МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

УПРАВЛЕНИЕ ДАННЫМИ

Методические указания для практических занятий

Направление подготовки 15.03.04 Автоматизация технологических процессов и производств

Направленность (профиль) Информационно-управляющие системы

Квалификация выпускника – бакалавр

Невинномысск 2026

ОГЛАВЛЕНИЕ

Практическая работа №1. Проектирование базы данных: 4
Практическая работа №2. Установка соединения с сервером Microsoft SQL Server и принципы создания баз данных 39
Практическая работа №3. Разработка таблиц и ограничений 63
Практическая работа №4. Введение в язык SQL. Создание таблиц и ограничений на SQL 74
Практическая работа №5. Создание запросов на выборку. Отбор строк по условию 104
Практическая работа №6. Создание многотабличных запросов. Запросы на соединение 118
Практическая работа №7. Создание запросов на группировку и сортировку данных. Запросы на изменение. Использование встроенных функций. 130
Практическая работа №8. Создание и управление представлениями 150
Практическая работа №9. Основы программирования с помощью встроенного языка Transact-SQL в Microsoft SQL Server 152
Практическая работа №10. Создание, изменение, применение и удаление функций и хранимых процедур 171
Практическая работа №11. Создание, программирование и управление триггерами 176
Практическая работа №12. Создание и управление транзакциями 180
Практическая работа №13. Создание, применение и управление курсорами 184
Практическая работа №14. Система безопасности SQL Server 187
Практическая работа №15. Администрирование сервера баз данных MS SQL Server 195
Практическая работа №16. Администрирование сервера баз данных MS SQL Server (продолжение) 201
Библиографический список 205

ПРАКТИЧЕСКАЯ РАБОТА №1. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ:

Цель практической работы

Получить теоретические знания и практические навыки реализации баз данных (БД). Осуществить анализ предметной области. Освоить концептуальное проектирование и научиться определять сущности и атрибуты БД. Научиться разрабатывать инфологическую модель БД в виде ER-диаграмм. Получить теоретические знания и практические навыки при физическом проектировании баз данных (БД). Научиться создавать даталогическую модель БД.

I. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ БАЗЫ ДАННЫХ

1.1. Понятие БД и СУБД

Информационная система - система, реализующая автоматизированный сбор, обработку и манипулирование данными и включающая технические средства обработки данных, программное обеспечение и соответствующий персонал.

Цель любой информационной системы - обработка данных об объектах реального мира. Основой информационной системы является **база данных**. В широком смысле слова **база данных** - это совокупность сведений о конкретных объектах реального мира в какой-либо предметной области.

Под **предметной областью** принято понимать часть реального мира, подлежащего изучению для организации управления и в конечном счете автоматизации, например, предприятие, вуз и т.д.

Создавая базу данных, пользователь стремится упорядочить информацию по различным признакам и быстро производить выборку с произвольным сочетанием признаков. Большое значение при этом приобретает структурирование данных.

Структурирование данных - это введение соглашений о способах представления данных.

Неструктурированными называют данные, записанные, например, в текстовом файле.

Ниже приведен пример неструктурированных и структурированных данных, содержащих сведения о студентах (номер личного дела, фамилию, имя, отчество и год рождения).

Неструктурированные данные:

Личное дело № 16493. Сергеев Петр Михайлович, дата рождения 1 января 1976 г.; Л/д № 16593, Петрова Анна Владимировна, дата рожд. 15 марта 1975 г.; № личн. дела 16693, д.р. 14.04.76, Анохин Андрей Борисович

Легко убедиться, что сложно организовать поиск необходимых данных, хранящихся в неструктурированном виде.

Структурированные данные:

№ личного	Фамилия	Имя	Отчество	Дата рождения
16493	Сергеев	Петр	Михайлович	01.01.76
16593	Петрова	Анна	Владимировна	15.03.75
16693	Анохин	Андрей	Борисович	14.04.76

В современной технологии баз данных предполагается, что создание базы данных, ее поддержка и обеспечение доступа пользователей к ней осуществляются централизованно с помощью специального программного инструментария - **системы управлении базами данных (СУБД).**

База данных (БД) - это поименованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области.

Объектом называется элемент предметной области, информацию о котором мы сохраняем.

Объект может быть **реальным** (например, человек, изделие; или населенный пункт) и **абстрактным** (например, событие, счет покупателя или изучаемый студентами курс).

Так, в области продажи автомобилей примерами **объектов** могут служить **МОДЕЛЬ АВТОМОБИЛЯ**, **КЛИЕНТ** и **СЧЕТ**. На товарном складе - это **ПОСТАВЩИК**, **ТОВАР, ОТПРАВЛЕНИЕ** и т. д.

Понятие базы данных тесно связано с такими понятиями структурных элементов, как **поле, запись, файл (таблица)** (рис.1).

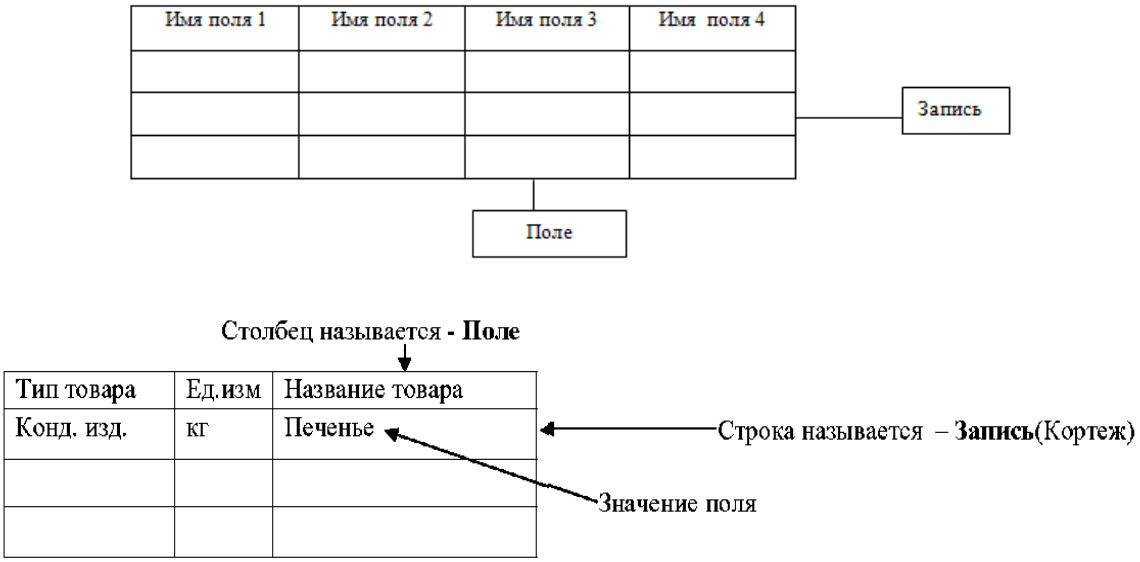


Рис.1 Основные структурные элементы БД

1.2. Структурные элементы базы данных

Поле - элементарная единица логической организации данных, которая соответствует неделимой единице информации - реквизиту. Для описания поля используются следующие характеристики:

- ***имя***, например, **Фамилия, Имя, Отчество, Дата рождения**;
- ***тип***, например, символьный, числовой, денежный;
- ***длина***, например, 15 байт, причем будет определяться максимально возможным количеством символов;
- ***точность*** для числовых данных, например два десятичных знака для отображения дробной части числа,

Запись - совокупность логически связанных полей.

Экземпляр записи - отдельная реализация записи, содержащая конкретные значения ее полей.

Файл (таблица) - совокупность экземпляров записей одной структуры.

Описание логической структуры записи файла содержит последовательность

расположения полей записи и их основные характеристики, как это показано на рис. 2 и 3.

Имя файла					
Поле		Признак ключа	Формат поля		
Имя (обозначение)	Полное наименование		Тип	Длина	Точность (для чисел)
имя 1					
имя N					

Рис. 2. Описание логической структуры записи файла

В структуре записи файла указываются поля, значения которых являются ключами: **первичными (ПК) и внешними (ВК),**

Первичный ключ (ПК) - это одно или несколько полей, однозначно идентифицирующих запись. Если первичный ключ состоит из одного поля, он называется **простым**, если из нескольких полей - **составным** ключом.

Внешний ключ (ВК) - это одно или несколько полей, которые выполняют роль поисковых или группировочных признаков. В отличие от первичного, значение внешнего ключа может повторяться в нескольких записях файла, то есть он не является уникальным. Если по значению первичного ключа может быть найден один единственный экземпляр записи, то по внешнему - несколько.

Имя файла: СТУДЕНТ					
Поле		Признак ключа	Формат поля		
Обозначение	Наименование		Тип	Длина	Точность
Номер	№ личного дела	•	Симв	5	
Фамилия	Фамилия студента		Симв	15	
Имя	Имя студента		Симв	10	
Отчество	Отчество студента		Симв	15	
Дата	Дата рождения		Дата	8	

Рис. 3. Описание логической структуры записи файла **СТУДЕНТ**

1.3. Понятие модели данных

Для того, чтобы спроектировать структуру базы данных, необходима исходная информация о предметной области. Желательно, чтобы эта информация была представлена в формализованном виде.

Такое формализованное описание предметной области будем называть **инфологической** (infological) **моделью предметной области** (ИЛМ) или **концептуальной моделью** (КМ).

Ядром любой базы данных является модель данных. **Модель данных** представляет собой множество структур данных, ограничений целостности и операций манипулирования данными. С помощью модели данных могут быть представлены объекты предметной области и взаимосвязи между ними.

Модель данных - совокупность структур данных и операций их обработки.

СУБД основывается на использовании **иерархической, сетевой или реляционной модели**, на комбинации этих моделей или на некотором их подмножестве.

Самой распространенной моделью данных является - **реляционная**.

1.3.1. Иерархическая модель данных

Иерархическая модель организует данные в виде древовидной структуры

К основным понятиям иерархической структуры относятся: **уровень, элемент (узел), связь**. Дерево представляет собой иерархию элементов, называемых узлами. **Узел** - это совокупность атрибутов данных, описывающих некоторый объект. На самом верхнем уровне иерархии имеется один и только один узел - **корень**. Каждый узел, кроме корня, связан с одним узлом на более высоком уровне, называемым **исходным** для данного узла. Ни один элемент не имеет более одного исходного. Каждый элемент может быть связан с одним или несколькими элементами на более низком уровне. Они называются **порожденными** (рис. 4).

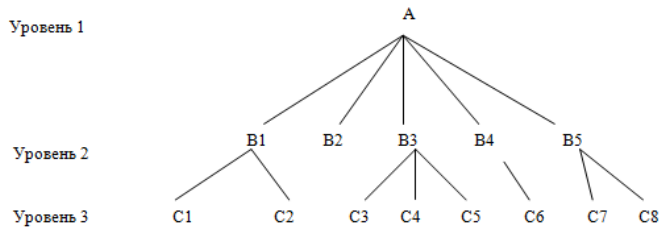


Рис. 4. Графическое изображение иерархической структуры БД

К каждой записи базы данных существует только один (иерархический) путь от корневой записи. Например, как видно из рис. 4, для записи С4 путь проходит через записи А и В3.

1.3.2. Сетевая модель данных

Сетевая модель организует данные в виде сетевой структуры.

Структура называется сетевой, если в отношениях между данными порожденный элемент имеет более одного исходного.

В сетевой структуре при тех же основных понятиях (уровень, узел, связь) каждый элемент может быть связан с любым другим элементом.

На рис. 5 изображена сетевая структура базы данных в виде графа.

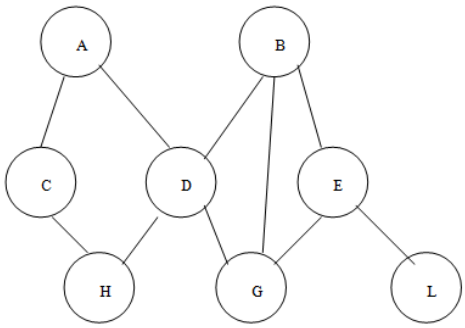


Рис. 5. Графическое изображение сетевой структуры

1.3.3. Реляционная модель данных

Реляционная модель данных является совокупностью взаимосвязанных двумерных таблиц объектов модели.

Например, реляционной таблицей можно представить информацию о студентах, обучающихся в вузе (рис. 6).

№ личного дела	Фамилия	Имя	Отчество	Дата рождения	Группа
16493	Сергеев	Петр	Михайлович	01.01.76	111
16593	Петрова	Анна	Владимировна	15.03.75	112
16693	Анохин	Андрей	Борисович	14.04.76	111

Рис. 6. Пример реляционной таблицы

Связи между двумя логически связанными таблицами в реляционной модели устанавливаются по равенству значений одинаковых атрибутов этих таблиц.

Каждая реляционная таблица представляет собой двумерный массив и обладает следующими ***свойствами:***

- **- каждый элемент таблицы - один элемент данных;**
- **- все столбцы в таблице однородные, т.е. все элементы в столбце имеют одинаковый тип (числовой, символьный и т.д.) и длину;**
- **- каждый столбец имеет уникальное имя;**
- **- одинаковые строки в таблице отсутствуют;**
- **- порядок следования строк и столбцов может быть произвольным.**

При описании реляционной модели часто используют следующие термины: **отношение, кортеж, домен.**

Отношения представлены в виде таблиц, строки которых соответствуют **записям (кортежам)**, а столбцы полям, атрибутам отношений (**доменам**).

Поле, каждое значение которого однозначно определяет соответствующую запись, называется простым ключом (ключевым полем). Если записи однозначно определяются значениями нескольких полей, то такая таблица базы данных имеет **составной ключ.**

В примере, показанном на рис.6, ключевым полем таблицы является "№ личного дела".

Между двумя реляционными таблицами могут быть сформированы **связи.** Различные таблицы, могут быть **связаны между собой через общее поле данных**.

На рис. 7 показан пример реляционной модели, построенной на основе отношений: **СТУДЕНТ, СЕССИЯ, СТИПЕНДИЯ**.

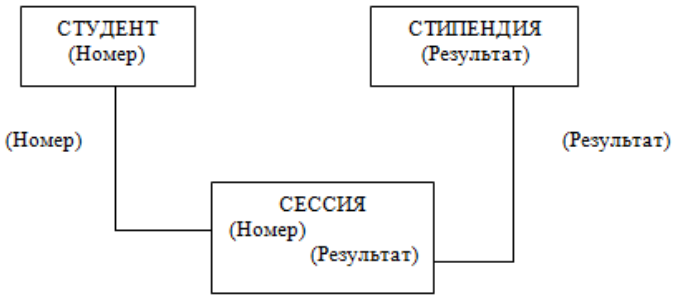


Рис.7. Пример реляционной модели

Таблица **СТУДЕНТ** имеет поля: Номер, Фамилия, Имя, Отчество, Дата рождения, Группа;

СЕССИЯ - Номер, Оценка 1, Оценка 2, Оценка 3, Оценка 4, Результат;

СТИПЕНДИЯ - Результат, Процент

Таблицы **СТУДЕНТ** И **СЕССИЯ** имеют совпадающие ключи (**Номер**), что дает возможность легко организовать связь между ними.

Таблица **СЕССИЯ** имеет **первичный ключ Номер** и содержит **внешний ключ Результат**, который обеспечивает ее связь с таблицей **СТИПЕНДИЯ**.

Благодаря имеющимся связям достигаются следующие преимущества:

1. Удается избежать дублирования информации. Все необходимые данные можно хранить только в одной таблице. Так, например, нет необходимости в таблице **СЕССИЯ** хранить номер группы каждого студента, сдающего экзамены, достаточно задать связь с таблицей **СТУДЕНТ**.

2. В реляционных базах данных легко производить изменения Если в таблице

СЕССИЯ изменить какие-нибудь значения, то правильная информация автоматически будет связана с другими таблицами, ссылающимися на первую (например, таблица **СТИПЕНДИЯ**).

3. В нереляционных базах данных сложно передать все имеющиеся зависимости, т.е. связать друг с другом данные из различных таблиц Реляционная база данных выполняет все эти действия автоматически.

4. В реляционных базах данных удается легко избежать установления ошибочных связей между различными таблицами данных, а необходимый объем памяти сокращен до минимума.

2. ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

2.1. Анализ и описание предметной области БД

Предметной областью называется фрагмент реальности, который описывается или моделируется с помощью БД и ее приложений. В предметной области выделяются информационные объекты – идентифицируемые объекты реального мира, процессы, системы, понятия и т.д., сведения о которых хранятся в БД.

Анализ предметной области выполняется в следующем порядке: название предприятия, цель деятельности предприятия, структура предприятия, информационные потребностей пользователей. Для описания ПО следует привести основные решаемые задачи БД. Дать словесное описание процесса функционирования ПО, проанализировать основные хозяйственные операции, которые совершаются в ПО. Привести анализ структуры предприятия, перечислить задачи, решаемые отдельными подразделениями.

Анализ предметной области целесообразно разбить па три фазы:

1. анализ концептуальных требований и информационных потребностей;
2. выявление информационных объектов и связей между ними;
3. построение концептуальной модели предметной области и проектирование концептуальной схемы БД.

2.2. Анализ концептуальных требований и информационных потребностей

Требования пользователей к разрабатываемой БД представляют собой список запросов с указанием их интенсивности и объемов данных. Эти сведения разработчики БД получают в диалоге с ее будущими пользователями. Здесь же выясняются требования к вводу, обновлению и корректировке информации. Требования пользователей уточняются и дополняются при анализе имеющихся и перспективных задач.

Рассмотрим примерный состав вопросника при анализе различных предметных областей.

Пример. Предлагается разработать БД для учета студентов вуза.

Анализ предметной области:

1. Сколько студентов учится в вузе? 2. Сколько факультетов и отделений в вузе? 3. Как распределены студенты по факультетам отделений и курсам? 4. Сколько дисциплин читается на каждом курсе по каждой специальности? 5. Как часто обновляется информация в БД? 6. Сколько преподавателей в вузе? 7. Сколько иногородних студентов живет в общежитии, на частных квартирах? 8. Сколько лекционных аудиторий и аудиторий для проведения практических занятий, лабораторий? 9. Какая преемственность существует между читаемыми курсами? 10. Как информация, представленная в п.п. 1-9, используется в настоящее время (расписание занятий, экзаменов, зачетов и т.д.) и как ее собираются использовать? 11. Сколько раз в день, сколько человек и кто пользуются БД?

2.3. Выявление информационных объектов и связей между ними

Вторая фаза анализа предметной области состоит в выборе информационных объектов, задании необходимых свойств для каждого объекта, выявлении связей между объектами, определении ограничений, накладываемых на информационные объекты, типы связей между ними, характеристики информационных объектов- Проанализируем предметную область на примере БД "Видеомагнитофоны".

При выборе информационных объектов постараемся ответить на ряд вопросов:

1. На какие классы можно разбить данные, подлежащие хранению в БД?
2. Какое имя можно присвоить каждому классу данных?
3. Какие наиболее интересные характеристики (с точки зрения пользователя) каждого класса данных можно выделить?
4. Какие имена можно присвоить выбранным наборам характеристик?

Пример. БД "Видеомагнитофоны", рассчитанной на пользователей, которые хотят приобрести данный вид техники.

После беседы с различными пользователями и просмотра каталогов было выяснено, что интерес представляют три информационных объекта: видеомагнитофон, видеоплеер, видеокассета. Рассмотрим наиболее существенные характеристики каждого информационного объекта.

Объект - ВИДЕОМАГНИТОФОН.

Атрибуты - страна-изготовитель, фирма-изготовитель, № модели, телевизионные системы, число кассетных гнезд, ресурс непрерывной работы, система автопоиска, напряжение в сети, наличие таймера, число программ, габаритные размеры, масса, цена в долларах, год выпуска.

Объект - ВИДЕОПЛЕЙЕР,

Атрибуты - страна-изготовитель, фирма-изготовитель, № модели, телевизионные системы, число воспроизводящих головок, ресурс непрерывной работы, напряжение в сети, наличие таймера, габаритные размеры, масса, цена в долларах, год выпуска.

Объект - ВИДЕОКАССЕТА.

Атрибуты - наименование, страна-изготовитель, фирма-изготовитель, тип кассеты, время проигрывания, цена в долларах.

Далее выделим связи между информационными объектами. В ходе этого процесса постараемся ответить на следующие вопросы:

1. Какие типы связей между информационными объектами?
2. Какое имя можно присвоить каждому типу связей?
3. Каковы возможные типы связей, которые могут быть использованы впоследствии?
4. Имеют ли смысл какие-нибудь комбинации типов связей?

Попытаемся задать ограничения на объекты и их характеристики.

Под **ограничением целостности** обычно понимают логические ограничения, накладываемые на данные. Ограничение целостности - это такое свойство, которое мы задаем для некоторого информационного объекта или его характеристики и которое должно сохраняться для каждого их состояния.

Введем следующие ограничения:

1. Значение атрибута "число кассетных гнезд" изменяется от 1 до 2.
2. Значение атрибута "ресурс непрерывной работы" изменяется от 4 до 24.
3. Значение атрибута "напряжение в сети" изменяется от 110 до 240 В.
4. Значение атрибута "число программ" изменяется от 1 до 20 и т.д.

Связи между различными классами объектов.

Помимо классов объектов в ИЛМ отображают связи между различными классами объектов. Такие связи моделируют отношения между объектами различных видов в реальном мире. При отборе связей помещаемых в ИЛМ следует руководствоваться информационными потребностями пользователей базы данных.

Каждая связь характеризуется именем, типом, классом принадлежности и направлением. Имя связи должно быть глагольным оборотом, например **«Принадлежит», «Закреплены за», «Входит в»** и т.д.

Все информационные объекты предметной области связаны между собой.

Соответствия, отношения, возникающие между объектами предметной области, называются связями. Различаются связи нескольких типов, для которых введены следующие обозначения:

Различают четыре типа связи:

- **«один к одному» (1:1);**
- **«один ко многим» (1:M);**
- **«многие к одному» (M:1)**
- **«многие ко многим» (M:M).**

Рассмотрим эти типы связей на примерах.

Пример. Дана совокупность информационных объектов, отражающих учебный процесс в вузе:

СТУДЕНТ (Номер, Фамилия, Имя, Отчество, Пол, Дата рождения, Группа)
СЕССИЯ (Номер. Оценка 1, Оценка 2, Оценка 3, Оценка 4, Результат)
СТИПЕНДИЯ (Результат, Процент)
ПРЕПОДАВАТЕЛЬ (Код преподавателя, Фамилия, Имя, Отчество)

Связь один к одному (1:1) предполагает, что в каждый момент времени одному экземпляру информационного объекта А соответствует не более одного экземпляра информационного объекта В и наоборот. Рис. 8 иллюстрирует указанный тип отношений.

Рис. 8. Графическое изображение реального отношения 1:1

Примером связи 1:1 может служить связь между информационными объектами СТУДЕНТ и СЕССИЯ:

СТУДЕНТ <-> СЕССИЯ

Каждый студент имеет определенный набор экзаменационных оценок в сессию.

При связи **один ко многим (1 : M**) одному экземпляру информационного объекта А соответствует 0, 1 или более экземпляров объекта В, но каждый экземпляр объекта В связан не более чем с 1 экземпляром объекта А. Графически данное соответствие имеет вид, представленный на рис. 9.

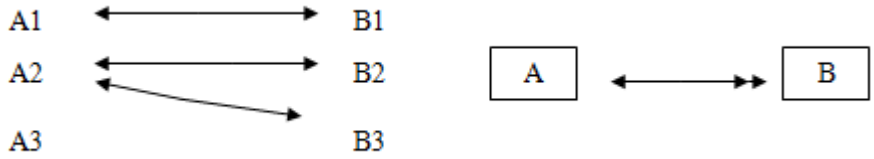


Рис. 9. Графическое изображение реального отношения 1:M

Примером связи 1: M служит связь между информационными объектами СТИПЕНДИЯ и СЕССИЯ:

СТИПЕНДИЯ < -- >> СЕССИЯ

Установленный размер стипендии по результатам сдачи сессии может повторяться многократно для различных студентов.

Связь **многие ко многим** (M:M) предполагает, что в каждый момент времени одному экземпляру информационного объекта A соответствует 0, 1 или более экземпляров объекта B и наоборот. На рис. 10 графически представлено указанное соответствие.

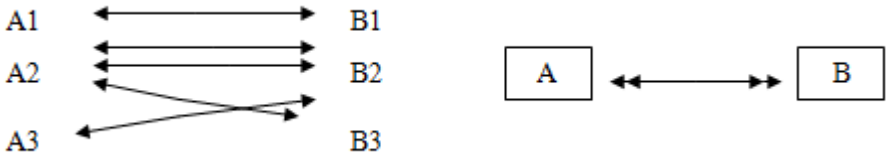


Рис. 10. Графическое изображение реального отношения M : M

Примером данного отношения служит связь между информационными объектами СТУДЕНТ и ПРЕПОДАВАТЕЛЬ:

СТУДЕНТ <<—>>ПРЕПОДАВАТЕЛЬ

Один студент обучается у многих преподавателей, один преподаватель обучает многих студентов.

Связь «многие к одному» при создании БД физически обычно организуется путем введения дополнительного поля в таблицу со стороны «много». Это поле называется **внешний ключ.** На рис. 10. код группы - внешний ключ.

Студент

Фамилия	Имя	...	Код группы
Петров.	Иван	...	2
Сидоров	Сергей	...	2
Лаптева	Марина	...	1
Киров	Андрей	...	3
Жоголева	Ирина	...	3

Группа

Код группы	Название группы	Код факультета
1	КТ171	
2	МТ181	
3	ЕК382	

Рис. 10. Введение внешнего ключа в БД

2.4. Построение инфологической (концептуальной модели) предметной области

Заключительная фаза анализа предметной области состоит в проектировании ее инфологической структуры или концептуальной модели.

Концептуальная модель включает описания объектов и их взаимосвязей, представляющих интерес в рассматриваемой предметной области (ПО) и выявляемых в результате анализа данных.

Концептуальная модель применяется для структурирования предметной области с учетом информационных интересов пользователей системы. Она дает возможность систематизировать информационное содержание предметной области, позволяет как бы

"подняться вверх" над ПО и увидеть ее отдельные элементы. При этом уровень детализации зависит от выбранной модели.

Концептуальная модель является представлением точки зрения пользователя на предметную область и не зависит ни от программного обеспечения СУБД, ни от технических решений.

Концептуальная модель должна быть стабильной. Могут меняться прикладные программы, обрабатывающие данные, может меняться организация их физического хранения, концептуальная модель остается неизменной или увеличивается с целью включения дополнительных данных.

Одной из распространенных моделей концептуальной схемы является модель **«сущность - связь» (*ER-моделей*** (или ER-диаграмм))**.** Основными конструкциями данной модели являются сущности и связи.

Под **сущностью** понимают основное содержание объекта ПО, о котором собирают информацию. В качестве сущности могут выступать место, вещь, личность, явление.

Экземпляр **сущности** - конкретный **объект.**

Например:

сущность (объект) - служащий экземпляр сущности - Иванов А.В.; сущность (объект) - институт экземпляр сущности - МГУ.

Сущность принято определять **атрибутами** - поименованными характеристиками.

Например:

сущность - служащий атрибуты: ФИО, год рождения, адрес, образование и т.д.

Чтобы задать атрибут в модели, ему надо присвоить имя и определить область допустимых значений. Одно из назначении атрибута - идентифицировать сущность.

Связь определяет отношения между сущностями. **Типы связей: один к одному, один ко многим, многие ко многим.**

При построении модели «сущность - связь» используют **графические диаграммы**. При этом обозначают: сущности - прямоугольниками, атрибуты - овалами, связи - ромбами, см. рис.11.

На практике приходится строить несколько вариантов моделей, из которых выбирается одна, наиболее полно отображающая предметную область.

Классом объектов называют совокупность объектов, обладающих одинаковым набором свойств. Например, для объектов класса «СТУДЕНТ» таким набором свойств являются: «ГОД_РОЖДЕНИЯ», «ПОЛ» и др.

Объекты могут быть реальными, как названный выше объект «СТУДЕНТ», и абстрактными, как, например, «ПРЕДМЕТЫ», которые изучают студенты.

Пример. Спроектировать БД "Сессия". База данных должна выдавать оперативную информацию об успеваемости студентов на факультетах в семестре. Результатами сессии считать только экзамены.

По сути дела *и* БД исходя из формулировки задания можно выделить лишь одно приложение. Речь идет об успеваемости студентов разных факультетов по тем или иным дисциплинам. Более конкретно речь идет о выдаче справок по результатам сессии каждого студента, учебной группы, курса, факультета, а также об автоматизированном составлении ведомости

Выберем следующие сущности:

ИНСТИТУТ, ФАКУЛЬТЕТ, СТУДЕНТ, ПРЕПОДАВАТЕЛЬ, ДИСЦИПЛИНА.

В данном примере можно выделить сущность ЭКЗАМЕН или ВЕДОМОСТЬ, но можно не выделять, а сформировать ведомость из имеющихся данных посредством связей.

Зададим каждую сущность набором атрибутов:

ИНСТИТУТ (название, подчиненность, адрес, телефон, ФИО ректора)

ФАКУЛЬТЕТ (название, код специальности, данные о кафедрах, число выпускников, декан).

СТУДЕНТ (ФИО, группа, курс, номер текущего семестра, пол).

ПРЕПОДАВАТЕЛЬ (ФИО, должность, звание, кафедра, стаж).

ДИСЦИПЛИНА (название, число часов, код дисциплины, виды занятий, число читаемых семестров, номера текущих семестров, на каких курсах преподается)

В каждом наборе атрибутов, характеризующих сущность, необходимо выбрать ключевые атрибуты, т.е. атрибуты, делающие сущность уникальной. При задании атрибутов ключевые атрибуты подчеркивались.

Определим связи между сущностями.

Название связи	**Связи между сущностями**
учится	студент, факультет
изучает	студент, дисциплина
имеет	институт, факультет
работает	преподаватель, факультет
преподает	преподаватель, дисциплина
экзамен	студент, дисциплина, преподаватель

После выбора сущностей, задания атрибутов и анализа связей можно перейти к проектированию информационной (концептуальной) схемы БД.

Концептуальная схема БД "Успеваемость» представлена на рис.11 (атрибуты сущностей на диаграмме не показаны).

Рассмотрим некоторые **ограничения** в рассматриваемом примере:

1 Значение атрибута "телефон" (сущность - ИНСТИТУТ) задается целым положительным шестизначным числом.

2 Значение атрибута "код факультета" (сущность - ФАКУЛЬТЕТ) лежит в интервале 1-10.

3. Значение атрибута "курс" (сущность - СТУДЕНТ) лежит в интервале 1 - 6

4. Значение атрибута "семестр" (сущность - СТУДЕНТ, ДИСЦИПЛИНА) лежит в интервале 1-12.

5. Значение атрибута "число часов" (сущность - ДИСЦИПЛИНА) лежит в интервале 1-300.

6. Одному студенту может быть приписана только одна группа.

7. Один студент может учиться только на одном факультете.

8. Один студент в семестре сдает от 3 до 5 дисциплин

9. Один студент изучает в семестре от 6 до 12 дисциплин.

10. Одному преподавателю приписывается только одна кафедра.

11. Один студент может пересдавать одну дисциплину не более трех раз.

12. Ключи: название института, название факультета, ФИО и группа студента, ФИО и кафедра преподавателя, название дисциплины.

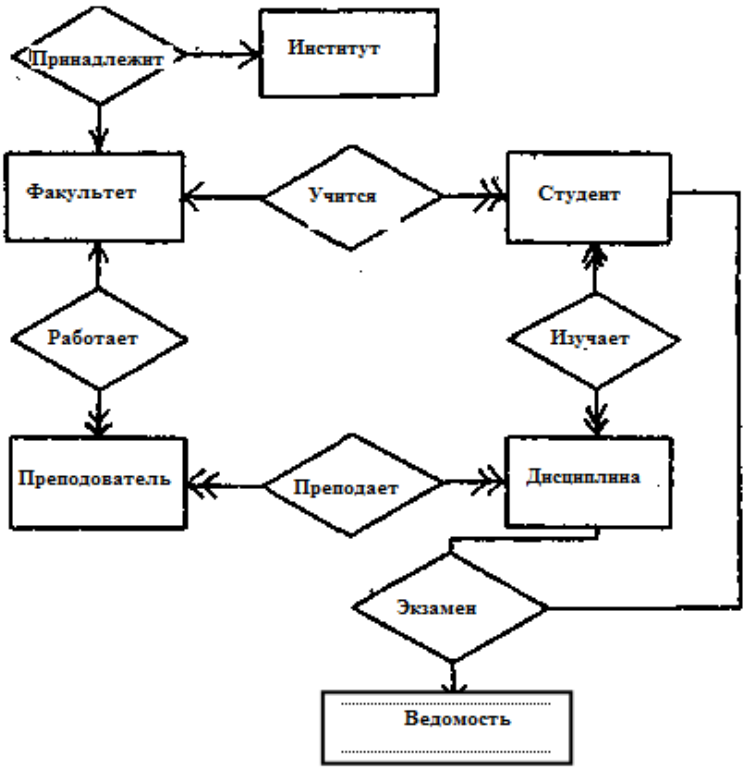


Рис 11. Концептуальная схема БД «Успеваемость»

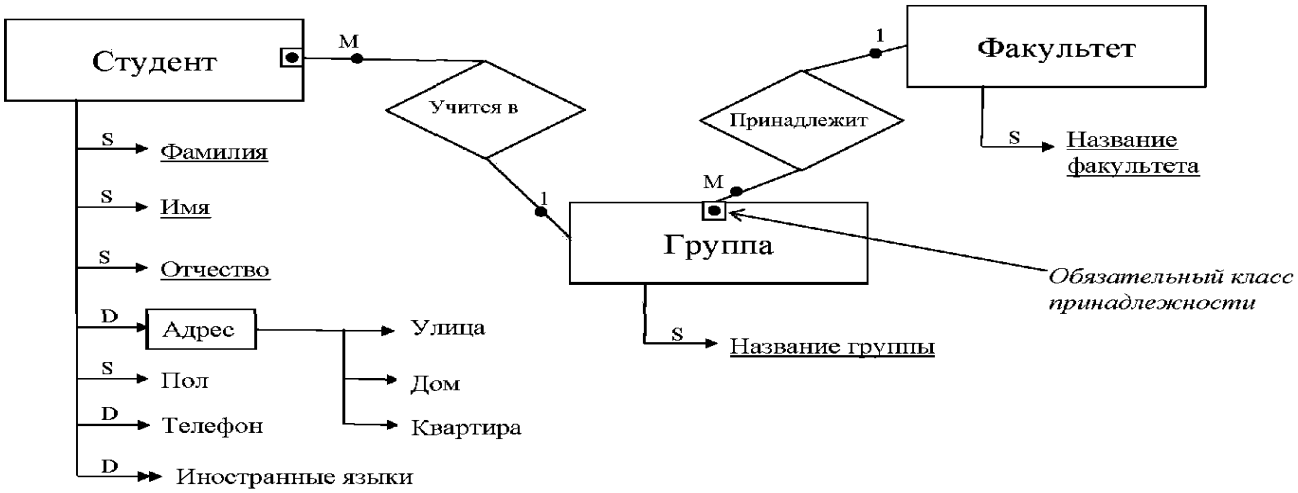


Рис. 12. Пример построения инфологической модели данных.

3. ФИЗИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

3.1. Даталогическое проектирование

Инфологическая модель является исходной для построения ***даталогической*** модели БД и служит промежуточной моделью для специалистов предметной области (для которой создается банк данных (БнД)) и администратора БД в процессе проектирования и разработки конкретной БД.

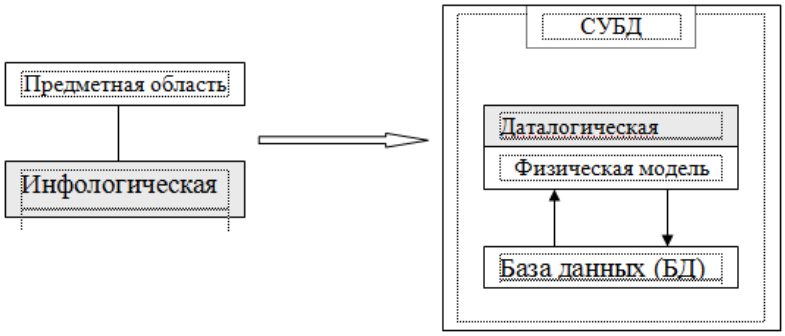


Рис. 13. Структура проектирования БД

Под ***даталогической*** понимается модель, отражающая логические взаимосвязи между элементами данных безотносительно их содержания и физической организации. При этом даталогическая модель разрабатывается с учетом конкретной реализации СУБД, также с учетом специфики конкретной предметной области на основе ее инфологической модели.

Инфологическая модель предметной области строится первой. Предварительная инфологическая модель строится еще на предпроектной стадии и затем уточняется на более поздних стадиях проектирования баз данных. Затем на ее основе строятся концептуальная (логическая), внутренняя (физическая) и внешняя модели.

Конечным результатом даталогического проектирования является описание логической структуры базы данных на языке программирования. Однако если проектирование выполняется «вручную», то для большей наглядности сначала строится схематическое графическое изображение структуры базы данных. При этом должно быть обеспечено однозначное соответствие между конструкциями языка описания данных и графическими обозначениями информационных единиц и связей между ними.

Графическое представление используется и при автоматизированном проектировании структуры базы данных как интерфейсное средство общения с проектировщиком, и при документировании проекта.

Спроектировать логическую структуру базы данных означает определить все информационные единицы и связи между ними, задать их имена; если для информационных единиц возможно использование разных типов, то определить их тип. Следует также задать некоторые количественные характеристики, например, длину поля.

3.2. Описание датологической модели.

Даталогическая модель представляет собой описание базы данных, выполненное в терминах используемой СУБД. Наиболее часто при разработках баз данных применяют реляционные СУБД. Для СУБД этого типа даталогическая удобно представить в виде набора таблиц специальной формы (табл. 1.4.).

Такая таблица составляется для каждого отношения, используемого в базе банных. Отношения в базе соответствуют классам объектов из инфологической модели. Кроме того, отношения могут представлять некоторые связи предметной области.

Каждой таблице нужно поставить в соответствие ее **ключи**. Схема ключа представляет собой перечисление атрибутов отношения, составляющих ключ.

Различают простые и составные ключи. **Простой ключ** строится на основе одного атрибута. **Составной ключ** строится на базе использования нескольких атрибутов.

Ключи принято разделять на **первичные, внешние** и **вспомогательные.**

Первичный индекс должен быть только один для каждой таблицы. Значения атрибутов, используемых для формирования первичного ключа, должны быть

уникальными для каждой записи в таблице. Значения первичного ключа уникально идентифицируют каждую запись. Не может быть двух записей в таблице с одинаковым значением первичного ключа. Например, в качестве первичного ключа для отношения «Сотрудники» можно выбрать атрибут «Табельный номер», значение которого является уникальным для каждой записи о сотруднике.

Внешние ключи используются для реализации связей типа **1:M** между отношениями. Внешний ключ строится для отношения, находящегося на стороне **«много»** связи 1:M. Для каждого такого отношения на даталогической модели должен быть показан внешний ключ. Внешний ключ всегда должен иметь соответствующий ему первичный ключ отношения, находящегося на стороне «один» связи типа 1:M.

Для связанных ключа «внешний - первичный» соединяются на схеме даталогической модели ломаной линией. Например, если в таблице «Сотрудники» имеется атрибут «Шифр категории», то этот атрибут можно использовать в качестве внешнего ключа для связи с таблицей «Категории». В последней таблице должен быть первичный ключ по полю «Шифр категории». Внешних ключей может быть несколько для одной таблицы.

Следует отметить, что первичные и внешние ключи строятся как правило на основе целочисленных атрибутов, а не атрибутов, имеющих строковый или вещественный тип.

Кроме первичных и внешних ключей часто используют **вспомогательные индексы.** Они применяются для реализации связей, получения нужного упорядочения при выводе на экран и создании отчетов и т.д. В каждом случае использования вспомогательного индекса, его необходимость должна быть обоснована. Применение большого количества индексов замедляет работу СУБД, т.к. операции над записями отношения требуют корректировки всех индексов.

II. ЗАДАНИЕ ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ РАБОТЫ

Практическая работа №1 выполняется письменно и в конце занятия сдается на проверку. После проверки будет выставлена оценка.

2.1. Выбор задания

Выбрать из таблицы «**Варианты заданий для лаб.работы №1.doc**» вариант задания, соответствующий **номеру студента в списке учебной группы**. Для всех последующих практических работ вариант остается неизменным. Каждому студенту предоставляется свой вариант предметной области (ПО), который он будет использовать в процессе выполнения всех практических работ.

2.2. Анализ предметной области.

На основании выбранного варианта привести: название предприятия, цель деятельности предприятия, структура предприятия, информационные потребностей пользователей (кратко).

2.3. Описание основных сущностей ПО.

Здесь следует привести описание основных сущностей (объектов) ПО. Отбор сущностей производится на основе анализа информационных потребностей. Необходимо привести таблицы описания сущностей (сущностей должно быть не менее 3-х)

Таблица 1.1. Список сущностей предметной области.

N п.п.	Наименование сущности	Краткое описание

Здесь же приводится отбор атрибутов (не менее 5-ти) для каждого экземпляра сущности. Отбираются только те атрибуты сущностей, которые необходимы для формирования ответов на регламентированные и непредусмотренные запросы. Для каждого объекта следует привести таблицы его атрибутов.

Таблица 1.2. Список атрибутов.

N п.п.	Наименование атрибута	Краткое описание

На основе анализа информационных запросов следует выявить связи между сущностями. Для выявленных связей также нужно заполнить таблицу 1.3.

Таблица 1.3. Список связей ПО.

N п.п.	Наименование связи	Сущности, участвующие в связи	Краткое описание

2.4. Построение инфологической модели.

На основании ранее выбранного варианта и таблиц 1.1-1.3:

- описать классы объектов (сущностей) и их свойства,
- расставить существующие связи между ними,
- на основании табл. 1.3. в письменной форме обосновать типы связей (1:1, 1:M и т.д.).

При графическом построении ИЛМ следует придерживаться единого масштаба для всей схемы. Все прямоугольники, обозначающие классы объектов, должны быть одного размера. Аналогично, все ромбы с именами связей также должны иметь одинаковый размер.

2.5. Построение даталогической модели.

На основании ранее выбранного варианта и таблиц 1.1-1.3, инфологической модели и нормализации БД необходимо:

- провести соответствие ключей для каждой таблицы 1.1-1.3,
- заполнить для каждой таблицы БД форму, согласно табл. 1.4.

Таблица 1.4. Структура таблицы для даталогической модели.

N п.п.	Наименование реквизита	Идентификатор	Тип	Длина	Формат изображения	Ограничения и комментарий

III. СОДЕРЖАНИЕ ОТЧЕТА

1. Название и цель работы.
2. Словесный и схематический анализ предметной области (ПО), включая схему структуры предприятия.
3. Заполненные таблицы 1.1 - 1.3. с описанием основных сущностей ПО.
4. Инфологическая модель БД, согласно варианту.
5. Обоснование типов связи в инфологической модели данных.
6. Даталогическая модель БД (табл. 1.4.).

IV. ПРИМЕР ОФОРМЛЕНИЯ

Пример. Разработать базу данных «Учеба студентов».

Решение.

Шаг первый. Анализ предметной области.

Студенты учатся на одном из факультетов, возглавляемом деканатом, в функции

которого входит контроль за учебным процессом. В учебном процессе участвуют преподаватели кафедр, административно относящиеся к одному из факультетов. Каждому факультету могут принадлежать несколько кафедр. Студенты кафедр организованные в группы.

Преподаватели кафедр характеризуются фамилией именем и отчеством, должностью, научным званием, ставкой и стажом работы, адресом проживания, возрастом.

Каждая кафедра читает определенный набор закрепленных за ней дисциплин. Каждая дисциплина характеризуется своим полным названием, указанием общего количества часов и формы контроля (зачет, экзамен).

В конце каждого семестра составляется экзаменационно-зачетные ведомости, в которых указываются дисциплины и для каких групп проводится форма контроля, фамилия преподавателя и учебный год и семестр. В каждой такой ведомости составляется список студентов и выставляется оценка.

Шаг второй. Описание основных сущностей ПО.

В результате проведенного анализа предметной области базы данных «Учеба студентов» легко перечислить основные сущности этой БД. Так как на физическом уровне сущности соответствует таблица, то просто перечислим основные таблицы БД.

В реляционную модель проектированной БД будут входить следующие таблицы (сущности): Факультет, Кафедра, Преподаватели, Группы, Студенты, Дисциплины, Ведомости.

Список сущностей.

№	Название	Назначение
1	Факультет	Описание факультета и его деканата
2	Кафедра	Описание кафедры
3	Преподаватели	Описание состава сотрудников кафедр
4	Группы	Перечень групп, закрепленных за каждой кафедрой
5	Студенты	Перечень студентов каждой группы
6	Дисциплины	Перечень дисциплин, закрепленных за каждой кафедрой
7	Ведомости	Экзаменнационно-зачетные ведомости с перечнем студентов и их оценками
8	Подчиненная ведомость	Это таблица внутри таблицы ведомости. Отражает связь один-ко-многим. Так как каждая ведомость выписывается каждой конкретной группе, а студентов в ней много.

Для каждой таблицы (сущности) приведем описание ее атрибутов. Атрибут на физическом уровне – это колонки таблицы и выражает определенное свойство объекта.

