

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

Институт профессионального образования

Кафедра информатики и информационных систем

Составитель О. С. Семенова

## **Основы проектирования баз данных**

**Методические указания к практическим работам**

Рекомендовано ЦМК  
специальности 09.02.07 Информационные системы и программирования  
в качестве электронного издания для использования  
в образовательном процессе

Кемерово 2023

Рецензенты:

Е. А. Ощепкова – преподаватель кафедры информатики и информационных систем.

**Семенова Ольга Сергеевна**

**Основы проектирования баз данных** : методические указания к практическим работам для обучающихся специальности СПО 09.02.07 «Информационные системы и программирование» / сост. О. С. Семенова, Кузбасский государственный технический университет имени Т. Ф. Горбачева. – Кемерово, 2023. – Текст : электронный.

Приведенные методические указания к практическим работам по курсу «Основы проектирования баз данных» позволяют углубить знания, полученные в ходе аудиторных занятий; способствуют закреплению теоретических положений; развивают навыки по их практическому применению.

© Кузбасский государственный  
технический университет  
имени Т. Ф. Горбачева, 2023  
© Семенова О. С.,  
составление, 2023

## Практическая 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 1968г. 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 повторяются. 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 случаях, 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 практической 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.

## **Практическая работа №2**

### **Проектирование реляционной БД. Нормализация таблиц**

**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  $t$ . 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 практической работе №2

1. Разработайте структуру БД методом нормализации.
  - а) Библиотечная система;
  - б) Почтовое отделение;
  - в) Автовокзал.

## Практическая работа №3 Проектирование реляционной БД. ER-метод

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

### Теоретические положения

При создании моделей данных может использоваться метод семантического моделирования. Семантическое моделирование основывается на значении структурных компонентов или характеристик данных, что способствует правильности их интерпретации (понимания, разъяснения). В качестве инструмента семантического моделирования используются различные варианты *диаграмм сущность-связь* (ER — Entity-Relationship) — ERD.

Существуют различные варианты отображения ERD, но все варианты диаграмм сущность-связь исходят из одной идеи — рисунок всегда нагляднее текстового описания. ER-диаграммы используют графическое изображение сущностей предметной области, их свойств (атрибутов), и взаимосвязей между сущностями.

*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 (запись, кортеж) — это конкретный представитель данной сущности. Атрибут сущности (поле, домен) — это именованная характеристика, являющаяся некоторым свойством сущности.

*Связь* — это некоторая ассоциация между двумя сущностями. Одна сущность может быть связана с другой сущностью или сама с собою. Связи позволяют по одной сущности находить другие сущности, связанные с ней.

Каждая связь может иметь один из следующих типов связи:

- один-к-одному,
- многое-ко-многим,
- один-ко-многим.

Связь типа *один-к-одному* означает, что один экземпляр первой сущности (левой) связан с одним экземпляром второй сущности (правой). Связь один-к-одному чаще всего свидетельствует о том, что на самом деле мы имеем всего одну сущность, неправильно разделенную на две.

Связь типа *многие-ко-многим* означает, что каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и каждый экземпляр второй сущности может быть связан с несколькими экземплярами первой сущности. Тип связи многие-ко-многим является временным типом связи, допустимым на ранних этапах разработки модели. В дальнейшем этот тип связи должен быть заменен двумя связями типа один-ко-многим путем создания промежуточной сущности.

Связь типа *один-ко-многим* означает, что один экземпляр первой сущности (левой) связан с несколькими экземплярами второй сущности (правой). Это наиболее часто используемый тип связи. Левая сущность (со стороны «один») называется родительской, правая (со стороны «много») – дочерней.

При разработке ER-моделей необходимо обследовать предметную область (организацию, предприятие) и выявить:

- 1) Сущности, о которых хранятся данные в организации (предприятии), например, люди, места, идеи, события и т.д. (будут представлены в виде блоков);
- 2) Связи между этими сущностями (будут представлены в виде линий, соединяющих эти блоки);
- 3) Свойства этих сущностей (будут представлены в виде имен атрибутов в этих блоках).

