризованный индекс занимает дополнительное место на диске, ускоряет поиск, но замедляет вставку/обновление данных.

Пример создания создания некластеризованного индекса:

```
CREATE NONCLUSTERED INDEX IX_Students_name
ON Students(name);
```

Некластеризованные индексы предназначены для оптимизации запросов.

Пример. Оптимизировать выборку всех заказов по указанной дате:

```
SELECT num, order_date, status  FROM Orders
WHERE order_date='04/05/2025';
```

Создаем индекс:

```
CREATE INDEX IX_Orders_order_date
ON Orders(order_date DESC)
INCLUDE (num, status);
```

Для упрощения оптимизации запросов в Management Studio в редактор кода встроен функционал, который позволяет посмотреть план предполагаемый выполнения запросов (рис.6).

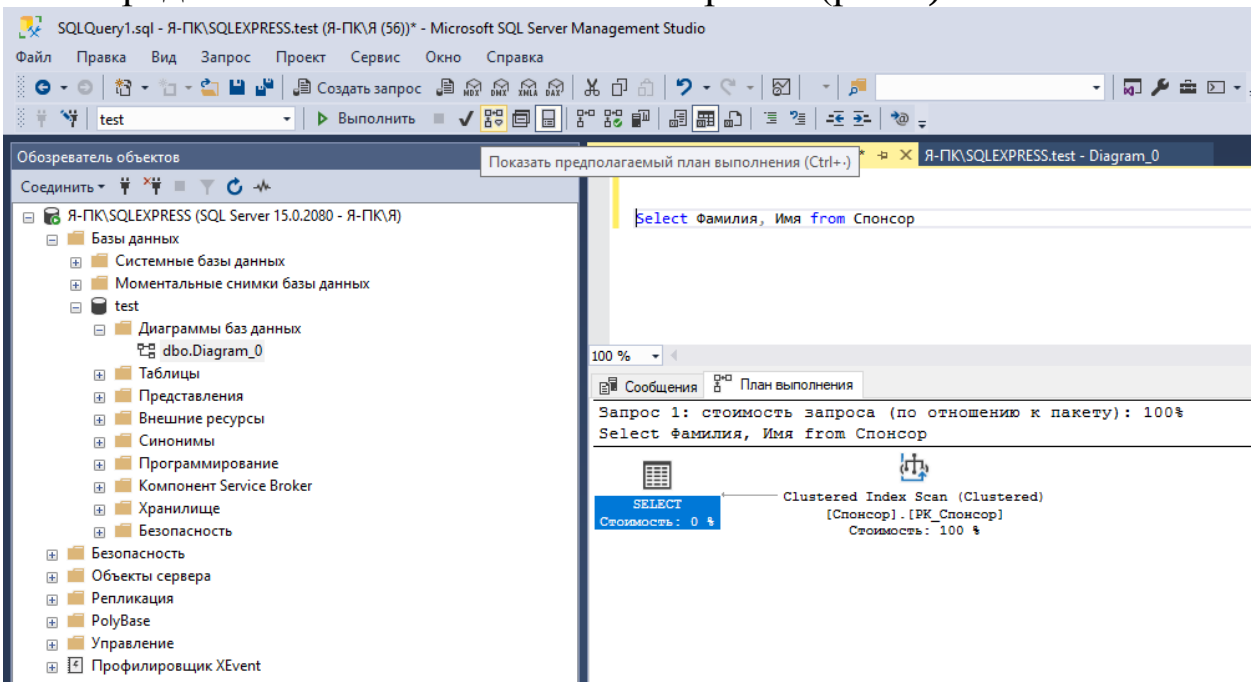


Рисунок 6 – План выполнения запроса

Действительный план выполнения аналогичен предполагаемому плану выполнения, но включает также действительные (не предполагаемые) значения количества строк, количества вложенных циклов и т. д.

Задание к лабораторной работе №16

1. Создать «покрывающий» некластеризованный индекс для повышения производительности запросов. Запросы придумать самостоятельно. Количество запросов/индексов – 4 шт.

2. Просмотреть предполагаемый и действительный план выполнения каждого запроса.

3. Созданные индексы используются при выполнении запросов? Если нет, то удалить индекс и создать другой.

Содержание самостоятельной работы

Цель самостоятельной работы обучающихся – получить новые знания по дисциплине «Проектирование и разработка баз данных».

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 3.

Таблица 3 – Виды самостоятельной работы

№ п/п	Вид СРС
1	Изучение литературы
2	Изучение стандартов оформления ER-диаграмм: нотация IDEF1x, Чена, «воронья лапка»
3	Изучение стандартов SQL (ANSI SQL:2016, SQL:2019), анализ различий
4	Изучение CASE-средств (ERwin) для проектирования БД
5	Изучение встроенных инструментов мониторинга и диагностики сервера СУБД

Обучающиеся должны изучить интернет-ресурсы и литературу по вопросам, представленным в таблице 3.

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