Список атрибутов таблицы «Факультеты»

Ключевое поле	Название	Назначение
ПК (первичный ключ)	Код факультета	Ключевое поле, предназначенное для однозначной идентификации каждой записи в таблице. Представляет собой первичный ключ. Это уникальное значение, соответствующее каждому факультету. Это целое число. Т.е. для идентификации каждого факультета будет применятся не названия самих факультетов, а определенный номер. Этот номер может быть случайным целым числом или счетчик по порядку.
	Название факультета	

	ФИО декана	
	Номер комнаты деканата	
	Телефон деканата	

Список атрибутов таблицы «Кафедра»

Ключевое поле	Название	Назначение
ПК (первичный ключ)	Код кафедры	Ключевое поле. Представляет собой первичный ключ. Это уникальное значение, соответствующее каждой кафедре. Однако для идентификации каждой кафедры первичного ключа недостаточно, так как каждая кафедра принадлежит определенному факультету. Для этого будем использовать внешний ключ.
ВК (внешний ключ)	Код факультета	Внешний ключ – это атрибут отношения, который является первичным ключом другого отношения. В нашем случае это атрибут таблицы факультеты. С помощью внешнего ключа будет определено к какому факультету принадлежит каждая кафедра.
	Название кафедры	
	ФИО заведующего	
	Номер комнаты кафедры	
	Телефон кафедры	

Список атрибутов таблицы «Преподаватели»

Ключевое поле	Название	Назначение
ПК (первичный ключ)	Код преподавателя	Ключевое поле. Представляет собой первичный ключ. Это уникальное значение, соответствующее каждому преподавателю. Это например, может быть его табельный номер. Однако для идентификации каждого преподавателя первичного ключа недостаточно, так как каждый сотрудник принадлежит определенной кафедры. Для этого будем использовать внешний ключ.
ВК (внешний ключ)	Код кафедры	С помощью данного внешнего ключа будет определено к какой кафедры принадлежит каждый преподаватель.
	ФИО	
	должность	Ассистент, доцент, процессор, ст. преподаватель
	научное звание	К.т.н., проф., магистр, ст.н.с., м.н.с.
	ставка	
	стаж работы,	
	адрес проживания	
	возраст	

Список атрибутов таблицы «Группы»

Ключевое	Название	Назначение

поле		
ПК (первичный ключ)	Код группы	Ключевое поле. Представляет собой первичный ключ. Это уникальное значение, соответствующее каждой группе. Однако для идентификации каждой группы первичного ключа недостаточно, так как каждая группа принадлежит определенной кафедре. Для этого будем использовать внешний ключ.
ВК (внешний ключ)	Код кафедры	С помощью данного внешнего ключа будет определено к какой кафедре принадлежит каждая группа.
	Номер группы	
	Год поступления	
	Курс обучения	

Список атрибутов таблицы «Студенты»

Ключевое поле	Название	Назначение
ПК (первичный ключ)	Код студента	Ключевое поле. Представляет собой первичный ключ. Это уникальное значение, соответствующее каждому студенту. Однако для идентификации каждого студента первичного ключа недостаточно, так как каждый студент принадлежит определенной группе. Для этого будем использовать внешний ключ.
ВК (внешний ключ)	Код группы	С помощью данного внешнего ключа будет определено к какой группе принадлежит каждый студент.
	ФИО	
	Год рождения	
	Адрес проживания	

Список атрибутов таблицы «Дисциплины»

Ключевое поле	Название	Назначение
ПК (первичный ключ)	Код дисциплины	Ключевое поле. Представляет собой первичный ключ. Это уникальное значение, соответствующее каждой дисциплине. Однако для идентификации каждой дисциплины первичного ключа недостаточно, так как каждая дисциплина принадлежит определенной кафедре. Для этого будем использовать внешний ключ.
ВК (внешний ключ)	Код кафедры	С помощью данного внешнего ключа будет определено к какой кафедре принадлежит каждая дисциплина.
	Название дисциплины	
	Расчасовка	
	Форма контроля	

Список атрибутов таблицы «Ведомости»

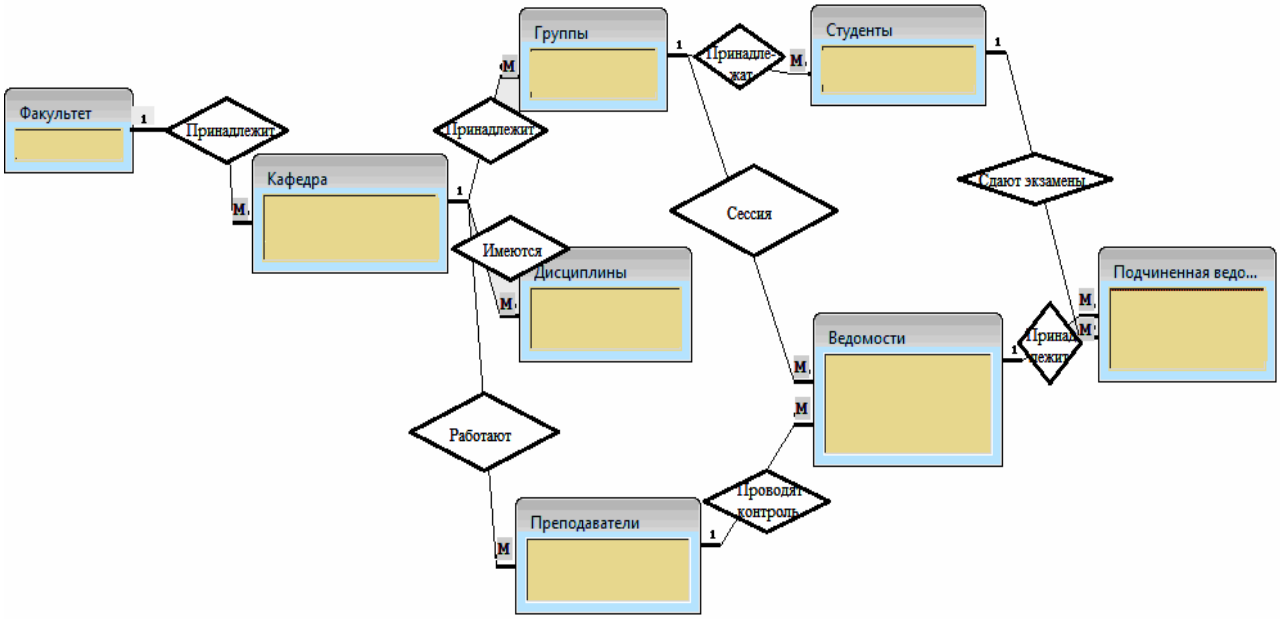
Ключевое поле	Название	Назначение
ПК (первичный ключ)	Код ведомости	Ключевое поле. Представляет собой первичный ключ. Это уникальное значение, соответствующее каждой учебной ведомости. Однако для идентификации каждой ведомости первичного ключа недостаточно, так как каждая ведомость выписывается для определенной учебной группе по определенной дисциплине и преподавателя. Для этого будем использовать внешние ключи.
ВК (внешний ключ)	Код группы	С помощью данного внешнего ключа будет определено для какой группы выписывается ведомость.
ВК (внешний ключ)	Код дисциплины	С помощью данного внешнего ключа будет определено для какой дисциплины выписывается ведомость.
ВК (внешний ключ)	Код преподавателя	С помощью данного внешнего ключа будет определено какому преподавателю выписывается ведомость.
	Учебный год	
	Семестр	

Список атрибутов таблицы «Подчиненная таблица Ведомости»

Ключевое поле	Название	Назначение
ПК (первичный ключ)	Код под_ведомости	Ключевое поле. Представляет собой первичный ключ. Это уникальное значение, соответствующее каждой подведомости. Однако для идентификации каждой подведомости первичного ключа недостаточно, так как каждая подведомость принадлежит определенной ведомости. Для этого будем использовать внешний ключ.
ВК (внешний ключ)	Код ведомости	С помощью данного внешнего ключа будет осуществлена связь с таблицей ведомости.
ВК (внешний ключ)	Код студента	С помощью данного внешнего ключа будет определен студент
	Оценка	

Шаг третий. Построение **инфологической модели**.

Инфологическую модель лучше представить графически, где будут изображены все таблицы и связи между ними. В нашем случае схема связей представлена на рисунке.



Для выявленных связей заполним таблицу

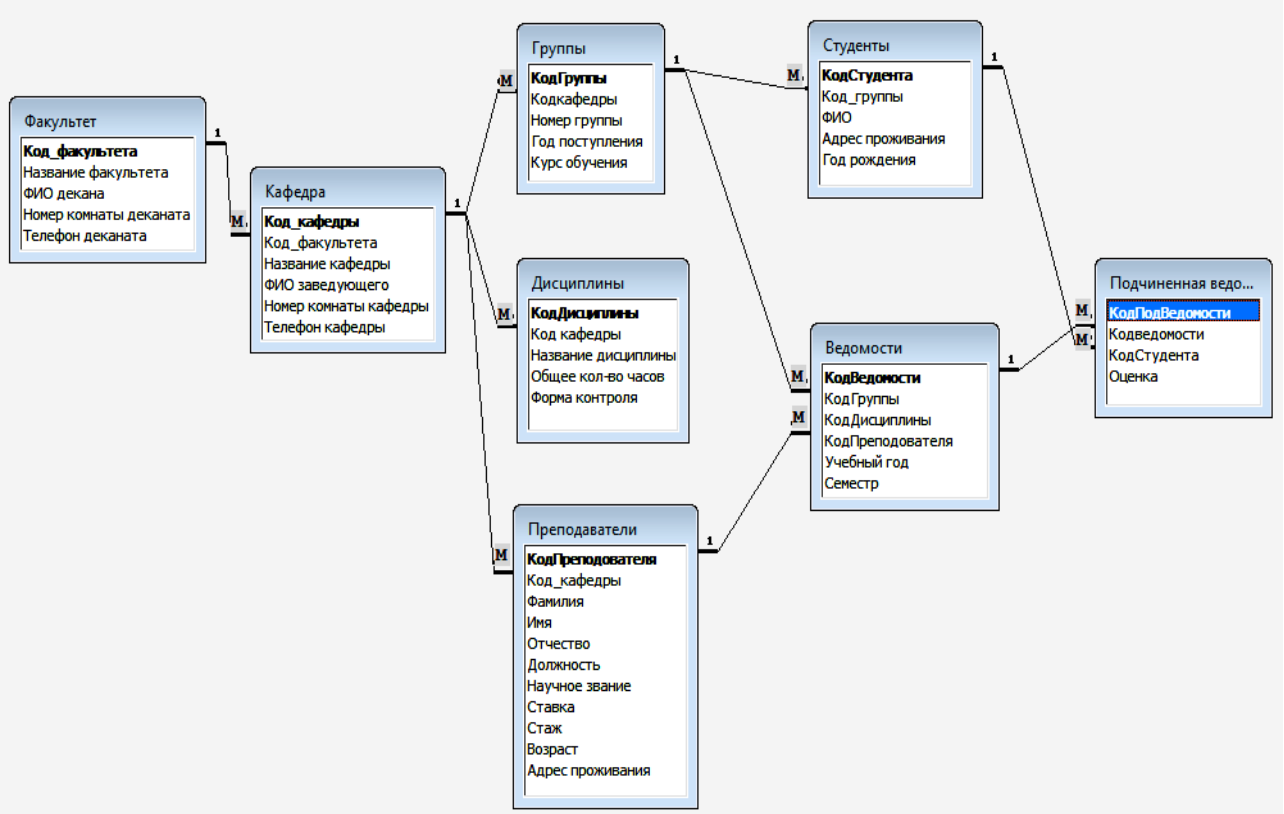
Список связей.

№	Название связи	Сущности, участвующие в связи	Назначение
1	1:M	Факультет-Кафедра	Одному факультету могут принадлежать несколько кафедр
2	1:M	Кафедра - Группа	Одной кафедре может принадлежать несколько групп
3	1:M	Кафедра - Дисциплины	Одной кафедре могут принадлежать несколько читаемых дисциплин
4	1:M	Кафедра - Преподаватели	На одной кафедре работает более одного преподавателя
5	1:M	Группа-Студенты	В каждой группе учится множество студентов
6	1:M	Группа - Ведомость	Каждой группе выписывают несколько ведомостей
7	1:M	Дисциплины - Ведомость	Ведомость выписывается из множества дисциплин
8	1:M	Преподаватели - Ведомость	Ведомость выписывается конкретному преподавателю
9	1:M	Ведомость-Подчиненная ведомость	Подчиненная ведомость принадлежит одной конкретной ведомости
10	1:M	Студенты-Подчиненная ведомость	В подчиненной ведомости перечислены все студенты группы

Шаг четвертый. Построение даталогической модели БД.

Даталогическая модель отражается графически в виде схемы базы данных, где указываются имена сущностей, их атрибуты и связи между сущностями.

В нашем случае схема связей представлена на рисунке.



Даталогическая модель БД представляется в виде набора таблиц специальной формы, в которых указываются наименование атрибута, идентификатор, тип, длина, формат, ограничения.

Таблица «Факультеты»

№	Название	Идентификатор	Тип	Не пусто	Ограничение
1	Код факультета	Kod_fakulteta	Числовой	Да	ПК (первичный ключ)
2	Название факультета	Name_fakulteta	Текстовый	Нет	
3	ФИО декана	FIO	Текстовый	нет	
4	Номер комнаты деканата	N_komnatu_dekanata	Текстовый	Нет	Например, 123/а
5	Телефон деканата	Telefon_dekanata	Текстовый	Нет	Например, 41-69-99

Список атрибутов таблицы «Кафедра»

№	Название	Идентификатор	Тип	Не пусто	Ограничение
1	Код кафедры	Kod_kafedru	Числовой	Да	ПК (первичный ключ)
2	Код факультета	Kod_fakulteta	Числовой	Да	ВК (внешний ключ)
3	Название кафедры	Name_kafedru	Текстовый		
4	ФИО	FIO	Текстовый	нет	

	заведующего				
5	Номер комнаты кафедры	N_komnatu_kafedru	Текстовый	Нет	Например, 123/а
6	Телефон кафедры	Telefon_kafedru	Текстовый	Нет	Например, 41-69-99

Список атрибутов таблицы «Преподаватели»

№	Название	Идентификатор	Тип	Не пусто	Ограничение
1	Код преподавателя	Kod_prepodavately	Числовой	Да	ПК (первичный ключ)
2	Код кафедры	Kod_kafedru	Числовой	Да	ВК (внешний ключ)
3	ФИО	FIO	Текстовый	Нет	
4	должность	Dolgnost	Текстовый	Нет	
5	научное звание	Zvanie	Текстовый	Нет	
6	ставка	Stavka	Числовой	Нет	Вещественное число Например, 0.5, 0.75, 1
7	стаж работы,	Stag	Числовой	Нет	Вещественное число
8	адрес проживания	Address	Текстовый	Нет	
9	возраст	Vozrast	Числовой	нет	

Список атрибутов таблицы «Группы»

№	Название	Идентификатор	Тип	Не пусто	Ограничение
1	Код группы	Kod_grupu	Числовой	Да	ПК (первичный ключ)
2	Код кафедры	Kod_kafedru	Числовой	Да	ВК (внешний ключ)
3	Номер группы	N_grupu	Текстовый	Нет	Например, МТ-461
4	Год поступления	God_post	Числовой	нет	
5	Курс обучения	Kurs	Числовой	Нет	Вычисляемое поле, как разность между текущей датой и годом поступления

Список атрибутов таблицы «Студенты»

№	Название	Идентификатор	Тип	Не пусто	Ограничение
1	Код студента	Kod_studenta	Числовой	Да	ПК (первичный ключ)
2	Код группы	Kod_grupu	Числовой	Да	ВК (внешний ключ)
3	ФИО	FIO	Текстовый	Нет	
4	Год рождения	God_rogdeniya	Числовой	нет	
5	Адрес проживания	Address	Текстовый	Нет	

Список атрибутов таблицы «Дисциплины»

№	Название	Идентификатор	Тип	Не пусто	Ограничение
1	Код дисциплины	Kod_disciplinu	Числовой	Да	ПК (первичный ключ)
2	Код кафедры	Kod_kafedru	Числовой	Да	ВК (внешний ключ)
3	Название дисциплины	Name_dis	Текстовый	Нет	
4	Расчасовка	Raschasovka	Числовой	нет	
5	Форма контроля	Kontrol	Текстовый	Нет	Два значения – экзамен или зачет

Список атрибутов таблицы «Ведомости»

№	Название	Идентификатор	Тип	Не пусто	Ограничение
1	Код ведомости	Kod_vedomopsti	Числовой	Да	ПК (первичный ключ)
2	Код группы	Kod_grupu	Числовой	Да	ВК (внешний ключ)
3	Код дисциплины	Kod_disciplinu	Числовой	Да	ВК (внешний ключ)
4	Код преподавателя	Kod_prepodavately	Числовой	Да	ВК (внешний ключ)
5	Учебный год	God	Числовой	Нет	
6	Семестр	Semester	Числовой	Нет	Диапазон от 1-10

Список атрибутов таблицы «Подчиненная таблица Ведомости»

№	Название	Идентификатор	Тип	Не пусто	Ограничение
1	Код под_ведомости	Kod_pod_vedomopsti	Числовой	Да	ПК (первичный ключ)
2	Код ведомости	Kod_ edomopsti	Числовой	Да	ВК (внешний ключ)
3	Код студента	Kod_studenta	Числовой	Да	ВК (внешний ключ)
4	Оценка	Osenka	Числовой	Нет	Диапазон от 0-12

Таблица 1. Варианты заданий для практической работы №1

№ варианта	Условие
Вариант №1	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – успеваемость студентов ВУЗА**. БД состоит из следующих таблиц: факультеты, кафедры, учебные группы, студенты, ведомости успеваемости.

	Таблица факультеты имеет следующие атрибуты: название факультета, ФИО декана, номер комнаты, номер корпуса, телефон. **Таблица кафедра** имеет следующие атрибуты: название кафедры, факультет, ФИО заведующего, номер комнаты, номер корпуса, телефон, кол-во преподавателей. **Таблица учебные группы** имеет следующие атрибуты: название группы, год поступления, курс обучения, кол-во студентов в группе. **Таблица студенты** имеет следующие атрибуты: студента, фамилия, имя, отчество, группа, год рождения, пол, адрес, город, телефон. **Таблица ведомости успеваемости** имеет следующие атрибуты: группа, студент, предмет, оценка.
Вариант №2	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – информационная система супермаркета**. БД состоит из следующих таблиц: отделы, сотрудники, товары, продажа товаров, должности. **Таблица отделы** имеет следующие атрибуты: название отдела, кол-во прилавков, кол-во продавцов, номер зала. **Таблица сотрудники** имеет следующие атрибуты: фамилия, имя, отчество, отдел, год рождения, год поступления на работу, стаж, должность, пол, адрес, город, телефон. **Таблица должности** имеет следующие атрибуты: название должности, сумма ставки. **Таблица товары** имеет следующие атрибуты: название товара, отдел, страна производитель, условия хранения, сроки хранения . **Таблица продажа товаров** имеет следующие атрибуты: сотрудника являющегося продавцом, товара дата, время, кол-во, цена, сумма.
Вариант №3	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – информационная система военного округа**. БД состоит из следующих таблиц: места дислокации, вид войск, части, роты, личный состав. **Таблица вид войск** имеет следующие атрибуты: название. **Таблица места дислокации** имеет следующие атрибуты: страна, город, адрес, занимаемая площадь. **Таблица части** имеет следующие атрибуты: номер части, место

	дислокации, вид войск, кол-во рот. **Таблица роты** имеет следующие атрибуты: название роты, кол-во служащих. **Таблица личный состав** имеет следующие атрибуты: фамилия, рота, должность, год рождения, год поступления на службу, выслуга лет, награды, участие в военных мероприятиях.
Вариант №4	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – информационная система библиотеки**. БД состоит из следующих таблиц: библиотеки, фонд библиотеки, тип литературы, сотрудники, пополнение фонда. **Таблица библиотеки** имеет следующие атрибуты: название, адрес, город. **Таблица фонд библиотеки** имеет следующие атрибуты: название фонда, библиотека, кол-во книг, кол-во журналов, кол-во газет, кол-во сборников, кол-во диссертаций, кол-во рефератов. **Таблица тип литературы** имеет следующие атрибуты: название типа. **Таблица сотрудники** имеет следующие атрибуты: фамилия сотрудника, библиотека, должность, год рождения, год поступления на работу, образование, зарплата. **Таблица пополнение фонда** имеет следующие атрибуты: фонд, сотрудник, дата, название источника литературы, тип литературы, издательство, дата издания, кол-во экземпляров.
Вариант №5	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – информационная система туристического агентства**. БД состоит из следующих таблиц: пансионаты, туры, клиенты, путевки, вид жилья. **Таблица пансионаты** имеет следующие атрибуты: название пансионата, адрес, город, страна, телефон, описание территории, кол-во комнат, наличие бассейна, наличие медицинских услуг, наличие спа-салона, уровень пансионата, расстояние до моря. **Таблица вид жилья** имеет следующие атрибуты: название (дом, бунгало, квартира, 1-я комната, 2-я комната и т.д.), категория жилья (люкс, полулюкс, и т.д.), пансионат, описание условий проживания, цена за номер в сутки. **Таблица туры** имеет следующие атрибуты: название тура (Европа, средняя Азия, тибет и т.д.), вид транспорта, категория жилья на ночь

	(гостиница, отель, палатка и т.д.), вид питания (одноразовое, двухразовое, трехразовое, завтраки), цена тура в сутки. **Таблица клиенты** имеет следующие атрибуты: фамилия, имя, отчество, паспортные данные, дата рождения, адрес, город, телефон. **Таблица путевки** имеет следующие атрибуты: клиент, пансионата, вид жилья, дата заезда, дата отъезда, наличие детей, наличие мед. страховки, кол-во человек, цена, сумма.
Вариант №6	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – информационная система автопредприятия города**. БД состоит из следующих таблиц: автотранспорт, водители, маршруты, обслуживающий персонал, гаражное хозяйство. **Таблица автотранспорт** имеет следующие атрибуты: название транспорта (автобусы, такси, маршрутные такси, прочий легковой транспорт, грузовой транспорт и т.д.), кол-во наработки, пробег, кол-во ремонтов, характеристика. **Таблица маршруты** имеет следующие атрибуты: название маршрута, транспорт, водитель, график работы. **Таблица водители** имеет следующие атрибуты: фамилия, имя, отчество, год рождения, год поступления на работу, стаж, должность, пол, адрес, город, телефон. **Таблица обслуживающий персонал** имеет следующие атрибуты: должность (техники, сварщики, слесари, сборщики и др.), фамилия, имя, отчество, год рождения, год поступления на работу, стаж, пол, адрес, город, телефон. **Таблица гаражное хозяйство** имеет следующие атрибуты: название гаража, транспорт на ремонте, вид ремонта, дата поступления, дата выдачи после ремонта, результат ремонта, персонал, производящего ремонт.
Вариант №7	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – информационная система поликлиники**. БД состоит из следующих таблиц: врачи, пациенты, история болезней, отделения, обслуживающий персонал. **Таблица отделения** имеет следующие атрибуты: название отделения (хирургия, терапия, нервология и т.д.), этаж, номера комнат, ФИО заведующего. **Таблица врачи** имеет следующие атрибуты: фамилия, имя, отчество,

	должность, стаж работы, научное звание, адрес, номер отделения, в котором он работает. **Таблица пациенты** имеет следующие атрибуты: фамилия, имя, отчество, адрес, город, возраст, пол. **Таблица диагнозы** имеет следующие атрибуты: название диагноза, признаки болезни, период лечения, назначения. **Таблица история болезни** имеет следующие атрибуты: пациент, врач, диагноз, лечение, дата заболевания, дата вылечивания, вид лечения (амбулаторное, стационарное).
Вариант №8	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – информационная система больницы**. БД состоит из следующих таблиц: врачи, пациенты, история болезней, операции, лист лечения. **Таблица врачи** имеет следующие атрибуты:, фамилия, имя, отчество, должность, стаж работы, научное звание, адрес. **Таблица пациенты** имеет следующие атрибуты: фамилия, имя, отчество, адрес, город, возраст, пол. **Таблица история болезни** имеет следующие атрибуты:, пациента врач, диагноз, дата заболевания, дата вылечивания, вид лечения (амбулаторное, стационарное), код операции. **Таблица лист лечения** имеет следующие атрибуты: дата лечения, история болезни, лекарства, температура, давление, состояние больного (тяжелое, среднее, и т.д.). **Таблица операции** имеет следующие атрибуты: описание операции (удаление аппендицита, пластическая операция и т.д.), врач, дата операции, пациент, результат операции.
Вариант №9	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – информационная система библиотек города**. БД состоит из следующих таблиц: библиотеки, читальные залы, литература, читатели, выдача лит-ры. **Таблица библиотеки** имеет следующие атрибуты: название, адрес, город. **Таблица читальные залы** имеет следующие атрибуты: название читального зала, библиотека, кол-во единиц лит-ры, кол-во посадочных мест, время работы, этаж, кол-во сотрудников. **Таблица читатели** имеет следующие атрибуты: фамилия, имя, отчество, категория читателя, место работы или обучения, возраст, дата регистрации в библиотеке.

	Таблица литература имеет следующие атрибуты: название, категория литературы, авторы, издательство, год издательства, кол-во страниц, читальный зал. **Таблица выдача литературы** имеет следующие атрибуты: читатель, литература, дата выдачи, срок выдачи, вид выдачи, наличие залога.
Вариант №10	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – информационная система автосалона**. БД состоит из следующих таблиц: автомобили, марка автомобиля, сотрудники, продажа автомобилей, покупатели. **Таблица марка автомобиля** имеет следующие атрибуты: название марки, страна производитель, завод производитель, адрес. **Таблица автомобиля** имеет следующие атрибуты: название автомобиля, марка, год производства, цвет, категория, цена. **Таблица покупатели** имеет следующие атрибуты: фамилия, имя, отчество, паспортные данные, адрес, город, возраст, пол. **Таблица сотрудника** имеет следующие атрибуты: фамилия, имя, отчество, стаж, зарплата. **Таблица продажа автомобилей** имеет следующие атрибуты: дата, сотрудник, автомобиль, покупатель.
Вариант №11	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – успеваемость студентов кафедры**. БД состоит из следующих таблиц: кафедры, дисциплины, преподаватели, студенты, ведомости успеваемости. **Таблица кафедра** имеет следующие атрибуты: название кафедры, факультет, ФИО заведующего, номер комнаты, номер корпуса, телефон, кол-во преподавателей. **Таблица преподаватели** имеет следующие атрибуты: фамилия, имя, отчество, кафедра, год рождения, год поступления на работу, стаж, должность, пол, адрес, город, телефон. **Таблица студенты** имеет следующие атрибуты: фамилия, имя, отчество, кафедра, год рождения, пол, адрес, город, телефон. **Таблица дисциплины** имеет следующие атрибуты: название дисциплины, кафедра, читаемой эту дисциплину, кол-во часов, вид итогового контроля.

	Таблица ведомости успеваемости имеет следующие атрибуты: преподаватель, дисциплина, студент, оценка.
Вариант №12	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – торговая организация**. БД состоит из следующих таблиц: торговая организация, торговая точка, продавцы, поставщики, заказы поставщикам. **Таблица торговая организация** имеет следующие атрибуты: название торговой организации, адрес, ФИО директора, налоговый номер. **Таблица торговая точка** имеет следующие атрибуты: название торговой точки, тип (универмаги, магазины, киоски, лотки и т.д.), торговая организация, адрес, ФИО заведующего. **Таблица продавцы** имеет следующие атрибуты: фамилия, имя, отчество, торговая точка, должность, год рождения, пол, адрес проживания, город. **Таблица поставщики** имеет следующие атрибуты: название поставщика, тип деятельности, страна, город, адрес. **Таблица заказы поставщикам** имеет следующие атрибуты: дата заказа, торговая точка, поставщик, название товара, кол-во, цена.
Вариант №13	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – проектная организация**. БД состоит из следующих таблиц: отделы, сотрудники, организации, договора, проектные работы. **Таблица отделы** имеет следующие атрибуты: название отдела, этаж, телефон, начальник отдела. **Таблица сотрудники** имеет следующие атрибуты: ФИО, должность (конструкторы, инженеры, техники, лаборанты, прочий обслуживающий персонал), номер отдела, в котором работает, пол, адрес, дата рождения. **Таблица организации** имеет следующие атрибуты: название организации, тип деятельности, страна, город, адрес, ФИО директора. **Таблица договора** имеет следующие атрибуты: номер договора, дата заключения договора, организация, стоимость договора. **Таблица проектные работы** имеет следующие атрибуты: дата начала проектной работы, дата завершения проектной работы, номер договора, отдел, осуществляющий разработку.
Вариант	На основании выбранного варианта выполнить следующее:

№14	1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – информационная система военно-морского флота**. БД состоит из следующих таблиц: базы, части, личный состав, корабли, учения. **Базы военно-морского флота** имеет следующие атрибуты: название базы, георгафическое расположение, кол-во частей. **Таблица части** имеет следующие атрибуты: номер части, база флота, место базирования, вид войск (морская авиация, морская пехота и т.д.). **Таблица личный состав** имеет следующие атрибуты: фамилия, часть, должность, год рождения, год поступления на службу, выслуга лет, награды, **Таблица корабли** имеет следующие атрибуты: идентификационный номер корабля, название корабля, тип корабля, дата создания, наработка, кол-во посадочных мест, устройство двигателя (парусное, гребное, пароход, теплоход, турбоход, и т.д.), тип привода (самоходное, несамоходное), размещение корпуса (подводная лодка, ныряющее, полупогружное, и т.д.) **Таблица учения**: часть, корабль, дата учения, место проведения, оценка.
Вариант №15	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – туристическая фирма**. БД состоит из следующих таблиц: туристы, туристичексая группа, состав групп, гостиницы, ведомости продажей. **Таблица туристы** имеет следующие атрибуты: ФИО, паспортные данные, пол, возраст, дети. **Таблица туры** имеет следующие атрибуты: название, страна, города, тип передвижения, тип питания, цена тура, тип проживания. **Таблица туристическая группа** имеет следующие атрибуты: название, дата отправления, дата прибытия, тур, кол-во туристов. **Таблица состав групп** имеет следующие атрибуты: дата продажи, турист, группа, цена билета. **Таблица гостиницы** имеет следующие атрибуты: название гостиницы, страна, город, адрес, кол-во мест, тип гостиницы. **Таблица ведомость продажей** имеет следующие атрибуты: дата, туристическая группа, гостиница, общая стоимость.
Вариант №16	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации;

	2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – цирк**. БД состоит из следующих таблиц: работники цирка, представления, расписание гастролей, труппа цирка, программа цирка. **Таблица работники цирка** имеет следующие атрибуты: фамилия, имя, отчество, год рождения, год поступления на работу, стаж, должность (акробат, клоун, гимнаст, музыкант, постановщик, служащий и т.д.), пол, адрес, город, телефон. **Таблица представления** имеет следующие атрибуты: название, режиссер-постановщик, художник-постановщик, дирижер-постановщик, автор, жанр, тип. **Таблица расписание гастролей** имеет следующие атрибуты: представление, дата начала, дата окончания, места проведения гастроли. **Таблица труппа представления цирка** имеет следующие атрибуты: представление, актер цирка, название роли. **Таблица программа цирка** имеет следующие атрибуты: представление, дата премьеры, период проведения, дни и время, цена билета.
Вариант №17	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – аптека**. БД состоит из следующих таблиц: лекартства, покупатели, продавцы, рецепты, продажа лекарств. **Таблица лекарства** имеет следующие атрибуты: название, тип (готовое, изготовляемое), вид (таблетки, мази, настойки), цена. **Таблица покупатели** имеет следующие атрибуты: фамилия, имя, отчество, адрес, город, телефон. **Таблица продавцы** имеет следующие атрибуты: фамилия, имя, отчество, дата поступления, дата рождения, образование. **Таблица рецепты** имеет следующие атрибуты: номер рецепта, дата выдачи, ФИО больного (покупатель), ФИО врача, диагноз пациента. **Таблица продажа лекарств** имеет следующие атрибуты: дата, лекарство, кол-во, рецепт, продавец.
Вариант №18	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние

	ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – городская телефонная сеть**. БД состоит из следующих таблиц: АТС, абонент, ведомость звонков, прайс АТС, ведомость абонентской платы. **Таблица АТС** имеет следующие атрибуты: название АТС, вид (городские, ведомственные и учрежденческие), адрес, город, кол-во абонентов. **Таблица абоненты** имеет следующие атрибуты: фамилия, имя, отчество, вид телефона (основной, параллельный или спаренный), номер телефона, межгород (открыт/закрыт), льгота (да/нет), адрес: индекс, район, улица, дом, квартира. **Таблица ведомость звонков** имеет следующие атрибуты: абонент, дата звонка, время начала, время окончания, межгород (да/нет). **Таблица прайс АТС** имеет следующие атрибуты: АТС, цена на городские, цена на межгород. **Таблица ведомость абонентской платы** имеет следующие атрибуты: абонент, месяц, год, кол-во минут на городские, кол-во минут на межгород, стоимость, сумма льготы, общая стоимость.
Вариант №19	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – аэропорт**. БД состоит из следующих таблиц: работники аэропорта, расписание вылетов, самолеты, бригады самолетов, ведомость продаж билетов. **Таблица работники аэропорта** имеет следующие атрибуты: фамилия, имя, отчество, год рождения, год поступления на работу, стаж, должность (пилотов, диспетчеров, техников, кассиров, работников службы безопасности, справочной службы и других,), пол, адрес, город, телефон. **Таблица расписание вылетов** имеет следующие атрибуты: самолет, дата вылета, время вылета, место выбытия, место прибытия, маршрут (начальный и конечный пункты назначения, пункт пересадки), стоимость билета. **Таблица самолеты** имеет следующие атрибуты: номер, год выпуска, кол-во посадочных место, грузоподъемность. **Таблица бригады самолетов** имеет следующие атрибуты: номер бригады, самолет, работник аэропорта (пилоты, техники и обслуживающий персонал)ю **Таблица ведомость продажи билетов** имеет следующие атрибуты: дата и время продажи, ФИО пассажира, паспортные данные, номер рейса, кол-во билетов, наличие льгот (пенсионеры, дети-сироты и т.д.), багаж (да/нет), стоимость.

Вариант №20	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – театр**. БД состоит из следующих таблиц: работники театра, спектакли, расписание гастролец, труппа спектакля, репертуар театра. **Таблица работники театра** имеет следующие атрибуты: фамилия, имя, отчество, год рождения, год поступления на работу, стаж, должность (актеров, музыкантов, постановщиков и служащих), пол, адрес, город, телефон. **Таблица спектакли** имеет следующие атрибуты: название, режисер-постановщик, художник-постановщик, дирижер-постановщик, автор, жанр (музыкальная комедия, трагедия, оперетта и пр), тип (детские, молодежные и пр.). **Таблица расписание гастролей** имеет следующие атрибуты: название, дата начала, дата окончания, места проведения гастроли, спектакль. **Таблица труппа спектакля** имеет следующие атрибуты: спектакль, актер, название роли. **Таблица репертуар театра** имеет следующие атрибуты: спектакль, дата премьеры, период проведения, дни и время, цена билета.
Вариант №21	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – железнодорожный вокзал**. БД состоит из следующих таблиц: работники ж.д.вокзала, расписание движения поездов, поезда, бригады поездов, ведомость продаж билетов. **Таблица работники ж.д.вокзала** имеет следующие атрибуты: фамилия, имя, отчество, год рождения, год поступления на работу, стаж, должность (машинист, диспетчеров, проводник, ремонтников подвижного состава, путей, кассиров, работников службы подготовки составов, справочной службы и других,), пол, адрес, город, телефон. **Таблица расписание движения поездов** имеет следующие атрибуты: поезд, дата отправления, время отправления, место отправления, дата прибытия, время прибытия, место прибытия, маршрут ((начальный и конечный пункты назначения, основные узловые станции), стоимость билета. **Таблица поезда** имеет следующие атрибуты: номер, год выпуска, кол-во вагонов, тип поезда (общий, скоростной, высокоскоростной).

	Таблица бригады поездов имеет следующие атрибуты: номер бригады, поезд, работник ж.д.вокзала (машинисты, техники, проводники и обслуживающий персонал). **Таблица ведомость продажи билетов** имеет следующие атрибуты: дата и время продажи, ФИО пассажира, паспортные данные, номер рейса, кол-во билетов, наличие льгот (пенсионеры, дети-сироты и т.д.), стоимость.
Вариант №22	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – информационная система ВУЗА**. БД состоит из следующих таблиц: факультеты, кафедры, преподаватели, дисциплины, учебная нагрузка. **Таблица факультеты** имеет следующие атрибуты: название факультета, ФИО декана, номер комнаты, номер корпуса, телефон. **Таблица кафедра** имеет следующие атрибуты: название кафедры, ФИО заведующего, номер комнаты, номер корпуса, телефон, кол-во преподавателей. **Таблица дисциплины** имеет следующие атрибуты: название дисциплины, кол-во часов, цикл дисциплин. **Таблица преподаватели** имеет следующие атрибуты: фамилия, имя, отчество, кафедра, год рождения, год поступления на работу, стаж, должность, пол, город. **Таблица учебная нагрузка** имеет следующие атрибуты: преподаватель, дисциплина, учебный год, семестр, группы, кол-во студентов, вид итогового контроля.
Вариант №23	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – информационная система военного округа**. БД состоит из следующих таблиц: места дислокации, вид войск, части, роты, личный состав. **Таблица вид войск** имеет следующие атрибуты: название вида войск. **Таблица места дислокации** имеет следующие атрибуты: страна, город, адрес, занимаемая площадь, кол-во сооружений. **Таблица части** имеет следующие атрибуты: номер части, место дислокации, вид войск, кол-во рот, кол-во техники, кол-во вооружений. **Таблица техника** имеет следующие атрибуты: название техники, часть, характеристики. **Таблица вооружения** имеет следующие атрибуты: название вооружения,

	часть, характеристики.
Вариант №24	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – информационная система супермаркета**. БД состоит из следующих таблиц: отделы, клиенты, товары, продажа товаров, поставщики. **Таблица отделы** имеет следующие атрибуты: название отдела, кол-во прилавков, кол-во продавцов, номер зала. **Таблица клиенты** имеет следующие атрибуты: название клиента, адрес, вид оплаты. **Таблица поставщики** имеет следующие атрибуты: название поставщика, адрес, страна, вид транспорта, вид оплаты. **Таблица товары** имеет следующие атрибуты: название товара, отдел, поставщик, условия хранения, сроки хранения . **Таблица продажа товаров** имеет следующие атрибуты: клиент, товар, дата, время, кол-во, цена, сумма.
Вариант №25	На основании выбранного варианта выполнить следующее: 1. Выполнить анализ предметной области исследуемой организации; 2. Описать основные сущности предметной области; 3. Расставить существующие связи между сущностями: самостоятельно добавить в каждую сущность первичные ключи и установить внешние ключи между сущностями; 4. Построить инфологическую модель базы данных организации; 5. Построить даталогическую модель базы данных организации. **БД – информационная система больницы**. БД состоит из следующих таблиц: врачи, пациенты, история болезней, отделения, лист лечения. **Таблица отделения** имеет следующие атрибуты: название отделения (хирургия, терапия, нервология и т.д.), этаж, номера комнат, ФИО заведующего. **Таблица врачи** имеет следующие атрибуты: фамилия, имя, отчество, должность, стаж работы, научное звание, адрес. **Таблица пациенты** имеет следующие атрибуты: фамилия, имя, отчество, адрес, город, возраст, пол. **Таблица история болезни** имеет следующие атрибуты: пациент, врач, диагноз, дата заболевания, дата вылечивания, вид лечения (амбулаторное, стационарное). **Таблица лист лечения** имеет следующие атрибуты: дата лечения, история болезни, лекарства, температура, давление, состояние больного (тяжелое, среднее, и т.д.).

ПРАКТИЧЕСКАЯ РАБОТА №2. УСТАНОВКА СОЕДИНЕНИЯ С СЕРВЕРОМ MICROSOFT SQL SERVER И ПРИНЦИПЫ СОЗДАНИЯ БАЗ ДАННЫХ

2.1. Цель практической работы

Познакомиться с основными принципами создания базы данных в MS SQL Server. Изучить операции, проводимые с базами данных в целом. Получить навыки использования программы "SQL Server Management Studio" для создания, удаления, регистрации, подключения, извлечения метаданных, резервного копирования и восстановления базы данных. Изучить SQL-операторы для создания, подключения и удаления базы данных. Познакомиться с основными принципами управления учетными записями и ролями.

2.2. Исходные данные

Студент получает индивидуальный вариант исходных данных с кратким описанием предметной области, который используется при выполнении всех описанных в данном пособии практических работ. При этом каждая очередная Практическая работа является продолжением выполненной ранее и поэтому они должны обязательно выполняться последовательно.

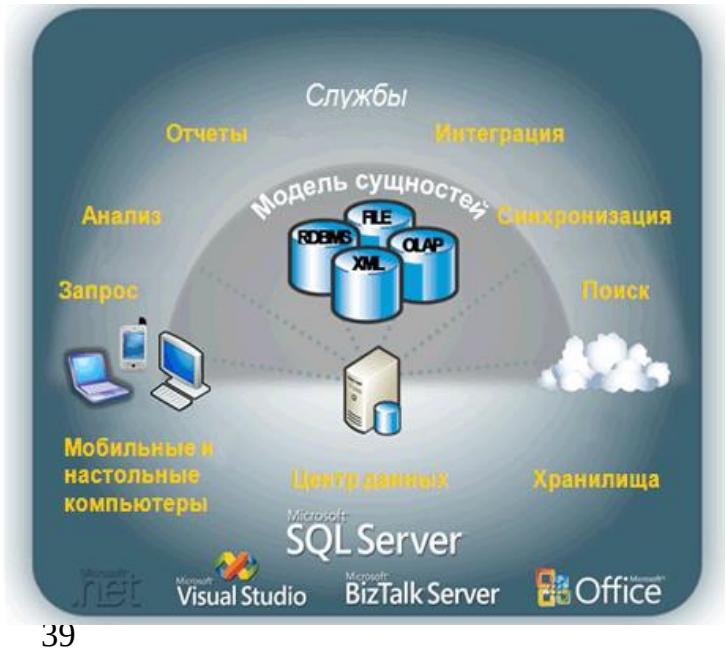
2.3. Используемые программы

1. Работающий на компьютере **сервер "MS SQL Server 2008 R2**".
2. Установленная платформа **.NET Framework 2.0, 3.0, 3.5 или 4.0**.
3. Операционная система **Microsoft Windows 2000/XP/2003/Vista/Windows 7/Windows 8.**
4. Приложение "**SQL Server Management Studio 2008 rus**", установленное на локальном компьютере.

2.4. Теоретические сведения

На сегодняшний день известно более двух десятков серверных СУБД, из которых наиболее популярными являются Oracle, Microsoft SQL Server, Informix, DB2, Sybase, InterBase, MySQL.

Для выполнения практических работ будет использоваться сервер " Microsoft SQL Server 2008", установленный на сервере кафедры компьютерных систем и сетей (компьютер pi_srv).



Microsoft® SQL Server™ — это система анализа и управления реляционными базами данных в решениях электронной коммерции, производственных отраслей и хранилищ данных.

Microsoft SQL Server — система управления реляционными базами данных (СУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — **Transact-SQL**, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта

ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

В SQL Server 2008 имеется большой набор интегрированных служб, расширяющих возможности использования данных: вы можете составлять запросы, выполнять поиск, проводить синхронизацию, делать отчеты, анализировать данные. Все данные хранятся на основных серверах, входящих в состав центра обработки данных. К ним осуществляется доступ с настольных компьютеров и мобильных устройств. Таким образом, вы полностью контролируете данные независимо от того, где вы их сохранили.

Система SQL Server 2008 позволяет обращаться к данным из любого приложения, разработанного с применением технологий Microsoft .NET и Visual Studio, а также в пределах сервисно-ориентированной архитектуры и бизнес-процессов — через Microsoft BizTalk Server. Сотрудники, отвечающие за сбор и анализ информации, могут работать с данными, не покидая привычных приложений, которыми они пользуются каждый день, например приложений выпуска 2007 системы Microsoft Office.

В Microsoft SQL базы данных хранятся в виде обычных файлов на диске. Как минимум на одну БД приходится таких **файлов 2**: ***.mdf и *.ldf.** В первом хранятся сами данные, таблицы, индексы и пр., а во втором находится т.н. transaction log, в котором находится информация необходимая для восстановления БД.

Файл с базой данных представляет собой набор страниц одинакового размера. Размер страницы задается при создании базы данных и может быть изменен только при ее восстановлении из резервной копии. Чтение и запись данных в базе данных осуществляется постранично.

Все операции с базой данных должны производиться только посредством команд к SQL-серверу. Для клиентских приложений эти файлы абсолютно бесполезны и при правильной организации доступа пользователей к файлам в сети, вообще не должны быть доступны.

Сервер СУБД не имеет интерфейса пользователя и для выполнения операций с базой данных ему необходимо посылать команды либо с помощью командной строки или с помощью какой-либо прикладной программы.

Для выполнения операций с базой данных при проведении практических работ предлагается использовать программу " **SQL Server Management Studio 2008 Rus**" (рис. 1), представляющую собой наиболее распространенное и удобное средство администрирования баз данных под управлением MS SQL Server (Среда Management Studio Express доступна для свободной загрузки из центра загрузки Майкрософт - http://download.microsoft.com/download/5/C/0/5C0C5CE4-10EB-4623-A63E-8D850D55D8EF/SQLEXPR_x86_RUS.exe).

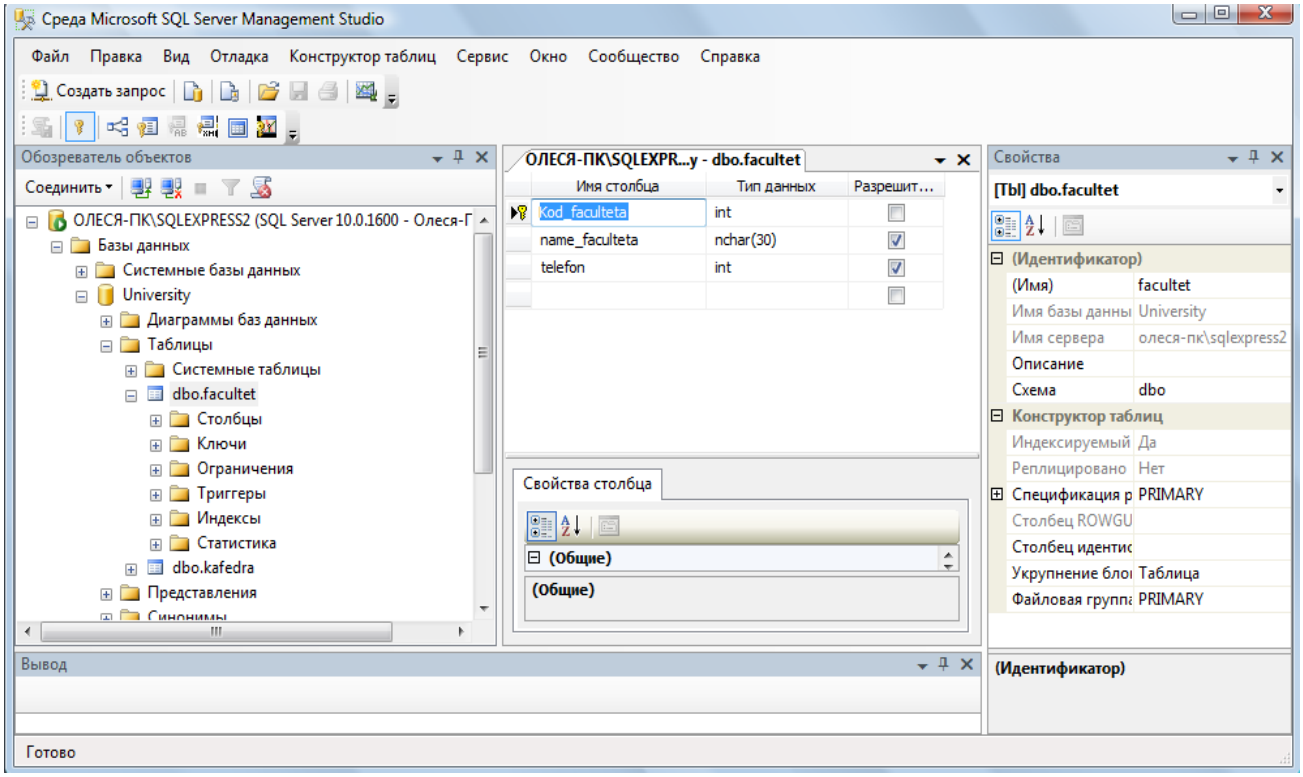


Рис. 1. Программа SQL Server Management Studio

Среда SQL Server Management Studio — это интегрированная среда для доступа, настройки, управления, администрирования и разработки всех компонентов SQL Server. Среда SQL Server Management Studio объединяет большое число графических средств с набором полнофункциональных редакторов сценариев для доступа к SQL Server разработчиков и администраторов с любым опытом работы.