Рассмотрим пример разработки информационной системы «Контингент студентов института». Для этого необходимо изучить предметную область (образовательное учреждение) и процессы, происходящие в ней.

В первую очередь требуется ознакомиться с нормативной документацией, изучить существующий документооборот, проанализировать связи. В результате определяется цель и задачи системы и формулируется постановка задачи.

Главная задача системы – сбор и обработка информации об основных участниках учебного процесса: студентах и преподавателях, формирование необходимых печатных форм (документов), используемых преподавателями в период зачётной недели и экзаменационной сессии, генерация сводных отчётов по результатам сессии для работников дирекции. При разработке системы следует учитывать, что она основывается на документации, поступающей из приёмной комиссии, дирекции и других подразделений института. Информация об успеваемости студентов должна накапливаться и храниться в течение всего пе-

риода обучения. В системе должен использоваться справочник специальностей и дисциплин (предметов), изучаемых студентами.

Таким образом, проектируемая система должна выполнять следующие действия:

1. Хранить информацию о студентах и их успеваемости.
2. В институтах/факультетах по определённой специальности печатать экзаменационные ведомости и другие документы.

Выделим все существительные в этих предложениях — это предполагаемые сущности и проанализируем их:

- Студент — явная сущность.
- Успеваемость — явная сущность.
- ?Институт/факультет — нужно выяснить один или несколько институтов в университете? Если несколько, то это — предполагаемая новая сущность.
- ? Специальность — нужно выяснить одна или несколько специальностей в институте/факультете? Если несколько, то это — ещё одна сущность.
- Предмет — предполагаемая сущность.

На первоначальном этапе моделирования данных информационной системы явно выделены две основные сущности: Студент и Успеваемость. Критерием успеваемости является наличие отметки о сдачи экзаменов. Сразу возникает очевидная связь между сущностями — «студент сдаёт несколько экзаменов» и «экзамены сдаются каждым студентом». Явная связь Один-ко-многим. Первый вариант диаграммы представлен на рисунке 1.



Рисунок 1 – Первый вариант ER-диаграммы

Мы знаем, что студенты учатся в институтах/факультетах, на определённой специальности и сдают экзамены по дисциплинам (предметам). Анализ предметной области показал, что студенты учатся в нескольких институтах университета по нескольким специальностям и сдают экзамены по определённому перечню предметов.

Исходя из этого, мы добавляем в ER-модель ещё несколько сущностей. В результате она будет выглядеть так, как показано на рисунке 2.

На следующей стадии проектирования модели вносим атрибуты сущностей в диаграмму (предполагаем, что атрибуты выявлены на стадии обследования объекта и при анализе аналогов существующих систем) и получаем окончательный вариант ER-диаграммы (рисунок 3).