Среда SQL Server Management Studio обеспечивает следующие основные возможности:

- поддерживает большинство административных задач для SQL Server;
- единая интегрированная среда для управления SQL Server Database Engine и разработки;
- новые управляющие диалоговые окна для управления объектами в компоненте SQL Server Database Engine, службах Analysis Services, Reporting Services, Notification Services и выпуске SQL Server Compact 3.5 с пакетом обновления 1 (SP1), позволяющие выполнять действия немедленно, направлять их в редактор кода или включать эти действия в сценарий для последующего выполнения;
- экспорт и импорт регистрации сервера среды SQL Server Management Studio из одной среды Management Studio в другую;
- сохранение и печать XML-файлов плана выполнения и взаимоблокировок, созданных приложением SQL Server Profiler, просмотр их в любое время и отправка для анализа администратору;
- новые окна сообщений об ошибках и информационных сообщений, предоставляющие гораздо больше сведений и позволяющие отправлять в Майкрософт комментарии о сообщениях, копировать сообщения в буфер обмена и отправлять их по электронной почте в службу поддержки;
- встроенный веб-обозреватель для быстрого обращения к библиотеке MSDN или получения интерактивной справки;
- встроенная справка от сообществ в Интернете и т.д.

Большинство действий с базой данной MS SQL Server в среде Среда SQL Server Management Studio может быть осуществлено двумя способами: **либо выполнением**

операторов языка SQL в окнах "**Script Execute**" (подключение к базе данных не обязательно) и "**SQL Editor**" (требуется подключение к базе данных), либо с использованием меню и диалоговых окон. В последнем случае операторы SQL, которые требуются для выполнения данного действия, будут сгенерированы и выполнены средой SQL Server Management Studio автоматически.

2.5. Задание

Практическую работу следует выполнять в следующем порядке:

1. Создать на сервере pi_srv (или на локальном компьютере, если нет сервера) рабочую папку для хранения файлов, получаемых при выполнении практической работы. Эта папка должна располагаться в папке **Базы данных\Группа\Студент** и соответствовать номеру выполняемой практической работы.
2. На основании индивидуального задания выбрать имя файла создаваемой базы данных. Для имени лучше всего выбрать одно или несколько английских слов, соответствующих наименованию предметной области. Использование для имени русских слов, записанных латинскими буквами, не допускается.
3. Открыть приложение " Среда SQL Server Management Studio ". Для этого можно либо воспользоваться меню Пуск (**Пуск/Программы/ Microsoft SQL Server 2008 / Среда SQL Server Management Studio**).
4. Создать соединение с локальным или удаленным сервером.
5. Создать базу данных для своей предметной области с помощью диалога, выбрав сервер "pi_srv" или локальный сервер "**Имя_компьютера\SQLEXPRESS**"
6. Создать базу данных и указать в качестве имени файла **"\Базы данных\Группа\ФИО_студента\Название_БД".**
7. Извлечь метаданные для автоматической генерации команды создания базы данных.
8. Удалить базу данных, выполнив команду "**Database/Drop Database**" (База данных/Удалить базу данных).
9. Создать базу данных вторым способом, выполнив в окне "**Script Executive**" операторы, полученные при извлечении метаданных перед предыдущим удалением.
10. Создать резервную копию базы данных.
11. Удалить базу данных.
12. Восстановить базу данных из резервной копии.
13. Сохранить файл сценария на сервере в папке "Студент", дав ему имя «лаб.№1» и стандартное расширение "***.sql**".

2.6. Ход работы

2.6.1. Создание соединения с сервером

Выполните следующие инструкции:

Работа с приложением **SQL Server Management Studio** начинается с создания соединения с установленным сервером. Убедитесь вначале, что сервер Microsoft SQL Server (2008) на локальной машине или на сервере компьютерного класса установлен и работает.

Откройте приложение " SQL Server Management Studio ". Для этого можно либо воспользоваться меню Пуск (**Пуск/Программы/ Microsoft SQL Server 2008 / Среда SQL Server Management Studio**).

В диалогом окне **Соединение с сервером** подтвердите заданные по умолчанию параметры и нажмите кнопку **Соединить**, см. рис.2.

Для соединения необходимо, чтобы поле **Имя сервера** содержало имя компьютера,

на котором установлен SQL Server.

Если компонент **Database Engine** является именованным экземпляром, то поле **Имя сервера** должно также содержать имя экземпляра в формате **<имя_компьютера>\<имя_экземпляра>**.

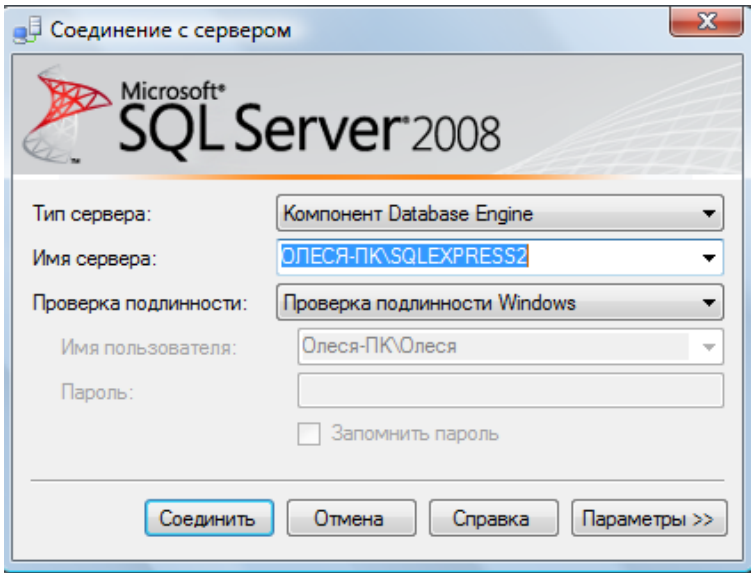


Рис. 2. Создание соединения с сервером

В параметрах указываем:

Тип сервера – Компонент **Database Engine.**

Имя сервера. Подключение может быть ***локальным*** или ***удаленным***. Представляет собой название компьютера в сети, на котором установлен сервер СУБД. Если сервер установлен на том же компьютере, где сейчас работает пользователь, то в качестве имени используется имя компьютера и идентификатор сервера;

проверка подлинности – Windows (по умолчанию),

имя пользователя –имя пользователя по умолчанию, зарегистрированного на сервере MS SQL Server (задается при установке сервера),

пароль – пусто или пароль для пользователя, заданного для сервера MS SQL Server;

Нажмите кнопку **Соединить**. Если соединение будет совершенно успешно, то на экране появятся данные сервера.

Среда Management Studio представляет данные в виде окон, выделенных для отдельных типов данных. Сведения о базе данных отображаются в обозревателе объектов и окнах документов.

Обозреватель объектов является представлением в виде дерева, в котором отображаются все объекты базы данных на сервере. Он может содержать базы данных компонента SQL Server Database Engine, служб Analysis Services, служб Reporting Services, служб Integration Services и SQL Server Compact 3.5 с пакетом обновления 1 (SP1).

Обозреватель объектов включает сведения по всем серверам, к которым он подключен. При открытии среды Management Studio пользователю предлагается применить при подключении обозревателя объектов параметры, которые использовались в прошлый раз. Чтобы подключиться к любому из серверов, следует дважды щелкнуть его в компоненте «**Зарегистрированные серверы**», однако регистрировать его не обязательно, см. рис.1.

Окно документов представляет собой наиболее крупную часть среды Management Studio. В окнах документов могут размещаться редакторы запросов и окна обзора. По

умолчанию отображается страница «**Сводка**», подключенная к экземпляру компонента Database Engine на текущем компьютере.

2.6.2. Общие сведения о базах данных MS SQL Server

Кроме четырех системных баз, SQL Server может обрабатывать до **32 734** баз данных, определяемых пользователем.

База данных представляет собой:

• набор взаимосвязанных таблиц;
• связанный набор страниц, выделенных для хранения данных MS SQL Server;
• совокупность данных при архивации;
• два и более файла;
• важную совокупность данных для целей защиты и управления.

Файлы базы данных

База данных состоит из двух и более файлов, каждый из которых может использоваться лишь одной базой.

У файлов существуют два имени: **логическое и физическое. Логическое имя** подчиняется стандартным правилам выбора имен объектов SQL Server. **Физическое имя** представляет собой полное имя любого локального или сетевого файла. Максимальное число файлов в базе данных — 32 768. **Файлы делятся на три типа:**

• **Первичные файлы**. Используются для хранения данных и информации, определяющих начальные действия с базой. База данных содержит лишь один первичный файл. Стандартное расширение — **.mdf.**

• **Вторичные файлы**. Одна или несколько вспомогательных областей для хранения данных. Могут использоваться для распределения операций чтения/записи по нескольким дискам. Стандартное расширение — **.ndf.**

• **Файлы журналов**. Содержат журналы транзакций базы данных. База данных содержит по крайней мере один файл журнала. Стандартное расширение — **.ldf**. Перед непосредственной записью транзакций в файл данных все вносимые изменения записываются в журнал.

Группы файлов

Группы файлов предназначены для объединения нескольких файлов. Каждый файл может входить не более чем в одну группу. Файлы журналов не могут принадлежать никаким группам. Группы файлов используются для распределения операций чтения/записи по нескольким дискам. Если группа содержит более одного файла, операции записи распределяются между файлами группы. Базы данных могут содержать до 32 768 групп файлов.

У каждой базы данных имеется **первичная группа файлов**. Она содержит первичный файл данных и все файлы, которые не были явно назначены в другую группу файлов. Имя первичной группы файлов — **PRIMARY.**

2.6.3. Создание и регистрация базы данных

Для создания базы данных можно использовать один из **двух способов**:

Первый способ создания БД. Выполнить команду "**База данных/Создать базу данных**..." в программе SQL Server Management Studio, ввести параметры создаваемой базы данных в диалоговом окне "**Создание базы данных**" (рис. 3) и нажать кнопку [OK].

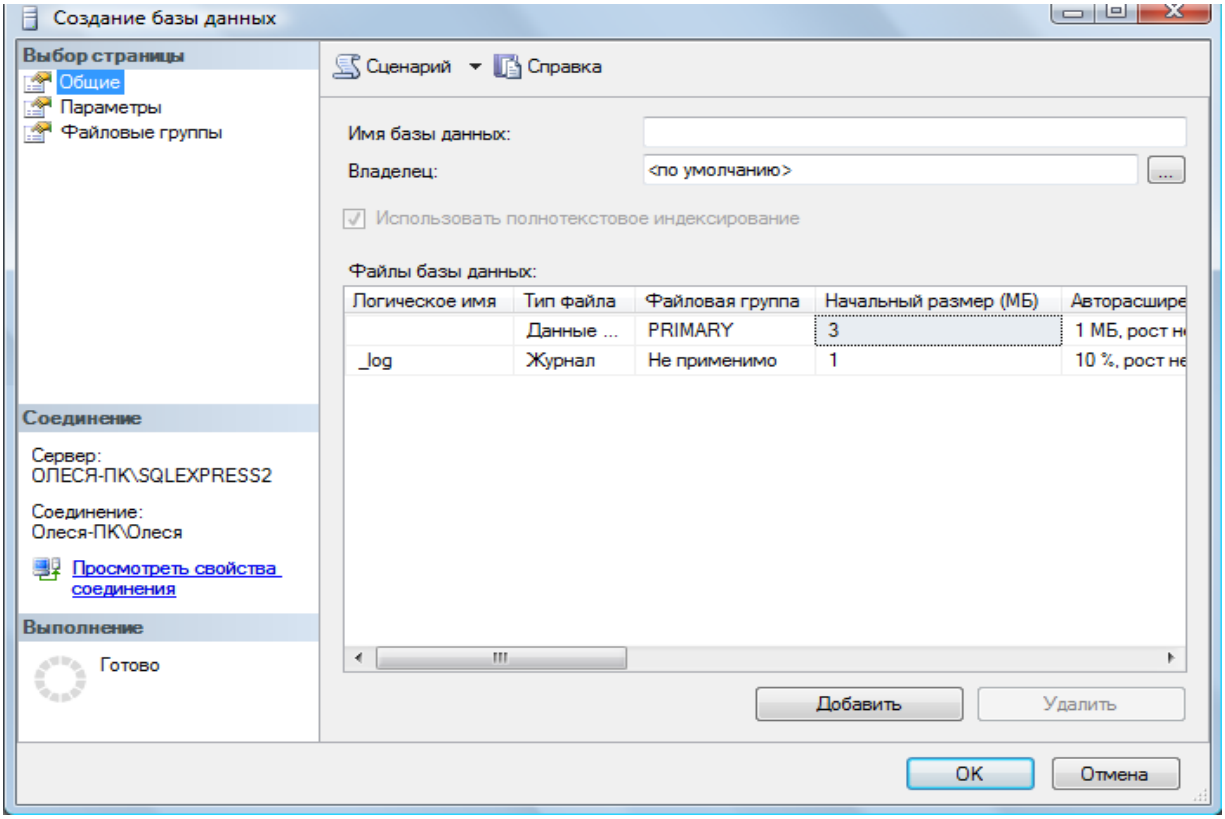


Рис. 3. Диалоговое окно создания базы данных

В поле **Имя базы данных** введите имя нашей будущей базы данных, например – **University**.

Поле **Владелец** - задан по умолчанию, в зависимости от настройки сервера.

Папка с базой данных будет создана по умолчанию на диске **C:\Program Files\Microsoft SQL Server\MSSQL10.SQLEXPRESS2\MSSQL\DATA \.**

Прежде чем нажать кнопку **Добавить**, просмотрите **Параметры** и **Файловые группы** для создаваемой базы данных.

После нажатия на кнопку [**OK**] программа " SQL Server Management Studio " создаст базу данных, имя которой вы увидите в обозревателе объектов, а также сгенерирует необходимый SQL-код для создания базы данных с теми свойствами, которые указаны в этом диалоговом окне и передаст его серверу СУБД для выполнения.

Пример этих операторов приведен на рис. 4. (нажмите на имени базы данных **University** правой клавишей и из контекстного меню выберите **Создать скрипт как.. CREATE**). Если параметры введены правильно, база данных будет создана.

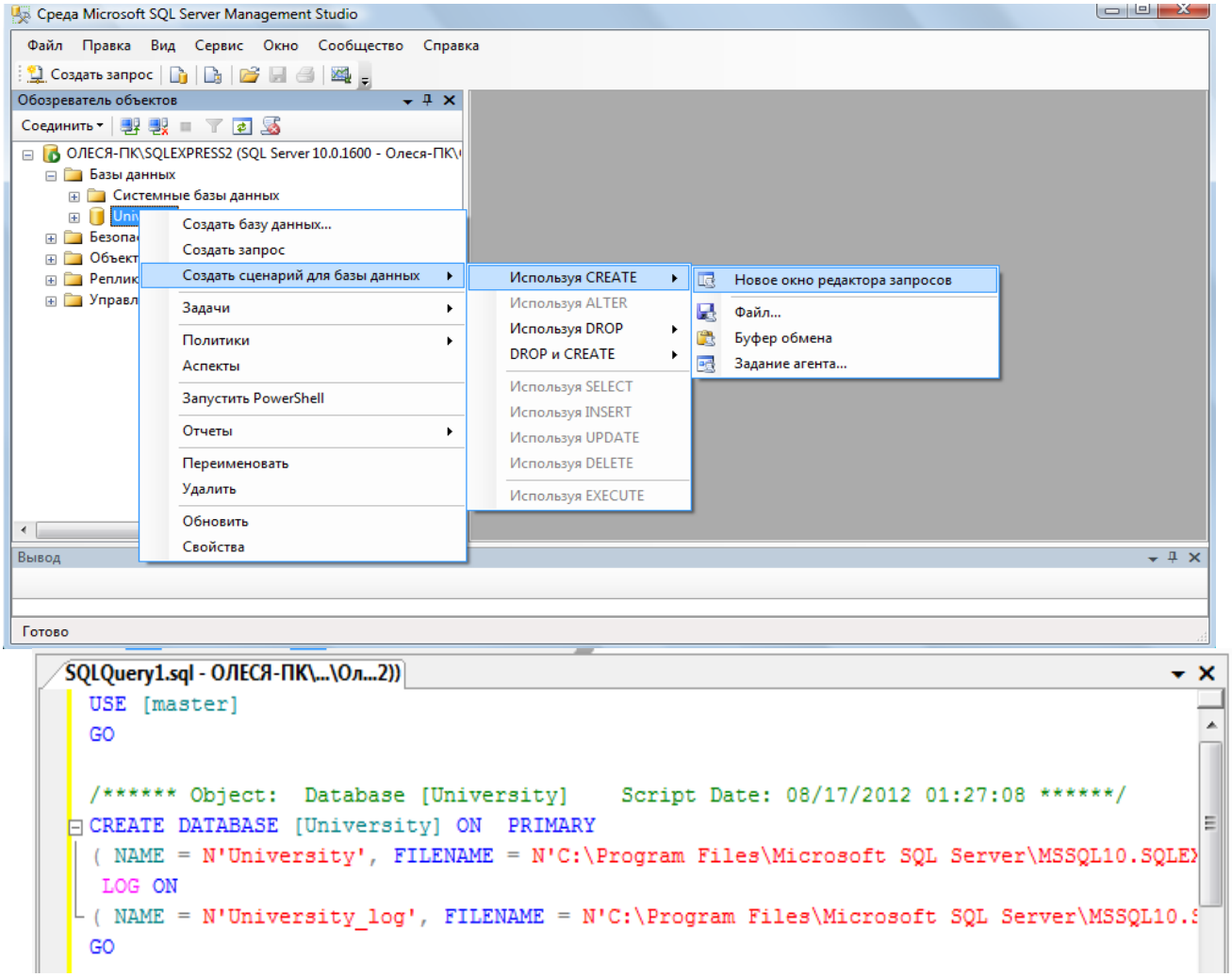


Рис. 4. Сгенерированный sql-код созданной базы данных

Содержащиеся в сценарии операторы отделяются друг от друга символом **";"**.

Сценарий может содержать поясняющие комментарии двух видов: многострочный комментарий (начинается символами "/*" и заканчивается символами "*/") и однострочный комментарий, который начинается символами "--" и продолжается до конца строки.

При создании базы данных возможны следующие типичные ошибки:

1. На целевом компьютере не запущен или не установлен сервер СУБД – т.е. выполнять команду создания базы данных просто некому.
2. На целевом компьютере нет каталога, в котором предполагается создать базу данных.
3. Файл, в котором должна будет находиться база данных на сервере, уже существует.

После создания базы данных вся введенная о базе данных информация запоминается программой SQL Server Management Studio и в окно редактора в дерево на вкладке "Проводник" добавляется узел с зарегистрированной базой данных (рис. 5).

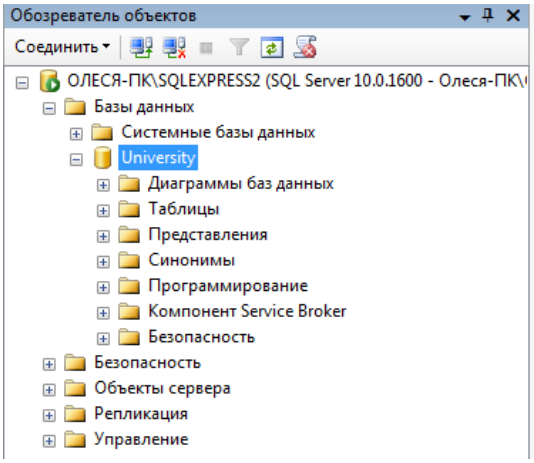


Рис. 5. Перечень зарегистрированных баз данных в SQL Server Management Studio

Второй способ создания БД. Выполнить в программе SQL Server Management Studio команду "**Создать запрос**" Создать запрос на панели инструментов, затем ввести команду, создающую базу данных в окне "Script Execute" (рис. 3) и нажать кнопку Выполнить.

Команда CREATE DATABASE - Создание базы данных MS SQL Server

Базы данных создаются командой **CREATE DATABASE**. Создание баз данных разрешается любому пользователю с ролью системного администратора или всем, кому системный администратор предоставил такое право. Команда **CREATE DATABASE** имеет следующий синтаксис:

```
01. CREATE DATABASE имя_базы
02. [ ON [PRIMARY] ([ NAME = логическое_имя_файла, ]
03. FILENAME = 'имя_файла_ОС'
04. [, SIZE = размер]
05. [, MAXSIZE = { максимальный_размер | UNLIMITED } ]
06. [, FILEGROWTH = приращение] )
07. | {FILEGROUP имя_группы_файлов FILEDEFINITIONS}
08. [,...n] ]
09. [LOG ON {[ NAME = логическое_имя_файла, ]
10. [FILENAME = 'имя_файла_ОС'
11. [, SIZE = размер]
12. [, MAXSIZE = { максимальный_размер | UNLIMITED } ]
13. [, FILEGROWTH = приращение] } [,...n]
14. [FOR LOAD | FOR ATTACH]
```

Если при создании базы не указан первичный файл данных и/или файл журнала, то отсутствующий файл (или файлы) создается с именем по умолчанию.

Физические файлы будут находиться в стандартном каталоге.

Первичному файлу присваивается имя **имя_базы.mdf**, а файлу журнала — **имя_базы_log.ldf**.

Если размер файлов не задан, то при создании размер первичного файла совпадает с размером первичного устройства базы **model**, а размер файла журнала и вторичных файлов данных равен 1 Мбайт. Он может быть и больше, если размер первичного файла базы данных model превышает 1 Мбайт. Хотя имена и размеры файлов указывать не обязательно, на практике это всегда следует делать. SQL Server создает базу данных за два этапа. На первом этапе база model копируется в новую базу данных, а на втором этапе инициализируется все неиспользуемое пространство.

Команда **CREATE DATABASE** имеет следующие параметры:

• **PRIMARY** — файл определяется как первичное устройство.

• **NAME** — логическое имя; по умолчанию совпадает с именем файла.
• **FILENAME** — полное имя файла на диске.
• **SIZE** — исходный размер файла. Минимальный размер файла журнала равен 512 Кбайт.
• **MAXSIZE** — максимальный размер файла.
• **UNLIMITED** — размер файла не ограничивается.
• **FILEGROWTH** — приращение размера в мегабайтах (MB), килобайтах (KB) или процентах (%). По умолчанию приращение равно 10%.
• **FOR LOAD** — обеспечивает обратную совместимость со сценариями SQL, написанными для предыдущих версий SQL Server.
• **FOR ATTACH** — указывает, что файлы базы данных уже существуют.

Пользователь, создавший базу данных, является ее владельцем. Все параметры конфигурации базы копируются из базы model, если только при создании базы не был указан параметр **FOR ATTACH.** В этом случае параметры конфигурации читаются из существующей базы данных. Рассмотрим некоторые примеры команды **CREATE DATABASE**:

/* База данных со стандартным размером и именами файлов */

```
01. CREATE DATABASE test1
02. /* Данные – 2 Мбайт, файл журнала – по умолчанию */
03. CREATE DATABASE test2
04. ON (FILENAME = 'c:\d1.mdf', SIZE = 2, NAME = 'd1')
05. /* Первичный файл – 10 Мбайт, одна группа файлов
06. g1 и журнал размером 10 Мбайт */
07. CREATE DATABASE test3
08. ON PRIMARY (FILENAME ='c:\test3.mdf',
09. SIZE = 10 , NAME = 'd1'),
10. FILEGROUP g1 (FILENAME – 'c:\g1.mdf',
11. SIZE = 10 , NAME = 'g1')
12. LOG ON (FILENAME ='c:\test3.ldf',
13. SIZE = 10, NAME = 'log1')
```

Задача 1. Создайте sql-скрипт создания новой базы данных под именем **Educator** на **"D:\Базы данных\Группа\ФИО_студента\Название_БД.mdf**, с первичным устройством, с исходным размером файла в 10 Мбайт и запустите на выполнение скрипт (кнопка на панели инструментов). Выполните в окне обозревателя объектов **Обновление**. Сохраните созданный скрипт в текущую папку под именем **1.sql**.

После успешного выполнения и обновления проводника у вас должна появится новая база данных.

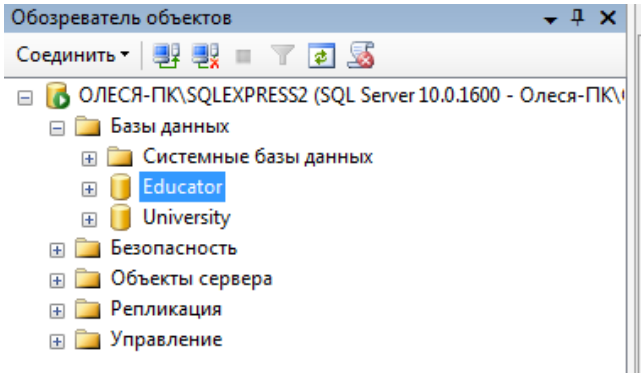


Рис. 6. Окно проводника после выполнения сценария создания базы данных

2.6.4. Подключение к базе данных

Чтобы подключиться к зарегистрированной базе данных, надо выбрать нужную базу данных в списке (рис. 5) и сделать двойной щелчок мышкой на выбранной базе

данных.

Если все параметры подключения были введены правильно, то произойдет подключение к базе данных, название подключенной базы данных в окне "Обозревателя объектов" будет выделено жирным шрифтом, а также появятся вложенные узлы с объектами, содержащимися в подключенной базе данных (рис. 7).

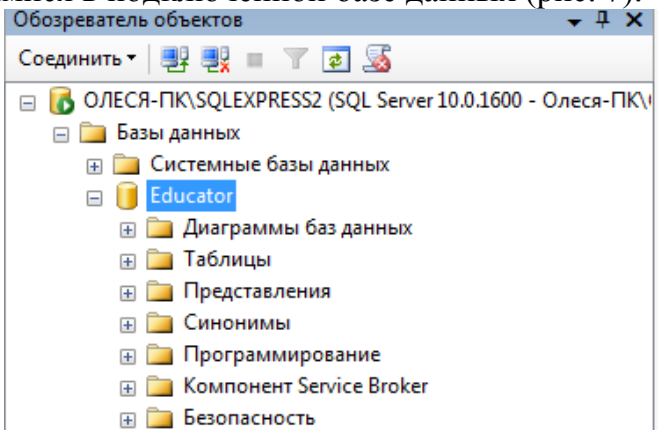


Рис. 7. Зарегистрированные базы данных в SQL Server Management Studio

После подключения к базе данных можно просматривать имеющиеся объекты, создавать новые, вносить и просматривать данные, а также проводить операции с имеющимися объектами.

После создания БД в окне **Обозревателя объектов** (его можно вызвать по **<F8>**) выбираем **DataBases (Базы данных)** и откроется список БД, в котором откроем созданную БД (если она не появилась, то в окне **Object Explorer** нажать **<F5>** для обновления списков), которая состоит из восьми вложенных разделов (некоторые содержат еще дополнительные разделы), соответствующих объектам СУБД SQL Server:

Database Diagrams (Диаграммы БД)	**Views (Представления)**	**Programmability (Объекты программирования)**
Tables (Таблицы)	**Synonyms (Синонимы)**	**Security (Безопасность)**
Service Broker	**Storage**	

На начальном этапе раздел созданной БД пуст, за исключением некоторых объектов, которые создаются по умолчанию, например в разделе **Security/ Users** создаются пользователи, которые имеют право на доступ к объектам БД, их можно изменить.

2.6.5. Удаление базы данных

Для удаления базы данных можно использовать один из трех способов:

1. Выполнить в программе " SQL Server Management Studio " команду контекстного меню "**Удалить**" , выбрав перед этим в списке базу данных, а затем подтвердить свое желание в диалоговом окне.
2. Выполнить оператор **DROP DATABASE** в SQL-редакторе.
3. Удалить файл с базой данных.

Синтаксис оператора **DROP DATABASE:**

DROP DATABASE database_name;

2.6.6. Резервное копирование и восстановление

Резервное копирование (**backup**) базы данных и ***восстановление*** из резервной копии (**restore**) – два важнейших и наиболее частых процесса, осуществляемых

администраторами баз данных.

Резервное копирование базы данных – единственный надежный способ предохранить данные от потери в результате поломки диска, сбоев электропитания, действий злоумышленников и ошибок в программах. В процессе резервного копирования создается независимый от платформы "снимок" базы данных, с помощью которого можно перенести данные на другую операционную систему или даже другую платформу.

Полный цикл: резервное копирование и восстановление из резервной копии приводит к корректировке статистической информации, является средством от излишнего "разбухания" базы данных и необходимой операцией обслуживания базы данных. Кроме того, миграция от одной версии сервера к другой также происходит при помощи процесса **backup/restore**.

Для создания резервной копии базы данных с помощью программы " SQL Server Management Studio " необходимо подключиться к базе данных, выбрать из контекстного меню базы данных **Задачи/ Создать резервную копию**. В открывшемся диалоговом окне "**Мастер резервного копирования**" задать несколько параметров и нажать кнопку [**Выполнить**], см. рис.8.

После выбора пути и файла для резервной копии в окне **Back Up Database** нажатием на OK запускаем процесс создания резервной копии. В случае успешной работы появится сообщение.

В результате будет создан файл с резервной копией. Стандартным расширением таких файлов для " SQL Server Management Studio " является "***.bak**". Файл с резервной копией базы данных обычно на порядок меньше оригинала.

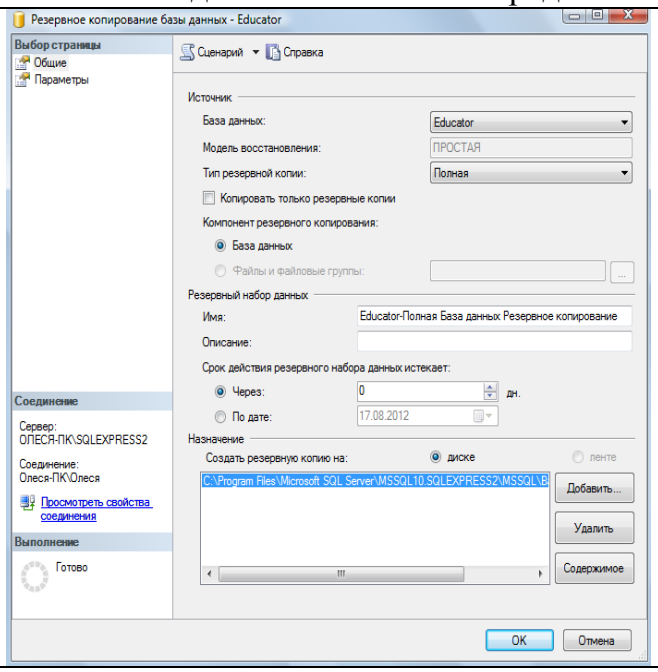


Рис.8. Создание резервной копии базы данных

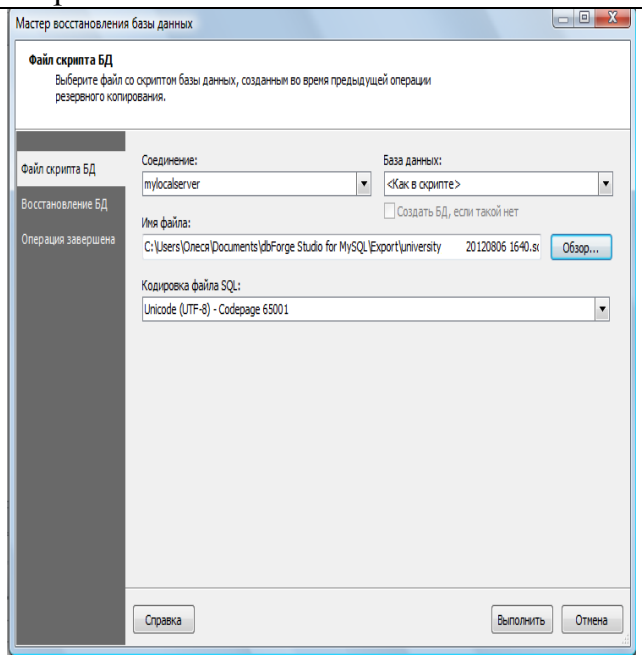


Рис.9. Восстановление базы данных

Для восстановления базы данных из резервной копии используется команда "**База данных/ Восстановление базы данных**. В результате откроется диалоговое окно "**Мастер восстановления баз данных**", в котором надо выбрать имя БД куда будет восстанавливаться база данных, в которую будет помещен результат, способ восстановления, файл, из которого будет восстанавливаться база данных, отмечаем выбранную резервную копию, и нажать кнопку [**Восстановить**], см.рис.9. Запускаем процесс восстановления. В случае успешного выполнения получим сообщение.

Резервное копирование и восстановление базы данных, наряду с процессом

извлечения метаданных и последующего выполнения полученного сценария, можно использовать при переносе разрабатываемой базы данных между различными компьютерами для обеспечения самостоятельной работы студентов над практическими работами и курсовым проектом.

Самостоятельно Выполните вначале резервирование, а затем восстановление базы данных.

Удалите базу данных **Educator** с помощью скрипта сохраните sql-запрос.

2.7. Копирование и перенос на другой сервер БД

Для просмотра, запуска, остановки служб MS SQL Server необходимо запустить утилиту **SQL Server Configuration Manager** (рис. 10).

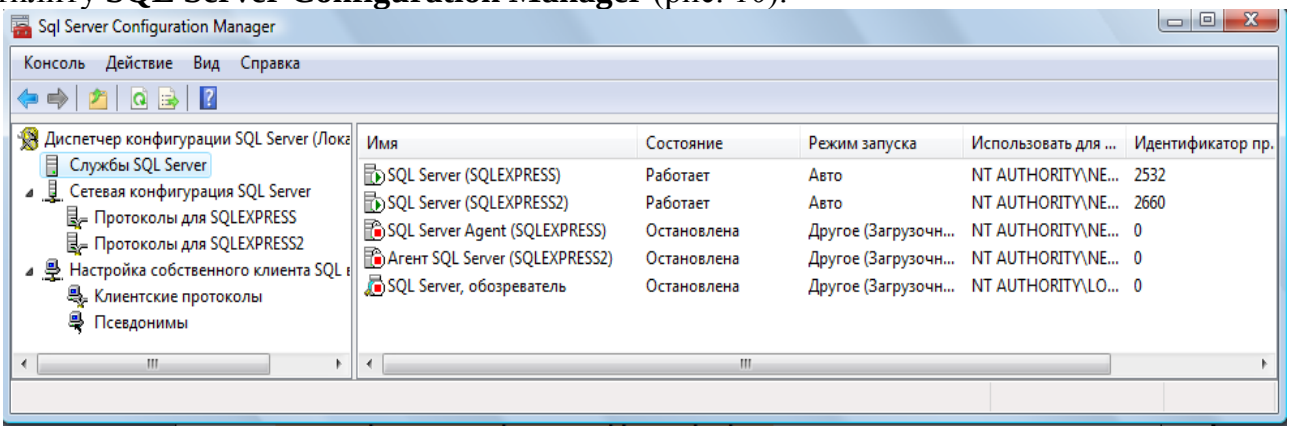


Рис.10. Список служб сервера БД

Для того **чтобы скопировать БД** необходимо остановить службу **SQL Server** (в ее контекстном меню выбрать **Stop**). Далее в подпапке **…\MSSQL.1\MSSQL\Data** скопировать файлы с вашим названием БД (по умолчанию их два). Не забудьте потом снова запустить службу **SQL Server** (в ее контекстном меню выбрать **Start**).

Для того **чтобы подключить** скопированную **БД** на другом сервере, нужно предварительно скопировать ваши файлы в папку **…\MSSQL.1\MSSQL\Data** соответствующего сервера. Далее запустить утилиту **SQL Server Management Studio.** В появившемся окне с названием **Object Explorer** Проводник объектов (его можно вызвать по <**F8**>) выбираем **DataBases** (**Базы данных**) и по <правой кнопке мыши> в контекстном меню (рис. 5) выбираем **Attach…** (**Присоединить…**). В появившемся окне **Attach DataBases** (Присоединение базы данных) нажать <**Add**> и выбрать ваш файл БД с расширением **.mdf**.

2.8. *Системные базы данных*

Системные базы данных сервера, создаваемые при установке, и их файлы представлены в таблице 1.

Название	Назначение	Размещение
Master	Хранит всю информацию сервера, включая учетные записи и параметры, сведения о всех базах и нахождении их первичных файлов с данными об инициализации баз данных пользователя.	Master.mdb – файл данных (75.mb) Mastlog.ldf – журнал транзакций (1 mb)
TempDB	Хранит все временные системные и пользовательские объекты: таблицы, переменные, хранимые процедуры, курсоры и т.п.	Temppdb.mdf – файл данных (8 mb) Templog.ldf – журнал транзакций (0.5 mb)
Model	Является шаблоном, задаваемых администратором и используемым для создания любых пользовательских баз данных. Содержит параметры по умолчанию, которые можно переопределять при создании соответствующей базы данных пользователя.	Model.mdf – файл данных (0.75 mb) Model.ldf – журнал транзакций (0.75 mb)
MSDB	Хранит информацию, относящуюся к автоматизации администрирования и управления сервером.	Msdbdata – файл данных (3.5 mb) Msdblog – журнал транзакций (0.75 mb)
		Всего – 22.75 mb

Все системные и пользовательские базы данных содержат в обязательном порядке 18 системных таблиц, которые хранят информацию, определяющие структуру и организацию соответствующей базы данных.

MSSQL Server поддерживает два основных класса приложений клиентского типа :

1. приложения реляционных баз данных, использующие команды Transact - SQL с расширениями ODBC и набор стандартных функций и объектно-ориентированных методов;
2. web - приложения , использующие команды Transact - SQL или запросы на языке Xpath и документы XML.

Оба класса приложений используют API интерфейс баз данных типа OLE DB или ODBC.

2.2. Основные принципы управления учетными записями и ролями в MS SQL Server

2.2.1. Список системных процедур и команд, которые позволяют реализовать политику разделения прав между пользователя БД.

Название встроенной процедуры	Описание
sp_grantlogin	– позволяет использовать пользователей или группы ОС для соединения с Microsoft SQL Server™ , используя **Windows Authentication**. Этот пример позволяет пользователю Windows NT Corporate\BobJ соединяться с SQL Server. Например, **EXEC sp_grantlogin 'Corporate\BobJ'**

sp_defaultdb	Изменяет для пользователя БД по умолчанию Этот пример устанавливает БД по умолчанию pubs для пользователя Victoria. Например, **EXEC sp_defaultdb 'Victoria', 'pubs'**
sp_grantdbaccess	Добавляет учетную запись из раздела security в текущую БД, для учетных записей Microsoft Windows также дает разрешение на доступ к текущей БД. **Синтаксис:** **EXEC sp_grantdbaccess [@loginame =] 'login'** **[,[@name_in_db =] 'name_in_db' [OUTPUT]]** Этот пример добавляет учетную запись Corporate\GeorgeW в текущую БД и присваивает псевдоним внутри БД Georgie. Например, **EXEC sp_grantdbaccess 'Corporate\GeorgeW', 'Georgie'**
sp_revokedbaccess	Удаляет информацию об учетной записи из текущей БД. **Синтаксис:** **EXEC sp_revokedbaccess [@name_in_db =] 'name'** Этот пример удаляет учетную запись Corporate\GeorgeW из текущей БД. **EXEC sp_revokedbaccess 'Corporate\GeorgeW'**
sp_addrole	Создает новую роль в текущей БД. Этот пример создает новую роль в текущей БД с названием Managers. **EXEC sp_addrole 'Managers'**
sp_addrolemember	В текущей БД назначает роль конкретному пользователю. **Пример A.** Этот пример добавляет учетную запись Corporate\JeffL из Windows NT в БД Sales как пользователя Jeff. Jeff затем получает роль Sales_Managers в БД Sales. **USE Sales** --сделать текущей БД Sales **GO –**выполнить команду, а потом запустить следующую **EXEC sp_grantdbaccess 'Corporate\JeffL', 'Jeff'** **GO** **EXEC sp_addrolemember 'Sales_Managers', 'Jeff'** **Пример B.** Этот пример добавляет пользователя SQL Server с именем Michael к роли Engineering в текущей БД. **EXEC sp_addrolemember 'Engineering', 'Michael'**
sp_helprotect	Показывает список привилегий, ассоциированных с ролью.
sp_helprolemember	Показывает список пользователей БД, входящих в указанную роль
sp_addsrvrolemember	Присвоение встроенной серверной роли для существующей учетной записи **sp_addsrvrolemember [@loginame =] 'login' , [@rolename =**

	] 'role' Например: **sp_addsrvrolemember 'Admin_DB', 'sysadmin'**
sp_dropsrvrolemember	Удаление встроенной серверной роли для учетной записи или группы **sp_dropsrvrolemember [@loginame =] 'login' , [@rolename =] 'role'** Например: **sp_dropsrvrolemember 'Admin_DB' , 'sysadmin'**
sp_helpsrvrole	Описание только встроенных ролей в SQL Server **sp_helpsrvrole [[@srvrolename =] 'role']** Например: **sp_helpsrvrole 'sysadmin'**
sp_helpsrvrolemember	Возвращает список ролей и учетных записей, которым присвоены эти роли **sp_helpsrvrolemember [[@srvrolename =] 'role']** Например: **sp_helpsrvrolemember 'sysadmin'**
sp_srvrolepermission	Возвращает список ролей и разрешений, которые присвоены этим ролям **sp_srvrolepermission [[@srvrolename =] 'role']** Например: **sp_srvrolepermission 'sysadmin'**
sp_addlogin

	и паролем Public_1 (пароль не должен совпадать с именем пользователя) с помощью следующих команд (не забудьте нажать F5 для запуска команд на выполнение): **EXEC sp_addlogin 'Public_','Public_1', 'DB_Books'** **use DB_Books** **EXEC sp_adduser 'Public_','Public_'** В БД DB_Books создан пользователь Public_ с ролью в БД DB_Books public.
Deny (отрицание)	Этот пример запрещает несколько системных привилегий для нескольких пользователей. Пользователи не могут использовать системные привилегии **CREATE DATABASE** or **CREATE TABLE**, если они не наделены имим через команду **GRANT**. **Пример:** **DENY CREATE DATABASE, CREATE TABLE** **TO Mary, John, [Corporate\BobJ]** **DENY SELECT, INSERT, UPDATE, DELETE ON authors TO Mary, John, Tom**
Grant (предоставлять)	This example grants multiple statement permissions to the users Mary and John, and the Corporate\BobJ Windows NT group. **GRANT CREATE DATABASE, CREATE TABLE** **TO Mary, John, [Corporate\BobJ]** Назначение разрешения на выборку (SELECT) для роли PUBLIC в таблице Authors: **GRANT SELECT ON Authors TO public**
Revoke (отменять)	This example revokes multiple statement permissions from multiple users. **REVOKE CREATE TABLE, CREATE DEFAULT** **FROM Mary, John** This example removes the denied permission from Mary and, through the SELECT permissions applied to the Budget role, allows Mary to use the SELECT statement on the table. **REVOKE SELECT ON Budget_Data TO Mary**

2.2.2. Создание пользователей для доступа к серверу через утилиту Microsoft SQL Server Management Studio

Создадим новую учетную запись для нашей базы данных **University**. Для этого выберите в Обозревателе объектов раздел **Безопасность/Имена входа**. Добавьте новое **имя входа – Proba**, установите опцию **Проверка подлинности SQL Server**, присвойте свой пароль, примените к выбранной базе данных, установите язык по умолчанию – русский.

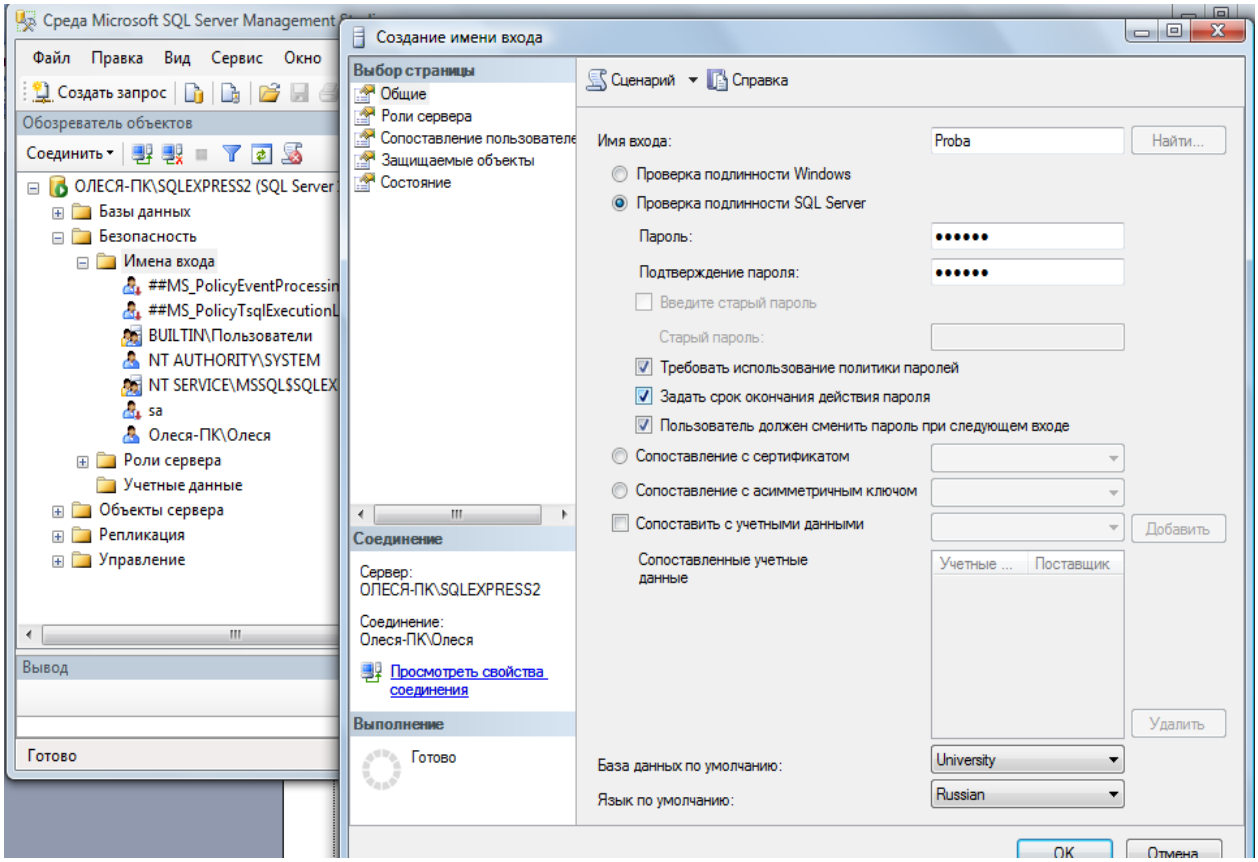


Рис. 2.1. Раздел Безопасность (Security) для работы с пользователями и создание нового пользователя (при SQL Server аутентификации нужно снять галочки с **Enforce password policy**)

Прежде чем добавлять нового пользователя просмотрите его назначенные серверные роли. Для этого в этом же окне выберите раздел **Роли сервера**. Установите для пользователя **Proba** роль **sysadmin**.

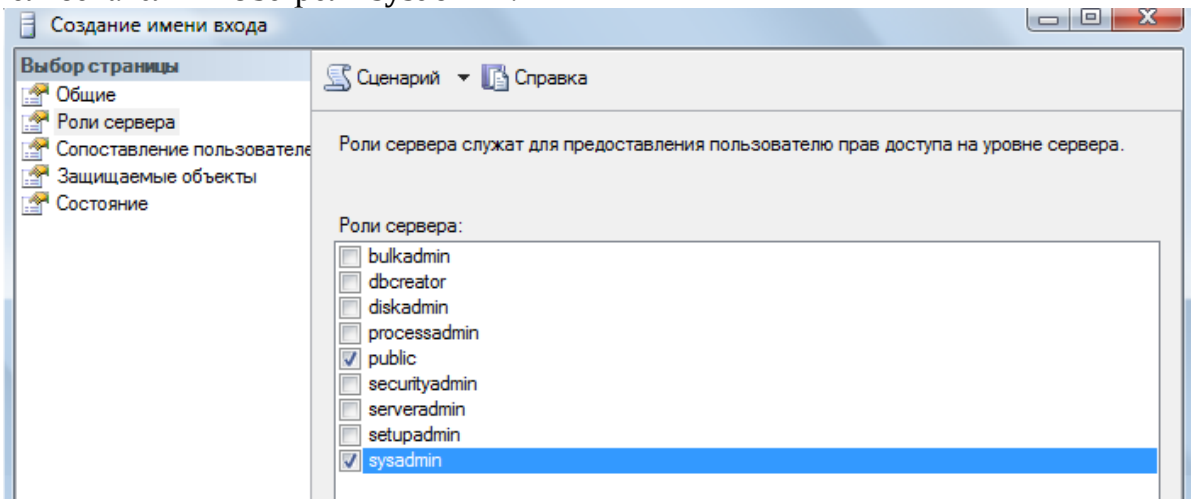


Рис. 2.2. Настройка серверной роли для нового пользователя (весь список серверных ролей с их привилегиями в конце работы)

Далее просмотрите раздел **Сопоставление пользователя**. Установите для базы данных **University** у пользователя **Proba** права доступа **Db_owner**, означающие, что пользователь может выполнять **любые действия с БД**. Ниже перечислены все возможные варианты прав доступа.

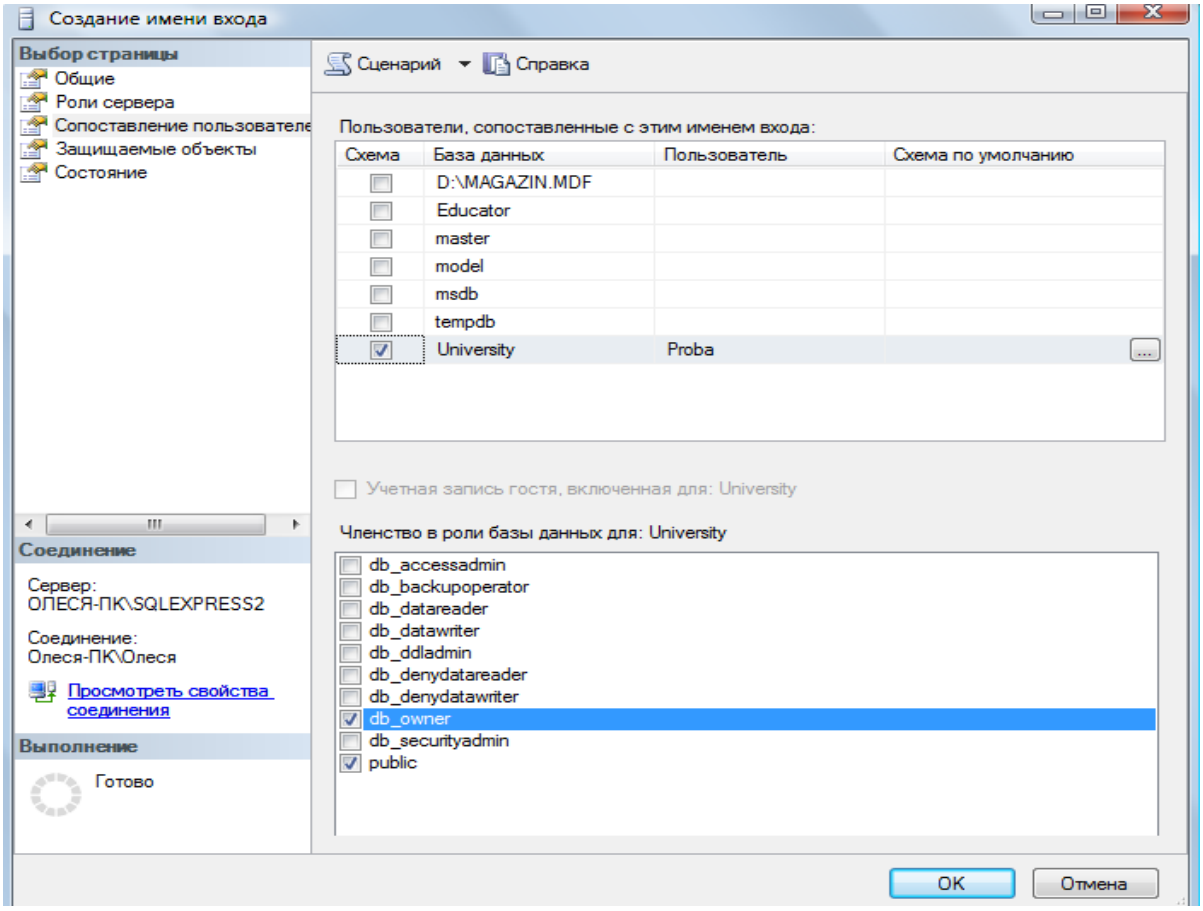


Рис. 2.3. Настройка роли базы данных для нового пользователя (весь список ролей баз данных с их привилегиями ниже)

Перечень ролей БД:

Public – минимальные права доступа к БД (на просмотр)

Db_owner – может выполнять любые действия с БД

Db_accessadmin – добавляет и удаляет пользователей БД

Db_sequrityadmin – управляет ролями в БД и разрешениями на запуск команд и работу с объектами БД

Db_ddladmin – добавляет, изменяет и удаляет объекты БД

Db_backupoperator – осуществляет резервное копирования БД

Db_dataSTUDENT – может просматривать все данные в каждой таблице в БД

Db_datawriter - может добавлять, удалять и изменять данные в каждой таблице в БД

Db_denydataSTUDENT – запрет на просмотр всех данных в каждой таблице в БД

Db_denydatawriter - запрет на добавление, удаление и изменение всех данных в каждой таблице в БД

Далее перейдите на раздел **Состояние**. Установите опции **Разрешение к подключению к ядру СУБД** – предоставить и **имя входа** включить.

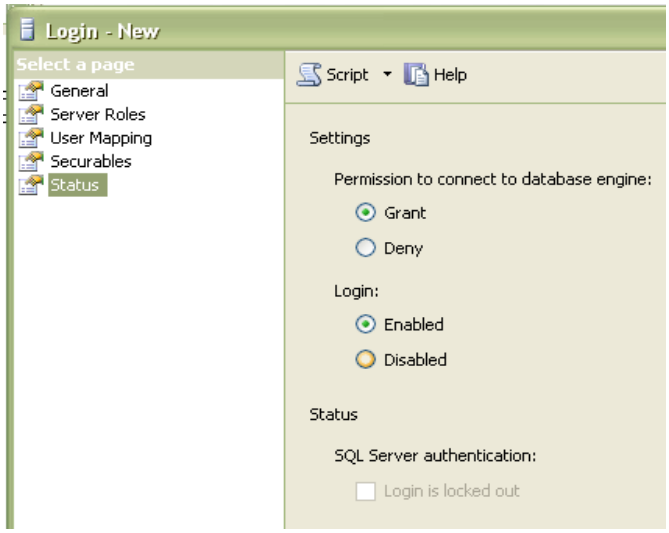


Рис. 4.4. Разблокирование создаваемой учетной записи

После нажатия на <OK> в БД появится пользователь **Proba** с правами собственника БД, который может выполнять все манипуляции с БД **University**.

Откройте в окне обозревателя объектов БД **University** и перейдите на вкладку **Безопасность**, там вы найдите только что созданного пользователя.

2.2.3. Создание ролей программно

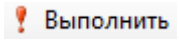
Для упрощения управления правами доступа в системе создаются **роли**, которые затем можно назначать **группе пользователей.**

Создадим для нашего примера роли декана (**DEKAN**) и студента (**STUDENT**).

Пример создания роли декана:

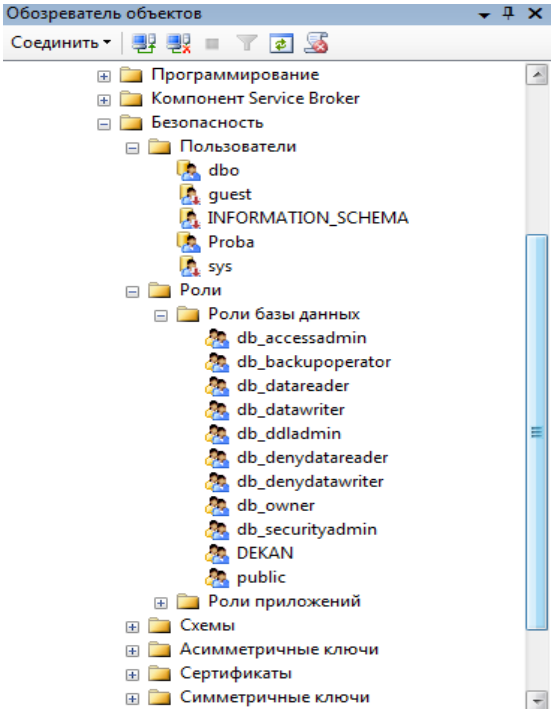
```
USE University  --сделать текущей БД University
EXEC sp_addrole 'DEKAN'
```

Эти операторы набрать на странице, вызванной нажатием кнопки <**Создать запрос**>.

Для запуска команд на выполнение нажать 

.

Сохраните запрос.



Повторный запуск тех же команд сгенерирует ошибки типа «**В БД уже существует роль DEKAN**».

Чтобы просмотреть, что роль добавлена, откройте вкладку **Безопасность/Роли/Роли базы данных**.

Пример создания роли студента:

USE University --сделать текущей БД university
EXEC sp_addrole 'STUDENT'

Декан должен обладать правами на **чтение, удаление, изменение, добавление во все таблицы БД University**, а также должен иметь возможность запускать на исполнение процедуры и функции БД University. Поэтому роли декана из системных привилегий назначаем **EXECUTE**, а из привилегий доступа к объектам назначаем **DELETE, INSERT, UPDATE, SELECT.**

Студент должен обладать правами **на чтение из таблиц**. Поэтому роли читателя из привилегий доступа к объектам назначаем **SELECT**.

Оператор представления привилегий

Синтаксис:
GRANT <привилегия>, ...
ON < объект >, …
TO <имя>
[WITH grant option];

Атрибут **WITH GRANT OPTION** дает право пользователю самому раздавать права, которые он получил.

С помощью оператора **GRANT** для каждого пользователя формируется список привилегий, привилегии управляют работой сервера данных с точки зрения защиты данных. Выполнению каждой транзакции предшествует проверка привилегий пользователя, сеанс которого породил транзакцию.

Например (не выполнять):

GRANT select, update (Sales, num) ON Sales_data TO user1
WITH GRANT OPTION

Пользователь, предоставивший привилегию другому, называется **грантор** (grantor — предоставитель). Привилегия является предоставляемой, если право на нее можно предоставить другим пользователям.

PUBLIC — имя роли, которую получает пользователь при добавлении в список пользователей конкретной БД, включает в себя минимальный набор прав на чтение данных из таблиц и представлений в БД.

Для примера (немного забегая вперед) создадим таблицу **Discuplinu**. Без объяснения синтаксиса выполните следующий sql-запрос:

```
USE University  --сделать текущей БД university
create table Discuplinu (
   Kod_Discuplinu int NOT NULL primary key,
        name_Discuplinu nchar(30) NULL,
        kol_chasov int NULL
);
```

Выполните код и обновите вкладку **Таблицы**. Вы должны увидеть созданную таблицу для сохранения данных о всех дисциплинах. Эта таблица пока пустая с тремя столбцами **Kod_Discuplinu, name_Discuplinu, kol_chasov.**

Роль декана названа **DEKAN**. Операторы назначения прав доступа для этой роли представлены ниже:

```
GRANT DELETE, INSERT, UPDATE, SELECT ON Discuplinu TO DEKAN
GRANT EXECUTE TO DEKAN
```

Роль студента названа STUDENT. Операторы назначения прав доступа для этой роли представлены ниже:

```
GRANT SELECT ON Discuplinu TO STUDENT
```

Примените роли декена и студента к созданной таблице.

Создание пользователей с определенной ролью

Пример создания декана Ivanov_Dek и присвоения ему роли:

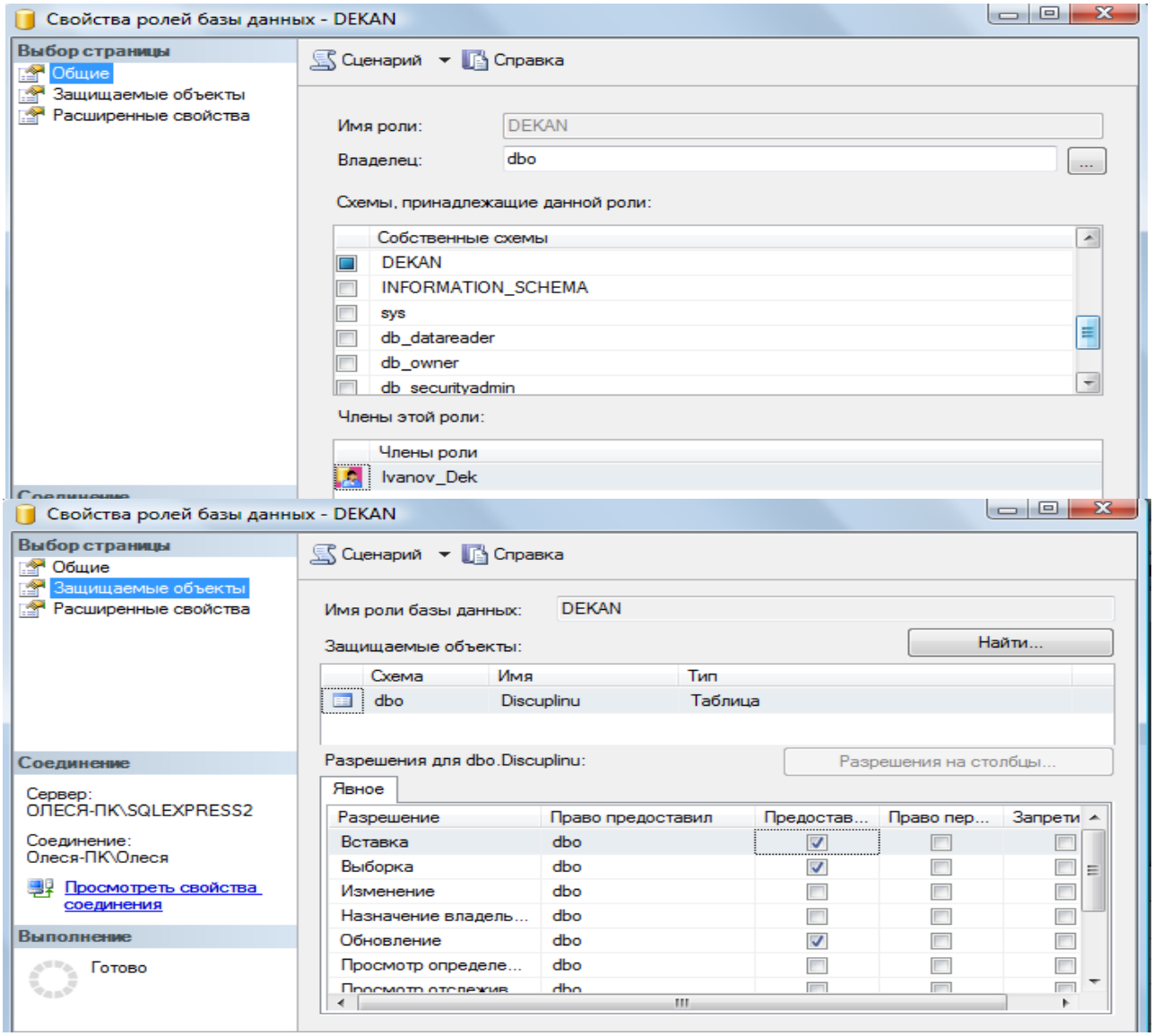
```
EXEC sp_addlogin 'Ivanov_Dek','Ivanov', 'University'
use University
EXEC sp_adduser 'Ivanov_Dek','Ivanov_Dek'
EXEC sp_addrolemember 'DEKAN', 'Ivanov_Dek'
```

Пример создания студент Petrov_Stud и присвоения роли:

```
EXEC sp_addlogin 'Petrov_Stud','Petrov', 'University'
use University
EXEC sp_adduser 'Petrov_Stud','Petrov_Stud'
EXEC sp_addrolemember 'STUDENT', 'Petrov_Stud'
```

Выполните команды. Перейдите в окне Обозреватель объектов на **Роли/Роли базы данных/Dekan** и просмотрите его свойства. Просмотрите назначенные общие свойства, защищаемые объекты и расширенные свойства.

Самостоятельно просмотрите свойства роли базы данных **Student.** Просмотрите назначенные общие свойства, защищаемые объекты и расширенные свойства.



Оператор отмены привилегий

Синтаксис отмены привилегий:

REVOKE [with grant option]

< привилегии >,…

ON < объект >,…

FROM <имя_пользователя>;

Предложение **with grant option** сохраняет за пользователем перечисленные привилегии, но отменяет его право передавать их кому-либо другому.

Пример:

REVOKE SELECT ON Discuplinu FROM STUDENT

Выполните команду.

Оператор изымания роли у пользователя

Revoke <список ролей> from <список пользователей>.

Пример:

use University

EXEC sp_droprolemember 'STUDENT', 'Petrov_Stud'
Выполните команду и просмотрите результат.

Задание для практической работы №2

- Создать файл базы данных, согласно номеру варианта, выданного в практической работе №1 с помощью sql-команды.
- Создать резервную копию базы данных.
- Определить **2-3** должностных лица, которые смогут работать с таблицами БД. Для каждого должностного лица определить набор привилегий, которыми он может пользоваться.
- В утилите **SQL Server Management Studio** создать под каждое должностное лицо соответствующую роль, наделить эту роль определенными привилегиями. Далее создать по одному пользователю на каждую должность и присвоить им соответствующие роли.
- Сохранить последовательно SQL-операторы с указанием заданий в файле с названием **ФамилияСтудента_Лаб_2.**
- Создать текстовый отчет, в котором отобразить sql-команды разработанных запросов и скриншоты результатов работы из СУБД **SQL Server Management Studio**.

ПРАКТИЧЕСКАЯ РАБОТА №3. РАЗРАБОТКА ТАБЛИЦ И ОГРАНИЧЕНИЙ

3.1. Цель практической работы

Изучить способы создания, изменения и удаления таблиц. Получить навыки использования приложения " SQL Server Management Studio " для создания, удаления и изменения структуры таблиц. Изучить SQL-операторы для работы с таблицами и индексами. Изучить используемые в SQL Server типы ограничений. Получить навыки использования программы " SQL Server Management Studio " для создания, изменения и удаления ограничений. Изучить SQL-операторы для работы с ограничениями.

3.2. Исходные данные

Исходными данными является индивидуальное задание и результат предыдущих практических работ.

3.3. Используемые программы

Программа " SQL Server Management Studio " и установленный сервер Microsoft SQL Server .

Часть 1. Создание таблиц

3.4. Теоретические сведения

3.4.1. Логическая структура и физическая реализация баз данных MS SQL Server

Данные в сервере SQL Server хранятся в базах данных. Структуру баз данных необходимо рассматривать на двух уровнях: **логическом и физическом.**

Логическая структура базы данных определяет структуру таблиц, взаимосвязи между ними, список пользователей, хранимые процедуры, правила, умолчания и другие объекты базы данных.

Физическая структура базы данных включает в себя описание файлов и групп файлов базы данных, журнала транзакций, первоначальный размер файлов, шаг прироста базы данных, ее максимальный размер, параметры конфигурации и другие физические характеристики.

Все логические компоненты базы данных SQL Server называются объектами и подразделяются на **11 типов.**

Для объектов можно устанавливать различные правила доступа со стороны пользователей.

Функциональное назначение объектов базы данных сервера SQL Server можно кратко определить следующим образом:

1. Table - таблица, представляющая собой матрицу из строк и столбцов.

Каждая строка (или запись) состоит из значений атрибутов конкретного объекта.

Столбец (или поле записи) содержит совокупность атрибутов рассматриваемых объектов некоторой предметной области. Некоторые столбцы таблицы могут быть вычисляемыми. В этих случаях для них задается расчетная формула.

2. User-defined data type - пользовательский тип данных, создаваемый на основе системных. Имя нового типа должно быть уникальным в пределах владельца.

3.View - представление, являющееся виртуальной таблицей, содержимое которой определяется запросом. Эта таблица не содержит данных, а только их представляет, возможно из нескольких таблиц. Данные из представления не сохраняются в базе данных. Физически представление реализуется в виде запроса **SELECT**. Представления используются в следующих случаях:

a) Для ограничения доступа пользователей к определенным строкам таблицы;
b) Для ограничения доступа пользователей к определенным столбцам таблицы;
c) Для представления данных столбцов разных таблиц в виде одного объекта;

d) Для просмотра информации, получающейся в результате преобразования данных столбцов.

4.Stored procedure - хранимая процедура, представляющая собой группу команд Transact-SQL, объединенных в один модуль. Каждая хранимая процедура имеет уникальное , в пределах базы данных, имя, по которому она вызывается. Хранимая процедура может вызывать другие хранимые процедуры. В состав SQL Server входит большое количество встроенных процедур, которые называются системными и имена которых начинаются с префикса **sp_.**

5.Trigger - триггер, представляющий собой специальную хранимую процедуру, автоматически запускаемую при добавлении, изменении или удалении данных из таблицы. Триггеры делятся на три категории:

UPDATE TRIGGER - триггеры изменения;

INSEART TRIGGER - триггеры вставки;

DELETE TRIGGER - триггеры удаления.

Действия, выполняемые в одном триггере, могут вызвать другие триггеры (вложенные триггеры).

6.Index - индекс, представляющий собой структуру, связанную с таблицей или представлением и предназначенную для ускорения поиска информации в этой таблице или представлении. Индекс определяется для одного или нескольких столбцов, называемых индексированными столбцами. Индекс содержит отсортированные значения индексированного столбца или столбцов со ссылкой на соответствующую строку исходной таблицы или представления. Алгоритмы поиска в отсортированных данных гораздо эффективнее, чем в неотсортированных.

7.Rule - правило, используемое для ограничения значений, хранимых в столбце таблицы или в пользовательском типе данных. Одно и тоже правило может связываться с множеством столбцов различных таблиц и пользовательских типов данных только в текущей базе данных. Правило создается командой **CREATE RULE** и связывается с объектом базы данных с помощью процедуры spbindrule. Правила оставлены в Transact-SQL для совместимости со старыми версиями сервера.

8.Constraint - ограничение целостности, представляющее собой механизм, обеспечивающий автоматический контроль соответствия данных установленным условиям, или ограничениям целостности. Ограничения целостности имеют приоритет над триггерами, правилами и значениями по умолчанию. Имеется пять ограничений целостности, различающихся по функциональности и области применения:

NULL - действует на уровне столбца и пользовательского типа данных и либо разрешает (NULL), либо запрещает (NOT NULL) хранение значений NULL.

CHECK - действует на уровне столбца и ограничивает диапазон значений, которые могут быть сохранены в столбце, путем проверки логического условия для вводимых данных. При вводе или изменении данных вводимое значение подставляется в условие. Если полученный результат

TRUE, то изменения данных принимаются, иначе - отвергаются и генерируется сообщение об ошибке. Для одного столбца можно задать несколько ограничений типа **CHECK** (проверок):

CONSTRAINT humanavance

CHECK (human_ avance BEETWEEN 0 and 700)).

UNIQUE - действует на уровне столбца и гарантирует уникальность в столбце вводимых значений. В отличии от ограничения **PRIMARY KEY**, это ограничение допускает хранение значений NULL.

PRIMARY KEY - действует на уровне столбца или таблицы и гарантирует уникальность в пределах таблицы первичного ключа, состоящего из одного или нескольких столбцов. Ни для одного из столбцов ключа не должно быть установлено свойство NULL. Когда используется один столбец, то для него необходимо также задать и свойство UNIQUE. В таблице создается только один первичный ключ. При его выборе надо учитывать требования удобства и функциональности.

FOREIGN KEY - действует на уровне таблицы и связывается с одним из кандидатов на первичный ключ в другой таблице. Таблица, в которой определен внешний ключ с помощью этого ограничения, называется зависимой, а таблица с кандидатом на первичный ключ - главной. В зависимую таблицу нельзя вставить строку, если внешний ключ не имеет соответствующего значения в главной таблице. Из главной таблицы нельзя удалить строку, если с ней связана хотя бы одна строка в зависимой таблице. Формат задания ограничения таков:

FOREIGN KEY REFERENCES имя главной таблицы (кандидат на первичный ключ или ее ключ).

9.Default - умолчание, представляющее собой значение, которое будет присвоено элементу столбца таблицы при вставке строки (записи), если в команде вставки явно не указано значение для этого столбца. Умолчаниями могут быть константы, а также встроенные функции и математические выражения, возвращающие конкретные значения. Создание умолчания выполняется командой **CREATE DEFAULT**, а удалить само умолчание командой **DROP DEFAULT.**

10.Function - функция, представляющая собой программный модуль, выполняющий некоторые часто используемые действия над данными и возвращающий значение какого-либо типа. Имя функции, осуществляющее ее вызов, может указываться в любом выражении языка Transact-SQL.

Встроенные функции (built-in functions) являются составной частью среды программирования сервера, выполняют заранее предопределенную последовательность команд и не могут изменяться пользователем: **COUNT, SUM, MIN, MAX** и т. д. **Функции пользователя** (user_ defind function) создаются пользователем по правилам языка Transact-SQL для реализации разрабатываемых алгоритмов.

Каждая копия, или **экземпляр сервера SQL Server имеет 4 системных базы данных** с именами **master, msdb, tempdb и model**, а также несколько созданных администратором пользовательских баз данных. Набор системных баз постоянен и не может быть изменен. Обращаться к этим базам напрямую запрещено. Обращаться к ним можно только с помощью специально разработанных интерфейсов:

a) С помощью системных хранимых процедур;
b) С помощью интегрированной среды **Enterprice Manager**;
c) Используя программные интерфейсы **Transact-SQL (API).**

При соединении с экземпляром сервера организуется связь с конкретной базой данных на сервере. Эта база данных называется ***текущей.*** Она определяется для каждого пользователя системным администратором.

Пользователь может переключаться на другую базу, используя команду

USE < имя базы данных>

или функции API для изменения текущей базы данных.

SQL Server позволяет соединить базы данных от одной копии сервера и подсоединить к другой, а затем восстановить подсоединение к прежней копии.

Все базы данных SQL Server, как системные, так и пользовательские, физически **организованы одинаково. Каждая база данных хранится в отдельных файлах**. В отдельном файле хранится **журнал транзакций**, создаваемому автоматически для каждой базы данных. Это повышает надежность системы. Если произойдет сбой и файлы базы данных будут повреждены, то можно восстановить базу данных из резервной копии, а затем восстановить сделанные изменения из журнала транзакций, который останется не поврежденным.

Таким образом, каждая база данных имеет, минимум, **два файла: один для базы данных и один для журнала транзакций**. Эти файлы имеют различную структуру и при работе с ними применяются разные правила.

Основная **единица хранения данных** на уровне файла базы данных - **это страница**, которая участвует в операциях ввода-вывода как единое целое даже тогда, когда требуется всего одна строка. Размер страницы равен 8 Кбайт.

Файл журнала транзакций не имеет страниц и экстентов. Он содержит только последовательность записей транзакций, выполняемых в базе данных.

Каждая страница файла базы данных имеет объем 8192 байт. Первые 96 байт страницы отводятся под заголовок, в котором хранится системная информация: тип страницы, объем свободного места на странице, идентификационный номер таблицы или индекса - владельца страниц:

Имеется шесть типов страниц:

1.Data. В страницах этого типа хранятся собственно данные, исключая данные типа text, ntext и image.

2.Index. Страницы этого типа используются для хранения информации об индексированных таблицах.

3.Text/Image. В страницах этого типа хранятся данные типа text, ntext и image.

4.Global Allocation Map (GAM). В страницах данного типа хранится информация об использовании экстентов (групп страниц). Экстент состоит из 8 страниц (64 Кбайт).

5.Page Free Space. В страницах этого типа хранится информация о свободном пространстве на страницах.

6.Index Allocation Map (IAM). Страницы этого типа хранят информацию об экстентах, используемых таблицами или индексами.

Более сложные базы данных имеют несколько файлов для данных и для транзакций. В этом случае они объединяются в группы для упрощения администрирования базы данных. Любой из таких файлов может располагаться на отдельном диске.

Многофайловая база данных имеет в своем составе файлы следующих типов:

Primary (*.mdf) - основной файл, который содержит системную информацию о самой базе данных и ее объектах. В этом файле размещаются системные таблицы и описание объектов базы данных. Здесь могут храниться и данные. В базе данных такой файл только один и его наличие в базе обязательно.

Secondary (*.ndf) - вторичный файл, который используется только для хранения данных и не содержит системной информации. Он может отсутствовать в базе данных. Если задано несколько файлов этого типа, то они могут быть организованны в группы и распределены по разным физическим дискам.

Transaction Log (*.ldf) - файл журнала транзакций. Можно использовать несколько таких файлов для ускорения операций ввода-вывода, так как транзакции записываются параллельно во все файлы.

Каждый файл, используемый в базе данных, имеет два имени:

Logical File Name - логическое имя файла, которое используется в командах Transact-SQL при ссылке на конкретный файл;

OS File Name - имя файла в операционной системе, которое используется в операционной системе.

Для каждого файла базы данных можно задать свойство автоматического роста и шаг прироста в мегабайтах или в процентах от первоначального роста, а также максимальный размер, до которого возможен рост файла.

SQL-Server 2000 обеспечивает создание групп следующих трех типов:

Primary File Group - основная группа файлов, которая включает первичный файл и все файлы, не включенные в другие группы. База данных может иметь только одну основную группу файлов.

3.4.1. Таблицы (Tables)

Microsoft SQL Server – реляционная СУБД, поэтому все данные в Sql хранятся в виде **двумерных таблиц** со строками и столбцами. **Строки** называются кортежами

или записями, а **столбцы** – доменами или полями.

В этой практической работе рассматриваются способы создания таблиц.

Основные ограничения, которым должны удовлетворять таблицы:

1. Каждый столбец в таблице имеет уникальное имя.
2. Все данные в столбце должны быть одного типа.
3. Порядок строк и столбцов в таблице не имеет значения.
4. В таблице не может быть двух одинаковых строк.

3.4.2. Индексы

Microsoft SQL Server (как и другие реляционные СУБД) хранит записи в таблицах в неупорядоченном виде. Записи, добавляемые в таблицу одна за другой, не обязательно окажутся "рядом". Данные, извлекаемые из таблицы, также не имеют какого-либо порядка, кроме того, который явно указан в запросе на выборку информации.

Индекс – это упорядоченный указатель на записи таблицы. Индекс состоит из пар значений "значение поля" – "физическое расположение записи", поэтому по значению поля (или полей), входящего в индекс, при помощи индекса можно быстро найти место в таблице, где располагается запись, содержащая это значение.

Устройство индексов на физическом уровне для нас совершенно не имеет значения. Важно только знать, что создание индексов может привести к значительному ускорению процессов поиска и сортировки.

Не следует создавать индекс на поля с ограниченным набором значений – например, на поле, хранящие пол человека, которое содержит только два значения – "м" и "ж".

Использование индексов имеется два отрицательных последствия:

1. Для индексов дополнительно тратится дисковое пространство.
2. Наличие индексов замедляет модификацию данных в таблице.

3.5. Ход работы

Выполнение операций создания таблиц и индексов в диалоговом режиме.

Откройте среду SQL Server Management Studio, выполните соединение с сервером, откройте созданную базу данных **University** в лаб.раб. №1.

Для создания таблицы в <u>диалоговом режиме,</u> нажмите в окне "**Обозревателя объектов**" правую клавишу мыши на узле "**Tables**" (Таблицы) или на одной из имеющихся таблиц и в открывшемся меню выберите команду "**New Table**…(Создать таблицу)" (рис. 1). В результате откроется окно создания таблицы (рис. 2).

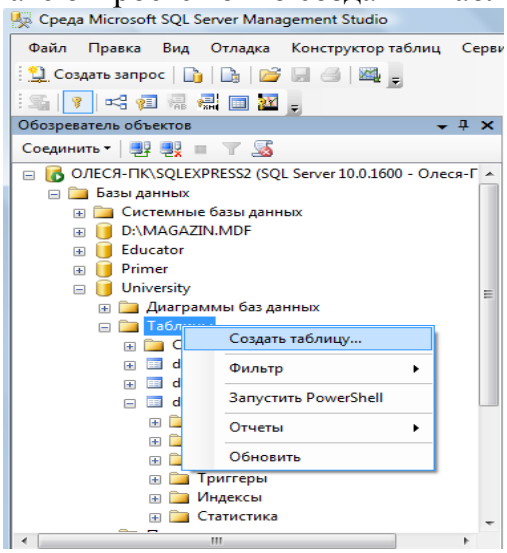


Рис. 1. Окно "Проводник" с перечнем таблиц

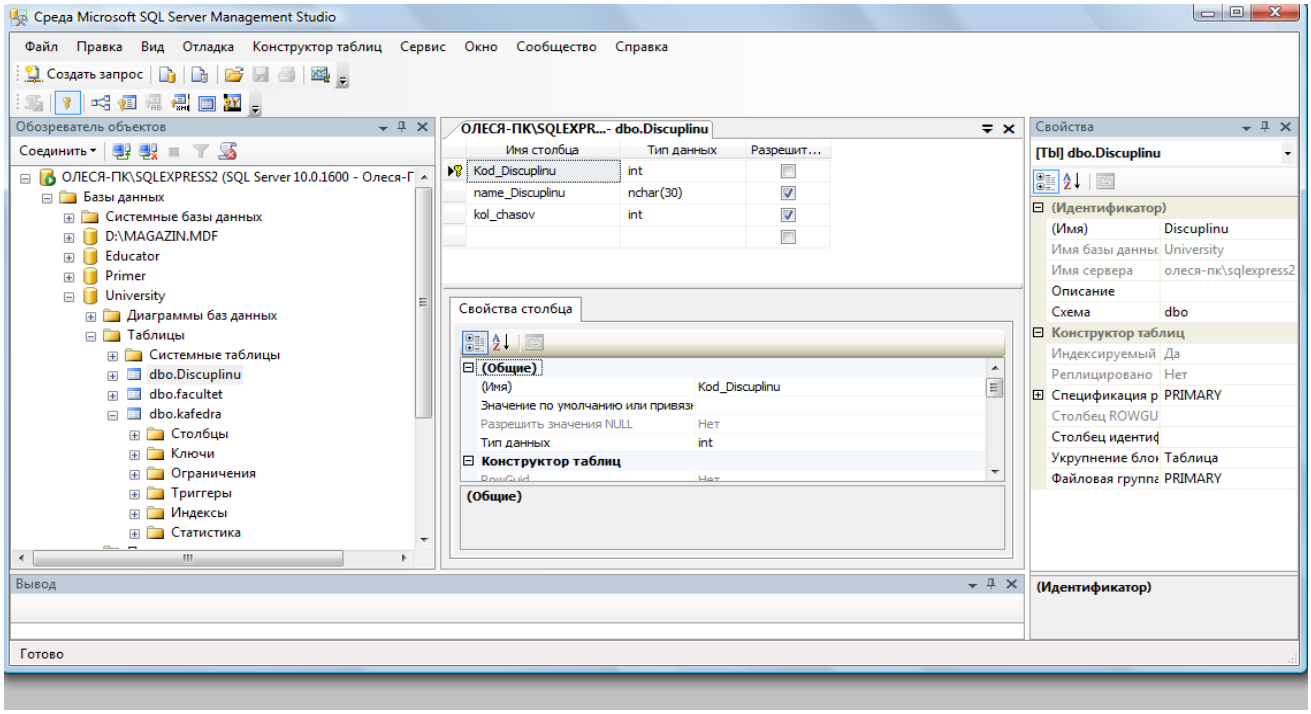


Рис. 2. Окно формирования таблицы в диалоговом режиме

Сетка в средней части окна содержит сведения о полях таблицы.

Чтобы добавить поле в таблицу, следует нажать правую клавишу и выбрать из контекстного меню [**Вставить столбец**].

В колонке "**Имя столбца**" вводится имя создаваемого поля, в колонке "**Тип данных**" выбирается тип данных. Чтобы задать полю ограничение "**NOT NULL**" достаточно установить флажок в колонке "**Разрешить значения NULL (пустые значения)**".

Чтобы присвоить полю статус первичного ключа необходимо в колонке **ПК** щелкнуть правой клавишей мыши и выбрать из контекстного меню **Создать первичный ключ.**

Дополнительные свойства можно настраивать с помощью окна **Свойства столбца,** которое находится внизу рабочей области.

Самостоятельно

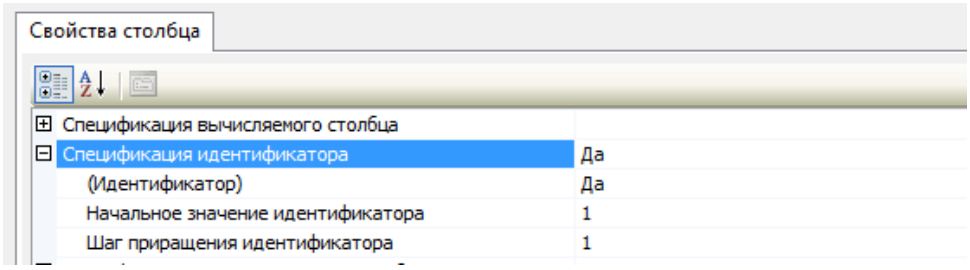
Создайте новую таблицу, хранящую информацию о всех факультетах в университете.

Присвойте ей имя – **Facultet.**

Добавьте поля **Kod_faculteta** (уникальный номер факультета, тип – целый **int**, первичный ключ, автоинкремент), **Name_faculteta** (название факультета, тип текстовый - **varchar**, длина 255 символов, не допускается пустых значений), **Fio_Decana** (ФИО декана факультета, тип – текстовый **varchar**), **Nomer_komnatu** (номер комнаты деканата, тип – символьный **varcha**r(допускается запись 134-2, где 134 – номер комнаты, а 2 – номер корпуса)), **Tel_decanata** (телефон деканата, тип – длинное целое число **bigint** с точностью в 10 символов, значение по умолчанию ‘999999’,ограничение на значение меньше ‘1 000 000’).

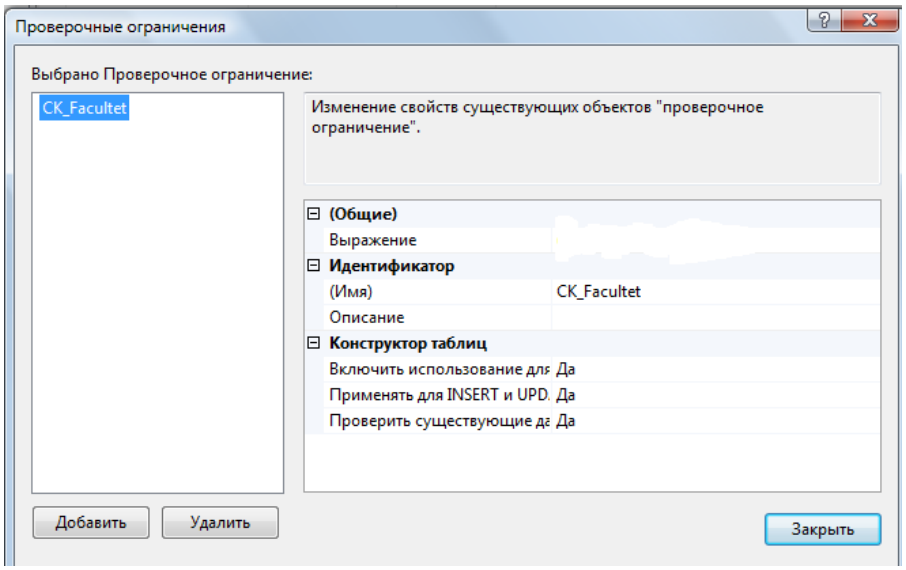
Примечание.

1). Для того, чтобы сделать **автоприращение** ключевого поля **kod_faculteta** измените в дополнительных свойствах столбца свойство – **Спецификация идентификатора**:



2). Для того, чтобы добавить проверочное ограничение на столбец **Tel_decanata** , выделите его и правой клавишей из меню выберите **Проверочные ограничения**. В появившемся окне нажмите кнопку **Добавить**. Будет создано ограничение под именем по умолчанию, для которого необходимо создать выражение вычисления ограничения на вводимые значения выбранного поля. В нашем случае ограничение на значение меньше ‘1 000 000’ будет иметь вид

Tel_decanata< 1 000 000.



После введения данных о всех полях таблицы следует нажать кнопку [**Сохранить**] на панели инструментов.

Создадим для данной таблицы индексы (для других таблиц создавать индексы не нужно!!).

Для этого выберите из контекстного меню **"Индексы и ключи"** (рис. 15).

В нашем случае для первичного ключа автоматически присваивается уникальный индекс по возрастанию, см. рис.15.

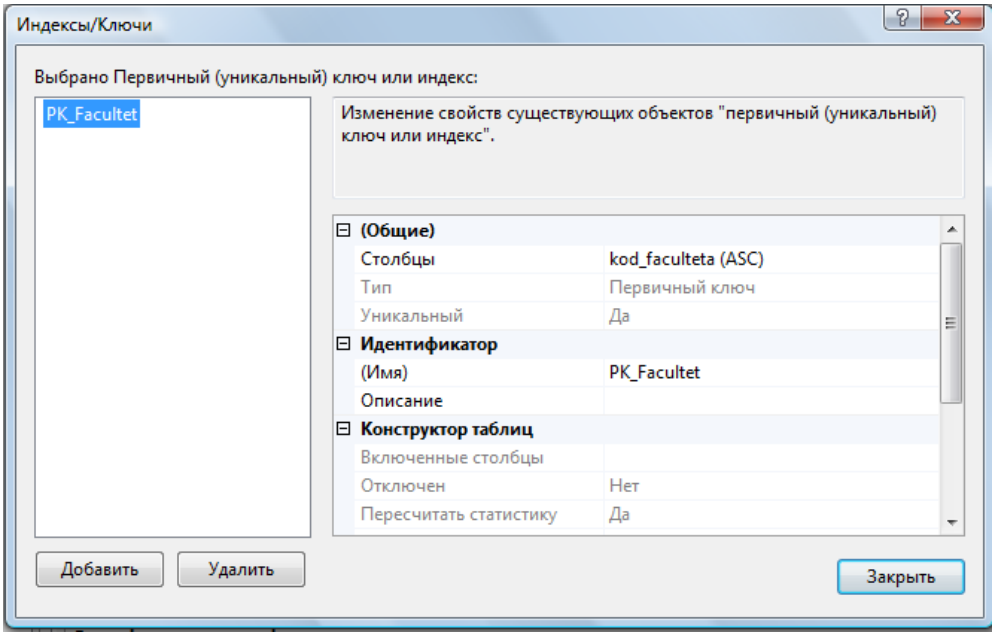


Рис. 15. Окно просмотра и редактирования индексов

Мастер позволяет просматривать, редактировать, создавать и удалять индексы.

Для создания индекса выполните следующие действия:

1. Нажмите в этой сетке клавишу [**Добавить**]. В результате будет вставлена новая строка.
2. Задайте в колонке "**имя**" имя индекса – **My_Index_Facultet**.
3. Нажмите кнопку … в колонке "**Столбцы**". В результате откроется окно с для выбора столбцов и порядка сортировки. В левом списке "**Имя столбца**" будут находиться поля, которые можно добавить к индексу, в правом списке "**Сортировка**" находиться список сортировки. Для формирования перечня полей, которые будут входить в индекс, добавьте нужные поля.
4. Если создается уникальный индекс, то изменяется свойство в колонке "**Уникальный**" .
5. Чтобы создать индекс нажмите кнопку **сохранить.**

После создания индекса его можно в любой момент изменить, если изменить параметры индекса и снова нажать кнопку **Сохранить.**

Для заполнения таблиц данными вначале выделите таблицу в окне проводника и из контекстного меню выберите **Изменить первые 200 строк**. Вид окна представлен на рис. 17.

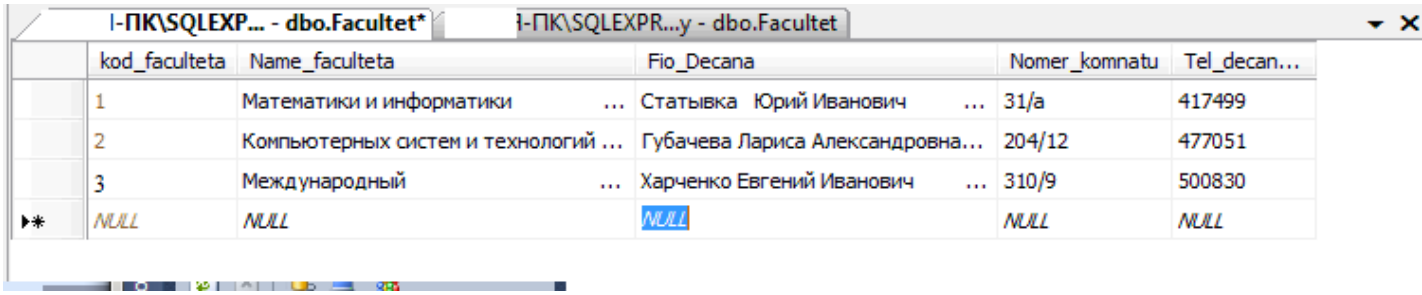


Рис. 17. Диалог создания записей в таблице

Введите самостоятельно три записи. Для добавления новой записи воспользуйтесь кнопкой перехода .

После ввода данных нажмите на панели кнопку о подтверждении изменения данных **Выполнить SQL – код.**

Примечание.

При вводе данных в поле **kod_faculteta** заполнять числами не нужно, они будут добавлены автоматически согласно формуле инкремента.

При вводе данных в поле **tel_decanata** значение можно оставить не заполненным. В результате выполнения sql-кода поле будет заполнено значением по-умолчанию.

При вводе данных в поле **tel_decanata** значение более 1 000 000 будет отображаться ошибка ввода связанной с проверочным ограничением.

Самостоятельно создайте новую таблицу, присвойте ей имя **Kafedra.**

Добавьте следующие поля в таблицу: **Kod_kafedru** (уникальный номер кафедры, тип – целый, первичный ключ, автоинкремент), **kod_faculteta** (номер факультета, к которому принадлежит кафедра, тип – целый, не допускается пустых значений), **Name_kafedru** (название кафедры, тип текстовый, длина 50 символов, не допускается пустых значений), **Fio_zavkaf** (ФИО заведующего кафедрой, тип – текстовый), **Nomer_komnatu** (номер комнаты кафедры, тип – числовой, **num_korpusa** (номер корпуса, тип – числовой, с ограничением – номер корпуса может быть, только от 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12), **Tel_kafedru** (телефон кафедры, тип – текстовый).

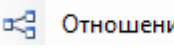
Примечание.

Для создания ограничений для столбца **num_korpusa** используйте в выражении функцию

IN (‘знач1’,’знач2’, ..,’значn’)

Создадим связь между двумя таблицами Факультет и Кафедра.