Рисунок 2 – Второй вариант ER-диаграммы

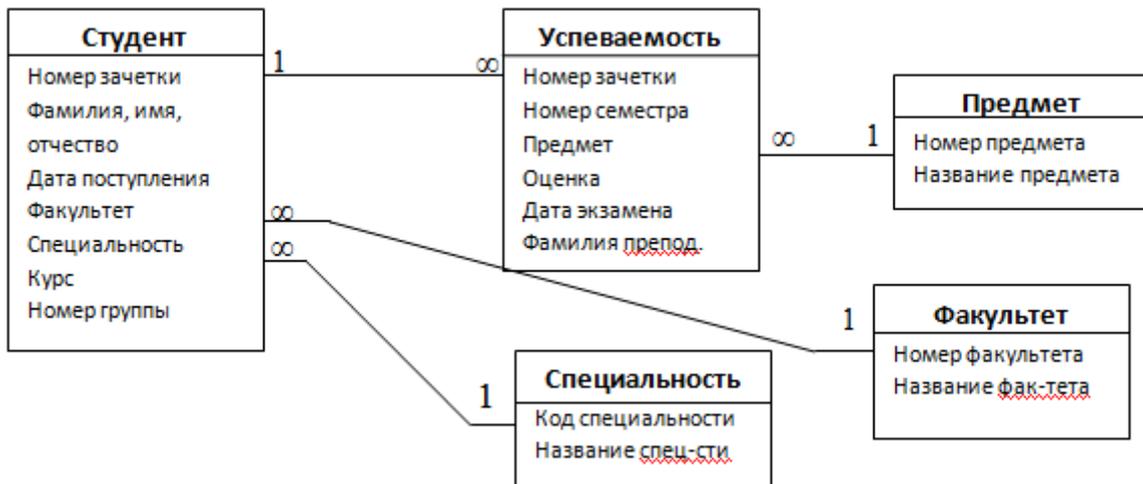


Рисунок 3 – ER-диаграмма БД

Предложенные этапы моделирования являются условными и нацелены на формирование общих представлений о процессе моделирования.

Разработанный выше пример ER-диаграммы является примером концептуальной диаграммы, не учитывающей особенности конкретной СУБД. На основе данной концептуальной диаграммы можно построить физическую диаграмму, которая будут учитывать такие особенности СУБД, как допустимые типы, наименования полей и таблиц, ограничения целостности и т.п.

Для преобразования концептуальной модели в физическую необходимо знать, что:

- каждая сущность в ER-диаграмме представляет собой таблицу базы данных.
- каждый атрибут становится полем соответствующей таблицы.
- В некоторых таблицах необходимо вставить новые атрибуты (поля), которых не было в концептуальной модели — это ключевые атрибуты родительских таблиц, перемещённых в дочерние таблицы для того, чтобы обеспечить связь между таблицами посредством внешних ключей.

### **Задания к практической работе №3**

1. Провести инфологическое проектирование предметной области «Студент». Предусмотреть хранение в БД следующей информации: 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. Выбор первичного ключа 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.

2.2. Проверить каждую таблицу на соответствие 1,2,3 NF. Провести нормализацию отношений.

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). Выбрать стратегию обеспечения ссылочной целостности для каждой связи (RI Action).

## Практическая работа №4

### Создание проекта БД. Создание БД. Редактирование и модификация таблиц

**Цель работы:** Научиться создавать таблицы различными способами.

#### Теоретические положения

Техническое задание на проектирование базы данных предоставляет заказчик.

При подготовке технического задания составляют:

- список исходных данных, с которыми работает заказчик;
- список выходных данных, которые необходимы заказчику для управления структурой своего предприятия;
- список выходных данных, которые не являются необходимыми для заказчика, но которые он должен предоставить в другие организации (в вышестоящие структуры, в органы статистического учета, прочие административные и контролирующие организации).

При этом очень важно не ограничиваться взаимодействием с головным подразделением заказчика, а провести обсуждение со всеми службами и подразделениями, которые могут оказаться поставщиками данных в базу или их потребителями.

Выяснив основную часть данных, которые заказчик потребляет или поставляет, можно приступить к созданию структуры базы, то есть структуры ее основных таблиц.

1. Работа начинается с составления генерального списка полей – он может насчитывать десятки и даже сотни позиций.
2. В соответствии с типом данных, размещаемых в каждом поле, определяют наиболее подходящий тип для каждого поля.
3. Далее распределяют поля генерального списка по базовым таблицам. На первом этапе распределение производят по функциональному признаку. Цель – обеспечить, чтобы ввод данных в одну таблицу производился, по возможности, в рамках одного подразделения, а еще лучше – на одном рабочем месте.
4. В каждой из таблиц намечают *ключевое поле*. В качестве такого выбирают поле, данные в котором повторяться не могут. Например, для таблицы данных о студентах таким полем может служить индивидуальный шифр студента или номер зачетной книжки (см. рисунок 3). Для таблицы, в которой содержатся расписание занятий, такого поля можно и не найти, но его можно создать искусственным комбинированием полей

«Время занятия» и «Номер аудитории». Эта комбинация неповторима, так как в одной аудитории в одно и то же время не принято проводить два различных занятия. Если в таблице нет полей, которые можно было бы использовать, как ключевые, всегда можно ввести дополнительное поле типа Счетчик – оно не может содержать повторяющихся данных по определению.

5. С помощью карандаша и бумаги расчерчивают связи между таблицами. Такой чертеж называется *схемой данных*. Связь между таблицами организуется на основе общего поля, причем в одной из таблиц оно обязательно должно быть ключевым.
6. Разработкой схемы данных заканчивается «бумажный» этап работы над техническим предложением. Эту схему можно согласовать с заказчиком, после чего приступить к непосредственному созданию базы данных.

Базы данных могут содержать различные объекты. Основными объектами любой базы данных являются таблицы. Простейшая база данных имеет хотя бы одну таблицу. Соответственно, структура простейшей базы данных тождественно равна структуре ее таблицы.

Структуру двумерной таблицы образуют столбцы и строки. Их аналогами в простейшей базе данных являются *поля и записи*. Если записей в таблице пока нет, значит, ее структура образована только набором полей.

Таблицы баз данных, как правило, допускают работу с гораздо большим количеством разных типов данных. Так, например, базы данных SQL Server работают со следующими типами данных (таблица 1).

Таблица 1

Тип данных	Описание
<b>binary(n)</b>	двоичные данные фиксированной длины до 8000 байт
<b>char(n)</b>	строковый тип данных фиксированной длины без поддержки Unicode длиной до 8000 байтов; данные зависят от установленной кодовой страницы;
<b>Varchar(n)</b>	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.
<b>Nchar(n)</b>	строковый тип, но с поддержкой Unicode; в этом случае для строковых констант надо задавать впереди букву N:

	N'ABC'.
<b>Nvarchar (n)</b>	строковый тип, как varchar(n), но с поддержкой Unicode.
<b>Int</b>	целый тип длиной в 8 байт и с диапазоном от-263 до 263-1.
<b>Decimal[(p[,s])]</b>	десятичный двоично-кодированный тип с p десятичными разрядами, из которых s - дробных;
<b>Float[(n)]</b>	плавающий (приблизительный) тип длиной в 4 байта
<b>Datetime</b>	тип данных для хранения даты (4 первых байта) и времени (4 последних байта)
<b>Smalldatetime</b>	тип данных для хранения даты (первых 2 байта) и времени (последние 2 байта)
<b>Money</b>	тип данных для хранения больших денежных величин с точностью до 4 знаков после запятой; для хранения данных отводится 8 байт.
<b>Smallmoney</b>	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.
<b>Bit</b>	битовый (логический) тип со значениями 0 и 1; для хранения выделяется 1 разряд байта памяти.

Создать таблицу в Microsoft SQL Server можно тремя способами:

- с помощью визуального интерфейса SQL Server Management Studio (SSMS), выбрав в обозревателе объектов Tables и нажав в контекстном меню New (рисунок 4),
- с помощью инструкции на языке T-SQL,
- с помощью графического конструктора SSMS, в существующей диаграмме баз данных.