Между данными таблицами можно установить связь «**один-ко-многим**», так как на одном факультете может быть несколько кафедр. Например, к факультету **Математики и информатики** относятся кафедры **Информатики, Компьютерные сети и системы, Прикладной математики и Математического анализа.**

Чтобы создать связь «**один-ко-многим**» необходимо вызвать в окне редактирования таблицы **Кафедра** каскадное меню и выбрать Отношения 

.

Внешний ключ будет создан к полю **kod_faculteta** и оно будет связано с полем первичным ключом **kod_faculteta** внешней таблицы **Факультет.**

Для этого в появившемся окне **Связи по внешнему** (см. рис. 18) перейдите на поле **Спецификация таблиц и столбцов** и нажмите кнопку напротив для его изменения.

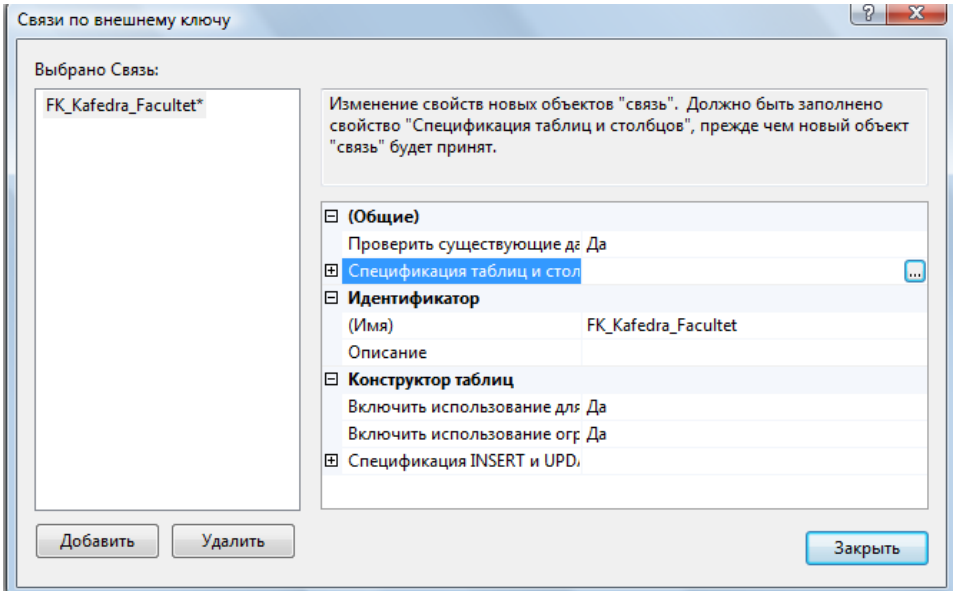


Рис. 18. *Диалог создания внешних ключей*

Вам будет открыто диалоговое окно создания **Внешних ключей** (рис. 19), где имя связи по умолчанию присвоенное внешнему ключу можете оставить без изменения.

Выберите в качестве таблицы первичного ключа – **Facultet** и поле **kod_faculteta.**

Выберите в качестве таблицы внешнего ключа – **Kafedra** и поле **kod_faculteta.**

Нажмите Ок и вернитесь в предыдущее окно.

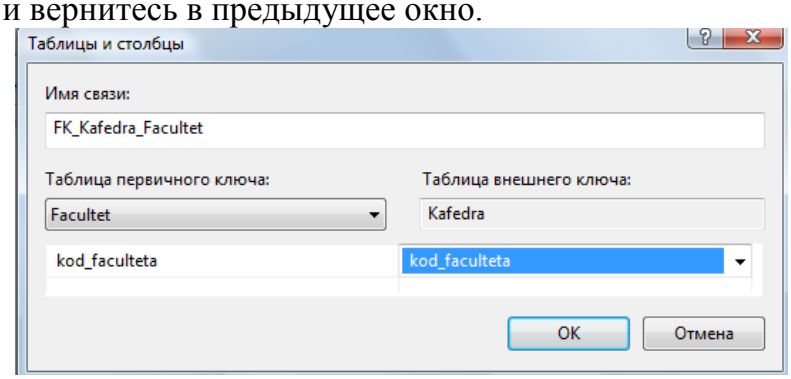
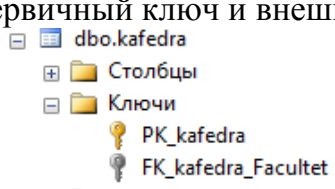


Рис. 19. Создание Внешних ключей

Установите правила удаления и обновления данных. Для этого перейдите на поле **Спецификации INSERT и UPDATE** и выберите правило **каскадного** удаления и обновления.

Сохраните таблицу.

После подтверждения операции у вас в таблице **Кафедра** появится внешний ключ по полю **kod_faculteta.** Для этого в окне проводника выберите в таблице **Кафедра** элемент **Ключи** и вы увидите первичный ключ и внешний ключ.



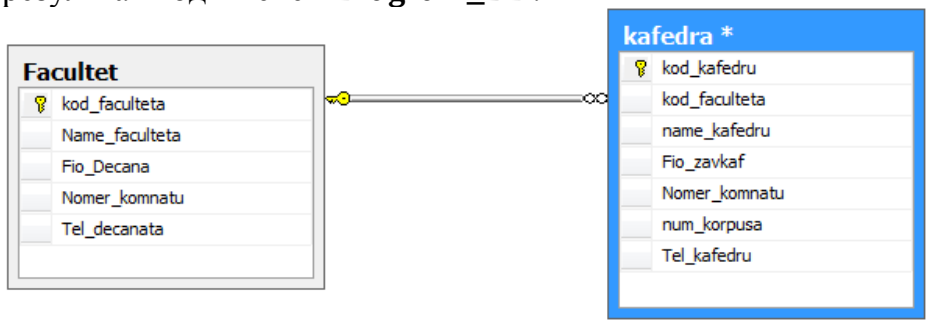
Введите самостоятельно название кафедр и их реквизиты на все созданные факультеты. Для добавления новой записи воспользуйтесь кнопкой перехода ✚.

\COMP\SQLEX...15 - dbo.Kafedra

Kod_kafe...	kod_faculteta	Name_kafedru	Fio_zavkaf	Nomer_ko...	num_korpusa	Tel_kafedru
1	1	Компьютерные системы и сети	Соловьев	414	1	899028
2	1	Прикладная математика	Кочевский	205	1	425262
3	1	Математического анализа	Арлинский	61	1	627272
4	1	Информатики	Пожидаев	404	1	738328
5	2	Автоматизации компьютерных технологий	Малахов	123	12	637327
6	2	Компьютерных технологий на промышленном транспорте	Губачева	342	12	373728
7	2	Системная инженерия	Ульшин	234	12	727287
8	3	Иностранных языков	Краснопольский	123	2	762728
9	3	Компьютерных наук	Дядечев	703	9	563272
NULL	NULL	NULL	NULL	NULL	NULL	NULL

После ввода данных нажмите на панели кнопку о подтверждении .

Для визуального отображения связей между таблицами воспользуйтесь средством **Диаграммы базы данных**, добавьте две созданных вами таблицы в проект и сохраните получившийся результат под именем **Diagram_BD**.



Задание для практической работы №3

Для ранее созданной базы данных по номеру варианта (созданной в лаб.работе №2) в СУБД **SQL Server Management Studio**:

1. Создайте все таблицы базы данных, ключи, ограничения и связи.
2. Каждая таблица должна иметь ограничение первичного ключа.
3. С помощью ограничений внешнего ключа должны быть заданы все имеющиеся связи между таблицами.
4. В зависимости от условий выданного задания в некоторых таблицах могут быть наложены дополнительные ограничения целостности на столбцы или должны быть разработаны вычисляемые поля.
5. Создайте диаграмму базы данных.
6. Заполните таблицы данными не менее 5 записей в каждой.
7. Создать текстовый отчет, в котором отобразить скриншоты результатов работы в СУБД **SQL Server Management Studio** (окно с базой данных с перечнем всех таблиц, проекты таблиц с перечнем столбцов, окна ограничений внешних ключей (создание), окно с перечнем ключей для каждой таблицы, окна с данными для каждой таблицы, диаграмма базы данных).

ПРАКТИЧЕСКАЯ РАБОТА №4. ВВЕДЕНИЕ В ЯЗЫК SQL. СОЗДАНИЕ ТАБЛИЦ И ОГРАНИЧЕНИЙ НА SQL

4.1. Цель практической работы

Изучение структурированного языка запросов Transact - SQL, являющегося основой системы программирования SQL Server, и приобретение навыков применение инструментальных средств разработки и программирования объектов создаваемых баз данных. Изучить SQL-операторы для работы с таблицами и индексами. Изучить sql-команды для создания, изменения и удаления таблиц. Изучить используемые в SQL Server типы ограничений. Изучить SQL-операторы для работы с ограничениями.

4.2. Исходные данные

Исходными данными является индивидуальное задание и результат предыдущих практических работ.

4.3. Используемые программы

Программа " SQL Server Management Studio " и установленный сервер Microsoft SQL Server .

Выполнение операций над таблицами, индексами и ограничениями с помощью языка SQL

4.4. Теоретические сведения

4.4.1.Основы программирования на Transact - SQL

Система программирования **SQL Server** относится к классу командно-интерпретирующих систем сверхвысокого уровня. Единицами действий системы являются команды, исполняемые в режиме интерпретации сразу же по мере их поступления в сервер. Основой этой системы программирования является проблемно-ориентированный структурированный язык запросов **(Structured Query Language) Transact - SQL**, который расширяется и развивает возможности стандарта **ANSI SQL - 92**.

Transact - SQL включает следующие средства:

1. данные различного типа баз данных и переменных;
2. константы, стандартные и ограниченные идентификаторы;
3. арифметические и логические выражения, включающие следующие операнды: константы, переменные, имена столбцов таблиц, функции, подзапросы и условные выражения, а также выражения, взятые в круглые скобки;
4. SQL - команды для создания, изменения и удаления баз данных и их объектов, а также для определения запросов на ввод, обработку и извлечение данных;
5. управляющие программные структуры, определяющие условия и порядок выполнения команд в заданной последовательности или пакете команд;
6. встроенные (системные) и определяемые пользователем функции;
7. встроенные (системные) и определяемые пользователем хранимые процедуры.

В системе могут храниться, помимо функций и процедур, последовательности (пакеты) команд, которые называются **скриптами**. Если скрипт описывает процесс создания базы данных, или каких-либо ее объектов, то такой скрипт называется **сценарием**. Сценарии позволяют переносить структуру базы данных от одного сервера к другому, а также структуру таблиц и других объектов в различные базы данных. Скрипты хранятся в текстовых файлах.

Функции и хранимые процедуры баз данных позволяют уменьшить объем запросов, передаваемых от клиента к серверу, что повышает общую производительность системы. Наличие исходного кода для этих объектов позволяет упростить сопровождение программных комплексов и внесение изменений в них.

Обычно все бизнес-правила и алгоритмы обработки данных реализуются на сервере баз данных и доступны конечному пользователю в виде набора функций и хранимых процедур, которые и представляют интерфейс обработки данных. Для обеспечения целостности данных, а также в целях безопасности, приложению обычно не предоставляется прямой доступ к данным. Вся работа ведется с помощью указанного интерфейса.

Подобный подход делает весьма простым изменение алгоритмов обработки данных и обеспечивает возможность расширения системы без внесения изменений в само приложение. Достаточно изменить хранимую процедуру на сервере баз данных, и сделанные изменения тотчас станут доступными всем пользователям сети.

В языке Transact - SQL имеются следующие виды констант:

1. **битовые**: 0 и 1;
2. **логические**: FALSE и TRUE;
3. **бинарные в шестнадцатеричном представлении**: 0*9E70DA;
4. символьные: 'ABC'; "ABC" (если QUOTEDIDENTIFIER = OFF); N 'ABC' (Unicode); N "ABC" (Unicode);
5. **целые**: 1; 2; 175;
6. **с фиксированной точкой**: 12.35; - 16.753;
7. **с плавающей точкой**: 1.75E5; 3.84E - 3;
8. **для даты**: " April 15.2003"; "4/15/2003"; "20031207";
9. **для времени**: 14:30; 14:30:20:999; 4am; 4pm;
10. **денежные**: $100;?200; 2.15.

Комментарии в языке бывают двух типов: сточные, начинающиеся с двух символов минуса **-** и блочные, заключаемые символами **/*** и ***/**.

Все объекты базы данных должны иметь имена, которые используются в командах для ссылки на эти объекты. Любой объект базы данных должен быть уникально идентифицирован. Помимо программных имен сервер автоматически генерирует внутренние уникальные имена для идентификации объектов баз данных, например, **PK Table X 014543FA**.

Программные имена задаются идентификаторами двух типов:

1. **стандартными идентификаторами**: **Table X; Key Col;**
2. **ограниченными идентификаторами**: **[My Table]; [Order]; "My Table"; "Order" (если QUOTEDIDENTIFIER = ON).**

Длина идентификатора - от 1 до 128 символов.

Идентификатором не может быть какое-либо зарезервированное ключевое слово языка.

Стандартный идентификатор в качестве первого символа может иметь любую латинскую или русскую букву, знаки #, ##, @, @@ и знак подчеркивания _. Последующими знаками, помимо указанных, могут быть еще и десятичные цифры.

Ограниченные идентификаторы могут включать и другие символы, в том числе зарезервированные слова. В этом случае они должны заключаться в квадратные скобки или двойные кавычки.

В соответствии с идеологией SQL Server каждый объект создается определенным пользователем и принадлежит той или иной базе данных. В свою очередь база данных

расположена на конкретном сервере. Из имен объекта, пользователя, базы данных и сервера создается **полное имя (complete name)** или **полностью определенное имя (full qualified name)**, записываемое в следующем виде:

[[[server.].[database].[owner_name].] objectname

Варианты обращения к объектам базы данных:
A.B.C.D; A.B..D; A..C.D; A..D; B.C.D; B..D; C.D; D

Чтобы сослаться на конкретный столбец таблицы или представления, необходимо в полном имени указать пятый элемент: **A.B.C.E.**

Операторами выражения могут быть унарные (+ и -), бинарные арифметические операторы **(+, -, *, %**), оператор присваивания **(**=), строковая операция конкатенации (+), операторы сравнения **(=, >, <, <=, >=, =, !=** или **<>, !<, !>**), логические операторы (**NOT, AND, OR, ALL, ANY, BETWEEN, EXIST, IN, LIKE, SOME**) и битовые операторы **(&,** |, ^).

Константы, переменные, операнды и выражения используются при записи команд и программирования функций и хранимых процедур, которые, все вместе взятые, составляют основную часть системы программирования SQL Server и определяют ее выразительную мощь. Команды позволяют создавать, модифицировать и удалять базы данных и их объекты, формировать сложные запросы на ввод, обработку и извлечение данных из баз знаний, выполнять функции администрирования и обслуживания баз данных. Функции и хранимые процедуры реализуют разнообразные алгоритмы обработки данных или выполнение служебных функций сервера.

При формировании запросов очень часто используются специальные **логические операторы**, синтаксис которых записывают следующим образом:

1. Выражение **{= | < > | ! = | > | >= | ! >|, = | !<} ALL** подзапрос.

Здесь скалярное выражение вычисляется и сравнивается с каждым значением, возвращаемым подзапросом. Если сравнение дает истину для всех возвращаемых подзапросом значений, то этот оператор возвращает истину.

2. Если вместо **ALL** записать **SOME** или **ANY**, то результатом будет истина, если хотя в одной строке будет выполняться заданное сравнение.

3. Выражение **[NOT] BETWEEN Н Выражение AND В Выражение** возвращает истину, когда значение выражения лежит в диапазоне значений **Н выражения** и **В Выражения** (или не лежит).

4. Оператор **EXISTS** (подзапрос) возвращает значение истина, если подзапрос возвращает хотя бы одну строку.

5. Выражение **[NOT] IN (подзапрос \ выражение [,.. n])** возвращает значение истина, если значение левого выражения совпадает с одним из значений подзапроса или списка значений правых выражений (или не совпадает).

6. Выражение **[NOT] LIKE шаблон [ESCAPE знак]** дает истину, если значение выражения соответствует или не соответствует шаблону, в котором **"%"** означает любое количество произвольных символов, **"_"** - один произвольный символ, **"[символы]"** - один из указанных в скобках, **"[символы]"** - все символы, кроме указанных. Знак после слова **ESCAPE** позволяет указать, что следующий за ним знак шаблона не является управляющим знаком шаблона, т.е. знаком **"%", "_"** и т.д., а представляет обычный знак строки.

Раздел документации сервера T - SQL Help содержит описание каждой команды языка Transact - SQL В и набор примеров их использования. Синтаксис команды

определяется с помощью специального метаязыка, основанного на **нормальных формах Бекуса Наура (БНФ)**.

Ключевые слова определения без всякого изменения переходят в саму команду; метапеременные, написанные курсивом, должны быть заменены программными именами; разделители (запятая, равно, апостроф и т.д.) также переходят в качестве разделителей в команду.

Метасинтаксические знаки имеют следующий смысл:

::= — есть по определению;
| — выбор альтернативы;
[ ] — возможное отсутствие части определения;
{ } — объединение частей определения для выюора или повторения;
[,...n] — повторение предшествующей части 1, 2, ..., n раз с разделителем запятая для этой части (разделитель может быть любой;)
<...> — метапеременная, которая имеет свое определение.

4.4.2. Команды SQL для создания, удаления и изменения таблицами

Часть языка SQL, которая управляет метаданными, называется **Data Definition Language (DDL)**.

К DDL относятся операторы для определения любых содержащихся в базе данных объектов, в том числе и таблиц.

Операторы, определяющие структуру таблиц в MS SQL Server, соответствуют стандарту SQL, и поэтому без изменений будут работать и во многих других СУБД.

Таблицы создаются, изменяются и удаляются соответственно командами **Transact-SQL**:

- **CREATE TABLE,**
- **ALTER TABLE**
- **DROP TABLE.**

При создании новой базы данных сервер автоматически создает **18 системных таблиц** для хранения информации о ее структуре и организации, доступ к которым со стороны пользователя запрещен.

Помимо основных и системных таблиц, которые, как правило, постоянно хранятся в базе данных, можно использовать **временные таблицы** для временного хранения информации, которые автоматически уничтожаются при закрытии соединения с базой данных.

В языке Transact-SQL используются следующие типы данных:

binary(n)	-двоичные данные фиксированной длины до 8000 байт; для n байтов выделяется n+4 байта памяти; значения задаются с помощью 16-ичных чисел 0 x<шестнадцатеричные цифры>; функция DATALENGTH позволяет определить длину поля в байтах; дополнение нулевыми байтами производится справа.
image	- двоичные данные длиной до 2 - 1; место выделяется в виде цепочки страниц.
char(n)	-строковый тип данных фиксированной длины без поддержки Unicode длиной до 8000 байтов; данные зависят от установленной кодовой страницы; если для столбца не задана опция NULL, то строка при необходимости будет дополняться справа пробелами; если эта опция задана, то дополнение пробелами будет иметь место

	при условии ANSI_PADDING=ON, в противном случае пробелы добавляться не будут.
var char(n)	-строковый тип, как и char^^ro не с фиксированной длиной; если ANSI_PADDING=OFF, то будет выполняться удаление конечных пробелов, если ANSI_PADDING=ON, то удаление пробелов производиться не будет.
Nchar(n)	-строковый типкак и char(n), но с поддержкой Unicode, поэтому максимальное количество символов составляет 4000; в этом случае для строковых констант надо задавать впереди букву N: N'ABC'.
Nvarchar(n)	-строковый тип, как varchar(n), но с поддержкой Unicode.
Text	-строковый тип без поддержки Unicode длиной до 2 Гбайт; память выделяется страницами по 8 Кбайт, связываемыми в цепочку; можно использовать встроенные функции: DATALENGTH, PATINDEX, SUBSTRING, TEXTPTR, TEXTVALID, READTEXT, SET TEXTSIZE, UPDATETEXT, WRITETEXT.
Ntext	-строковый тип как и text, но с поддержкой Unicode, поэтому длина строки не более 1 Гбайта.
Int	-целый тип длиной в 4 байта и с диапазоном от -2^{31} до $2^{31}-1$. **Smalling** -целый тип длиной в 2 байта с диапазоном от -2^{15} до $2^{15}-1$. **Tinyint** -целый тип длиной в 1 байт и диапазоном от 0 до 255.
Bigint	-целый тип длиной в 8 байт и с диапазоном от-2^{63} до $2^{63}-1$. **Decimal[(p[,s])]** -десятичный двоично-кодированный тип с p десятичными разрядами, из которых s - дробных; максимальное значение p достигает 38, поэтому диапазон значений составляет от $-(10^{38}-1)$ до $10^{38}-1$.
Numeric[(p[,s])]	-тип, аналогичный типу decimal[(p[,s])].
Float[(n)]	-плавающий (приблизительный) тип длиной в 4 байта и с диапазоном от $-1.79x10^{308}$ до $1.79x10^{308}$; значение n определяет количество бит для хранения мантисы и может принимать значения от 1 до 53.
Real	-плавающий тип, являющийся аналогом float(240).
Datetime	-тип данных для хранения даты (4 первых байта) и времени (4 последних байта) в диапазоне от 1.1.1753 и до 31.12.9999 года; дата хранится в виде смещения относительно базовой даты 1.1.1753, а время является количеством миллисекунд после полуночи; формат для пользователя: MMM DD YYYY hh:
Smalldatetime	-тип данных для хранения даты (первых 2 байта) и времени (последние 2 байта) в диапазоне от 1.1.1900г. до 6.6.2079г., время задается с точностью до минуты. **Money** -тип данных для хранения больших денежных величин с точностью до 4 знаков после запятой в диапазоне от -922 337 203 685 477.5808 до +922 337 203 685 477.5807; 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.
Timestamp	-тип данных временный штамп для учета числа изменений данных в записи (версий строки row version); значение timestamp уникально в пределах базы данных и позволяет идентифицировать конкретное значение записи; тип аналогичен binary(8), если хранение NULL не

	разрешено и varbinary(8), если разрешено. **Uniqueidentifier** -тип данных для хранения глобальных уникальных идентификаторов длиной в 16 байт, генерируемых функций NEWID и используемых для идентификации строк (записей); при генерации используется номер сетевой карты компьютера и текущее время.
Sysname	-тип данных для хранения имен объектов базы данных; аналог nvarchar (128). **Sql_variant** -вариантный тип данных для хранения данных любого типа, кроме text, ntext,image,timestamp; для получения информации о природе хранимых данных используется функция SQL_VARIANT_PROPERTY().
Table	-тип таблицы для временного хранения наборов данных с использованием переменных.

Инструкция CREATE TABLE (Transact-SQL)

Для создания таблиц используется оператор "**CREATE TABLE**", который приводит к созданию **пустой** таблицы без строк. При создании таблиц задается имя таблицы, описание набора столбцов с их именами, типами и размерами, а также ограничения на хранящуюся в таблице информацию.

Имена таблиц в пределах базы данных должны быть уникальны.

Каждый столбец в таблице должен иметь имя, уникальное в пределах таблицы, а также либо тип данных, ограничения целостности, либо выражение для вычисления значения столбца.

Общий синтаксис

```
CREATE TABLE
   [ database_name . [ schema_name ] . | schema_name . ] table_name
   [ AS FileTable ]
   ( { <column_definition> | <computed_column_definition>
      | <column_set_definition> | [ <table_constraint> ] [ ,...n ] } )
   [ ON { partition_scheme_name ( partition_column_name ) | filegroup
      | "default" } ]
   [ { TEXTIMAGE_ON { filegroup | "default" } ]
   [ FILESTREAM_ON { partition_scheme_name | filegroup
      | "default" } ]
   [ WITH ( <table_option> [ ,...n ] ) ]
[ ; ]

<column_definition> ::=
column_name <data_type>
   [ FILESTREAM ]
   [ COLLATE collation_name ]
   [ NULL | NOT NULL ]
   [
      [ CONSTRAINT constraint_name ] DEFAULT constant_expression ]
    | [ IDENTITY [ ( seed ,increment ) ] [ NOT FOR REPLICATION ]
   ]
   [ ROWGUIDCOL ] [ <column_constraint> [ ...n ] ]
   [ SPARSE ]
```

```
<data type> ::=
[ type_schema_name . ] type_name
   [ ( precision [ , scale ] | max |
      [ { CONTENT | DOCUMENT } ] xml_schema_collection ) ]

<column_constraint> ::=
[ CONSTRAINT constraint_name ]
{    { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ]
      [
         WITH FILLFACTOR = fillfactor
       | WITH ( < index_option > [ , ...n ] )
      ]
      [ ON { partition_scheme_name ( partition_column_name )
         | filegroup | "default" } ]
  | [ FOREIGN KEY ]
      REFERENCES [ schema_name . ] referenced_table_name [ ( ref_column ) ]
      [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
      [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
      [ NOT FOR REPLICATION ]
  | CHECK [ NOT FOR REPLICATION ] ( logical_expression )
}

<computed_column_definition> ::=
column_name AS computed_column_expression
[ PERSISTED [ NOT NULL ] ]
[
   [ CONSTRAINT constraint_name ]
   { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ]
      [
         WITH FILLFACTOR = fillfactor
       | WITH ( <index_option> [ , ...n ] )
      ]
   | [ FOREIGN KEY ]
      REFERENCES referenced_table_name [ ( ref_column ) ]
      [ ON DELETE { NO ACTION | CASCADE } ]
      [ ON UPDATE { NO ACTION } ]
      [ NOT FOR REPLICATION ]
   | CHECK [ NOT FOR REPLICATION ] ( logical_expression )
   [ ON { partition_scheme_name ( partition_column_name )
      | filegroup | "default" } ]
]

<column_set_definition> ::=
column_set_name XML COLUMN_SET FOR ALL_SPARSE_COLUMNS

< table_constraint > ::=
[ CONSTRAINT constraint_name ]
{
```

```
    { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ]
      (column [ ASC | DESC ] [ ,...n ] )
      [
        WITH FILLFACTOR = fillfactor
       |WITH ( <index_option> [ , ...n ] )
      ]
      [ ON { partition_scheme_name (partition_column_name)
        | filegroup | "default" } ]
    | FOREIGN KEY
      ( column [ ,...n ] )
      REFERENCES referenced_table_name [ ( ref_column [ ,...n ] ) ]
      [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
      [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
      [ NOT FOR REPLICATION ]
    | CHECK [ NOT FOR REPLICATION ] ( logical_expression )
  }
  <table_option> ::=
  {
    [DATA_COMPRESSION = { NONE | ROW | PAGE }
     [ ON PARTITIONS ( { <partition_number_expression> | <range> }
     [ , ...n ] ) ]]
    [ FILETABLE_DIRECTORY = <directory_name> ]
    [ FILETABLE_COLLATE_FILENAME = { <collation_name> | database_default
} ]
    [ FILETABLE_PRIMARY_KEY_CONSTRAINT_NAME = <constraint_name> ]
    [ FILETABLE_STREAMID_UNIQUE_CONSTRAINT_NAME =
<constraint_name> ]
    [ FILETABLE_FULLPATH_UNIQUE_CONSTRAINT_NAME =
<constraint_name> ]
  }

  <index_option> ::=
  {
    PAD_INDEX = { ON | OFF }
   | FILLFACTOR = fillfactor
   | IGNORE_DUP_KEY = { ON | OFF }
   | STATISTICS_NORECOMPUTE = { ON | OFF }
   | ALLOW_ROW_LOCKS = { ON | OFF}
   | ALLOW_PAGE_LOCKS ={ ON | OFF}
   | DATA_COMPRESSION = { NONE | ROW | PAGE }
      [ ON PARTITIONS ( { <partition_number_expression> | <range> }
      [ , ...n ] ) ]
  }
  <range> ::=
  <partition_number_expression> TO <partition_number_expression>
```

Рассмотрим и расшируем назначение ключевых слов и аргументов команды **CREATE TABLE**.

Аргументы:

database_name

Имя базы данных, в которой создается таблица. Параметр **database_name** должен указывать имя существующей базы данных. Если аргумент **database_name** не указан, по умолчанию список стоп-слов создается в текущей базе данных. Имя входа для текущего соединения должно быть связано с идентификатором пользователя, существующего в базе данных, указанной аргументом **database_name**, а этот пользователь должен обладать разрешениями **CREATE TABLE**.

schema_name

Имя схемы, которой принадлежит новая таблица.

table_name

Имя новой таблицы. Имена таблиц должны соответствовать правилам для идентификаторов. Аргумент **table_name** может состоять не более чем из 128 символов, за исключением имен локальных временных таблиц (имена с одним префиксом номера #), длина которых не должна превышать 116 символов.

AS FileTable

Создает новую таблицу FileTable. Нет необходимости указывать столбцы, так как таблица FileTable имеет фиксированное схему. Дополнительные сведения о таблицах FileTable см. в разделе Таблицы FileTable (SQL Server).

column_name

Имя столбца в таблице. Имена столбцов должны соответствовать правилам именования идентификаторов и быть уникальными в рамках таблицы. Аргумент **column_name** может иметь длину не более 128 символов. Аргумент **column_name** может быть опущен для столбцов, создаваемых с типом данных timestamp. Если аргумент **column_name** не указан, столбцу типа timestamp по умолчанию присваивается имя timestamp.

computed_column_expression

Выражение, определяющее значение вычисляемого столбца. Вычисляемый столбец представляет собой виртуальный столбец, физически не хранящийся в таблице, если для него не установлен признак **PERSISTED**. Значение столбца вычисляется на основе выражения, использующего другие столбцы той же таблицы. Например, определение вычисляемого столбца может быть следующим:

cost AS price * qty.

Выражение может быть именем невычисляемого столбца, константой, функцией, переменной или любой их комбинацией, соединенной одним или несколькими операторами. Выражение не может быть вложенным запросом или содержать псевдонимы типов данных.

Вычисляемые столбцы могут использоваться в списках выбора, предложениях **WHERE, ORDER BY** и в любых других местах, в которых могут использоваться обычные выражения, за исключением следующих случаев.

- Вычисляемый столбец нельзя использовать ни в качестве определения ограничения **DEFAULT** или **FOREIGN KEY**, ни вместе с определением ограничения **NOT NULL**. Однако вычисляемый столбец может использоваться в качестве ключевого столбца индекса или части какого-либо ограничения **PRIMARY KEY** или **UNIQUE,** если значение этого вычисляемого столбца определяется детерминистическим выражением и тип данных результата разрешен в столбцах индекса.

- Вычисляемый столбец не может быть целевым столбцом инструкций **INSERT** или **UPDATE**. Примечание

Каждая строка таблицы может содержать различные значения столбцов, задействованных в вычисляемом столбце; таким образом, значение вычисляемого столбца не будет одним и тем же в каждой строке.

PERSISTED - указывает, что компонент Компонент SQL Server Database Engine будет физически хранить вычисляемые значения в таблице и обновлять их при изменении любого столбца, от которого зависит вычисляемый столбец. Указание **PERSISTED** для вычисляемого столбца позволяет создать индекс по вычисляемому столбцу, который будет детерминистическим, но неточным.

ON { <partition_scheme> | filegroup | "default" }

Указывает схему секционирования или файловую группу, в которой хранится таблица. Если аргумент **<partition_scheme>** указан, таблица будет разбита на секции, хранимые в одной или нескольких файловых группах, указанных аргументом <**partition_scheme**>. Если указан аргумент **filegroup**, таблица сохраняется в файловой группе с таким именем. Это должна быть существующая файловая группа в базе данных. Если указано значение "**default**" или параметр **ON** не определен вообще, таблица сохраняется в файловой группе по умолчанию. Механизм хранения таблицы, указанный в инструкции **CREATE TABLE**, изменить в дальнейшем невозможно.

TEXTIMAGE_ON { filegroup| "default" }

Указывают, что столбцы типов **text, ntext, image, xml, varchar(max), nvarchar(max), varbinary(max),** а также определяемых пользователем типов данных **CLR** (включая **geometry** и **geography**) хранятся в указанной файловой группе.

FILESTREAM_ON { partition_scheme_name | filegroup | "default" }

Задает файловую группу для данных **FILESTREAM.**

Если таблица содержит данные **FILESTREAM** и является секционированной, необходимо включить предложение **FILESTREAM_ON** и указать схему секционирования файловых групп файлового потока. В этой схеме секционирования должны использоваться те же функции и столбцы секционирования, что и в схеме секционирования для таблицы; в противном случае возникает ошибка.

[type_schema_name.] type_name

Указывает тип данных столбца и схему, к которой он принадлежит. Тип данных может быть одним из следующих.

- Системный тип данных.
- Псевдонимы типа на основе системного типа данных SQL Server.
- Определяемый пользователем тип данных **CLR**. Прежде чем определяемые пользователем типы данных **CLR** можно будет использовать в определении таблицы, их нужно создать с помощью инструкции **CREATE TYPE**. Для создания столбца с определяемым пользователем типом данных **CLR** требуется разрешение **REFERENCES** на этот тип.

precision

Точность указанного типа данных.

scale

Масштаб указанного типа данных.

CONTENT

Указывает, что каждый экземпляр типа данных xml в столбце **column_name** может содержать несколько элементов верхнего уровня. Аргумент **CONTENT** применим только к данным типа xml.

DOCUMENT

Указывает, что каждый экземпляр типа данных xml в столбце **column_name** может содержать только один элемент верхнего уровня. Аргумент DOCUMENT применим только к данным типа xml.

DEFAULT

Указывает значение, присваиваемое столбцу в случае отсутствия явно заданного значения при вставке. Определения **DEFAULT** могут применяться к любым столбцам, кроме имеющих тип timestamp или обладающих свойством IDENTITY. Определения DEFAULT удаляются, когда таблица удаляется из памяти. В качестве значения по умолчанию могут использоваться только константы (например, символьные строки), скалярные функции (системные, определяемые пользователем или функции CLR) или значение NULL.

constant_expression

Константа, значение NULL или системная функция, используемая в качестве значения столбца по умолчанию.

IDENTITY

Указывает, что новый столбец является столбцом идентификаторов. При добавлении в таблицу новой строки компонент Компонент Database Engine формирует для этого столбца уникальное последовательное значение. Столбцы идентификаторов обычно используются с ограничением **PRIMARY KEY** для поддержания уникальности идентификаторов строк в таблице.

Свойство **IDENTITY** присвоено столбцам типа **tinyint, smallint, int, bigint, decimal(p,0)** или **numeric(p,0).** Для каждой таблицы можно создать только один столбец идентификаторов. Ограниченные значения по умолчанию и ограничения DEFAULT не могут использоваться в столбце идентификаторов.

Необходимо указать как начальное значение, так и приращение, или же не указывать ничего. Если ничего не указано, применяется значение по умолчанию (**1,1**).

seed

Значение, используемое для самой первой строки, загружаемой в таблицу.

increment

Значение приращения, добавляемое к значению идентификатора предыдущей загруженной строки.

NOT FOR REPLICATION

В инструкции **CREATE TABLE** предложение **NOT FOR REPLICATION** может указываться для свойства **IDENTITY**, а также ограничений **FOREIGN KEY** и **CHECK.** Если это предложение указано для свойства **IDENTITY**, значения в столбцах идентификаторов не приращиваются, если вставку выполняют агенты репликации. Если ограничение сопровождается этим предложением, оно не выполняется, когда агенты репликации выполняют операции вставки, обновления или удаления.

ROWGUIDCOL

Указывает, что новый столбец является столбцом идентификаторов GUID строки. Только один столбец типа uniqueidentifier в таблице может быть назначен в качестве столбца ROWGUIDCOL. Применение свойства ROWGUIDCOL позволяет ссылаться на столбец с помощью ключевого слова $ROWGUID. Свойство ROWGUIDCOL может быть присвоено только столбцу типа uniqueidentifier. Ключевым словом ROWGUIDCOL нельзя обозначать столбцы определяемых пользователем типов данных.

SPARSE

Указывает, что столбец является разреженным столбцом. Хранилище разреженных столбцов оптимизируется для значений NULL. Для разреженных столбцов нельзя указать параметр NOT NULL. Дополнительные ограничения и сведения о разреженных столбцах см. в разделе Использование разреженных столбцов.

FILESTREAM

Допустимо только для столбцов типа varbinary(max). Указывает хранилище FILESTREAM для данных BLOB типа varbinary(max).

COLLATE collation_name

Задает параметры сортировки для столбца. Могут использоваться параметры сортировки Windows или параметры сортировки SQL. Параметр **collation_name** применим только к столбцам типов данных **char, varchar, text, nchar, nvarchar** и **ntext**. Если этот аргумент не указан, столбцу назначаются либо параметры сортировки определяемого пользователем типа, если столбец принадлежит к определяемому пользователем типу данных, либо установленные по умолчанию параметры сортировки для базы данных.

CONSTRAINT

Необязательное ключевое слово, указывающее на начало определения ограничения **PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY** или **CHECK**.

constraint_name

Имя ограничения. Имена ограничений должны быть уникальными в пределах схемы, к которой принадлежит таблица.

NULL | NOT NULL

Определяет, допустимы ли для столбца значения NULL.

PRIMARY KEY

Ограничение, которое обеспечивает целостность сущностей для указанного столбца или столбцов с помощью уникального индекса. Можно создать только одно ограничение **PRIMARY KEY** для таблицы.

UNIQUE

Ограничение, которое обеспечивает целостность сущностей для указанного столбца или столбцов с помощью уникального индекса. В таблице может быть несколько ограничений **UNIQUE**.

FOREIGN KEY REFERENCES

Ограничение, которое обеспечивает ссылочную целостность данных в этом столбце или столбцах. Ограничения **FOREIGN KEY** требуют, чтобы каждое значение в столбце существовало в соответствующем связанном столбце или столбцах в связанной таблице.

Ограничения **FOREIGN KEY** могут ссылаться только на столбцы, являющиеся ограничениями **PRIMARY KEY** или **UNIQUE** в связанной таблице или на столбцы, на

которые имеются ссылки в индексе UNIQUE INDEX связанной таблицы. Внешние ключи в вычисляемых столбцах должны быть также помечены как PERSISTED.

[schema_name.] referenced_table_name]

Имя таблицы, на которую ссылается ограничение **FOREIGN KEY**, и схема, к которой она принадлежит.

(ref_column [,... n])

Столбец или список столбцов из таблицы, на которую ссылается ограничение **FOREIGN KEY**.

ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }

Определяет операцию, которая производится над строками создаваемой таблицы, если эти строки имеют ссылочную связь, а строка, на которую имеются ссылки, удаляется из родительской таблицы. Параметр по умолчанию — NO ACTION.

NO ACTION

Компонент Компонент Database Engine формирует ошибку, и выполняется откат операции удаления строки из родительской таблицы.

CASCADE

Если из родительской таблицы удаляется строка, соответствующие ей строки удаляются и из ссылающейся таблицы.

SET NULL

Все значения, составляющие внешний ключ, при удалении соответствующей строки родительской таблицы устанавливаются в NULL. Для выполнения этого ограничения внешние ключевые столбцы должны допускать значения NULL.

SET DEFAULT

Все значения, составляющие внешний ключ, при удалении соответствующей строки родительской таблицы устанавливаются в значение по умолчанию. Для выполнения этого ограничения все внешние ключевые столбцы должны иметь определения по умолчанию. Если столбец допускает значения NULL и значение по умолчанию явно не определено, значением столбца по умолчанию становится NULL.

ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }

Указывает, какое действие совершается над строками в изменяемой таблице, когда эти строки имеют ссылочную связь и строка родительской таблицы, на которую указывает ссылка, обновляется. Параметр по умолчанию — NO ACTION.

NO ACTION

Компонент Компонент Database Engine возвращает ошибку, а обновление строки родительской таблицы откатывается.

CASCADE

Соответствующие строки обновляются в ссылающейся таблице, если эта строка обновляется в родительской таблице.

SET NULL

Всем значениям, составляющим внешний ключ, присваивается значение NULL, когда обновляется соответствующая строка в родительской таблице. Для выполнения этого ограничения внешние ключевые столбцы должны допускать значения NULL.

SET DEFAULT

Всем значениям, составляющим внешний ключ, присваивается их значение по умолчанию, когда обновляется соответствующая строка в родительской таблице.

CHECK

Ограничение, обеспечивающее целостность домена путем ограничения возможных значений, которые могут быть введены в столбец или столбцы. Ограничения **CHECK** в вычисляемых столбцах должны быть также помечены как **PERSISTED.**

logical_expression

Логическое выражение, возвращающее значения TRUE или FALSE..

column

Столбец или список столбцов (в скобках), используемый в ограничениях таблицы для указания столбцов, используемых в определении ограничения.

[ASC | DESC]

Указывает порядок сортировки столбца или столбцов, участвующих в ограничениях таблицы. Значение по умолчанию — ASC.

partition_scheme_name

Имя схемы секционирования, определяющей файловые группы, которым сопоставляются секции секционированной таблицы. Эта схема секционирования должна существовать в базе данных.

[partition_column_name.]

Указывает столбец, по которому будет секционирована таблица. Столбец должен соответствовать по типу данных, длине и точности столбцу, указанному в функции секционирования, используемой аргументом partition_scheme_name. Вычисляемый столбец, участвующий в функции секционирования, должен быть явно обозначен ключевым словом PERSISTED. Важно!

WITH FILLFACTOR =fillfactor

Указывает, насколько плотно компонент Компонент Database Engine должен заполнять каждую страницу индекса, используемую для хранения данных индекса.

column_set_name XML COLUMN_SET FOR ALL_SPARSE_COLUMNS

Имя набора столбцов. Набор столбцов представляет собой нетипизированное XML-представление, в котором все разреженные столбцы таблицы объединены в структурированные выходные данные.

< table_option> ::=

Указывает один или более параметров таблицы.

DATA_COMPRESSION

Задает режим сжатия данных для указанной таблицы, номера секции или диапазона секций. Ниже приведены доступные параметры.

ON PARTITIONS ({ <partition_number_expression> | <range> } [,...n])

Указывает секции, к которым применяется параметр DATA_COMPRESSION. Если таблица не секционирована, аргумент ON PARTITIONS приведет к формированию ошибки.

4.5. Упрощенный синтаксис оператора создания таблицы:

<определение_таблицы> ::=

CREATE TABLE [имя_базы_данных.[владелец]. | владелец.]
имя_таблицы
(<элемент_таблицы>[,...n])

где
<элемент_таблицы> ::=

{<определение_столбца>}
| <имя_столбца> AS <выражение>
|>ограничение_таблицы<

Обычно владельцем таблицы (**dbo**) является тот, кто ее создал.
<Выражение> задает значение для вычисляемого столбца.

<определение_столбца> ::=

{ имя_столбца <тип_данных>}
[[DEFAULT <выражение>]
| [IDENTITY (начало, шаг) [NOT FOR REPLICATION]]]
[ROWGUIDCOL][<ограничение_столбца>][...n]]

В определении столбца обратим внимание на параметр **IDENTITY**, который указывает, что соответствующий столбец будет **столбцом-счетчиком**. Для таблицы может быть определен только один столбец с таким свойством. Можно дополнительно указать начальное значение и шаг приращения. Если эти значения не указываются, то по умолчанию они оба равны 1. Если с ключевым словом **IDENTITY** указано **NOT FOR REPLICATION**, то сервер не будет выполнять автоматического генерирования значений для этого столбца, а разрешит вставку в столбец произвольных значений.

В качестве **ограничений** используются **ограничения столбца** и **ограничения таблицы**. Различие между ними в том, что ограничение столбца применяется только к определенному полю, а ограничение таблицы - к группам из одного или более полей. Различные типы ограничений рассмотрим позже.

Пример создания таблицы без ограничений.

Для выполнения данных запросов предварительно откройте **среду Micrpsoft SQL Server Managmant Studio**, выполните соединение с сервером и откройте базу данных **Educator**. Нажмите на панели инструментов команду **Создать запрос**.

Пример 4.1. Создание родительской таблицы Товар без ограничений.

```
use Educator
CREATE TABLE Товар
(КодТовара INT IDENTITY(1,1),
 Название  VARCHAR(50)      ,
 Цена      MONEY            ,
 Тип       VARCHAR(50)      ,
 Сорт      VARCHAR(50)      ,
 Город     VARCHAR(50)      ,
 Остаток   INT    );
```

Выполните sql-код. Обновите базу данных и просмотрите созданную таблицу. Сохраните sql-запрос под именем **Пример1.sql** в папке **ФИО_студента/Лаб4**.

Автоматическая генерация значения столбца **КодТовара** достигается за счет использования свойства **IDENTITY**, по умолчанию начальное значение, генерируемое с помощью **IDENTITY** равно 1, так же как и его приращение. Таким образом, следующее значение будет равно 2. Значения в **IDENTITY**-столбцах обязательно последовательные, то есть если приращение положительное, то следующее значение всегда больше предыдущего, если приращение отрицательное, то – всегда меньше. Приращение и начальное значение могут быть заданы, однако этот механизм чрезвычайно редко используется в реальных проектах.

4.6. Создание ограничений

В качестве ограничений используются ограничения столбца и ограничения таблицы. Различие между ними в том, что ограничение столбца применяется только к определенному полю, а ограничение таблицы - к группам из одного или более полей.

<ограничение_столбца>::=

```
[ CONSTRAINT имя_ограничения ]
{ [ NULL | NOT NULL ]
| [ {PRIMARY KEY | UNIQUE }
[ CLUSTERED | NONCLUSTERED ]
[ WITH FILLFACTOR=фактор_заполнения ]
[ ON {имя_группы_файлов | DEFAULT } ] ] ]
| [ [ FOREIGN KEY ]
REFERENCES имя_род_таблицы
   [(имя_столбца_род_таблицы) ]
[ ON DELETE { CASCADE | NO ACTION } ]
[ ON UPDATE { CASCADE | NO ACTION } ]
[ NOT FOR REPLICATION ]]
| CHECK [ NOT FOR REPLICATION](<лог_выражение>) }
```

<ограничение_таблицы>::=

```
[CONSTRAINT имя_ограничения ]
{ [ {PRIMARY KEY | UNIQUE }
   [ CLUSTERED | NONCLUSTERED ]
{(имя_столбца [ASC | DESC][,...n])}
[WITH FILLFACTOR=фактор_заполнения ]
[ON {имя_группы_файлов | DEFAULT } ]]
|FOREIGN KEY[(имя_столбца [,...n])]
REFERENCES имя_род_таблицы
   [(имя_столбца_род_таблицы [,...n])]
[ ON DELETE { CASCADE | NO ACTION } ]
[ ON UPDATE { CASCADE | NO ACTION } ]
| NOT  FOR REPLICATION ]
| CHECK [ NOT FOR REPLICATION ] (лог_выражение) }
```

Рассмотрим отдельные параметры представленных конструкций, связанные с ограничениями целостности данных. Ограничения целостности имеют приоритет над триггерами, правилами и значениями по умолчанию. К ограничениям целостности относятся ограничение первичного ключа **PRIMARY KEY**, ограничение внешнего ключа **FOREIGN KEY**, ограничение уникальности **UNIQUE**, ограничение значения **NULL**, ограничение на проверку **CHECK**.

4.6.1. Ограничение первичного ключа (PRIMARY KEY)

Таблица обычно имеет столбец или комбинацию столбцов, значения которых уникально идентифицируют каждую строку в таблице. Этот столбец (или столбцы) называется **первичным ключом таблицы** и нужен для обеспечения ее **целостности.** Если в первичный ключ входит более одного столбца, то значения в пределах одного столбца могут дублироваться, но любая комбинация значений всех столбцов первичного ключа должна быть уникальна.