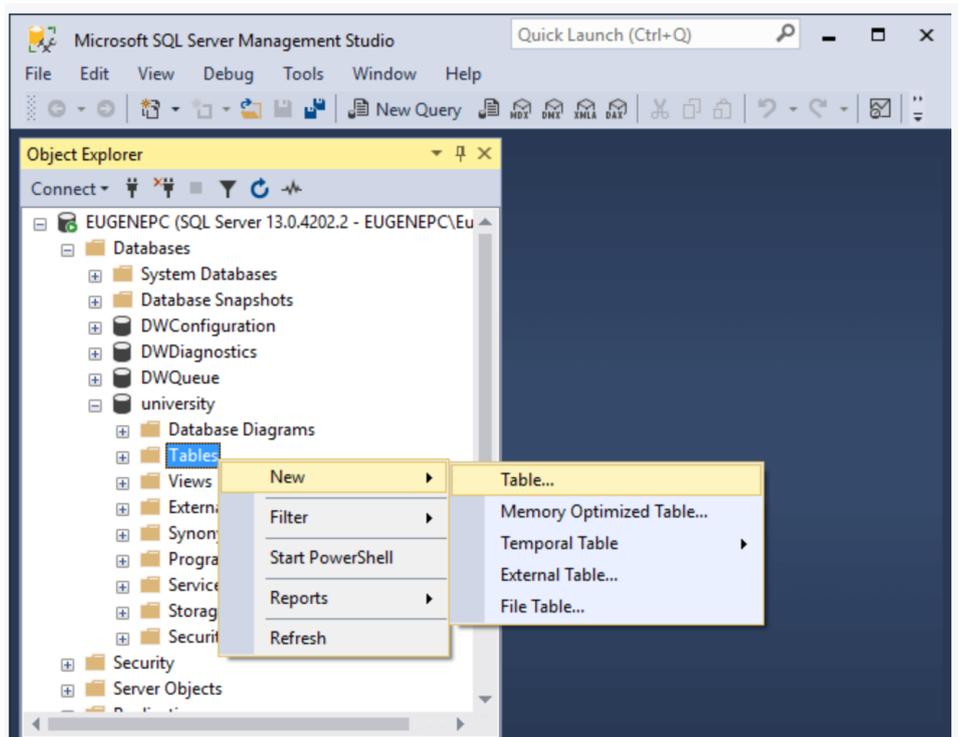


Рисунок 4 – Создание таблицы

#### Задание к практической работе №4

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.

**Практическая работа №5**  
**Создание ключевых полей.**  
**Задание индексов.**

**Установление и удаление связей между таблицами**

**Цель работы:** Научиться создавать ключевые поля в таблицах, создавать индексы, устанавливая и удаляя связи между таблицами.

**Теоретические положения**

Атрибуты используются не только для описания свойств сущностей и связей, но и для идентификации их экземпляров. Экземпляр сущности однозначно идентифицируется набором всех ее атрибутов.

Однако в большинстве случаев полный набор атрибутов является избыточным для целей идентификации.

Потенциальным ключом называется подмножество атрибутов сущности, которое функционально полно определяет значение любого атрибута. Термин «функциональная зависимость» означает однозначную зависимость от ключа.

Потенциальный ключ может быть:

- простой – состоит из одного атрибута, например, «табельный номер»;
- составной – состоит из нескольких атрибутов, например, («серия паспорта», «номер паспорта»).

Потенциальный ключ обладает следующими свойствами:

- А) Уникальностью
- Б) Неизбыточностью (никакое подмножество ключа не может выступать в роли ключа)
- В) Обязательностью (определенностью) – ни при каких условиях атрибуты ключа не могут принимать неопределенные (Null) значения.

Потенциальный ключ, чья идентифицирующая роль обусловлена семантикой данных, иногда называют естественным ключом. В качестве альтернативы часто используется т. н. искусственный (суррогатный) ключ. Искусственный ключ – это дополнительный атрибут, никак не связанный с содержанием данных.

Среди потенциальных ключей выбирают по соображениям простоты манипулирования один, первичный ключ, остальные ключи называются альтернативными.

Важной характеристикой связи между старшей сущностью (родительской или обобщением) и младшей (потомком или категорией) является внешний ключ.

Внешний ключ – это первичный ключ другой сущности (родительской), который мигрирует (копируется) в дочернюю сущность и служит для связи сущностей.

Ошибки при связывании таблиц возникают если:

- 1) Связываемые поля имеют различный тип данных
- 2) Данные в полях противоречат друг другу (внешний ключ содержит данные, отличные от значений первичного ключа).

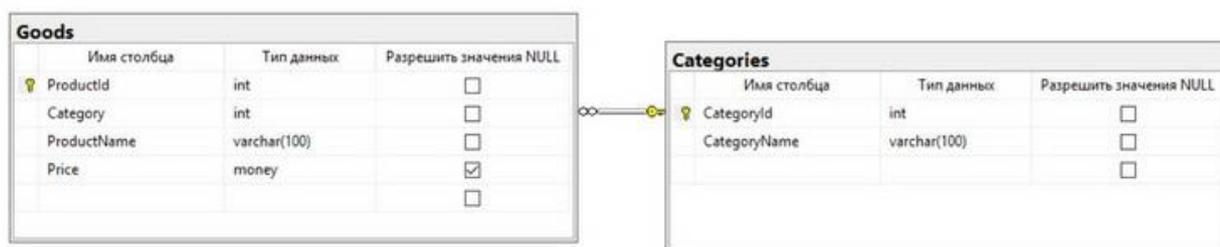


Рисунок 5 – Установка связей между таблицами

Другим элементом управления реализацией модели является т. н. не-уникальный индекс (инверсионный вход, IE). Статус индекса присваивается атрибуту или группе атрибутов, которые, не будучи ключом, позволяют получить доступ к экземплярам сущности в упорядоченном по индексу виде.

**Индексы** представляют собой структуру, позволяющую выполнять ускоренный доступ к строкам таблицы на основе значений одного или более ее столбцов. Наличие индекса может существенно повысить скорость выполнения некоторых запросов и сократить время поиска необходимых данных за счет физического или логического их упорядочивания.

Индексы можно создавать на большинстве типов данных, включая XML.

#### Создание стандартного индекса

1. Щелкните правой кнопкой мыши на таблице и выберите **Индексы/Ключи**. Открывается диалоговое окно **Индексы/Ключи**.
2. Нажмите кнопку **Добавить**.  
Новый индекс появится в списке **Выбранный первичный (уникальный) ключ или индекс** с именем по умолчанию, подобным IX\_Table1.
3. Выберите строку **Столбцы** и нажмите кнопку с многоточием. Отображается диалоговое окно **Столбцы индекса**.
4. Щелкните стрелку раскрывающегося списка под полем **Имя столбца** и выберите столбец, на котором необходимо создать индекс.

#### Задание к практической работе №5

1. В созданных таблицах определите первичные ключи, индексы.
2. Свяжите таблицы.

## Практическая работа №6 Создание диаграммы БД

**Цель работы:** Научиться создавать диаграммы базы данных.

### Теоретические положения

Диаграммы баз данных обеспечивают визуальное представление структуры и отношений таблиц в базе данных (схему базы данных). Включение их в состав базы данных является удобным способом документирования схемы, поскольку диаграммы автоматически отражают любые внесенные изменения.

Хотя возможно создать всю схему базы данных из окна Database Diagram (Диаграмма базы данных), чаще всего диаграммы создают из существующих таблиц. Для этого достаточно выбрать таблицы, которые необходимо включить в диаграмму (рисунок 6).