При создании первичного ключа SQL Server автоматически создает **уникальный индекс** для столбцов, входящих в первичный ключ. Он ускоряет доступ к данным этих столбцов при использовании первичного ключа в запросах.

Таблица может иметь только **одно ограничение PRIMARY KEY**, причем ни один из включенных в первичный ключ столбцов не может принимать значение **NULL**. При попытке использовать в качестве первичного ключа столбец (или группу столбцов), для которого ограничения первичного ключа не выполняются, первичный ключ создан не будет, а система выдаст сообщение об ошибке.

Поскольку ограничение **PRIMARY KEY** гарантирует уникальность данных, оно часто определяется для столбцов-счетчиков. Создание ограничения целостности **PRIMARY KEY** возможно как при создании, так и при изменении таблицы. Одним из назначений первичного ключа является обеспечение ссылочной целостности данных нескольких таблиц. Естественно, это может быть реализовано только при определении соответствующих внешних ключей в других таблицах.

Пример 2. Создание таблицы Товар с ограничением первичного ключа.

```
CREATE TABLE Товар
(КодТовара INT IDENTITY(1,1) PRIMARY KEY,
 Название  VARCHAR(50)     ,
 Цена      MONEY           ,
 Тип       VARCHAR(50)     ,
 Сорт      VARCHAR(50)     ,
 Город     VARCHAR(50)     ,
 Остаток   INT    );
```

Примечание. Прежде чем выполнять sql-код удалите ранее созданную таблицу **Товар** из базы данных и обновите ее.

Выполните sql-код. Обновите базу данных и просмотрите созданную таблицу. Сохраните sql-запрос под именем **Пример2.sql** в папке **ФИО_студента/Лаб4**.

Первичные ключи более чем одного поля

Ограничение **PRIMARY KEY** может также быть применено для многочисленных полей, составляющих уникальную комбинацию значений. Предположим что ваш первичный ключ - это имя, и вы имеете первое имя и последнее

имя сохраненными в двух различных полях (так что вы можете организовывать данные с помощью любого из них). Очевидно, что ни первое ни последнее имя нельзя заставить быть уникальным самостоятельно, но мы можем каждую из этих двух комбинаций сделать уникальной.

Мы можем применить ограничение таблицы **PRIMARY KEY** для пар:

Пример 3. Создание таблицы Сотрудники с ограничением первичного ключа.

```
CREATE TABLE  Сотрудники
( Фамилия       char (10),
  Имя           char (10) ,
  Город         char (10),
  PRIMARY KEY  ( Фамилия, Имя ));
```

Выполните sql-код. Обновите базу данных и просмотрите созданную таблицу. Сохраните sql-запрос под именем **Пример3.sql** в папке **ФИО_студента/Лаб4**.

Одна проблема в этом подходе та, что мы можем вынудить появление уникальности - например, введя **Иванов Андрей** и **Иванов А.** Это может ввести в заблуждение, потому что ваши служащие могут не знать кто из них кто.

Обычно более надежный способ чтобы определять числовое поле которое могло бы отличать одну строку от другой, это иметь первичный ключ, и применять ограничение **UNIQUE** для двух имен полей.

4.6.2. Использование ограничений для исключения пустых(NULL) указателей

Вы можете использовать команду **CREATE TABLE** чтобы предохранить поле от разрешения в нем пустых (**NULL**) указателей с помощью ограничения **NOT NULL**. Это ограничение накладывается только для разнообразных столбцов.

NULL - это специальное обозначение которое отмечает поле как пустое. **NULL** может быть полезен, когда имеются случаи, когда вы хотите быть от них гарантированы. Очевидно, что первичные ключи никогда не должны быть пустыми , поскольку это будет подрывать их функциональные возможности. Кроме того, такие поля как имена, требуют в большинстве случаев, определенных значений. Например, вы вероятно захотите иметь информацию о **должности** занимаемым каждым сотрудником в таблице **Сотрудники**.

Если вы поместите ключевые слова **NOT NULL** сразу после типа данных (включая размер) столбца, любая попытка поместить значение **NULL** в это поле будет отклонена. В противном случае, SQL понимает, что **NULL** раз решен.

Например, давайте улучшим наше определение таблицы Сотрудники, не позволяя помещать NULL значения в столбец **Должность** :

Пример 4. Создание таблицы Сотрудники с ограничением пустых значений.

```
CREATE TABLE  Сотрудники
( Фамилия       char (10),
  Имя           char (10) ,
  Город         char (10),
…….Должность ……char (10) NOT NULL,
  PRIMARY KEY  ( Фамилия, Имя ));
```

Выполните sql-код. Обновите базу данных и просмотрите созданную таблицу. Сохраните sql-запрос под именем **Пример4.sql** в папке **ФИО_студента/Лаб4**.

Важно помнить, что любому столбцу с ограничением **NOT NULL** должно быть установлено значение в каждом предложении **INSERT** воздействующем на таблицу. При отсутствии **NULL**, SQL может не иметь значений для установки в эти столбцы, если конечно значение по умолчанию, описанное ра нее в этой главе, уже не было назначено.

Если ваша система поддерживает использование **ALTER TABLE** чтобы добавлять новые столбцы к уже существующей таблице, вы можете вероятно помещать ограничение столбцов, типа **NOT NULL**, для этих новых столбцов. Однако, если вы предписываете новому столбцу значение **NOT NULL**, текущая таблица должна быть **пустой**!!!!.

4.6.2. Использование ограничений для уникальности значений

В предыдущей лаб.работе мы обсудили использование уникальных индексов чтобы зас тавить поля иметь различные значения для каждой строки. Эта практика - осталась с прежних времен, когда SQL поддерживал ограничение **UNIQUE.**

Уникальность - это свойство данных в таблице, и поэтому его более логично назвать как ограничение этих данных, а не просто как свойство логического отличия, связывающее объект данных (индекс).

Несомненно, уникальные индексы - один из самых простых и наиболее эффективных методов предписания уникальности. По этой причине, некото рые реализации ограничения **UNIQUE** используют уникальные индексы; то-есть они создают индекс не сообщая вам об этом. Остается фактом, что вероятность беспорядка в базе данных достаточно мала, если вы предписываете уникальность вместе с ограничением.

Уникальность как ограничение столбца

Время от времени, вы хотите убедиться, что все значения введеные в столбец отличаются друг от друга. Например, первичные ключи достаточно ясно это показывают. Если вы помещаете ограничение столбца **UNIQUE** в поле при создании таблицы, база данных отклонит любую попытку ввода в это поле для одной из строк, значения, которое уже представлено в другой строке. Это ограничение может применяться только к полям которые были обьявлены как непустые (**NOT NULL**), так как не имеет смысла позволить одной строке таблицы иметь значение **NULL**, а затем исключать другие строки с **NULL** значениями как дубликаты. Имеется дальнейшее усовершенствование нашей команды создания таблицы Сотрудники :

Пример 5. Создание таблицы Сотрудники с ограничением уникальности.

```
CREATE TABLE  Сотрудники
( Фамилия       char (10) NOT NULL UNIQUE,
  Имя           char (10) NOT NULL UNIQUE,
  Город         char (10),
…….Должность ……char (10) NOT NULL,
  PRIMARY KEY  ( Фамилия, Имя ));
```

Выполните sql-код. Обновите базу данных и просмотрите созданную таблицу. Сохраните sql-запрос под именем **Пример5.sql** в папке **ФИО_студента/Лаб4**.

Когда вы обьявляете поля **Фамилия** уникальным, убедитесь, что в вашей базе данных не будет двух Ивановых или Петровых. В то же время это не так уж необходимо с функциональной точки зрения - потому что поле **Имя** в качестве первичного ключа, все равно обеспечит отличие этих двух строк - что проще для людей использующих данные в таблицах, чем помнить, что эти Ивановы не идентичны.

Столбцы (не первичные ключи) чьи значения требуют уникальности, называются **ключами-кандидатами** или **уникальными ключами.**

4.6.3. Ограничение по умолчанию (DEFAULT)

Столбцу может быть присвоено значение по умолчанию. Оно будет актуальным в том случае, если пользователь не введет в столбец никакого иного значения.

Отдельно необходимо отметить пользу от использования значений по умолчанию при добавлении нового столбца в таблицу. Если для добавляемого столбца не разрешено хранение значений **NULL** и не определено значение по умолчанию, то операция добавления столбца закончится неудачей.

Когда вы вставляете строку в таблицу без указания значений в ней для каждого поля, SQL должен иметь значение по умолчанию для включения его в определенное поле, или же команда будет отклонена. Наиболее общим значением по умолчанию является - **NULL**. Это - значение по умолчанию для любого столбца, которому не было дано ограничение **NOT NULL** или который имел другое назначение по умолчанию.

Значение **DEFAULT(ПО УМОЛЧАНИЮ)** указывается в команде **CREATE TABLE** тем же способом что и ограничение столбца, хотя, с технической точки зрения, значение **DEFAULT** не ограничительного свойства - оно не ограничивает значения которые вы можете вводить, а просто определяет, что может случиться если вы не введете любое из них.

Предположим, что вы работаете в г. Москва и подавляющее большинство ваших сотрудников живут в этом городе. Вы можете указать г. Москва в качестве значения поля **Город**, по умолчанию, для вашей таблицы **Сотрудников**:

Пример 7. Создание таблицы Сотрудники с значением по умолчанию.

```
CREATE TABLE  Сотрудники
( Фамилия       char (10) NOT NULL UNIQUE,
  Имя           char (10) NOT NULL UNIQUE,
  Город         char (10) DEFAULT  'Москва',
…….Должность ……char (10) NOT NULL,
  PRIMARY KEY  ( Фамилия, Имя ));
```

Выполните sql-код. Обновите базу данных и просмотрите созданную таблицу. Сохраните sql-запрос под именем **Пример7.sql** в папке **ФИО_студента/Лаб4**.

Конечно, вводить значение Москва в таблицу каждый раз когда назна чается новый сотрудник, не такая уж необходимость, и можно просто пре небречь им (не вводя его) даже если оно должно иметь некоторое зна чение.

Другой способ использовать значение по умолчанию - это использовать его как альтернативу для **NULL**. Так как **NULL** (фактически) неверен при любом сравнении, ином чем **IS NULL**, он может быть исключен с помощью большинства предикатов.

Иногда, вам нужно видеть пустые значения ваших полей не обрабатывая их каким-то определенным образом. Вы можете уста-

новить значение по умолчанию, типа нуль или пробел, которые функцио нально меньше по значению чем просто не установленное значение - пус тое значение(**NULL**). Различие между ними и обычным **NULL** в том, что SQL будет обрабатывать их также как и любое другое значение.

4.6.4. Ограничение проверочное (CHECK)

Данное ограничение используется для проверки допустимости данных, вводимых в конкретный столбец таблицы, т.е. ограничение **CHECK** обеспечивает еще один уровень защиты данных.

Ограничения целостности **CHECK** задают диапазон возможных значений для столбца или столбцов. В основе ограничений целостности **CHECK** лежит использование логических выражений.

Допускается применение нескольких ограничений **CHECK** к одному и тому же столбцу. В этом случае они будут применимы в той последовательности, в которой происходило их создание. Возможно применение одного и того же ограничения к разным столбцам и использование в логических выражениях значений других столбцов.

Проверка значений полей

Конечно, имеется любое число ограничений, которые можно устанавливать для данных вводимых в ваши таблицы, чтобы видеть, например, находятся ли данные в соответствующем диапазоне или правильном формате, о чем SQL естественно не может знать заранее. По этой причине, SQL обеспечивает вас ограничением **CHECK,** которое позволяет вам установить условие которому должно удовлетворять значение вводимое в таблицу, прежде чем оно будет принято.

Ограничение **CHECK** состоит из ключевого слова **CHECK** сопровождаемого предложением предиката, который использует указанное поле. Любая попытка модифицировать или вставить значение поля которое могло бы сделать этот предикат неверным - будет отклонена.

Давайте рассмотрим таблицу **Продавцы**. Столбец комиссионных выражается десятичным числом и поэтому может быть умножен непосредственно на сумму приобретений в результате чего будет получена сумма комиссионных(в долларах) продавца с установленым справа значком доллара($) . Кто-то может использовать понятие процента, однако ведь, можно об этом и не знать. Если человек введет по ошибке 14 вместо .14 чтобы указать в процентах свои комиссионные, это будет расценено как 14.0 , что является законным десятичным значением, и будет нормально воспринято системой. Чтобы предотвратить эту ошибку, мы можем наложить ограничение столбца - **CHECK** чтобы убедиться, что вводимое значение меньше чем 1.

Пример 8. Создание таблицы Продавцы и с проверкой значений полей

```
CREATE TABLE Продавцы
  ( КодПродавца     integer NOT NULL PRIMARY KEY,
    Фамилия    char(10) NOT NULL UNIQUE,
    Город      char(10),
    Комиссионные     decimal CHECK (Комиссионные < 1 ));
```

Использование - CHECK, чтобы предопределять допустимое вводимое значение

Мы можем также использовать ограничение **CHECK** чтобы защитить от ввода в поле определенных значений, и таким образом предотвратить ошибку.

Например, предположим, что единственными городами в которых мы имели ведомства сбыта являются Лондон, Барселона, Сан Хосе, и Нью Йорк. Если вам известны все продавцы работающие в каждом из этих ведомств, нет необходимости позволять ввод других значений. Если же нет, использова ние ограничения может предотвратить опечатки и другие ошибки.

Пример 8. Создание таблицы Продавцы и с проверкой значений полей, чтобы предопределять допустимое вводимое значение

```
CREATE TABLE Продавцы
  ( КодПродавца     integer NOT NULL PRIMARY KEY,
    Фамилия    char(10) NOT NULL UNIQUE,
    Город      char(10)
CHECK    (Город IN (' Лондон ', 'Барселона', ' Сан Хосе ', ' Нью Йорк ')),
    Комиссионные     decimal CHECK (Комиссионные < 1 ));
```

Выполните sql-код. Обновите базу данных и просмотрите созданную таблицу. Сохраните sql-запрос под именем **Пример8.sql** в папке **ФИО_студента/Лаб4**.

Конечно, если вы собираетесь сделать это, вы должны быть уверены что ваша компания не открыла уже новых других ведомств сбыта. Большинство программ баз данных поддерживают команду **ALTER TABLE** , которая позволяет вам изменять определение таблицы, даже когда она находится в использовании. Однако, изменение или удаление ограничений не всегда возможно для этих команд, даже там где это вроде бы поддерживается.

Если вы использовали систему, которая не может удалять ограничения, вы будете должны создавать (**CREATE**) новую таблицу и передавать информацию из старой таблицы в нее всякий раз, когда вы хотите изменить ограничение. Конечно же Вы не захотите делать это часто, и со временем вообще перестанете это делать.

Проверка условий, базирующийся на многочисленых полях

Вы можете также использовать **CHECK** в качестве табличного ограничения. Это полезно в тех случаях, когда вы хотите включить более одного поля строки в условие. Предположим что комиссионные 0.15 и выше, будут разрешены только для продавца из Барселоны. Вы можете указать это со следующим табличным ограничением **CHECK** :

Пример 9. Создание таблицы Продавцы и с проверкой значений полей, базирующийся на многочисленых полях

```
CREATE TABLE Продавцы2
  ( КодПродавца     integer NOT NULL PRIMARY KEY,
    Фамилия    char(10) NOT NULL UNIQUE,
    Город      char(10) ,
Комиссионные     decimal,
CHECK (Комиссионные < 0.15 OR  Город='Барселона'));
```

Выполните sql-код. Обновите базу данных и просмотрите созданную таблицу. Сохраните sql-запрос под именем **Пример9.sql** в папке **ФИО_студента/Лаб4**.

Как вы можете видеть, два различных поля должны быть проверены чтобы определить, верен предикат или нет. Имейте в виду, что это - два разных поля одной и той же строки. Хотя вы можете использовать многочис леные поля, SQL не может проверить более одной строки одновременно. Вы не можете, например использовать ограничение **CHECK** чтобы удостовериться что все комиссионные в данном городе одинаковы. Чтобы сделать это, SQL должен всякий раз просматривая другие строки таблицы, когда вы модифицируете или вставляете строку, видеть, что значение комиссионных указано для текущего города. SQL этого делать не умеет.

Фактически, вы могли бы использовать сложное ограничение **CHECK** для вышеупомянутого, если бы знали заранее, каковы должны быть комиссионные в разных городах.

Самостоятельно измените ограничение в примере 9 на следующее:

- Если комиссионные равны 0.15 , то будут разрешены только для продавца из Лондана
- Если комиссионные равны 0.14 , то будут разрешены только для продавца из Барселоны
- Если комиссионные равны 0.13 , то будут разрешены только для продавца из Сан-Хосе
- Если комиссионные равны 0.12 , то будут разрешены только для продавца из Нью-Йорка

Вы получили идею. Чем налагать такой комплекс ограничений, вы могли бы просто использовать представление с предложением **WITH CHECK OPTION,** которое имеет все эти условия в своем предикате. Пользователи могут обращаться к представлению таблицы вместо самой таблицы. Одним из преимуществ этого будет то, что процедура изменения в ограничении не будет такой болезненной или трудоемкой. Представление с **WITH CHECK OPTION** - хороший заменитель ограничению **CHECK**.

Пример 10. Создание таблицы Клиент с ограничениями.

```
CREATE TABLE  Клиент
(КодКлиента INT IDENTITY(1,1) PRIMARY KEY,
 Фирма       VARCHAR(50)      NOT NULL,
 Фамилия     VARCHAR(50)      NOT NULL,
 Город       VARCHAR(50)      NOT NULL,
 Телефон     CHAR(10)         NOT NULL
  CHECK (Телефон LIKE  '[1-9][0-9]-[0-9][0-9]-[0-9][0-9]'));
```

Выполните sql-код. Обновите базу данных и просмотрите созданную таблицу. Сохраните sql-запрос под именем **Пример10.sql** в папке **ФИО_студента/Лаб4**.

4.6.5. Ограничение внешнего ключа (FOREIGN KEY)

Ограничение внешнего ключа - это основной механизм для поддержания ссылочной целостности между таблицами реляционной базы данных.

Столбец дочерней таблицы, определенный в качестве **внешнего ключа** в параметре **FOREIGN KEY**, применяется для ссылки на **столбец родительской таблицы**, являющийся в ней первичным ключом.

Имя родительской таблицы и столбцы ее первичного ключа указываются в предложении **REFERENCES**.

Данные в столбцах, определенных в качестве внешнего ключа, могут принимать только такие же значения, какие находятся в связанных с ним столбцах первичного ключа родительской таблицы.

Совпадение имен столбцов для связи дочерней и родительской таблиц необязательно.

Первичный ключ может быть определен для столбца с одним именем, в то время как столбец, на который наложено ограничение **FOREIGN KEY**, может иметь совершенно другое имя. Единственным требованием остается соответствие столбцов по типу и размеру данных.

На первичный ключ могут ссылаться не только столбцы других таблиц, но и столбцы, расположенные в той же таблице, что и собственно первичный ключ; это позволяет создавать рекурсивные структуры.

Внешний ключ может быть связан не только с первичным ключом другой таблицы. Он может быть определен и для столбцов с ограничением **UNIQUE** второй таблицы или любых других столбцов, но таблицы должны находиться в одной базе данных.

Столбцы внешнего ключа **могут содержать значение NULL**, однако проверка на ограничение **FOREIGN KEY** игнорируется. Внешний ключ может быть проиндексирован, тогда сервер будет быстрее отыскивать нужные данные. Внешний ключ определяется как при создании, так и при изменении таблиц.

Ограничение ссылочной целостности задает требование, согласно которому для каждой записи в дочерней таблице должна иметься запись в родительской таблице. При этом изменение значения столбца связи в записи родительской таблицы при наличии дочерней записи блокируется, равно как и удаление родительской записи (запрет каскадного изменения и удаления), что гарантируется параметрами **ON DELETE NO ACTION и ON UPDATE NO ACTION**, принятыми по умолчанию. Для разрешения каскадного воздействия следует использовать параметры **ON DELETE CASCADE** и **ON UPDATE CASCADE.**

Если пользователь предпринимает попытку удалить из родительской таблицы строку, на которую ссылается одна или более строк дочерней таблицы, язык SQL предоставляет следующие возможности:

CASCADE - выполняется удаление строки из родительской таблицы, сопровождающееся автоматическим удалением всех ссылающихся на нее строк дочерней таблицы;

SET NULL - выполняется удаление строки из родительской таблицы, а во внешние ключи всех ссылающихся на нее строк дочерней таблицы записывается значение NULL;

SET DEFAULT - выполняется удаление строки из родительской таблицы, а во внешние ключи всех ссылающихся на нее строк дочерней таблицы заносится значение, принимаемое по умолчанию;

NO ACTION - операция удаления строки из родительской таблицы отменяется. Именно это значение используется по умолчанию в тех случаях, когда в описании внешнего ключа фраза ON DELETE опущена.

Те же самые правила применяются в языке SQL и тогда, когда значение потенциального ключа родительской таблицы обновляется.

Определитель **MATCH** позволяет уточнить способ обработки значения **NULL** во внешнем ключе.

При определении таблицы предложение **FOREIGN KEY** может указываться произвольное количество раз.

В операторе **CREATE TABLE** используется необязательная фраза **DEFAULT**, которая предназначена для задания принимаемого по умолчанию значения, когда в операторе **INSERT** значение в данном столбце будет отсутствовать.

Фраза **CONSTRAINT** позволяет задать имя ограничению, что позволит впоследствии отменить то или иное ограничение с помощью оператора **ALTER TABLE**.

Пример 11. Создание таблицы Склад с ограничениями первичного ключа и внешнего ключа.

```
CREATE TABLE Сделка
(КодСделки  INT IDENTITY(1,1) PRIMARY KEY,
 КодТовара  INT      NOT NULL,
 КодКлиента INT      NOT NULL,
 Количество INT      NOT NULL DEFAULT 0,
 Дата       DATETIME  NOT NULL DEFAULT GETDATE(),
Остаток     INT,
CONSTRAINT  fk_Товар  FOREIGN KEY(КодТовара)  REFERENCES Товар,
CONSTRAINT  fk_Клиент  FOREIGN  KEY(КодКлиента)  REFERENCES
Клиент);
```

Выполните sql-код. Обновите базу данных и просмотрите созданную таблицу. Сохраните sql-запрос под именем **Пример11.sql** в папке **ФИО_студента/Лаб4**.

4.7. Изменение таблиц

Для внесения изменений в уже созданные таблицы стандартом SQL предусмотрен оператор **ALTER TABLE**, предназначенный для выполнения следующих действий:

- добавление в таблицу нового столбца;
- удаление столбца из таблицы;
- добавление в определение таблицы нового ограничения;
- удаление из определения таблицы существующего ограничения;
- задание для столбца значения по умолчанию;
- отмена для столбца значения по умолчанию.

Оператор изменения таблицы имеет следующий обобщенный формат:

```
<изменение_таблицы> ::=
ALTER TABLE имя_таблицы
[ADD [COLUMN]имя_столбца тип_данных
  [ NOT NULL ][UNIQUE]
[DEFAULT <значение>][ CHECK (<условие_выбора>)]]
[DROP [COLUMN] имя_столбца [RESTRICT | CASCADE ]]
[ADD [CONSTRAINT [имя_ограничения]]
[{PRIMARY KEY (имя_столбца [,...n])
   |[UNIQUE (имя_столбца [,...n])}
|[FOREIGN KEY (имя_столбца_внешнего_ключа [,...n])
   REFERENCES имя_род_таблицы
     [(имя_столбца_род_таблицы [,...n])],
```

[MATCH {PARTIAL | FULL}
[ON UPDATE {CASCADE| SET NULL |
SET DEFAULT | NO ACTION}]
[ON DELETE {CASCADE| SET NULL |
SET DEFAULT | NO ACTION}]
|[CHECK(<условие_выбора>)][,...n]}]
[DROP CONSTRAINT имя_ограничения
[RESTRICT | CASCADE]]
[ALTER [COLUMN] SET DEFAULT <значение>]
[ALTER [COLUMN] DROP DEFAULT]

Здесь параметры имеют то же самое назначение, что и в определении оператора **CREATE TABLE**.

Оператор **ALTER TABLE** реализован не во всех диалектах языка SQL. В некоторых диалектах он поддерживается, однако не позволяет удалять из таблицы уже существующие столбцы.

В дополнение к уже названным параметрам определим параметр **{ENABLE | DISABLE } TRIGGER ALL_,** предписывающий задействовать или отключить конкретный триггер или все триггера, связанные с таблицей.

Пример 12. Удаление ограничения внешнего ключа.
ALTER TABLE Сделка DROP CONSTRAINT fk_Товар

Пример 13. Добавления ограничения внешнего ключа, реализующего каскадные обновления и изменения.
ALTER TABLE Сделка ADD CONSTRAINT fk_Товар
FOREIGN KEY (КодТовара) REFERENCES Товар
ON UPDATE CASCADE ON DELETE CASCADE

Пример 14. Пример создания вычисляемого поля.
ALTER TABLE Товар ADD Налог AS Цена*0.05

Пример 15. Пример удаления поля
ALTER TABLE Товар DROP COLUMN Остаток

4.7. Удаление таблиц

Удаление таблицы выполняется командой:

DROP TABLE имя_таблицы

Удалить можно любую таблицу, даже системную. К этому вопросу нужно подходить очень осторожно. Однако удалению не подлежат таблицы, если существуют объекты, ссылающиеся на них. К таким объектам относятся таблицы, связанные с удаляемой таблицей посредством внешнего ключа. Поэтому, прежде чем удалять родительскую таблицу, необходимо удалить либо ограничение внешнего ключа, либо дочерние таблицы. Если с таблицей связано хотя бы одно представление, то таблицу также удалить не удастся. Кроме того, связь с таблицей может быть установлена со стороны функций и процедур. Следовательно, перед удалением таблицы необходимо удалить все объекты базы данных, которые на нее ссылаются, либо изменить их таким образом, чтобы ссылок на удаляемую таблицу не было.

Самостоятельно удалите таблицу **Продавцы.**

Задание для практической работы №4

Самостоятельно, используя **команды языка SQL,** в базе данных **Университет создать:**

1). Новую таблицу под именем **STUDENT (Студент)** с помощью sql-операторов с полями:

STUDENT_ID – целого типа для уникальной идентификации записей в таблице первичный ключ тип счетчик,

SUTNAME – текстового типа для обозначения имени студента,

SUTFNAME - текстового типа для обозначения фамилии,

STIPEND – действительного типа для обозначения стипендии. При этом на это поле наложено ограничение числом – величина размера стипендии должна быть меньше 500 грн.

KURS - целого типа для обозначения курса. При этом на это поле наложено ограничение – курс на котором может учиться студент может принимать значение от 1 до 5 ,

CITY - текстового типа для обозначения города,

BIRTDAY –типа даты/времени для обозначения день рождения,

GROUP - текстового типа для обозначения студенческой группы,

KOD_KAFEDRU – целого типа для обозначения названия кафедры, на которой учится студент. Поле **KOD_KAFEDRU** из таблицы **STUDENT** и поле **KOD_KAFEDRU** из таблицы **KAFEDRA** связаны тем, что описывают одни и те же данные, т.е. содержат идентификаторы кафедр, информация о которых содержит база данных. Более того, значение идентификаторов кафедр, которые допустимы в таблице **STUDENT**, должны выбираться только из списка значений поля **KOD_KAFEDRU**, т.е. принадлежащих реально описанных в базе данных кафедрам. Т.е. между этими полями имеется прямая связь. Т.о. поле **KOD_KAFEDRU** из таблицы **STUDENT** будет являться внешним ключом.

Кроме того, при определении таблицы **STUDENT** запрещено использовать значение **NULL** – значений для столбцов **STUDENT_ID, SUTNAME, SUTFNAME.**

В качестве первичного ключа принято значение столбца **STUDENT_ID.**

Выполните sql-код*. Обновите базу данных и просмотрите созданную таблицу. Сохраните sql-запрос под именем* ***Студент.sql*** *в папке* ***ФИО_студента/Лаб4.***

2). Новую таблицу под именем **TEACHER (Преподаватели)** с помощью sql-операторов. Эта таблица содержит информацию о преподавателях вуза. Каждый преподаватель может работать на одной кафедре, иметь множество лекционных занятий и быть куратором более чем одной группы.

Описание столбцов таблицы TEACHER

KOD_TEACHER Уникальный идентификатор преподавателя. Является первичным ключом

KOD_KAFEDRU Уникальный идентификатор кафедры, на которой работает преподаватель. Является внешним ключом

NAME_TEACHER Фамилия преподавателя

INDEF_KOD Идентификационный код. Является уникальным для преподавателя

DOLGNOST Должность, может принимать только определенные значения из списка, такие как 'профессор', 'доцент', 'старший преподаватель', 'ассистент'. Значение по умолчанию 'ассистент'.

ZVANIE Научное звание, может принимать только определенные

значения из списка, такие как 'к.т.н', 'к.г.у', 'к.с.н', 'к.ф.м.н.', 'д.т.н', 'д.г.у', 'д.с.н', 'д.ф.м.н', 'нет'. Значение по умолчанию 'нет'.

SALARY ставка зарплаты . Значение по умолчанию 1000 грн. Зарплата должна быть больше нуля.

RISE надбавка к зарплате. Ее значение по умолчанию =0 и не может быть отрицательным числом.

DATA_HIRE дата приема на работу. По умолчанию текущая дата.

BIRTHDAY день рождения

POL пол, может принимать только определенные значения из списка, 'ж', 'Ж', 'м', 'М'

TEL_TEACHER Телефон. Может принимать значения только в виде '[1-9][0-9]-[0-9][0-9]-[0-9][0-9]'.

В качестве первичного ключа принято значение столбца **KOD_TEACHER**.

Поле **KOD_KAFEDRU** из таблицы **TEACHER** и поле **KOD_KAFEDRU** из таблицы **KAFEDRA** связаны тем, что описывают одни и те же данные, т.е. содержат идентификаторы кафедр, информация о которых содержит база данных. Более того, значение идентификаторов кафедр, которые допустимы в таблице **TEACHER**, должны выбираться только из списка значений поля **KOD_KAFEDRU**, т.е. принадлежащих реально описанных в базе данных кафедрам. Т.е. между этими полями имеется прямая связь. Т.о. поле **KOD_KAFEDRU** из таблицы **TEACHER** будет являться внешним ключом.

Выполните sql-код**. Обновите базу данных и просмотрите созданную таблицу. Сохраните sql-запрос под именем **Преподаватель.sql** в папке **ФИО_студента/Лаб4.

САМОСТОЯТЕЛЬНО используя **команды языка SQL!!!**:

Создать **на языке Transact-SQL** файл базы данных согласно номеру варианта (присвоить ей новое имя, несовпадающие с именем базы данных созданной в лаб.№3). База данных разрабатывается на основе спроектированной концептуальной модели данных в лаб.№1.

Создать программно на языке SQL все таблицы, с указанием первичных и внешних ключей и ограничения целостности.

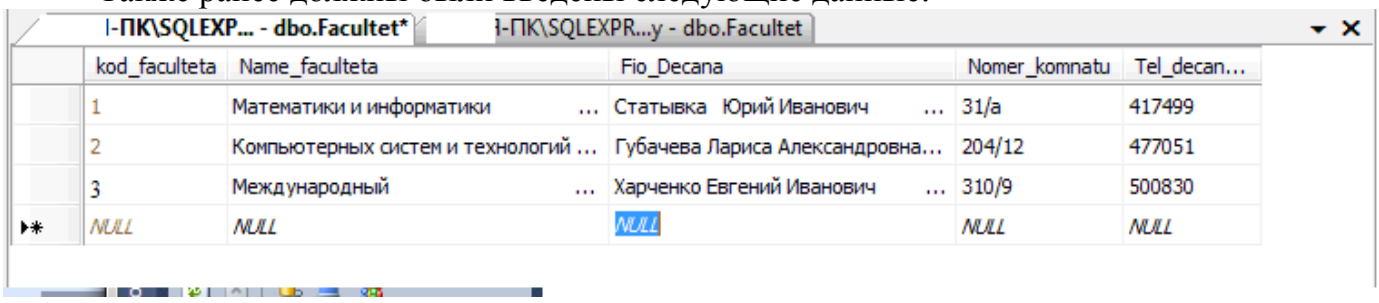
Все программные инструкции команд SQL сохранять в файлах с расширением ***.sql** в папке ***ФИО_студента/Лаб4***.

Заполнить таблицы данными по 5 записей в каждой.

Создать текстовый отчет, в котором отобразить sql-команды разработанных запросов и скриншоты результатов работы из СУБД **SQL Server Management Studio.**

Самостоятельно заполните вручную данными таблицы **Студент** и **Преподаватель** согласно рис. 1-2, приведенным ниже.

Также ранее должны были введены следующие данные:



kod_faculteta	Name_faculteta	Fio_Decana	Nomer_komnatu	Tel_decan...
1	Математики и информатики ...	Статывка Юрий Иванович ...	31/a	417499
2	Компьютерных систем и технологий ...	Губачева Лариса Александровна...	204/12	477051
3	Международный ...	Харченко Евгений Иванович ...	310/9	500830
NULL	NULL	NULL	NULL	NULL

\COMP\SQLEX...15 - dbo.Kafedra

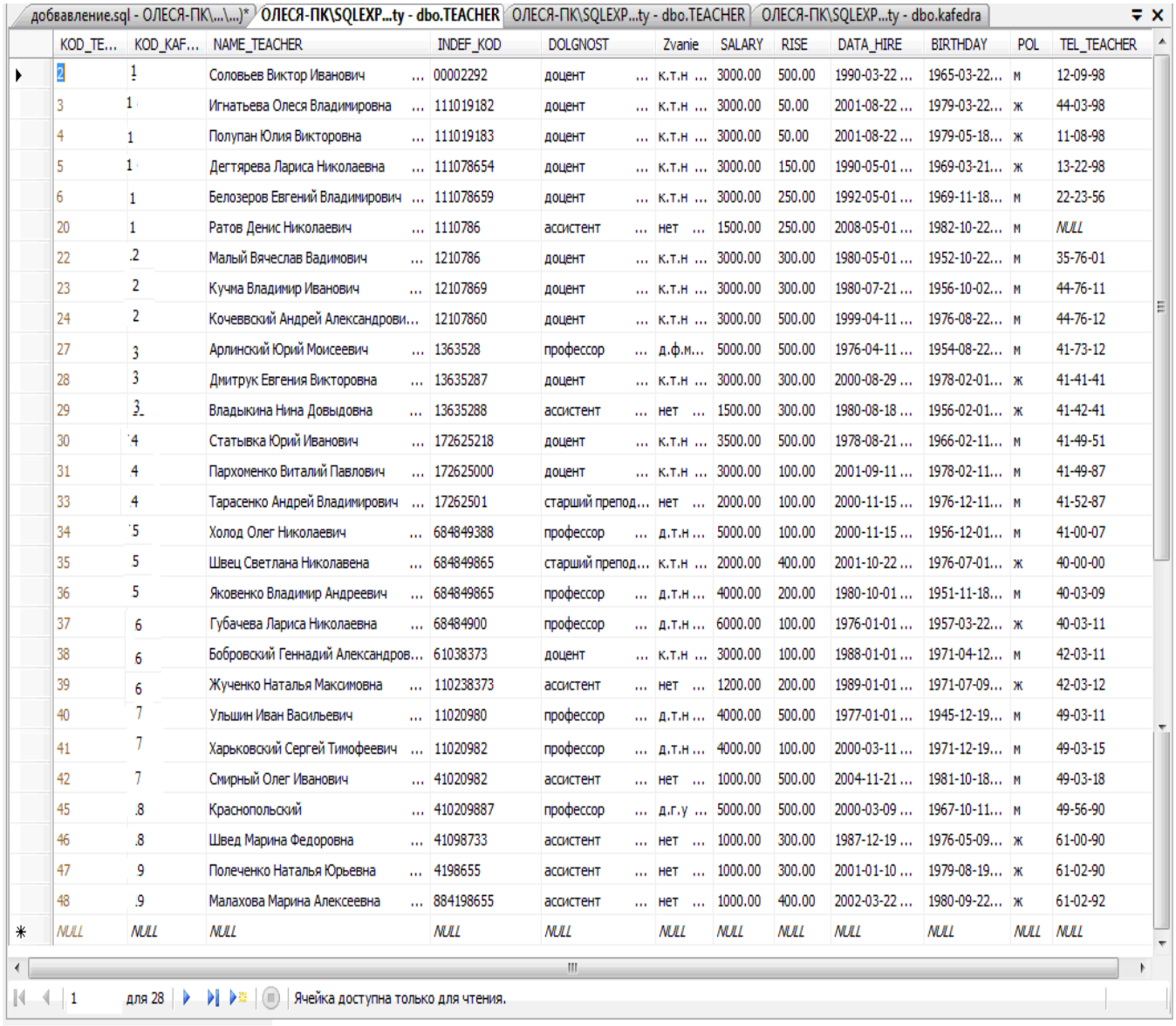
Kod_kafe...	kod_faculteta	Name_kafedru	Fio_zavkaf	Nomer_ko...	num_korpusa	Tel_kafedru
1	1	Компьютерные системы и сети	Соловьев	414	1	899028
2	1	Прикладная математика	Кочевский	205	1	425262
3	1	Математического анализа	Арлинский	61	1	627272
4	1	Информатики	Пожидаев	404	1	738328
5	2	Автоматизации компьютерных технологий	Малахов	123	12	637327
6	2	Компьютерных технологий на промышленном транспорте	Губачева	342	12	373728
7	2	Системная инженерия	Ульшин	234	12	727287
8	3	Иностранных языков	Краснопольский	123	2	762728
9	3	Компьютерных наук	Дядечев	703	9	563272
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Новые данные ввести вручную.

-ПК\SQLEXP...ty - dbo.kafedra | ПК\SQLEX...y - dbo.STUDENT | добавление.sql - (| I-ПК\...\...)* | -ПК\SQLEX...y - dbo.

STUDEN...	SUTNAME	SUTFNAME	STIPEND	KURS	CITY	BIRTHDAY	KOD_KAFEDRU	GROUP
2	Анна	Грешкина	450	2	Ростов-на-Дону	1989-01-01	1	МТ-401
3	Борис	Котовский	400	2	Ростов-на-Дону	1989-05-27	1	МТ-401
4	Петр	Комаров	300	2	Ростов-на-Дону	1989-11-18	1	МТ-401
5	Марина ...	Погребняк	353	2	Ростов-на-Дону	1989-10-21	1	МТ-402
6	Иванна ...	Смирнова	400	2	Ростов-на-Дону	1989-03-11	1	МТ-402
7	Роман	Сергеенко	0	2	Ростов-на-Дону	1989-07-04	1	МТ-402
8	Владимир ...	Невечеров	100	3	Ростов-на-Дону	1989-03-24	2	МТ-231
9	Мая	Соломка	450	3	Ростов-на-Дону	1988-01-12	2	МТ-231
10	Герман ...	Степанюга	300	3	Ростов-на-Дону	1988-11-18	2	МТ-231
11	Филипп ...	Мозговой	0	1	Ростов-на-Дону	1987-01-08	3	МТ-521
12	Федор ...	Филоненко	0	1	Донецк	1988-12-22	3	МТ-521
13	Наталья ...	Николенко	100	1	Москва	1989-12-02	3	МТ-521
14	Егор	Осадчий	100	5	Москва	1989-12-02	4	МТ-341
15	Анастасия ...	Петрова	489	5	Донецк	1989-07-22	4	МТ-341
16	Марьян ...	Шичанина	360	5	Донецк	1989-05-09	4	МТ-341
17	Роман	Тараненко	300	5	Донецк	1988-02-29	4	МТ-341
18	Виктория ...	Изотова	350	1	Донецк	1988-03-23	4	КТ-121
19	Валерий ...	Киреев	150	1	Донецк	1987-06-08	4	КТ-121
20	Наталья ...	Черткова	450	3	Ростов-на-Дону	1988-09-18	5	КТ-241
21	Андрей ...	Романенко	250	3	Ростов-на-Дону	1986-11-24	5	КТ-241
22	Владимир ...	Черней	480	3	Ростов-на-Дону	1986-11-11	5	КТ-241
23	Петр	Молчанов	450	3	Ростов-на-Дону	1986-11-11	6	КТ-341
24	Анастасия ...	Коваленко	350	3	Ростов-на-Дону	1986-11-11	6	КТ-341
25	Андрей ...	Жиркевич	250	3	Ростов-на-Дону	1986-07-08	6	КТ-341
26	Филипп ...	Шевцов	450	1	Донецк	1989-07-01	7	ИЯ-121
27	Максим ...	Громченко	300	1	Донецк	1988-05-11	7	ИЯ-121
28	Марина ...	Ващенко	100	1	Макарово	1988-11-23	8	ИЯ-121
29	Федор ...	Белянский	300	1	Макарово	1988-08-25	8	ИЯ-121
30	Юлия	Осадчая	400	2	Макарово	1988-08-25	9	КН-101
31	Федор ...	Христенко	200	2	Ростов-на-Дону	1986-01-15	9	КН-101
32	Марина ...	Романова	300	2	Ростов-на-Дону	1986-04-11	9	КН-101
33	Алина	Березова	400	2	Ростов-на-Дону	1986-03-22	9	КН-101
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

1 для 32 | Ячейка доступна только для чтения.



KOD_TE...	KOD_KAF...	NAME_TEACHER	INDEF_KOD	DOLGNOST	Zvanie	SALARY	RISE	DATA_HIRE	BIRTHDAY	POL	TEL_TEACHER
2	1	Соловьев Виктор Иванович	00002292	доцент	к.т.н	3000.00	500.00	1990-03-22	1965-03-22	м	12-09-98
3	1	Игнатьева Олеся Владимировна	111019182	доцент	к.т.н	3000.00	50.00	2001-08-22	1979-03-22	ж	44-03-98
4	1	Полупан Юлия Викторовна	111019183	доцент	к.т.н	3000.00	50.00	2001-08-22	1979-05-18	ж	11-08-98
5	1	Дегтярева Лариса Николаевна	111078654	доцент	к.т.н	3000.00	150.00	1990-05-01	1969-03-21	ж	13-22-98
6	1	Белозеров Евгений Владимирович	111078659	доцент	к.т.н	3000.00	250.00	1992-05-01	1969-11-18	м	22-23-56
20	1	Ратов Денис Николаевич	1110786	ассистент	нет	1500.00	250.00	2008-05-01	1982-10-22	м	NULL
22	2	Малый Вячеслав Вадимович	1210786	доцент	к.т.н	3000.00	300.00	1980-05-01	1952-10-22	м	35-76-01
23	2	Кучма Владимир Иванович	12107869	доцент	к.т.н	3000.00	300.00	1980-07-21	1956-10-02	м	44-76-11
24	2	Кочеввский Андрей Александрови	12107860	доцент	к.т.н	3000.00	500.00	1999-04-11	1976-08-22	м	44-76-12
27	3	Арлинский Юрий Моисеевич	1363528	профессор	д.ф.м	5000.00	500.00	1976-04-11	1954-08-22	м	41-73-12
28	3	Дмитрук Евгения Викторовна	13635287	доцент	к.т.н	3000.00	300.00	2000-08-29	1978-02-01	ж	41-41-41
29	3	Владыкина Нина Довыдовна	13635288	ассистент	нет	1500.00	300.00	1980-08-18	1956-02-01	ж	41-42-41
30	4	Статывка Юрий Иванович	172625218	доцент	к.т.н	3500.00	500.00	1978-08-21	1966-02-11	м	41-49-51
31	4	Пархоменко Виталий Павлович	172625000	доцент	к.т.н	3000.00	100.00	2001-09-11	1978-02-11	м	41-49-87
33	4	Тарасенко Андрей Владимирович	17262501	старший препод	нет	2000.00	100.00	2000-11-15	1976-12-11	м	41-52-87
34	5	Холод Олег Николаевич	684849388	профессор	д.т.н	5000.00	100.00	2000-11-15	1956-12-01	м	41-00-07
35	5	Швец Светлана Николавена	684849865	старший препод	к.т.н	2000.00	400.00	2001-10-22	1976-07-01	ж	40-00-00
36	5	Яковенко Владимир Андреевич	684849865	профессор	д.т.н	4000.00	200.00	1980-10-01	1951-11-18	м	40-03-09
37	6	Губачева Лариса Николаевна	68484900	профессор	д.т.н	6000.00	100.00	1976-01-01	1957-03-22	ж	40-03-11
38	6	Бобровский Геннадий Александров	61038373	доцент	к.т.н	3000.00	100.00	1988-01-01	1971-04-12	м	42-03-11
39	6	Жученко Наталья Максимовна	110238373	ассистент	нет	1200.00	200.00	1989-01-01	1971-07-09	ж	42-03-12
40	7	Ульшин Иван Васильевич	11020980	профессор	д.т.н	4000.00	500.00	1977-01-01	1945-12-19	м	49-03-11
41	7	Харьковский Сергей Тимофеевич	11020982	профессор	д.т.н	4000.00	100.00	2000-03-11	1971-12-19	м	49-03-15
42	7	Смирный Олег Иванович	41020982	ассистент	нет	1000.00	500.00	2004-11-21	1981-10-18	м	49-03-18
45	8	Краснопольский	410209887	профессор	д.г.у	5000.00	500.00	2000-03-09	1967-10-11	м	49-56-90
46	8	Швед Марина Федоровна	41098733	ассистент	нет	1000.00	300.00	1987-12-19	1976-05-09	ж	61-00-90
47	9	Полеченко Наталья Юрьевна	4198655	ассистент	нет	1000.00	300.00	2001-01-10	1979-08-19	ж	61-02-90
48	9	Малахова Марина Алексеевна	884198655	ассистент	нет	1000.00	400.00	2002-03-22	1980-09-22	ж	61-02-92
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

ПРАКТИЧЕСКАЯ РАБОТА №5. СОЗДАНИЕ ЗАПРОСОВ НА ВЫБОРКУ. ОТБОР СТРОК ПО УСЛОВИЮ

5.1. Цель практической работы

Изучить используемый в реляционных СУБД оператор извлечения данных из таблиц. Получить навыки работы с оператором SELECT в программе ‘SQL Server Managmant Studio’.

5.2. Исходные данные

Исходными данными является индивидуальное задание и результат предыдущих практических работ.

5.3. Используемые программы

Программы ‘ SQL Server Managmant Studio ‘.

5.4. Теоретические сведения

В SQL имеется единственный оператор, который предназначен для выборки данных из базы данных. Оператор относится к подмножеству **DML.**

Ниже приведен почти полный синтаксис оператора **SELECT.**

SELECT [**DISTINCT | ALL**]
{* | <величина> [, <величина> ...]**}**
[**INTO** :Переменная [, :Переменная ...]]
FROM <tableref> [**,** <tableref> **...**]
[**WHERE** <условие поиска>]
[**GROUP BY** Колонка [, Колонка ...]]
[**HAVING** <условие поиска>]
[**UNION** [**ALL**] <select_expr>]
[**ORDER BY** <список сортировки>];

<величина> = **{**Колонка | :Переменная | <константа>
| <выражение> | <функция>
| udf **(**[<величина> [, <величина> ...]]**)**
| **NULL | USER}** [**AS** Псевдоним]

<константа> = Число | 'Строка'

<выражение> = SQL выражение, возвращающее единичное значение

<функция> =
COUNT (* | [**ALL**] <величина> | **DISTINCT** <величина>**)**
| **SUM (**[**ALL**] <величина> | **DISTINCT** <величина>**)**
| **AVG** ([**ALL**] <величина> | **DISTINCT** <величина>**)**
| **MAX (**[**ALL**] <величина> | **DISTINCT** <величина>**)**
| **MIN (**[**ALL**] <величина> | **DISTINCT** <величина>**)**
| **CAST(**<величина> **AS** <тип данных>**)**
| **UPPER (**<величина>**)**
| **GEN_ID (**Имя_Генератора, <величина>**)**

<tableref> = {<joined_table> | table | view
| procedure[**(**<величина> [, <величина> ...]**)**]**}**

[Псевдоним]

<joined_table> = <tableref> <join_type> **JOIN** <tableref>
ON <условие поиска> | **(**<joined_table>**)**

<join_type> = [**INNER**] | **{LEFT** | **RIGHT** | **FULL }** [**OUTER**]

<условие поиска> =
<величина> <оператор сравнения>
{<величина> | **(**<select_one>**)}**
| <величина> [**NOT**] **BETWEEN** <величина> **AND** <величина>
| <величина> [**NOT**] **LIKE** <величина>
| <величина> [**NOT**] **IN**
(<величина> [, <величина> ...] | <select_list>**)**
| <величина> **IS** [**NOT**] **NULL**
| <величина> **{**>= | <=**}** <величина>
| <величина> [**NOT**] **{**= | < | >**}** <величина>
| **{ALL** | **SOME** | **ANY} (**<select_list>**)**
| **EXISTS (**<select_expr>**)**
| **SINGULAR (**<select_expr>**)**
| <величина> [**NOT**] **CONTAINING** <величина>
| <величина> [**NOT**] **STARTING** [**WITH**] <величина>
| **(**<условие поиска>**)**
| **NOT** <условие поиска>
| <условие поиска>**OR** <условие поиска>
| <условие поиска>**AND** <условие поиска>

<оператор сравнения> =
{= | < | > | <= | >= | !< | !> | <> | !=**}**

<select_one> = оператор SELECT, выбирающий одну колонку и возвращающий ровно одно значение

<select_list> = оператор SELECT, выбирающий одну колонку, возвращающий ноль или много значений

<select_expr> = оператор SELECT, выбирающий несколько величин и возвращающий ноль или много значений

<список сортировки> =
{Колонка | Номер**}**
[**ASC** | **DESC**]
[, <список сортировки> ...]

Некоторые параметры, входящие в этот оператор, описаны в табл. 5.1.

Таблица 5.1

Описание параметров оператора SELECT

Параметр	Описание
DISTINCT | ALL	**DISTINCT** – предотвращает дублирование данных, которые будут извлечены. **ALL** (по умолчанию) – приведет к извлечению всех
{* | <величина> [, <величина> ...]**}**	Звездочка (*****) означает, что надо извлекать все колонки из указанных таблиц. <величина> [, <величина> ...] – извлекает список указанных
INTO :Переменная [, :Переменная ...]	Используется только в триггерах и хранимых процедурах для операторов SELECT, возвращающих не более одной строки. Указывается список переменных, в которые извлекаются
FROM <tableref> [, <tableref> ...]	Указывает список таблиц, просмотров и хранимых процедур, из которых извлекаются данные. Список может включать соединения и соединения могут быть
table	Имя существующей в базе данных таблицы
view	Имя существующего базе данных просмотра
procedure	Имя существующей хранимой процедуры, предназначенной для использования в операторе SELECT
Псевдоним	Короткое альтернативное имя для таблицы, просмотра или колонки. После описания в <tableref>, псевдоним может использоваться для ссылок на таблицу или просмотр
join_type	Задает тип соединения, которое может быть внутренним или внешним
WHERE <условие поиска>	Указывает условие, которое ограничивает количество извлекаемых строк
GROUP BY Колонка [, Колонка ...]	Разбивает результат запроса на группы, содержащие все строки с идентичными значениями указанными в списке
HAVING <условие поиска>	Использует совместно с GROUP BY. Задает условие, которое ограничивает количество возвращаемых групп
UNION [**ALL**]	Объединяет результаты нескольких запросов. Все запросы должны извлекать одинаковое количество столбцов, тип данных каждого столба первого запроса должен совпадать с типом данных других запросов, имена столбцов в разных запросах мо- гут отличаться. Необязательный параметр ALL
ORDER BY <список сортировки>	Указывает колонки, по которым будет производиться сортировка извлекаемых строк. Можно указывать либо имена коло- нок, либо их порядковые номера в списке извлекаемых колонок. Если указать ASC, то строки будут выдаваться в порядке возрастания значений сортируемых полей, если DESC

Как видно из синтаксиса оператора **SELECT,** обязательными являются только предложение **SELECT** с перечнем выдаваемых колонок и предложение **FROM.**

Пример простейшего оператора SELECT:

```
-- Выдать перечень всех служащих:
SELECT * FROM Employee;
```

Ниже приведено несколько упрощенных вариантов синтаксиса оператора **SELECT**, помогающих научиться составлять простые запросы.

Упрощенный синтаксис внутреннего соединения (стандарт SQL-92):

SELECT Колонка [, Колонка ...] | ***
FROM <tableref_left> [**INNER**] **JOIN** <tableref_right>
[**ON** <условие поиска>]
[**WHERE** <условие поиска>];

Упрощенный синтаксис внешнего соединения:

SELECT Колонка [, Колонка ...] | ***
FROM <tableref_left>
{LEFT | RIGHT | FULL} [**OUTER**] **JOIN**
<tableref_right>
[**ON** <условие поиска>]
[**WHERE** <условие поиска>];

Упрощенный синтаксис использования подзапроса:

SELECT [**DISTINCT**] Колонка [, Колонка ...]
FROM <tableref> [, <tableref> ...]
WHERE
{expression **{**[**NOT**] **IN** | <оператор сравнения>**}**
| [**NOT**] **EXISTS**
}
(SELECT [**DISTINCT**] Колонка [, Колонка ...]
FROM <tableref> [, <tableref> ...]
WHERE <условие поиска>
);

5.5. Задание

Практическую работу следует выполнять в следующем порядке:

1. Изучить синтаксис оператора **SELECT** и примеры запросов к учебной базе данных '**University.mdf**'.

2. Выполнить в окне 'SQL Editor' **27** запроса к базе данных, согласно приведенным в практической работе образцам выполнения запросов и сохранять каждый под именами '**Lab5-k.sql'**, где k – номер запроса по порядку, в своей рабочей папке. Каждый запрос должен иметь комментарии с описанием, а файл в целом должен иметь комментарии со сведениями об авторе и дате создания.

5.6. Ход работы

Примечание. У вас должны быть перед выполнением этой практической работы созданы все таблицы базы данных университета, созданы ключи, а также заполнены данными.

Выполнение sql-запросов

Для выполнения запросов **SELECT** в программе 'SQL Server Managmant Studio' необходимо выполнить следующие действия:

1. Подключиться к базе данных и выполнить команду '**Создать запрос**'. В результате откроется окно '**Конструктора запросов**' (рис. 1).

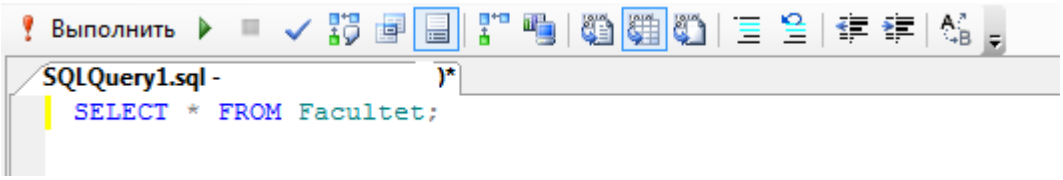


Рис. 1. Окно выполнения запросов

2. Ввести текст запроса согласно рис.1.
3. Нажать на панели инструментов кнопку [**Выполнить**] .
4. Если запрос правильный, то в результате произойдет его выполнение и результат будет отображен на вкладке '**Результаты**' (рис. 2).

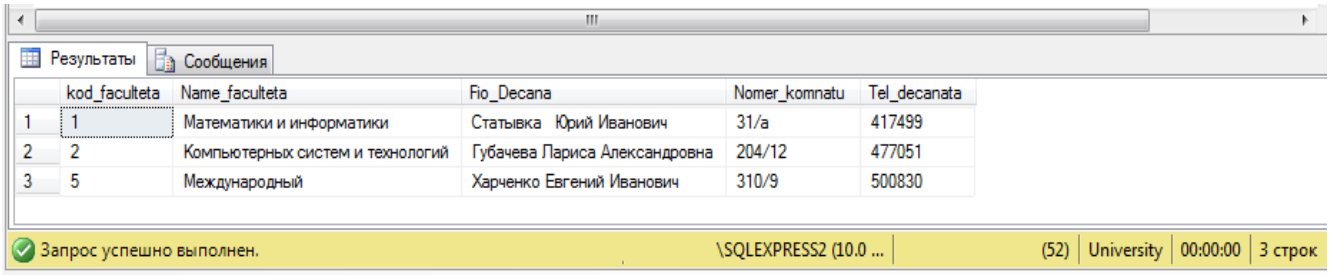


Рис. 2. Окно с результатом выполнения запроса

5. Количество извлеченных в результате выполнения запроса строк отображается над сеткой с данными справа. На рис.2 там содержится строка '3 строк'. В данном примере извлечено столько строк, сколько требуется, чтобы заполнить сетку (в ней помещается только 3 строки) * .
6. Чтобы узнать, сколько всего строк соответствуют выполненному оператору, надо перейти в конец отображаемого набора данных.

Чтобы выполнить другой запрос, надо вернуться на вкладку '**Редактора**', создать новый запрос и повторить те же действия.

К тексту ранее выполнявшихся правильных запросов можно вернуться, если перейти на вкладку '**История**'.

Тема Примеры создания запросов с отбором строк по условию.

SQL дает возможность определить критерии отбора необходимых строк во фразе WHERE предложения SELECT. В этом случае строки исходных таблиц будут включены в результирующую только если строка соответствует указанным критериям. *Условие* - это выражение, которое может быть истинным или ложным (логическое выражение или предикат), то есть принимать логические значения TRUE или FALSE соответственно. В результирующую таблицу включаются только те строки, для которых указанное во фразе WHERE условие равно TRUE (иными словами, которые удовлетворяют заданному условию).

В случае одной таблицы механизм работы предложения SELECT с фразой WHERE следующий.

1. Из таблицы, указанной во фразе **FROM**, выбирается очередная строка.
2. Она проверяется на соответствие условию во фразе **WHERE**.
3. Если результат равен **TRUE**, строка включается в результирующую таблицу и форматируется в соответствии с фразой **SELECT**, а если он равен **FALSE**, строка пропускается.

Далее будут рассмотрены основные выражения, допустимые для условия во фразе WHERE.

Использование простейших условий

Простейшими считаются условия, в которых используются операторы сравнения и логические операторы.

Хотя такие условия являются простейшими в смысле семантики конструкций, они могут иметь довольно сложную структуру из многих операторов сравнений и вложенных друг в друга логических связок.

Операторы сравнения

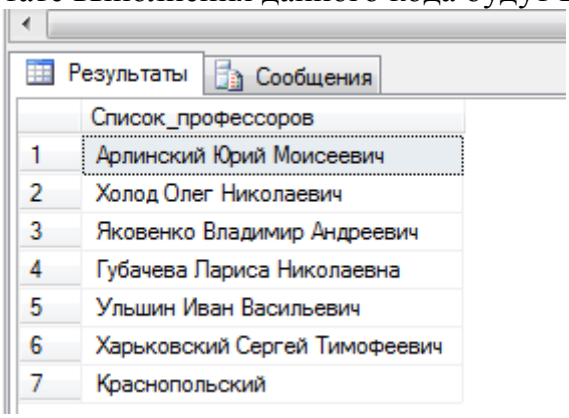
Особенностью операторов сравнения является то, что независимо от типов операндов их результатом являются логические значения. Предположим, вы хотите получить список всех профессоров.

Создайте новый запрос, введите sql-запрос, выполните его, сохраните его в рабочую папку ЛАБ6_SQL под именем 1.sql.

Запрос 1. Вывести фамилии профессоров.

```
SELECT NAME_TEACHER AS ‘Список профессоров’
FROM    TEACHER
WHERE  DOLGNOST = 'профессор';
```

Чтобы выполнить sql-команду нажмите на панели редактора кнопку . В результате выполнения данного кода будут выданы все профессора. Например,



	Список_профессоров
1	Арлинский Юрий Моисеевич
2	Холод Олег Николаевич
3	Яковенко Владимир Андреевич
4	Губачева Лариса Николаевна
5	Ульшин Иван Васильевич
6	Харьковский Сергей Тимофеевич
7	Краснопольский

Слово 'профессор' в запросе является строковой константой, поэтому ее следует заключить в кавычки. Обратите внимание, что мы указали фразу SELECT без ключевого слова DISTINCT, так как тогда от нас была бы скрыта информация о существовании среди профессоров однофамильцев. Чтобы при выводе результирующий столбец имел содержательный заголовок, мы поименовали его как Список профессоров.

Это первый пример использования предиката над строковым типом данных. Здесь столбец строкового типа сравнивается со строковой константой. Запрос выполнен правильно, однако нужно всегда помнить о том, что предикаты над строками являются чувствительными к регистру букв. Например, предикат 'ИВАНОВ' = 'Иванов' будет ложным. Поэтому, если для некоторого профессора его должность была введена в таблицу TEACHER как 'Профессор', он не будет найден по условию WHERE DOLGNOST = 'профессор'.

Чтобы на предикаты над строками не влиял регистр букв, нужно использовать обычно имеющиеся в СУБД функции преобразования букв в прописные и строчные. В стандарте SQL, например, указаны функции UPPER и LOWER, выполняющие такие преобразования. Следовательно, для предыдущего запроса правильней будет записать

условие фразы WHERE одним из следующих способов:

WHERE LOWER(DOLGNOST) = 'профессор'
WHERE UPPER(DOLGNOST) = 'ПРОФЕССОР'

Самостоятельно измените в исходном запросе строку условия с использованием функции изменения регистра.

Чтобы сохранить запрос нажмите правую кнопу и из контекстного меню выберите Сохранить в файл. Присвойте имя 1.sql и сохраните в папку ЛАБ5_SQL.

Запрос 2. Найти всех студентов с стипендией, превышающим 300 грн.

В sql-редакторе создайте новый запрос и введите следующий код:

```
SELECT SUTNAME, SUTFNAME
FROM   STUDENT
WHERE  STIPEND > 300;
```

Выполните его.

Приведем несколько примеров использования операторов сравнения для столбцов строкового и временного типа. Обратите внимание, что сравнивать можно не только значение столбца с константой, но и значения столбцов между собой.

Запрос 3. Вывести фамилии и должности преподавателей, принятых на работу после 01.01.2002.

```
SELECT NAME_TEACHER AS 'Фамилия', DOLGNOST AS 'Должность'
FROM   TEACHER
WHERE  DATA_HIRE > '1/01/2002';
```

Запрос 4. Вывести фамилии и должности преподавателей, фамилии которых в алфавитном порядке располагаются после фамилии Сычева.

```
SELECT NAME_TEACHER, DOLGNOST
FROM   TEACHER
WHERE  UPPER(NAME_TEACHER) > 'Сычева';
```

Самостоятельно создайте запрос 5. Вывести фамилии преподавателей, у которых надбавка меньше ставки в 2,5 и более раз.

Логические операторы

Операндами и результатом логических операторов являются логические значения.

Стандартными логическими операторами являются **AND, OR** и **NOT**. Их действие показано в так называемых истинностных таблицах. Использование операторов сравнения вместе с логическими операторами предоставляет возможность формулировать составные условия для отбора строк таблиц.

Использование логического оператора AND

Логический оператор **AND** во многих случаях действует как связка ‘и’ в русском языке. Рассмотрим несколько примеров с использованием этого оператора.

Запрос 6. Вывести фамилии студентов, проживающих в городе Макарово и имеющих стипендию больше 100 грн.

```
SELECT SUTFNAME
FROM   STUDENT
WHERE  CITY = 'Макарово' AND STIPEND >100;
```

Самостоятельно создайте запрос 7. **Вывести фамилии преподавателей, которые являются профессорами и ставка которых превышает 4500.**

Самостоятельно создайте запрос 8. Вывести фамилии студентов учащихся на кафедре под порядковым номером 2 (Прикладная математика) с стипендией в диапазоне 100-500 грн.

Использование логического оператора OR

Логический оператор OR во многих случаях действует как связка ‘или’ в русском языке. Рассмотрим несколько примеров.

Запрос 9. Вывести названия кафедр, расположенных либо в 1 либо в 8 корпусе.

```
SELECT NAME_KAFEDRU, NUM_KORPUSA
FROM   KAFEDRA
WHERE NUM_KORPUSA =1 OR NUM_KORPUSA =8;
```

Использование логического оператора NOT

Логический оператор NOT в русском языке передается словами ‘не’ и ‘кроме’.

Запрос 10. Вывести названия всех факультетов, кроме факультета математики и информатики.

```
SELECT NAME_FACULTETA
FROM   FACULTET
WHERE  NOT LOWER(NAME_FACULTETA) = 'математики и информатики';
```

Обратите внимание, что оператор NOT должен предшествовать выражению сравнения, а не ставиться перед оператором сравнения. То есть запись LOWER(NAME_FACULTETA) NOT ="математики и информатики" будет неверной. Учитывая, что отрицанием оператора = является оператор <>, вместо указанного условия можно было бы записать LOWER(NAME_FACULTETA) <> "математики и информатики". Это относится ко всем операторам сравнения, так как каждый из них имеет оператор, являющийся его отрицанием.

Комбинирование логических операторов

Логические операторы можно объединять, формируя составные условия. Возможность комбинирования обеспечивается тем, что любой логический оператор возвращает истинностное значение, а значит, его результат может использоваться в другом логическом операторе. Рассмотрим несколько примеров.

Запрос 11. Вывести фамилии, должность, ставку и надбавку ассистентов, у которых либо ставка меньше 550, либо надбавка больше 60.

```
SELECT NAME_TEACHER, DOLGNOST, Salary, Rise
FROM   TEACHER
WHERE LOWER(DOLGNOST) ='ассистент' AND
(Salary < 550 OR Rise > 60);
```

Использование выражений над столбцами

До сих пор в выражениях фразы WHERE мы использовали в качестве значения одного или обоих аргументов имена столбцов. Однако аргументами могут быть и выражения над столбцами.

Запрос 12. Показать фамилии преподавателей, чья зарплата (ставка плюс **надбавка) превышает 3500.**

```
SELECT NAME_TEACHER   AS ‘Фамилия преподавателя’,
  Salary + Rise AS ‘Его зарплата’
FROM   TEACHER
WHERE Salary + Rise > 3500;
```

Запрос 13. Показать фамилии преподавателей, половина зарплаты которых превышает пятикратную надбавку.

```
SELECT NAME_TEACHER
FROM    TEACHER
WHERE (Salary + Rise) / 2 > 5 * Rise;
```

Использование специальных операторов

В SQL имеются операторы сравнения, позволяющие проверять значения столбцов и выражений над ними на соответствие некоторым специальным условиям:

принадлежность множеству;

принадлежность диапазону;

соответствие шаблону;

соответствие регулярному выражению;

неопределенное значение.

В этом разделе вы узнаете, как их использовать и как с их помощью создавать составные условия.

Проверка на принадлежность множеству

Оператор IN позволяет проверить, входит ли значение в указанное множество значений. В простейшем случае этот оператор имеет следующий синтаксис:

имя_столбца [NOT] IN (список_значений)

Здесь список значений представляет собой перечень разделенных запятыми констант, тип которых должен соответствовать типу столбца, чье имя приведено слева. Семантика этого предиката такова: он принимает значение TRUE, если значение столбца соответствует одной из констант списка. Приведем пример.

Запрос 14. Вывести названия и номер корпуса кафедр, расположенных в корпусах 1, 3, 12.

```
SELECT Name_Kafedru, NUM_KORPUSA
FROM    KAFEDRA
WHERE NUM_KORPUSA IN ('1', '3', '12');
```

Использование отрицания

Так как предикат IN возвращает истинностное значение, к нему можно применить логическое отрицание. Для этого следует воспользоваться нотацией NOT IN. В этом случае предикат будет истинным, если значение столбца не входит в указанный список.

Запрос 15. Вывести названия и номер корпуса кафедр, расположенных в любых корпусах, кроме 1, 3, или 12.

```
SELECT Name_Kafedru AS ‘Название кафедры’,
  NUM_KORPUSA AS “Корпус’
FROM   KAFEDRA
WHERE  NUM_KORPUSA NOT IN (‘1', '3', '12');
```

Использование выражений над столбцами

В левой части оператора IN вместо имени столбца можно использовать любое допустимое над столбцами таблицы выражение языка.

Запрос 16. Вывести фамилии преподавателей, зарплата которых (ставка + надбавка) равна 800, 900, 1000, 1100 или 1200.

```
SELECT NAME_TEACHER AS ‘Фамилия преподавателя’,
Salary + Rise AS ‘Зарплата преподавателя’
FROM   TEACHER
WHERE Salary + Rise IN (1150, 2400, 3150, 4300);
```

Более того, элементами списка в правой части оператора IN тоже могут быть выражения над столбцами, как это показано в следующем примере:

```
Запрос 17.
SELECT NAME_TEACHER, Salary, Salary + Rise
FROM   TEACHER
WHERE Salary + Rise IN (Salary + 100, Salary + 200, Salary + 300, Salary + 400,
Salary + 500);
```

Проверка на принадлежность диапазону значений

Еще одной формой проверки вхождения элемента во множество является проверка на его принадлежность диапазону значений. Для этого применяется предикат BETWEEN, который определяет нахождение значения столбца между указанными минимальным и максимальным значениями. Синтаксис предиката следующий:

имя_столбца [NOT] BETWEEN минимум AND максимум

Проверять можно значения числовых, строковых и временных типов (для строк символов предполагается алфавитное упорядочение). Оператор BETWEEN является включающим - это означает, что крайние значения диапазона включаются в допустимые.

Запрос 18. Вывести фамилии преподавателей со ставкой в диапазоне 1000-2000.

```
SELECT NAME_TEACHER
FROM   TEACHER
WHERE  Salary BETWEEN 1000 AND 2000;
```

Использование строковых значений

Использование в операторе BETWEEN в качестве границ диапазона строковых значений имеет особенности, связанные с упорядочением (это же относится и к другим операторам сравнения).

Запрос 19. Вывести фамилии преподавателей, начинающиеся на буквы от ‘З’ до ‘Л’.

```
SELECT NAME_TEACHER
FROM   TEACHER
```

WHERE UPPER(NAME_TEACHER) BETWEEN '3' AND 'Л';

Среди строк результата нет фамилий, начинающихся на букву ‘Л’. Дело в том, что при сравнении строк символов разной длины SQL предварительно дополняет более короткую строку символами пробела, а он в упорядочениях символов предшествует всем остальным. Поэтому строка, состоящая из буквы ‘Л’ (дополненная пробелами), всегда будет меньше любой другой строки, в которой за начальной буквой 'Л’ следуют отличающиеся от пробела символы.

Чтобы это учесть, в качестве верхнего значения диапазона лучше всего указывать следующую по алфавиту букву (в данном случае — ‘М’).

Использование отрицания

Так как предикат BETWEEN возвращает истинностное значение, к нему можно применить логическое отрицание. Для этого следует воспользоваться нотацией NOT BETWEEN, в которой предикат будет истинным, только если значение столбца не входит в указанный диапазон. Представление отрицания нотацией NOT BETWEEN введено в язык для большей наглядности, так как с предикатом BETWEEN можно стандартным образом использовать логический оператор NOT (то есть ставить отрицание ко всему выражению, а не к предикату):

NOT (имя_столбца BETWEEN минимум AND максимум)

Круглые скобки в данном случае можно и опустить, так как они не меняют порядка исполнения операторов. В нотации NOT BETWEEN крайние значения в диапазон не включаются.

Запрос 20. Вывести названия и номер корпуса кафедр, которые не расположены в корпусах 1 и 3.

```
SELECT Name_Kafedru, NUM_KORPUSA
FROM    KAFEDRA
WHERE  NUM_KORPUSA NOT BETWEEN ‘1’ AND '3';

SELECT Name_Kafedru, NUM_KORPUSA
FROM   KAFEDRA
WHERE NOT (NUM_KORPUSA BETWEEN ‘1’ AND '3');
```

Использование выражений над столбцами

Как и в предикате IN, вместо имени столбца и границ диапазона можно использовать любое допустимое в языке выражение над столбцами таблицы, включая и функции.

Запрос 21. Показать фамилии преподавателей, принятых на работу между 01.01.2000 и 12.12.2001.

```
SELECT NAME_TEACHER, DATA_HIRE
FROM    TEACHER
WHERE DATA_HIRE BETWEEN ‘01/01/2000’ AND '12/12/2001';
```

Запрос 22. Вывести данные преподавателей, зарплата которых (ставка + надбавка) находится в диапазоне от удвоенной величины надбавки до утроенной надбавки плюс 50.

```
SELECT NAME_TEACHER, Salary + Rise, 2 * Rise, 3 * Rise + 50
FROM   TEACHER
WHERE Salary + Rise BETWEEN 2 * Rise AND 3 * Rise + 50;
```

Проверка на соответствие шаблону

Когда необходимо отобрать строки таблицы, в которых значение некоторого столбца совпадает с заданной строкой символов, следует использовать обычное сравнение, как это показано выше. Однако во многих случаях можно не знать точное представление в базе данных интересующего значения. Название одной и той же кафедры, например, может храниться в одном из следующих вариантов: ‘базы данных’, ‘организация баз данных’, ‘информационные системы и базы данных’, ‘базы данных и знаний’.

Такая же ситуация возникает, когда не известно точное написание фамилии преподавателя, название дисциплины, факультета и т. п. Специально для таких случаев предназначен оператор сравнения LIKE, позволяющий отобрать из таблицы строки на основе частичного соответствия. Упрощенный синтаксис оператора следующий:

имя_столбца [NOT] LIKE шаблон [ESCAPE символ_пропуска]

Его можно использовать только с символьными значениями.

Использование шаблона

Оператор LIKE сравнивает значение столбца с множеством значений, определяемых шаблоном. Он представляет собой строку, в которой помимо обычных символов, составляющих основу поискового выражения, можно использовать так называемые подстановочные символы (иногда они называются групповыми символами). Имеется всего два подстановочных символа, различающихся тем, что именно на их месте может стоять:

% — любая последовательность символов, включая их отсутствие;

_ — один любой символ.

Подстановочные символы могут находиться в любом месте шаблона в любом наборе.

Например, шаблону '%Иван%' соответствуют строки 'Иван', 'Иванов', 'Иванченко', 'Петр Иванович', а шаблону 'л_с_' - 'лист', 'леса', 'лоск' (ноне 'лес', 'листок', 'плес').

Оператор LIKE, как и все другие, работающие с символьными строками, чувствителен к регистру букв, поэтому при его использовании мы рекомендуем использовать уже известные вам функции UPPER() и L0WER().

Запрос 23. Найти фамилии преподавателей на букву ‘М’.

```
SELECT NAME_TEACHER
FROM   TEACHER
WHERE  UPPER(NAME_TEACHER) LIKE 'M%';
```

Имейте в виду, что если вы запишете условие фразы WHERE как UPPER(NAME_TEACHER) = 'M%' или даже как 'M%' LIKE UPPER(NAME_TEACHER), фамилии преподавателей будут сравниваться со строкой ' M%'. Во втором случае выражение является синтаксически правильным оператором LIKE, однако в нем строка 'M%' не выступает в качестве шаблона, так как расположена перед ключевым словом LIKE.

Запрос 24. Указать преподавателей, в фамилиях которых первой буквой является ‘М’, а четвертой – ‘ы’.

```
SELECT NAME_TEACHER
FROM   TEACHER
```

WHERE NAME_TEACHER LIKE 'М__ ы%';

Запрос 25. Вывести названия кафедр, в которых присутствует словосочетание ‘анализ’ (в различных грамматических формах).

```
SELECT Name_Kafedru
FROM   KAFEDRA
WHERE LOWER(Name_Kafedru) LIKE '%анализ%';
```

В левой части оператора LIKE может находиться не только имя столбца, но и любое допустимое над столбцами выражение, как это показано в следующем примере.

Запрос 26. Указать преподавателей, в фамилию и название должности которых входит в сумме не меньше пяти букв ‘о’.

```
SELECT NAME_TEACHER, DOLGNOST
FROM   TEACHER
WHERE LOWER(NAME_TEACHER + DOLGNOST) LIKE '%о%о%о%о%о%';
```

Проверка на неопределенное значение

Как мы уже отмечали, наличие значения NULL во фразе WHERE приводит к тому, что условие принимает истинностное значение UNKNOWN и соответствующая строка не включается в результат. Детальное описание работы с неопределенным значением вы можете найти в уроке 10, а здесь мы покажем, как обрабатывать значение NULL во фразе WHERE.

Чтобы проверить столбец на неопределенное значение, следует применить унарный оператор IS NULL, имеющий такой синтаксис:

имя_столбца IS [NOT] NULL

Этот оператор принимает истинностное значение TRUE, если столбец имеет неопределенное значение, и FALSE — в противном случае. В нотации IS NOT NULL его действие обратное.

Запрос 27. Вывести фамилии преподавателей, у которых не задан номер телефона или идентификационный код.

```
SELECT NAME_TEACHER, INDEF_KOD, TEL_TEACHER
FROM   TEACHER
WHERE INDEF_KOD IS NULL OR TEL_TEACHER IS NULL;
```

Задание для практической работы №5

Для созданной базы данных, согласно номеру варианта, самостоятельно создать **на языке Transact-SQL 15 запросов с отбором строк по условию:**

- 3 простейших запроса с использованием операторов сравнения;
- 3 запроса с использованием логических операторов AND, OR и NOT;
- 1 запрос на использование комбинации логических операторов;
- 1 запрос на использование выражений над столбцами;
- 2 запроса с проверкой на принадлежность множеству;
- 2 запроса с проверкой на принадлежность диапазону значений;
- 2 запроса с проверкой на соответствие шаблону;
- 1 запрос с проверкой на неопределенное значение.

Все программные инструкции команд SQL сохранять в файлах с расширением ***.sql** в папке ***ФИО_студента/Лаб5***.

Для каждого запроса сформулировать текстовое задание, которое должно быть выполнено к базе данных.

Создать текстовый отчет, в котором отобразить sql-команды разработанных запросов и скриншоты результатов работы из СУБД **SQL Server Management Studio**.

ПРАКТИЧЕСКАЯ РАБОТА №6. СОЗДАНИЕ МНОГОТАБЛИЧНЫХ ЗАПРОСОВ. ЗАПРОСЫ НА СОЕДИНЕНИЕ

6.1. Цель практической работы

Изучить используемый в реляционных СУБД оператор извлечения данных из таблиц. Получить навыки работы с оператором SELECT в программе ‘SQL Server Managmant Studio’.

6.2. Исходные данные

Исходными данными является индивидуальное задание и результат предыдущих практических работ.

6.3. Используемые программы

Программы ‘ SQL Server Managmant Studio ‘.

6.4. Теоретические сведения

При проектировании стремятся создавать таблицы, в каждой из которых содержалась бы информация об одном и только одном типе сущности. Это облегчает модификацию базы данных и поддержку ее целостности. Именно так мы поступили, создавая учебную базу данных. Однако сущности могут быть взаимосвязанными. Кафедры связаны с факультетами по признаку вхождения в их состав, преподаватели работают на кафедрах, студенты учатся на кафедрах и т. д.

Связь между таблицами устанавливается за счет размещения специального столбца первичного ключа одной таблицы, которая называется родительской, в другой таблице, которая называется дочерней. Столбец (или совокупность столбцов) дочерней таблицы, определенный для связи с родительской таблицей, называется внешним ключом.

Наличие внешних ключей является основной для инициирования поиска по многим таблицам.

Одна из наиболее важных особенностей предложения SELECT — это способность использования связей между различными таблицами, а также вывода содержащейся в них информации. Операция, которая приводит к соединению из двух таблиц всех пар строк, для которых выполняется заданное условие, называется соединением таблиц. Для того чтобы указать соединяемые таблицы, их следует перечислить через запятую во фразе FROM.

Декартово произведение таблиц

Соединение таблиц - это частный случай операции декартового произведения (или просто произведения). Декартово произведение двух таблиц — это таблица, состоящая из всех возможных пар строк обеих таблиц. Это определение можно естественным образом расширить на любое количество таблиц. В SQL декартово произведение выражается указанием имен перемножаемых таблиц во фразе FROM и указанием всех их столбцов во фразе SELECT.

Так, произведение таблиц FACULTET и KAFEDRA выражается следующим образом:

```
SELECT *
FROM   FACULTET, KAFEDRA
```

Так как результирующая таблица содержит много столбцов и они не помещаются по ширине страницы, мы приведем только интересующие нас столбцы произведения этих таблиц.

Запрос 28. Декартово произведение таблиц.

```
SELECT FACULTET.Name_faculteta, FACULTET. Kod_faculteta,
  KAFEDRA. Kod_faculteta, KAFEDRA.Name_Kafedru
FROM    FACULTET, KAFEDRA;
```

Каждая строка таблицы факультетов оказалась соединенной с каждой строкой таблицы кафедр, в результате получилось 27 строк (3 факультета х 9 кафедр = 27 комбинаций).

В произведении может участвовать много таблиц. Например, произведение таблиц факультетов, кафедр и преподавателей записывается следующим образом:

```
SELECT *
FROM   FACULTET, KAFEDRA, TEACHER
```

Условие соединения

Соединение таблиц может быть указано во фразе WHERE или во фразе FROM. Сначала рассмотрим первый вариант. Большинство запросов, имеющих несколько таблиц во фразе FROM, содержат фразу WHERE, в которой указаны условия, попарно сравнивающие столбцы из различных таблиц. Такое условие называется условием соединения. В этом случае SQL предполагает сцепление только тех пар строк из разных таблиц, для которых условие соединения принимает истинное значение. Теоретически при соединении сначала выполняется декартово произведение указанных таблиц в одну, а затем из нее отбираются строки согласно условию соединения. Естественно, ни одна СУБД не работает таким образом.

Фраза WHERE помимо условия соединения может также содержать другие условия, каждое из которых ссылается на столбцы соединенной таблицы. Эти условия производят отбор строк соединенной таблицы.

Соединения можно разделить на следующие категории.

Внутренние соединения (типичные операции соединения, использующие такие операторы сравнения, как = или <>). Они включают эквивалентные соединения и естественные соединения.

Внутренние соединения используют оператор сравнения для установки соответствия строк из двух таблиц на основе значений общих столбцов в каждой таблице. Примером может быть получение всех строк, в которых идентификационный номер студента одинаковый как в таблице students, так и в таблице courses.

Внешние соединения. Внешние соединения бывают левыми, правыми и полными.

Если внешние соединения задаются в предложении FROM, они указываются с одним из следующих наборов ключевых слов.

LEFT JOIN или LEFT OUTER JOIN

Результирующий набор левого внешнего соединения включает все строки из левой таблицы, заданной в предложении LEFT OUTER, а не только те, в которых соединяемые столбцы соответствуют друг другу. Если строка в левой таблице не имеет совпадающей строки в правой таблице, результирующий набор строк содержит значения NULL для всех столбцов списка выбора из правой таблицы.

RIGHT JOIN или RIGHT OUTER JOIN

Правое внешнее соединение является обратным для левого внешнего соединения. Возвращаются все строки правой таблицы. Для левой таблицы возвращаются значения NULL каждый раз, когда строка правой таблицы не имеет совпадающей строки в левой таблице.

FULL JOIN или FULL OUTER JOIN

Полное внешнее соединение возвращает все строки из правой и левой таблицы. Каждый раз, когда строка не имеет соответствия в другой таблице, столбцы списка выбора другой таблицы содержат значения NULL. Если между таблицами имеется соответствие, вся строка результирующего набора содержит значения данных из базовых таблиц.

Перекрестные с соединения

Перекрестное соединение возвращает все строки из левой таблицы. Каждая строка из левой таблицы соединяется со всеми строками из правой таблицы. Перекрестные соединения называются также декартовым произведением.

Таблицы или представления в предложении FROM могут указываться в любом порядке с внутренним соединением или полным внешним соединением. Однако важен порядок таблиц или представлений, заданных при использовании левого или правого внешнего соединения.

Соединение таблиц по равенству

Если таблицы соединяются по равенству значений пары столбцов (группы столбцов) из различных таблиц, такая операция называется соединением таблиц по равенству. Соединение по равенству, в отличие от декартового произведения, позволяет соединить только те пары строк, которые действительно взаимосвязаны друг с другом. Так, например, мы можем соединить таблицы факультетов и кафедр по условию FACULTET.Kod_faculteta = KAFEDRA.Kod_faculteta. В таком варианте мы соединяем таблицы осмысленно, так как каждая строка таблицы FACULTET соединяется только со строками соответствующих кафедр. На базе таблиц FACULTET и KAFEDRA мы получаем таблицу со столбцами из обеих таблиц, имеющую строки с понятным смыслом. Можно также сказать, что в таблицу KAFEDRA вместо столбца Kod_faculteta мы вставляем все характеристики (столбцы) соответствующего факультета из таблицы FACULTET.

Соединение таблиц используется, когда необходимо вывести значения столбцов:

- разных таблиц;
- одной таблицы, но отвечающих условию, заданному на другой таблице.

Эти два варианта, а также их комбинация, характерны для любого вида соединения, а не только по равенству. Перейдем к рассмотрению примеров.

Вывод столбцов разных таблиц

Этот вид запросов характерен тем, что фраза WHERE содержит только условие соединения, а список фразы SELECT содержит имена столбцов из различных таблиц.

Запрос 29. Вывести названия кафедр и номера их групп.

```
SELECT Name_Kafedru, [Group]
FROM   KAFEDRA, STUDENT
WHERE  KAFEDRA.kod_kafedru = STUDENT.kod_kafedru;
```

или

```
SELECT Name_Kafedru, student.[GROUP]
FROM   KAFEDRA, STUDENT
WHERE  KAFEDRA.kod_kafedru = STUDENT.kod_kafedru;
```

Мы привели два варианта запроса. В первом имена столбцов не уточняются именами таблиц, а во втором — уточняются. В данном случае это не имеет значения, оба запроса корректны.

Уточнение имен столбцов

До тех пор, пока запрос относится к одной таблице, обращение к столбцам по их именам не вызывает проблем — в таблице все имена столбцов должны быть неповторяющимися. Однако как только запрос соединяет несколько таблиц, может возникнуть неоднозначность при ссылках на столбцы с одинаковыми именами из разных таблиц. Для разрешения этой неоднозначности во фразах SELECT и WHERE (как и в некоторых других фразах) имена столбцов необходимо уточнять именами таблиц.

Запрос 30. Вывести названия факультетов и их кафедр.

```
SELECT FACULTET.NAME_FACULTETA, KAFEDRA.Name_Kafedru
FROM    FACULTET, KAFEDRA
WHERE  FACULTET.Kod_faculteta = KAFEDRA.Kod_faculteta;
```

В этом запросе мы уточнили имена столбцов во фразах SELECT и WHERE, хотя во втором случае это не обязательно, так как используются неповторяющиеся имена. Тем не менее, рекомендуем при соединении таблиц для наглядности уточнять имена столбцов. Обратите внимание на то, что в предыдущем примере отсутствует факультет математики — на нем нет кафедр.

Вывод столбцов с условием отбора

Вариант, когда отбираются строки одной таблицы, а условие задается с участием другой, используется довольно часто. Приведем примеры.

Запрос 31. Вывести названия кафедр факультета Математики и информатики.

```
SELECT KAFEDRA.Name_Kafedru AS ‘Кафедры факультета математики и информатики’
FROM    FACULTET, KAFEDRA
WHERE  FACULTET. Kod_faculteta = KAFEDRA. Kod_faculteta AND
LOWER(FACULTET.NAME_FACULTETA) = 'математики и информатики';
```

Запрос 32. Вывести фамилии доцентов кафедры информатики.

```
SELECT TEACHER.NAME_TEACHER AS ‘Доценты кафедры информатики’
FROM    KAFEDRA, TEACHER
WHERE  KAFEDRA.kod_kafedru = TEACHER. kod_kafedru AND
LOWER(KAFEDRA.Name_Kafedru) = ‘информатики' AND
LOWER(TEACHER.DOLGNOST) = 'доцент';
```

В последнем запросе помимо условия соединения используется также отбор строк по условиям, заданным для разных таблиц.

Синонимы таблиц

Синонимы таблиц часто используются для задания более лаконичного имени таблицы, по которому можно сослаться на нее в любых других местах запроса. Приведем пример.

Запрос 33. Вывести названия кафедр, на которых имеются студенты со стипендией >200 грн.

```
SELECT DISTINCT k.Name_Kafedru
FROM   KAFEDRA k, STUDENT s
WHERE k.Kod_kafedru = s. Kod_kafedru AND s.Stipend > 400;
```

Запросы по трем и более таблицам

SQL позволяет формулировать запросы, которые предполагают использование трех и более таблиц. При этом следует применять ту же методику соединения, что и для двух таблиц. Рассмотрим простой пример соединения трех таблиц.

Запрос 34. Вывести названия тех кафедр факультета математики и информатики, на которых работают профессора.

```
SELECT DISTINCT KAFEDRA.Name_Kafedru
FROM    FACULTET, KAFEDRA, TEACHER
WHERE FACULTET.Kod_faculteta = KAFEDRA.Kod_faculteta AND
KAFEDRA.Kod_kafedru = TEACHER.Kod_kafedru AND
FACULTET.Name_faculteta ='Математики и информатики' AND
TEACHER.DOLGNOST = 'профессор';
```

Для ответа на запрос необходимы три таблицы: на таблицах факультетов и преподавателей заданы условия отбора, а из таблицы кафедр следует вывести столбец названий. Поэтому три необходимые таблицы указываются во фразе FROM, а во фразе WHERE производится их соединение по условию равенства первичного и внешнего ключей:

FACULTET. Kod_faculteta = KAFEDRA. Kod_faculteta -- соединение таблиц факультетов и кафедр

KAFEDRA. Kod_kafedru = TEACHER. Kod_kafedru -- соединение таблиц кафедр и преподавателей

Таблица, образующаяся в результате соединений, будет иметь столько же строк, сколько имеется в таблице преподавателей (если все преподаватели работают на кафедрах). Выясним, почему это так, но сначала заметим, что результат соединения таблиц не зависит от порядка соединения. Поэтому рассмотрим случай, когда сначала мы соединяем таблицы кафедр и преподавателей, а затем результат соединяем с таблицей факультетов.

Так как между таблицами кафедр и преподавателей существует связь типа один-ко-многим, их соединение фактически означает приписывание к строке каждого преподавателя данных о его кафедрах. Количество строк этого соединения будет равным количеству преподавателей. Связь между таблицами факультетов и кафедр также имеет тип один-ко-многим, поэтому второе соединение означает, что к каждой строке таблицы, полученной после первого соединения, приписываются данные о факультете кафедры. Таким образом, количество строк останется равным числу преподавателей.

Вернемся к запросу. Последние два условия фразы WHERE отбирают строки из соединенной таблицы, а во фразе SELECT указан выводимый столбец. Ключевое слово DISTINCT указано в нем потому, что названия кафедр в соединенной таблице могут повторяться.

Запрос 35. Вывести фамилии ассистентов факультета математики и информатики.

```
SELECT TEACHER.NAME_TEACHER AS ‘Ассистенты ф-та математики и информатики’
FROM   FACULTET, KAFEDRA, TEACHER
WHERE FACULTET.Kod_faculteta = KAFEDRA.Kod_faculteta AND
KAFEDRA.Kod_kafedru = TEACHER.Kod_kafedru AND
FACULTET.Name_faculteta ='Математики и информатики' AND
TEACHER.DOLGNOST = 'ассистент';
```

В этом случае для ответа нужны две таблицы — факультетов и преподавателей. Однако они связаны между собой опосредованно, через таблицу кафедр. Поэтому для соединения таблиц факультетов и преподавателей следует использовать таблицу кафедр.

Сформулируем общую процедуру составления многотабличного запроса и приведем пример ее использования.

1. Определить множество таблиц, необходимых для ответа на запрос. В это множество должны входить таблицы, на столбцах которых сформулированы условия, а также те, столбцы которых необходимо вывести. Это так называемые базовые таблицы запроса.

2. В структуре взаимосвязанных таблиц найти путь, соединяющий базовые таблицы. Это так называемый путь вычисления запроса. В результате вы получите перечень таблиц, необходимых для формулировки запроса. Это так называемые таблицы запроса.

3. Во фразе FROM перечислить необходимые таблицы.

4. Во фразе WHERE соединить таблицы запроса и при необходимости задать условия отбора строк в базовых таблицах запроса.

5. Во фразе SELECT перечислить выводимые столбцы.

Вывод всех столбцов соединяемой таблицы

В многотабличном запросе конструкция SELECT * означает выбор всех столбцов соединенной таблицы. Например, результирующая таблица следующего запроса состоит из 21 столбца: 5 столбцов таблицы факультетов, 6 столбцов таблицы кафедр и 10 столбцов таблицы преподавателей.

Запрос 36.

```
SELECT *
FROM   FACULTET f, KAFEDRA k, TEACHER t
WHERE  f.Kod_faculteta = k.Kod_faculteta AND k.Kod_kafedru = t.Kod_kafedru;
```

При наличии в запросе многих таблиц конструкция SELECT * становится не очень практичной. В связи с этим в различных СУБД предоставляется возможность использовать во фразе SELECT многотабличных запросов выражение имя_таблицы.* для указания вывода всех столбцов конкретной таблицы. Например:

```
SELECT f.*,  k.FIO_ZAVKAF,  t.*
FROM   FACULTET f, KAFEDRA k, TEACHER t
WHERE  f.Kod_faculteta = k.Kod_faculteta AND k.Kod_kafedru = t.Kod_kafedru;
```

Другие виды соединений по равенству

Логическая связь между таблицами поддерживается взаимосоответствием столбцов первичного и внешнего ключей. Все рассмотренные до сих под запросы для соединения таблиц использовали именно эту связь. Однако SQL позволяет связывать таблицы по любой паре столбцов, которые имеют сравнимые типы данных, независимо от того, имеет ли эта связь какой-либо смысл. Рассмотрим ряд примеров.

Запрос 37. Если фамилия заведующего кафедры совпадает с фамилией декана какого-нибудь из факультетов, вывести название этой кафедры вместе с названием соответствующего факультета.

```
SELECT k.Name_Kafedru AS 'Название кафедры',
       f.NAME_FACULTETA AS 'Название факультета'
FROM   FACULTET f, KAFEDRA k
WHERE  f.FIO_DECANA = k.FIO_ZAVKAF;
```

Запрос 38. Вывести пары названий кафедр и фамилий преподавателей, у которых совпадают первичные ключи.

```
SELECT k.Name_Kafedru AS 'Название кафедры',
       t.Name_Teacher AS 'Фамилия преподавателя'
FROM   KAFEDRA k, TEACHER t
WHERE k.Kod_kafedru = t.Kod_kafedru;
```

Если первый запрос не лишен смысла, то последний абсолютно бессмысленный, так как в учебной базе данных первичные ключи лишены какого-либо содержания и используются только для идентификации строк своих таблиц.

Самосоединение таблицы

Как правило, взаимосвязи существуют и в пределах одной таблицы. В одних случаях эти связи являются явными, например, когда внешний ключ ссылается на первичный ключ той же самой таблицы. В других случаях эта связь присутствует неявно, например, кафедры могут быть связаны между собой на основании того свойства, что располагаются в одном корпусе.