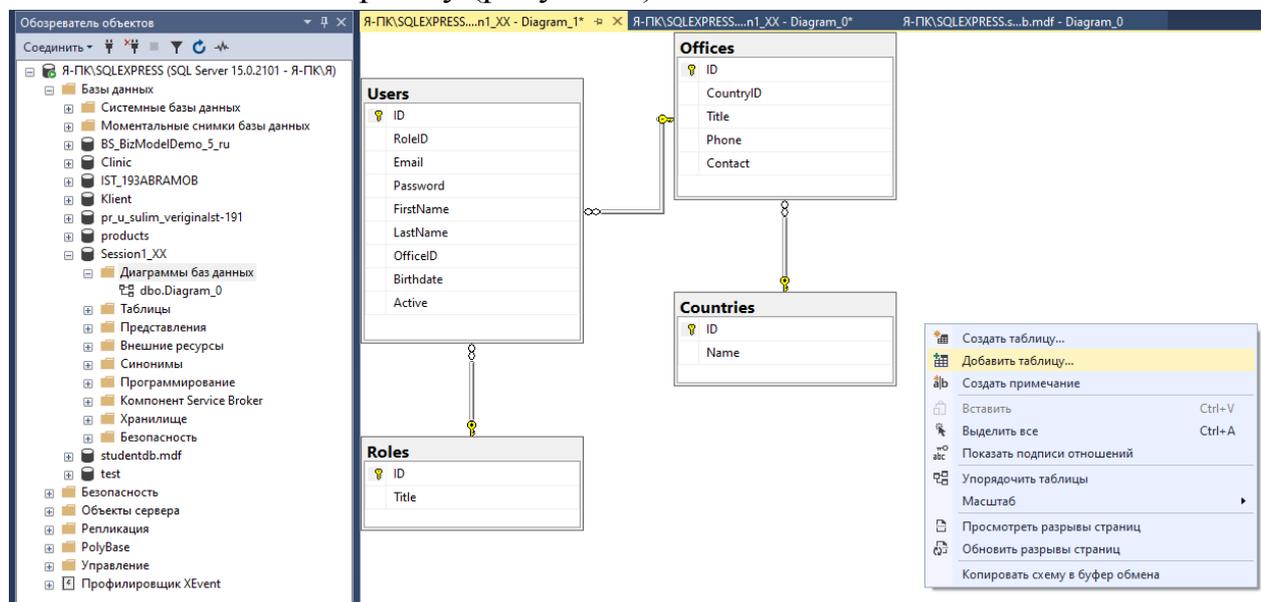


Рисунок 6 – Добавление таблицы в диаграмму БД

После того, как создана диаграмма базы данных, можно добавлять и удалять таблицы, как из нее, так и из базы данных. Все изменения в диаграмме не будут отправлены в базу данных до тех пор, пока диаграмма не будет сохранена. В случае возникновения проблем и конфликтов появляется диалоговое окно, содержащее дополнительные сведения.

При добавлении таблицы в диаграмму базы данных эта таблица появляется вместе с рядом отображаемых свойств. Количество отображаемых сведений можно задать в контекстном меню, выбрав пункт «Вид таблицы». Возможные варианты: стандартное, имена столбцов, ключи, только имя, другой, настраиваемый.

### Задание к практической работе №6

1. В программе ERWin сгенерировать скрипт для создания объектов БД.
2. Создать и присоединить БД к экземпляру SQL Server. Имя БД – Группа\_ФамилияИмя, например, «ИСТ191\_СмирновА».
3. Выполнить скрипт по созданию объектов в целевой БД.
4. Создать диаграмму БД.
5. Добавить к таблицам поля: «КраткаяХарактеристика», «ФотоПреподавателя».
6. Добавить к таблицам ограничения NOT NULL, PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, DEFAULT. Заполнить таблицы данными. Выбрать стратегию обеспечения ссылочной целостности для операций удаления, обновления и вставки данных в созданные таблицы.  
Например:
  - А) Для поля [Год поступления в ВУЗ] установить ограничение CHECK > 2000.
  - Б) Установить значение по умолчанию для поля E-mail студента «student@kuzstu.ru».
7. Для искусственных первичных ключей установить значение свойства «Автозаполнение» на True. Заполнить таблицы данными.

### Практическая работа №7

#### 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 использовать 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 один символ.
- Квадратные скобки ([]): означают любой один символ из заданного диапазона. Например, [a-z] означает любой один 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

Функция	Описание
Day()	Возвращает целое число, представляющее дату (день месяца) указанного значения типа <i>date</i> .
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 практической работе №7

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.

### Практическая работа №8

#### 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: у нас есть две таблицы – "Orders" (потомок) и "Customers" (родитель). 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, customers.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 практической работе №8

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 плаванием. 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.

## Практическая работа №9

### 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:

1. **SUM** – функция суммирования значений. Она принимает на вход столбец или выражение и возвращает сумму значений.

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

2. **COUNT** – функция подсчета количества значений. Она может принимать столбец или выражение и возвращает количество строк.

Пример: `SELECT COUNT(*) FROM products;`

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

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

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

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

5. **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 по-

ле, 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 практической работе №9**

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.

### **Практическая работа №10 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 практической работе №10

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.

## Практическая работа №11

### Создание запросов на обновление и добавление данных

**Цель работы:** Приобрести навыки создания запросов на обновление и добавление 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. 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;
```

Для добавления данных в таблицу используется оператор INSERT INTO. Синтаксис запроса выглядит так:

```
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. Например:

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

### **Задание к практической работе №11**

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.

### **Практическая работа №12**

#### **Создание запросов на удаление данных**

**Цель работы:** Приобрести навыки создания запросов на удаление данных.

#### **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.

### **Задание к практической работе №12**

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.

### Список литературы

1. Федорова, Г. Н. Основы проектирования баз данных [Текст] : учебное пособие для образовательных учреждений, реализующих программы среднего профессионального образования по специальности "Информационные системы (по отраслям)" : [для студентов СПО] / Г. Н. Федорова. – Москва : Академия, 2017. – 224 с. – Доступна электронная версия: <http://www.academia-moscow.ru/catalogue/4831/296505/>
2. Голицына, О. Л. Основы проектирования баз данных. – Москва : НИЦ ИНФРА-М, 2016. – 416 с. – Режим доступа: <http://znanium.com/go.php?id=552969>. – Загл. с экрана. (14.12.2018)
3. Шустова, Л. И. Базы данных. – Москва : НИЦ ИНФРА-М, 2018. – 304 с. – Режим доступа: <http://znanium.com/go.php?id=967755>. – Загл. с экрана. (14.12.2018)
4. Цветкова, М. С. Информатика [Электронный ресурс] : учебник для использования в учебном процессе образовательных учреждений СПО на базе основного общего образования с получением среднего общего образования / М. С. Цветкова, И. Ю. Хлобыстова. – Москва : Академия, 2017. – 352 с. – Режим доступа: <http://academia-moscow.ru/reader/?id=227485#copy>. – Загл. с экрана. (14.12.2018)

## Содержание

ПРАКТИЧЕСКАЯ РАБОТА №1. МОДЕЛИ ДАННЫХ.....	3
ПРАКТИЧЕСКАЯ РАБОТА №2. ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БД. НОРМАЛИЗАЦИЯ ТАБЛИЦ.....	6
ПРАКТИЧЕСКАЯ РАБОТА №3. ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БД. ER-МЕТОД...8	
ПРАКТИЧЕСКАЯ РАБОТА №4. СОЗДАНИЕ ПРОЕКТА БД. СОЗДАНИЕ БД. РЕДАКТИРОВАНИЕ И МОДИФИКАЦИЯ ТАБЛИЦ.....	13
ПРАКТИЧЕСКАЯ РАБОТА №5. СОЗДАНИЕ КЛЮЧЕВЫХ ПОЛЕЙ. ЗАДАНИЕ ИНДЕКСОВ. УСТАНОВЛЕНИЕ И УДАЛЕНИЕ СВЯЗЕЙ МЕЖДУ ТАБЛИЦАМИ.....	17
ПРАКТИЧЕСКАЯ РАБОТА №6. СОЗДАНИЕ ДИАГРАММЫ БД.....	19
ПРАКТИЧЕСКАЯ РАБОТА №7. СОЗДАНИЕ ЗАПРОСОВ НА ВЫБОРКУ ДАННЫХ ИЗ ОДНОЙ ТАБЛИЦЫ СОГЛАСНО ЗАДАННОМУ УСЛОВИЮ.....	20
ПРАКТИЧЕСКАЯ РАБОТА №8. СОЗДАНИЕ ЗАПРОСОВ НА ВЫБОРКУ ДАННЫХ ИЗ НЕСКОЛЬКИХ ТАБЛИЦ СОГЛАСНО ЗАДАННОМУ УСЛОВИЮ.....	24
ПРАКТИЧЕСКАЯ РАБОТА №9. ГРУППОВЫЕ ОПЕРАЦИИ. АГРЕГАТНЫЕ ФУНКЦИИ	26
ПРАКТИЧЕСКАЯ РАБОТА №10. ВЫЧИСЛЯЕМЫЕ ПОЛЯ. ПСЕВДОНИМЫ.....	27
ПРАКТИЧЕСКАЯ РАБОТА №11. СОЗДАНИЕ ЗАПРОСОВ НА ОБНОВЛЕНИЕ И ДОБАВЛЕНИЕ ДАННЫХ.....	29
ПРАКТИЧЕСКАЯ РАБОТА №12. СОЗДАНИЕ ЗАПРОСОВ НА УДАЛЕНИЕ ДАННЫХ ...	30
СПИСОК ЛИТЕРАТУРЫ.....	32