Для ответа на такие запросы следует осуществлять соединение таблицы со своей копией. Такое соединение иногда называют самосоединением таблицы. Несмотря на кажущуюся искусственность идеи самосоединения таблиц, существует множество запросов, которые требуют именно такого соединения. На приводимых далее примерах вы убедитесь в этом.

Чтобы произвести соединение таблицы со своей копией, необходимо указать во фразе FROM имя одной и той же таблицы два или большее количество раз, а во фразе WHERE — условие их самосоединения.

Однако в этом случае возникает следующая проблема — как ссылаться на столбцы различных копий таблицы- До сих пор проблема ссылки на столбцы с одинаковыми именами из разных таблиц разрешалась уточнением имени столбца именем таблицы. В нашем же случае соединяемые таблицы имеют одинаковые имена. Для разрешения этой проблемы без синонимов таблиц уже не обойтись. В нашем случае различным вхождениям одной и той же таблицы приписываются различные синонимы и именно по этим синонимам производится обращение к столбцам. Приведем примеры использования самосоединения.

Запрос 39. Вывести фамилии преподавателей, зарплата которых больше, чем у преподавателя Сидорова.

```
SELECT needed.NAME_TEACHER
FROM    TEACHER needed, TEACHER given
WHERE  needed.Salary + needed.Rise > given.Salary + given.Rise AND
given.NAME_TEACHER = 'Игнатьева Олеся Владимировна';
```

Симметричное соединение и удаление избыточности

При самосоединении по равенству обычно возникают избыточные строки. Рассмотрим следующий запрос.

Запрос 40. Вывести названия кафедр, располагающихся в том же корпусе, что и кафедра информатики.

```
SELECT needed.Name_Kafedru
FROM  KAFEDRA needed, KAFEDRA given
WHERE  needed.NUM_KORPUSA = given.NUM_KORPUSA AND
given.Name_Kafedru = 'Информатики';
```

Обратите внимание, что в результат включена и сама кафедра информатики. Для того чтобы избавиться от ненужной результирующей строки, следует добавить условие отбора:

needed.Name_Kafedru <> ' Информатика’

При самосоединении по равенству можно получить симметричную результирующую таблицу. Суть симметричности заключается в том, что в таблице содержатся строки:

с одинаковыми значениями всех столбцов;

со всеми возможными перестановками значений столбцов.

Запрос 41. Вывести пары номеров групп, которые принадлежат одной кафедре.

```
SELECT g1.[Group], g2.[Group], g1.kod_kafedru
FROM   STUDENT g1, STUDENT g2
WHERE g1.kod_kafedru = g2.kod_kafedru;
```

ПРИМЕЧАНИЕ В этом запросе мы дополнительно вывели столбец с номером (первичным ключом) кафедры для большей наглядности.

Результирующая таблица оказалась симметричной, и в связи с этим содержит избыточные строки.

Простой способ избежать этого состоит в том, чтобы наложить ограничение на выбираемые пары значений таким образом, чтобы первое выдаваемое значение было меньше другого (или предшествовало ему в алфавитном порядке). Это делает результат асимметричным, поэтому пары с одинаковыми значениями, а также пары, заданные в обратном порядке, не будут появляться. Покажем это на примере варианта предыдущего запроса.

```
SELECT g1.[Group], g2.[Group], g1.kod_kafedru
FROM   STUDENT g1, STUDENT g2
WHERE g1.kod_kafedru = g2.kod_kafedru
 AND g1.[Group] <g2.[Group];
```

Проверка правильности данных

Самосоединение можно использовать для проверки корректности данных. Например, мы точно знаем, что в нашем вузе нет однофамильцев, занимающих разные должности. С помощью самосоединения таблицы преподавателей мы можем убедиться, что их нет и в базе данных.

Запрос 42. Указать преподавателей-однофамильцев, которые занимают различные должности.

```
SELECT tch1.NAME_TEACHER AS ‘Препод. с различ. должностями’
FROM   TEACHER tch1, TEACHER tch2
WHERE  tch1.NAME_TEACHER = tch2.NAME_TEACHER AND
tch1.DOLGNOST <> tch2.DOLGNOST;
```

Внешнее соединение таблиц

Предположим, необходимо вывести список факультетов и их кафедр. Это достигается соединением таблиц FACULTET и KAFEDRA по равенству значений первичного и внешнего ключей и выбором столбцов с названиями факультетов и кафедр. Но в таком случае, если на факультете кафедр нет, он не будет включен в результат.

Для того чтобы в списке присутствовали все факультеты, даже без кафедр, необходимо использовать внешнее соединение, которое расширяет возможности обычного соединения. Внешнее соединение возвращает строки, которые удовлетворяют условию соединения, а также те строки одной из таблиц, для которых в другой не нашлось удовлетворяющих условию соединения строк.

Внутренние соединения возвращают результат, когда в обеих таблицах есть хотя бы одна строка, соответствующая условиям соединения. Внутренние соединения исключают строки, не соответствующие ни одной строке в другой таблице. Однако внешние соединения возвращают все строки хотя бы из одной таблицы или представления, упомянутых в предложении FROM, если они удовлетворяют условиям поиска WHERE или HAVING.

Все строки, получаемые из левой таблицы, образуют левое внешнее соединение, а строки, получаемые из правой таблицы, — правое внешнее соединение. Все строки их обеих таблиц возвращаются в полном внешнем соединении.

Для внешних соединений в предложении FROM SQL Server использует ключевые слова ISO:

LEFT OUTER JOIN или LEFT JOIN;
RIGHT OUTER JOIN или RIGHT JOIN;
FULL OUTER JOIN или FULL JOIN.

Работа с левыми внешними соединениями

Рассмотрим примеры.

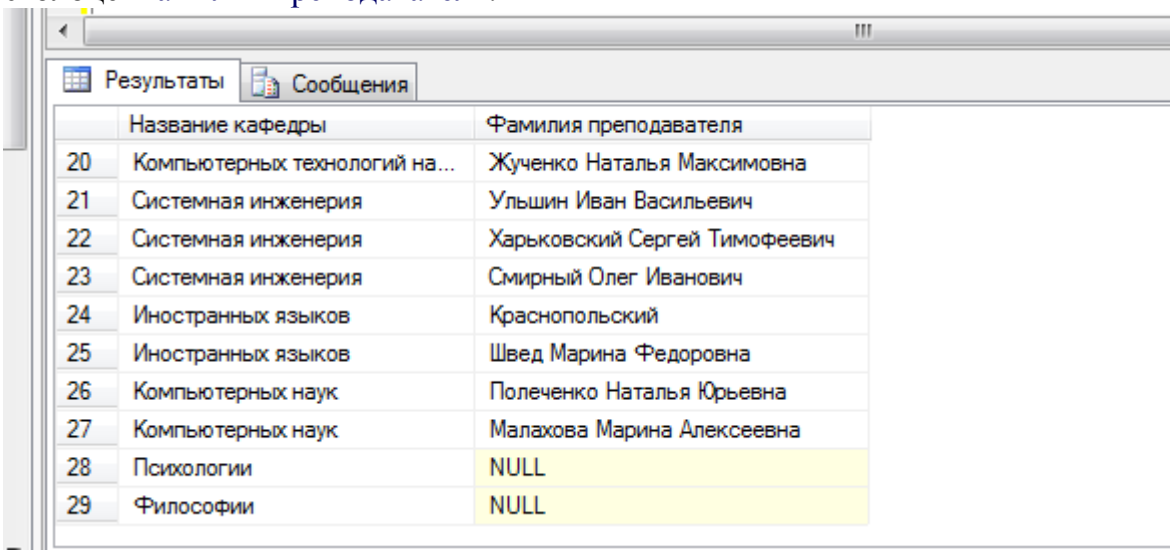
Рассмотрим соединение таблиц KAFEDRA и TEACHER по столбцам kod_kafedru. В результате будут выведены только те кафедры, для которых были написаны преподаватели.

Чтобы включить в результаты все кафедры, независимо от того, были ли написаны их преподаватели, используйте левое внешнее соединение ISO. Пример запроса:

Запрос 43. Вывести фамилии всех преподавателей с указанием их кафедры, если она есть.

```
SELECT KAFEDRA.Name_Kafedru AS 'название кафедры',
TEACHER.NAME_TEACHER AS 'фамилия преподавателя'
FROM   KAFEDRA LEFT OUTER JOIN TEACHER
ON   KAFEDRA.kod_kafedru = TEACHER.kod_kafedru;
```

Ключевые слова **LEFT OUTER JOIN** включают в вывод все строки таблицы KAFEDRA независимо от того, есть ли для них соответствующие значения в столбце kod_kafedru таблицы TEACHER. Обратите внимание на то, что в результатах, где для кафедры нет соответствующего преподавателя, строки содержат значение NULL в столбце Фамилия преподавателя.



Результаты | Сообщения

	Название кафедры	Фамилия преподавателя
20	Компьютерных технологий на...	Жученко Наталья Максимовна
21	Системная инженерия	Ульшин Иван Васильевич
22	Системная инженерия	Харьковский Сергей Тимофеевич
23	Системная инженерия	Смирный Олег Иванович
24	Иностранных языков	Краснопольский
25	Иностранных языков	Швед Марина Федоровна
26	Компьютерных наук	Полеченко Наталья Юрьевна
27	Компьютерных наук	Малахова Марина Алексеевна
28	Психологии	NULL
29	Философии	NULL

Работа с правыми внешними соединениями

Рассмотрим соединение таблиц **KAFEDRA** и **TEACHER** по столбцам **kod_kafedru**. Оператор правого внешнего соединения ISO, **RIGHT OUTER JOIN**, включает в результаты все строки второй таблицы независимо от того, есть ли для них совпадающие данные в первой таблице.

Чтобы включить в результаты всех преподавателей независимо от того, есть ли связанные с ними кафедры, используйте правое внешнее соединение ISO. Пример запроса Transact-SQL и результаты правого внешнего соединения:

Запрос 44. Вывести названия всех кафедр с указанием фамилий преподавателей, если они есть.

```
SELECT KAFEDRA.Name_Kafedru AS 'название кафедры',
TEACHER.NAME_TEACHER AS 'фамилия преподавателя'
FROM    KAFEDRA RIGHT OUTER JOIN TEACHER
ON   KAFEDRA.kod_kafedru = TEACHER.kod_kafedru;
```

Внешнее соединение и условие отбора

При внешнем соединении можно применять и дополнительные условия отбора строк. Как видно из следующих двух примеров, если условие относится к столбцам таблицы, к которой не применяется оператор внешнего соединения, то внешнее соединение происходит.

Запрос 45. Вывести названия всех кафедр корпуса 1 с указанием их преподавателей, если они есть.

```
SELECT KAFEDRA.Name_Kafedru AS 'название кафедры',
TEACHER.NAME_TEACHER AS 'фамилия преподавателя'
FROM    KAFEDRA LEFT OUTER JOIN TEACHER
ON   KAFEDRA.kod_kafedru = TEACHER.kod_kafedru
```

WHERE KAFEDRA.NUM_KORPUSA = '1';

Запрос 46. Вывести названия всех кафедр с указанием их преподавателей, если они есть, ставка которых больше 3000.

```
SELECT KAFEDRA.Name_Kafedru AS 'Название кафедры',
TEACHER.NAME_TEACHER AS 'Фамилия преподавателя'
FROM    KAFEDRA RIGHT OUTER JOIN TEACHER
ON   KAFEDRA.kod_kafedru = TEACHER.kod_kafedru
WHERE  TEACHER.salary  > 3000;
```

Работа с полными внешними соединениями

Чтобы сохранить в выводе не соответствующие друг другу строки из обеих таблиц, включив их в результаты соединения, используйте полное внешнее соединение. SQL Server предоставляет оператор полного внешнего соединения, **FULL OUTER JOIN**, включающий все строки из обеих таблиц вне зависимости от того, есть ли в них совпадающие значения.

Использование перекрестных соединений

Перекрестное соединение, не имеющее предложения WHERE, выполняет декартово произведение таблиц, вовлеченных в объединение. Размер результирующего набора декартова произведения вычисляется, как произведение количества строк в первой таблице на количество строк во второй таблице. Следующий пример показывает перекрестное соединение Transact-SQL.

```
SELECT KAFEDRA.Name_Kafedru AS 'название кафедры',
TEACHER.NAME_TEACHER AS 'фамилия преподавателя'
FROM    KAFEDRA CROSS JOIN  TEACHER
ORDER BY    KAFEDRA.kod_kafedru;
```

Результирующий набор содержит 297 строк (в **KAFEDRA** имеется 11 строк, а в таблице **TEACHER** существует 27 строк; 11, умноженное на 27, равно 297).

Внешнее соединение трех и более таблиц

В запросе может быть использовано внешнее соединение более чем двух таблиц.

Хотя в операции соединения указываются всего две таблицы, предложение FROM может содержать несколько операций объединения. Это позволяет соединять в одном запросе несколько таблиц.

При этом следует помнить, что если к столбцу таблицы A применен оператор внешнего соединения с таблицей B, то никакой другой столбец таблицы A не может содержать оператор внешнего соединения с таблицей, отличающейся от B.

В следующем примере внешнее соединение применяется для трех таблиц — факультетов, кафедр и преподавателей.

Запрос 47. Вывести список всех факультетов с указанием их кафедр и преподавателей.

```
SELECT f.NAME_FACULTETA AS 'Факультет',
k.Name_Kafedru AS 'Кафедра',
t.NAME_TEACHER AS 'Преподаватель'
FROM    FACULTET f JOIN KAFEDRA k
ON f.kod_faculteta =k.kod_faculteta
JOIN TEACHER t
ON k.kod_kafedru = t.kod_kafedru;
```

Следующий запрос Transact-SQL выполняет поиск наименований всех факультетов определенной кафедры и имена преподавателей этих кафедр.

Обратите внимание, что ни один из соединяемых столбцов — ни kod_faculteta, ни kod_kafedru, не включается в результаты. Тем не менее, соединение возможно только при использовании Kafedra в качестве промежуточной таблицы.

Среднюю таблицу соединения, Kafedra, можно назвать таблицей преобразования, или промежуточной таблицей, так как Kafedra является промежуточной точкой объединения, которая находится между двумя другими участвующими в объединении таблицами.

При наличии в инструкции нескольких операторов соединения, применяющихся либо при соединении более двух таблиц, либо при соединении более двух пар столбцов, выражения соединения могут быть связаны операторами AND или OR.

Задание для практической работы №6

Для созданной базы данных, согласно номеру варианта, самостоятельно создать **на языке Transact-SQL 15 многотабличных запросов:**

- 1 запрос с использованием декартового произведения двух таблиц;
- 3 запроса с использованием соединения двух таблиц по равенству;
- 1 запрос с использованием соединения двух таблиц по равенству и условием отбора;
- 1 запрос с использованием соединения по трем таблицам;
- создать копии ранее созданных запросов на соединение по равенству на запросы с использованием внешнего полного соединения таблиц (**JOIN**).
- 1 запрос с использованием левого внешнего соединения;
- 1 запрос на использование правого внешнего соединения;
- 1 запрос с использованием симметричного соединения и удаление избыточности.

Все программные инструкции команд SQL сохранять в файлах с расширением ***.sql** в папке ***ФИО_студента/Лаб6***.

Для каждого запроса сформулировать текстовое задание, которое должно быть выполнено к базе данных.

Создать текстовый отчет, в котором отобразить sql-команды разработанных запросов и скриншоты результатов работы из СУБД **SQL Server Management Studio**.

ПРАКТИЧЕСКАЯ РАБОТА №7. СОЗДАНИЕ ЗАПРОСОВ НА ГРУППИРОВКУ И СОРТИРОВКУ ДАННЫХ. ЗАПРОСЫ НА ИЗМЕНЕНИЕ. ИСПОЛЬЗОВАНИЕ ВСТРОЕННЫХ ФУНКЦИЙ.

7.1. Цель работы

Изучить используемый в реляционных СУБД оператор извлечения данных из таблиц SELECT и выполнение группировки и сортировки данных. Изучить синтаксис языка модификации данных. Научится использовать встроенные функции в запросах.

7.2. Исходные данные

Исходными данными является индивидуальное задание и результат предыдущих практических работ.

7.3. Используемые программы

Программа SQL Server Managmant Studio.

7.4. Задание

Практическую работу следует выполнять в следующем порядке:

1. Изучить синтаксис создания запросов с использованием функций, группировки и сортировки данных, язык манипулирования данных на примерах запросов, использовать встроенные функции к учебной базе данных "**University.mdf**".

2. Выполнить в окне "SQL Editor" **41** запросов к базе данных "**University.mdf**", согласно приведенным в практической работе образцам выполнения запросов и сохранить их в файле "**Lab7.sql**" в своей рабочей папке.

Тема 1. Создание запросов с использованием функций

Функции SQL подобны любым другим операторам языка в том смысле, что они производят действия с данными и возвращают результат в качестве своего значения. Функции имеют тип, который определяется типом возвращаемого значения, поэтому можно говорить о числовых, строковых, временных функциях и т. д. От обычных операторов функции отличаются форматом представления:

имя_функции[(аргумент[, аргумент]...)]

Этот формат допускает, что функции могут иметь ноль, один или более аргументов, причем при отсутствии аргументов круглые скобки не используются.

Имеется два основных класса функций SQL: встроенные и определяемые пользователем.

Встроенными являются функции, предопределенные в SQL. Ко второму классу относятся функции, которые пишутся пользователями на специальном языке, обеспечивающем использование всех возможностей SQL. Каждая СУБД использует для этого свой собственный язык.

SQL Server содержит множество встроенных функций, а также поддерживает создание определяемых пользователем функций.

В SQL определено множество встроенных функций различных категорий. На этом уроке мы рассмотрим:

–агрегатные (или групповые) функции, оперирующие значениями столбцов множества строк и возвращающие одно значение;

–функции одной строки, использующие в качестве аргументов значения столбцов одной строки и возвращающие одно значение.

Встроенные функции (Transact-SQL)

SQL Server содержит множество встроенных функций, а также поддерживает создание определяемых пользователем функций. Категории встроенных функций перечислены на этой странице.

Типы функций

Функция	Описание
Функции, возвращающие наборы строк.	Возвращают объект, который можно использовать так же, как табличные ссылки в SQL-инструкции.
Агрегатные функции	Обрабатывают коллекцию значений и возвращают одно результирующее значение.
Ранжирующие функции	Возвращают ранжирующее значение для каждой строки в секции.
Скалярная функция (описывается далее)	Обрабатывают и возвращают одиночное значение. Скалярные функции можно применять везде, где выражение допустимо.

Скалярные функции

Категория функции	Описание
Функции конфигурации	Возвращают сведения о текущей конфигурации.
Функции преобразования	Поддержка приведения и преобразования типов данных.
Функции работы с курсорами	Возвращают сведения о курсорах.
Функции и типы данных даты и времени	Выполняют операции над исходными значениями даты и времени, возвращают строковые и числовые значения, а также значения даты и времени.
Логические функции	Выполнение логических операций.
Математические функции	Выполняют вычисления, основанные на числовых значениях, переданных функции в виде аргументов, и возвращают числовые значения.
Функции метаданных	Возвращают сведения о базах данных и объектах баз данных.
Функции безопасности	Возвращают данные о пользователях и ролях.
Строковые функции	Выполняют операции со строковым (char или varchar) исходным значением и возвращают строковое или числовое значение.
Системные функции	Выполняют операции над значениями, объектами и параметрами экземпляра SQL Server и возвращают сведения о них.
Системные статистические функции	Возвращают статистические сведения о системе.
Функции обработки текста и изображений	Выполняют операции над текстовыми или графическими исходными значениями или столбцами и возвращают сведения о

Агрегатные функции

Аргументами агрегатных функций могут быть как столбцы таблиц, так и результаты выражений над ними. Агрегатные функции и сами могут включаться в другие арифметические выражения. В стандарте SQL определены следующие виды

агрегатных функций: унарные, бинарные, инверсного распределения, гипотетические функции множеств.

Мы будем рассматривать только определенные в стандарте SQL унарные агрегатные функции. Их перечень представлен в табл. 1.1. Конкретные СУБД расширяют этот список.

AVG - среднее

MIN - минимум

CHECKSUM_AGG - Возвращает контрольную сумму значений в группе. Значения NULL не учитываются.

SUM - сумма

COUNT - количество

STDEV – среднее квадратическое отклонение

COUNT_BIG - Возвращает количество элементов в группе.

STDEVP - Возвращает статистическое стандартное отклонение всех значений в указанном выражении.

GROUPING - Указывает, является ли указанное выражение столбца в списке GROUP BY статистическим или нет. В результирующем наборе функция GROUPING возвращает 1 (статистическое выражение) или ноль (нестатистическое выражение).

VAR - дисперсия

GROUPING_ID - Представляет собой функцию, которая вычисляет уровень группирования.

VARP - Возвращает статистическую дисперсию для заполнения всех значений в указанном выражении.

MAX - максимум

Общий формат унарной агрегатной функции следующий:

имя_функции([ALL | DISTINCT] выражение) [FILTER (WHERE условие)]

где **DISTINCT** указывает, что функция должна рассматривать только различные значения аргумента, а ALL — все значения, включая повторяющиеся (этот вариант используется по умолчанию). Фраза FILTER позволяет дополнительно отобрать строки таблицы, столбец которой используется в качестве аргумента функции.

Агрегатные функции применяются во фразах SELECT и HAVING. Здесь мы рассмотрим их использование во фразе SELECT. В этом случае выражение в аргументе функции применяется ко всем строкам входной таблицы фразы SELECT. Кроме того, во фразе SELECT нельзя использовать и агрегатные функции, и столбцы таблицы (или выражения с ними) при отсутствии фразы GROUP BY, которую мы рассмотрим в теме 2.

Функция COUNT

Функция COUNT имеет два формата. В первом случае возвращается количество строк входной таблицы, во втором случае — количество значений аргумента во входной таблице:

C0UNT(*)

C0UNT([DISTINCT | ALL] выражение)

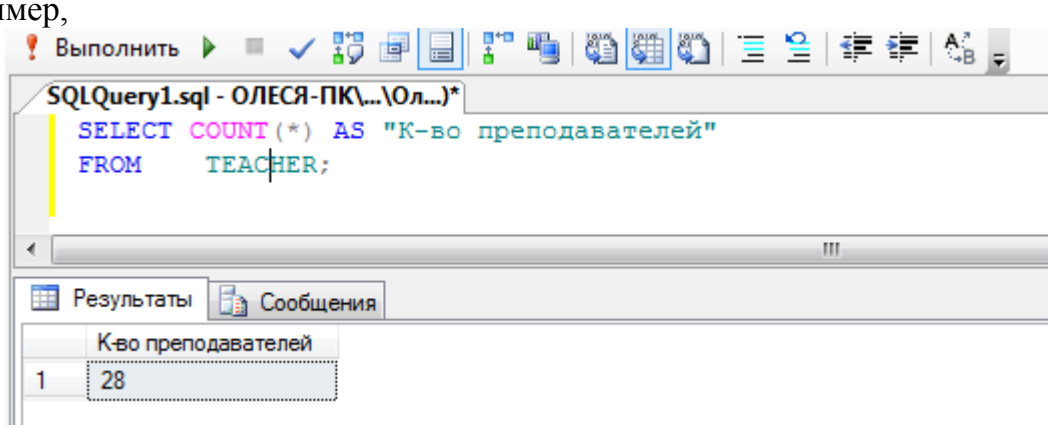
Простейший способ использования этой функции - подсчет количества строк в таблице (всех или удовлетворяющих указанному условию). Для этого используется первый вариант синтаксиса.

Создайте новый запрос, введите sql-запрос, выполните его, сохраните его в рабочую папку ЛАБ7_SQL под именем 1.sql.

Запрос 1. Информация о скольких преподавателях имеется в базе данных?

```
SELECT COUNT(*) AS "К-во преподавателей"
FROM    TEACHER;
```

Чтобы выполнить sql-команду нажмите на панели редактора кнопку Выполнить. В результате выполнения данного кода будет подсчитано кол-во всех преподавателей. Например,



Самостоятельно создать запрос 2. Сколько ассистентов не имеют телефонов?

Запросы с агрегатными функциями можно строить и с использованием нескольких таблиц, так как входная таблица и в этом случае будет только одна.

Самостоятельно создать запрос 3. Сколько кафедр на факультете математики и информатики?

Во втором варианте синтаксиса функции COUNT в качестве аргумента может быть использовано имя отдельного столбца. В этом случае подсчитывается количество либо всех значений в этом столбце входной таблицы, либо только неповторяющихся (при использовании ключевого слова DISTINCT).

Запрос 4. На скольких различных должностях работают преподаватели кафедры «Компьютерные системы и сети»?

```
SELECT C0UNT(DISTINCT DOLGNOST)
FROM   KAFEDRA d, TEACHER t
WHERE  d.KOD_KAFEDRU =  t.KOD_KAFEDRU AND
LOWER(d.NAME_KAFEDRU) = 'Компьютерные системы и сети';
```

Функция SUM

Эта агрегатная функция подсчитывает сумму значений аргумента для всех строк входной таблицы. Аргумент должен иметь числовой тип или быть временным промежутком. В качестве аргумента может выступать имя столбца или выражение над столбцами входной таблицы. В этой функции также допускается использовать ключевые слова DISTINCT и ALL. Приведем примеры.

Запрос 5. Какая суммарная ставка всех ассистентов?

```
SELECT SUM(Salary)
FROM   TEACHER
WHERE LOWER(DOLGNOST) = 'ассистент';
```

Функция AVG

Агрегатная функция AVG подсчитывает среднее значение аргумента для всех строк входной таблицы. Аргумент должен иметь числовой тип или быть временным промежутком. В качестве аргумента может выступать имя столбца или выражение над столбцами входной таблицы. Допускается использовать ключевые слова DISTINCT и ALL. Приведем ряд примеров.

Самостоятельно создать запрос 6. Какая средняя ставка среди всех преподавателей?

Самостоятельно создать запрос 7. Какое среднее значение ставки в вузе?

В данном запросе используйте ключевое слово DISTINCT, чтобы применить AVG не ко всем имеющимся в таблице TEACHER ставкам, а только к различным значениям ставки.

Функции MIN и MAX

Эти функции позволяют находить максимальное (MAX) и минимальное (MIN) значения аргумента для всех строк входной таблицы. Хотя и в этом допускается использование ключевых слов DISTINCT и ALL, они не оказывают влияния на результат. Аргумент этих функций может быть любого типа, для которого определено упорядочение, то есть числовой, строковый и временной.

Запрос 8. Какова максимальная зарплата преподавателя ?

```
SELECT MAX(Salary + Rise)
FROM   TEACHER;
```

Самостоятельно создать запрос 9. Когда в последний раз (максимальная дата приема на работу) принимали на работу преподавателя на кафедру информатики?

Выражения с использованием агрегатных функций

Агрегатные функции не только могут иметь выражение в своем аргументе, но и сами могут использоваться в выражениях.

Запрос 10. Вывести процентное соотношение суммарной ставки к суммарной зарплате и наоборот.

```
SELECT SUM(Salary)*100/SUM(Rise) AS "Процент зарплаты к зарплате",
SUM(Rise)*100/SUM(Salary) AS "Процент зарплаты к ставке"
FROM    TEACHER;
```

Однострочные функции

Напомним, что эти функции используют в качестве аргумента одно значение (одного столбца одной строки таблицы) и возвращают в качестве своего результата также единственное значение. Мы рассмотрим эти функции по типам их аргументов.

Строковые функции

Эти функции используют в качестве аргумента строку символов и в качестве результата возвращают также символьную строку. Стандарт SQL предлагает варианты таких функций и для двоичных строк.

Функции UPPER, LOWER

Эти функции мы уже рассматривали и многократно использовали. Они имеют следующий формат:

UPPER(строка)

LOWER(строка)

Приведем для них примеры .

Запрос 11. Вывести фамилии всех преподавателей прописными буквами.

```
SELECT UPPER(NAME_TEACHER) AS "Все прописные”
FROM  TEACHER;
```

Аналогично можно вывести все фамилии преподавателей строчными буквами.

Числовые функции над числами

Эти функции возвращают числовые значения на основании заданных в аргументе значений того же типа. Числовые функции используются для обработки данных, а также в условиях их поиска. Стандарт SQL предлагает ряд числовых функций с очевидной семантикой. Часть функций перечислены ниже:

ABS абсолютное значение

DEGREES Возвращает для значения угла в радианах соответствующее значение в градусах.

RAND – Возвращает псевдослучайное значение типа float от 0 до 1.

EXP экспонента

ROUND - Возвращает числовое значение, округленное до указанной длины или точности.

FLOOR Возвращает наибольшее целое число, меньшее или равное указанному числовому выражению.

LOG логорифм

SIN - синус

LOG10 десятичный логорифм

SQRT – корень квадратный

PI число 3.14

SQUARE – квадрат числа

POWER Возвращает значение указанного выражения, возведенное в заданную степень.

TAN - тангенс

Особой функцией является WIDTH_BUCKET, с помощью которой можно легко строить гистограммы:

WIDTH_BUCKET(число, миминум, максимум, количество)

Некоторые СУБД расширяют приведенный выше набор функций, включая другие числовые функции, например вычисления обычных и гиперболических тригонометрических функций.

Временные функции

Эти функции используют в качестве аргумента типы даты, времени, временной отметки или временного промежутка. Тип возвращаемого значения не всегда соответствует типу аргумента.

Функции даты и времени в Transact-SQL деляться на

Функции, получающие значения системной даты и времени

Функции, получающие компоненты даты и времени

Функции, получающие значения даты и времени из их компонентов

Функции, получающие разность даты и времени

Функции, изменяющие значения даты и времени

Функции, устанавливающие или получающие формат сеанса
Функции, проверяющие значения даты и времени.

Функции, получающие компоненты даты и времени.

Функция извлекает из операнда указанный компонент и возвращает его в виде числа.

DATENAME (datepart , date)

Здесь date - это выражение временного типа, а datepart - временная единица, которая может иметь одно из следующих значений: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND и т.д.

DATEPART (datepart,date) - Возвращает целое число, представляющее указанный компонентdatepart указанной даты date.

DAY (date) - Возвращает целое число, представляющее день указанной даты date.

MONTH (date) - Возвращает целое число, представляющее месяц указанной даты date.

YEAR (date) - Возвращает целое число, представляющее год указанной даты date.

Рассмотрим пример.

Запрос 12. Вывести фамилии всех преподавателей родившихся в 1979 году.

```
SELECT Name_teacher, BIRTHDAY
FROM   TEACHER
WHERE DATENAME(YEAR, BIRTHDAY)=1979;
```

Функции, получающие значения системной даты и времени

Функция CURRENT_TIMESTAMP - Возвращает значение типа datetime2(7), которое содержит дату и время компьютера, на котором запущен экземпляр SQL Server. Смещение часового пояса не включается.

Эта функция возвращает текущую дату. Аргументов она не имеет.

Функция GETDATE ( ) Возвращает значение типа datetime2(7), которое содержит дату и время компьютера, на котором запущен экземпляр SQL Server. Смещение часового пояса не включается.

Функция GETUTCDATE ( ) Возвращает значение типа datetime2(7), которое содержит дату и время компьютера, на котором запущен экземпляр SQL Server. Возвращаемые дата и время отображаются в формате UTC.

Многие СУБД существенно расширяют список функций, оперирующих датой и временем. Далее мы приведем некоторые из важных функций этого типа, которые используются в Oracle. Напоминаем, что тип DATE в Oracle содержит в себе как дату, так и время.

Функции, получающие значения даты и времени из их компонентов

Функция DATEADD (datepart, number , date) Возвращает новое значение datetime, добавляя интервал к указанной части datepart заданной даты date.

Добавляет к дате, указанной в первом аргументе, количество месяцев второго аргумента.

Dateadd (компонент, кол-во , дата)

Здесь кол-во - это количество прибавляемых лет, месяцев, дней и т.д., а компонент - временная единица, которая может иметь одно из следующих значений: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND.

Например, DATEADD(month, 1, '2006-08-30')

Запрос 13. Осуществить пересчет даты приема на работу преподавателя на фамилию начинающуюся на букву С в сторону увеличения на 3 месяца.

```
SELECT NAME_TEACHER, DATA_HIRE AS ' Дата приема ',
DATEADD(month, 3, DATA_HIRE) AS ' Плюс 3 месяца '
FROM    TEACHER
WHERE  (NAME_TEACHER) LIKE 'C%';
```

Функция EOMONTH

EOMONTH (start_date [, month_to_add])

Возвращает дату последнего дня того месяца, который указан в аргументе. Обычно используется для определения, сколько дней осталось до конца месяца.

LAST_DAY(дата)

Функция DATEDIFF

DATEDIFF (datepart , startdate , enddate)

Возвращает количество пересеченных границ (целое число со знаком), указанных аргументом datepart, за период времени, указанный аргументами startdate и enddate.

Запрос 14. Например, если вы хотите узнать, сколько месяцев уже проработал Статывка, можно выполнить такой запрос:

```
SELECT 'Статывка проработал ' ||
R0uND(DATEDIFF(month,GETDATE(), DATA_HIRE),1) ||
' месяцев' AS "Стаж Статывки"
FROM    TEACHER
WHERE NAME_TEACHER LIKE 'Статыв%';
```

Функция NEXT_DAY

Возвращает ближайшую к первому параметру дату, в которой название дня недели совпадает с указанным во втором параметре.

NEXT_DAУ(дата, день_недели)

Функции преобразования

Стандарт SQL предлагает единственную функцию преобразования данных из одного типа в другой — это функция CAST.

Функция CAST и CONVERT

Производит преобразование выражения, заданного первым аргументом, в тип, заданный вторым аргументом. Преобразование допускается только для определенных пар типов данных.

CAST (expression AS data_type [(length)])

CONVERT (data_type [(length)] , expression [, style])

Например

CAST(10.3496847 AS money)

CAST(10.6496 AS int)

Тема 2. Группировка и сортировка

В этом уроке мы рассмотрим еще три фразы предложения SELECT, а именно:

HAVING, GROUP BY и ORDER BY.

Первая из них позволяет группировать строки таблицы и применять к созданным группам агрегатные функции.

Рассмотрим простейшие варианты группировки. Фраза HAVING используется вместе с фразой GROUP BY и позволяет формулировать условия на группах строк для дополнительного отбора.

Наконец, фраза ORDER BY позволяет сортировать строки результирующей таблицы.

Запросы с группировкой строк

Часто при создании отчетов появляется необходимость в формировании промежуточных итоговых значений, то есть относящихся к данным не всей таблицы, а ее частей.

Именно для этого предназначена фраза GROUP BY. Она позволяет все множество строк таблицы разделить на группы по признаку равенства значений одного или нескольких столбцов (и выражений над ними).

Фраза GROUP BY должна располагаться вслед за фразой WHERE (если она отсутствует, то за фразой FROM).

Общий синтаксис фразы GROUP BY следующий:

GROUP BY выражение[, выражение]...

При наличии фразы GROUP BY фраза SELECT применяется к каждой группе, сформированной фразой группировки. В этом случае и действие агрегатных функций, указанных во фразе SELECT, будет распространяться не на всю результирующую таблицу, а только на строки в пределах каждой группы. Каждое выражение в списке фразы SELECT должно принимать единственное значение для группы, то есть оно может быть:

- константой;
- агрегатной функцией, которая оперирует всеми значениями аргумента в пределах группы и агрегирует их в одно значение (например, в сумму);
- выражением, идентичным стоящему во фразе GROUP BY;
- выражением, объединяющим приведенные выше варианты.

Рассмотрим возможности фразы GROUP BY, переходя от простых вариантов ее использования к более сложным.

Группировка по одному столбцу

Группировка по значениям одного столбца является самым простым вариантом использования фразы GROUP BY. Приведем примеры.

Запрос 15. Для каждого корпуса подсчитать количество находящихся в нем кафедр.

```
SELECT NUM_KORPUSA AS “Корпус”,
COUNT(*) AS "К-во кафедр"
FROM   KAFEDRA
GROUP BY NUM_KORPUSA ;
```

Самостоятельно создать запрос 16. Для каждой из должностей указать суммарный фонд заработной платы.

Если в запросе используются фразы и WHERE, и GROUP BY, строки, не удовлетворяющие условию фразы WHERE, исключаются до выполнения группировки. Вследствие этого группировка производится только по тем строкам, которые удовлетворяют условию.

В случае многотабличных запросов сначала производится соединение таблиц, а затем их группировка. Приведем примеры.

Самостоятельно создать запрос 17. Для каждого факультета, расположенного в корпусе 1, вывести количество групп и общее количество студентов по каждой кафедре.

Группировка по нескольким столбцам

SQL позволяет группировать строки таблицы и по нескольким столбцам. В этом случае имена столбцов перечисляются во фразе GROUP BY через запятую.

Запрос 18. Для каждого факультета, расположенного в корпусе 1, вывести сколько учится студентов по каждой группе.

```
SELECT f.Name_faculteta,
s."GROUP", count(s."GROUP")   AS "Кол-во студентов в группе"
FROM    FACULTET f, KAFEDRA d, STUDENT s
WHERE   f.KOD_FACULTETA = d.KOD_FACULTETA   AND
d.KOD_kafedru = s.KOD_kafedru AND
d.NUM_KORPUSA  = '1'
GROUP BY f.Name_faculteta,s."GROUP";
```

Самостоятельно создать запрос 19. Для каждой кафедры и должности вывести суммарную и среднюю зарплату преподавателей.

Даже при группировке по двум и более столбцам этот вариант фразы GROUP BY обеспечивает только один уровень группировки. Так, приведенный выше запрос обеспечивает только одну итоговую строку для пары значений кафедра-должность.

Использование выражений

Хотя стандарт SQL не допускает группировку по выражениям над столбцами, некоторые СУБД такую возможность предоставляют. В этом случае во фразе SELECT также можно использовать выражение группировки, однако нельзя выводить по отдельности столбцы, участвующие в этом выражении.

Запрос 20. Для каждого значения зарплаты, не превышающего 1500, вывести это значение и количество преподавателей, такую зарплату получающих.

```
SELECT Salary + Rise, C0UNT(*)
FROM    TEACHER
WHERE  Salary + Rise <= 1500
GROUP BY Salary + Rise;
```

Вложение агрегатных функций

Если фраза GROUP BY в запросе отсутствует, то во фразе SELECT нельзя вкладывать агрегатные функции друг в друга. Например, следующий запрос приведет к ошибке:

```
SELECT AVG(MIN(Salary))
FROM    TEACHER;
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

ORA-00978: вложенная групповая функция без GROUP BY

Однако при наличии фразы GROUP BY такое вложение допускается. Оно интерпретируется следующим образом: сначала для каждой группы выполняется вложенная агрегатная функция, затем к полученной таким образом промежуточной таблице применяется внешняя агрегатная функция. Двойное вложение, например MAX(AVG(MIN(Salary))), недопустимо. Приведем пример.

Запрос 21. Вывести среднее значение среди минимальных и максимальных ставок для каждой группы преподавателей, занимающих одну должность, а также минимальное и максимальное значения среди средних ставок.

```
SELECT AVG(MIN(Salary)) AS AVG_MIN,
AVG(MAX(Salary)) AS AVG_MAX,
MIN(AVG(Salary)) AS MIN_AVG,
MAX(AVG(Salary)) AS MAX_AVG
FROM   TEACHER
GROUP BY Dolgnost ;
```

Условие отбора групп

Предположим, что нужно вывести номера кафедр, у которых суммарное количество работающих профессоров более 1. Приведенная ниже формулировка запроса является неверной:

```
SELECT KOD_kafedru
FROM   TEACHER
WHERE  count(dolgnost) > 1 and dolgnost=’профессор’
GROUP BY KOD_kafedru;
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
WHERE  count(dolgnost) > 3 and dolgnost=’профессор’;
*
```

Ошибка в строке 3;

CRA-00934: групповая функция здесь не разрешена

Дело в том, что фраза WHERE проверяет на соответствие условию строки исходных таблиц, а мы указали в ней агрегатную функцию. Для отбора строк среди полученных групп следует применять фразу HAVING. Она играет такую же роль для групп, что и фраза WHERE для исходных таблиц, и может использоваться лишь при наличии фразы GROUP BY. В предложении SELECT фразы WHERE, GROUP BY и HAVING обрабатываются в следующем порядке.

Фразой WHERE отбираются строки, удовлетворяющие указанному в ней условию.

Фраза GROUP BY группирует отобранные строки.

Фразой HAVING отбираются группы, удовлетворяющие указанному в ней условию. В связи с вышесказанным, предыдущий запрос необходимо записать так.

Перепишем тогда запрос так:

Запрос 22. Вывести номера кафедр, у которых суммарное количество работающих профессоров более 1.

```
SELECT KOD_kafedru as "Номер кафедры" ,Count(*) as "Кол-во профессоров на кафедре"
FROM   TEACHER
WHERE   dolgnost='профессор'
GROUP BY KOD_kafedru
having count(dolgnost) > 1  ;
```

Использование столбцов группировки во фразе HAVING

Рассмотрим использование во фразе HAVING условий отбора, заданных для группируемых столбцов (или выражений над ними). Для этого усложним предыдущий запрос.

Запрос 23. Вывести названия кафедр факультета математики и информатики, на которых работают один и более профессоров. Указать также количество профессоров и их суммарную зарплату.

```
SELECT d.Name_kafedru, Count(*), SUM(t.salary + t.Rise)
FROM    FACULTET f, KAFEDRA d, TEACHER t
WHERE   f.KOD_FACULTETA = d.KOD_FACULTETA AND
d.KOD_kafedru = t.KOD_kafedru AND
LOWER(f.Name_faculteta) = 'математики и информатики' AND
LOWER(t.Dolgnost ) = 'профессор'
GROUP BY d.Name_kafedru
HAVING COUNT(*) > 0;
```

Фраза HAVING без фразы GROUP BY

Выше мы указали, что фраза HAVING может использоваться лишь при наличии фразы GROUP BY. Из этого правила синтаксис SQL допускает только одно исключение: когда вся таблица интерпретируется как одна группа. В этом случае в списке фразы SELECT можно использовать только константы, агрегатные функции и выражения над ними. Приведем примеры.

Запрос 24. Если суммарная зарплата всех преподавателей превышает 15 000, вывести их минимальную ставку, максимальную надбавку и суммарную зарплату.

```
SELECT MIN(Salary), MAX(Rise), SUM(Salary + Rise)
FROM    TEACHER
HAVING SUM(Salary + Rise) > 15000;
```

При наличии фразы WHERE сначала производится отбор строк согласно ее условию, и только после этого применяется условие фразы HAVING.

Запрос 25. Если суммарная зарплата всех ассистентов превышает 2500, вывести их среднюю ставку, среднюю надбавку и суммарную зарплату.

```
SELECT AVG(Salary), AVG(Rise), SUM(Salary + Rise)
FROM    TEACHER
WHERE LOWER(Dolgnost ) = 'ассистент'
HAVING SUM(Salary + Rise) > 2500;
```

На практике фраза HAVING очень редко используется без фразы GROUP BY, из-за чего такая возможность предоставляется не во всех СУБД.

Сортировка результирующих строк

Как мы уже отмечали, строки в таблицах базы данных неупорядочены. Также неупорядочены и строки результирующей таблицы запроса, однако для их упорядочения в предложении SELECT можно воспользоваться фразой ORDER BY. Она сортирует по значению указанных в ней столбцов (и выражений над столбцами) строки результирующей таблицы запроса. Синтаксис этой фразы следующий:

ORDER BY спецификация_сортировки[. спецификация_сортировки]...

где спецификация_сортировки имеет такой синтаксис:

выражение_сортировки [направление_сортировки] [положение_NULL]

Сортировать можно по столбцам (выражениям) тех типов, для которых определены операции сравнения. Это относится, в частности, к символьным строкам, числам и временным значениям. Можно указывать направление сортировки и место расположения строк, имеющих значение NULL для выражений сортировки.

Далее в этом уроке мы рассмотрим общие способы упорядочения результирующих строк.

Сортировка по столбцу или выражению

Сортировать строки результирующей таблицы запроса можно по отдельным столбцам, совокупности столбцов, а также по одному или нескольким выражениям над столбцами. Ниже рассматриваются все эти варианты.

Сортировка по столбцу

Простейший вариант сортировки - это сортировка по одному из столбцов результирующей таблицы.

Запрос 26. Вывести алфавитный список фамилий профессоров и доцентов.

```
SELECT NAME_TEACHER
FROM    TEACHER
wheRE  LOwER(Dolgnost ) =’профессор’ OR
LOWER(Dolgnost ) = 'доцент'
ORDER BY NAME_TEACHER;
```

Сортировка по выражению над столбцами

Упорядочивать строки можно не только по значению столбца, но и по значению выражения над столбцами.

Запрос 27. Вывести фамилии ассистентов и их зарплату по ее возрастанию.

```
SELECT Name_teacher, Salary + Rise
FROM   TEACHER
WHERE LOWER(Dolgnost ) = 'ассистент’
ORDER BY Salary + Rise;
```

Направление сортировки

Во всех до сих пор приводимых примерах сортировка производилась в порядке возрастания значений. В SQL такой порядок определен по умолчанию. Однако есть возможность и явно указать направление сортировки с помощью ключевых слов ASC (по возрастанию) и DESC (по убыванию), которые следует располагать после имени сортируемого столбца (выражается).

Запрос 28. Вывести фамилии ассистентов и дату их приема на работу по возрастанию даты.

```
SELECT Name_teacher, Data_hire
FROM    TEACHER
WHERE  LOWER(Dolgnost) = 'ассистент'
ORDER BY Data_hire ASC;
```

Самостоятельно создать запрос 29. Вывести фамилии доцентов в обратном алфавитном порядке и их зарплату.

Тема 3. Внесение изменений в базу данных

7.4.1. Добавление новых данных

Новые данные добавляются оператором INSERT. Наименьшей единицей информации, которую можно добавить в реляционную базу данных, является одна строка таблицы.

Немного упрощенный синтаксис оператора INSERT имеет вид:

INSERT INTO Имя_Таблицы [(Колонка [, Колонка ...])]
{VALUES(<величина> [, <величина> ...]) | <оператор SELECT>};
<величина> = {:Переменная | <константа> | <выражение>
| <функция> | udf([<величина> [, <величина> ...]])
| NULL | USER}

<константа> = Число | 'Строка'

<функция> = CAST(<величина> AS <тип данных>)
UPPER(<величина>)
GEN_ID(Имя_Генератора, <величина>)

<выражение> = SQL выражение, возвращающее единичное значение

В этом описании можно выделить два варианта оператора:

1. Вставка одной строки. Для этого после ключевого слова VALUES в круглых скобках указывают вставляемые величины.
2. Вставка в таблицу нескольких строк, выбранных с помощью оператора SELECT * .

В этой практической работе рассматривается только первый вариант оператора INSERT.

Пример, когда в качестве вставляемых величин применены константы:

INSERT INTO Person(Pr_ID, Pr_LastName, Pr_FirstName)
VALUES(150, 'Иванов', 'Петр');

7.4.2. Удаление существующих данных

Для удаления строк из таблицы используется оператор DELETE. Вот его упрощенный синтаксис:

DELETE FROM Имя_Таблицы
[WHERE <условие поиска>];

<условие поиска> = как в операторе SELECT

Если не использовать предложение WHERE, то будут удалены все строки в таблице.

-- Удаление всех служащих:
DELETE FROM Employee;

-- Удаление всех людей с номерами 150 и больше:
DELETE FROM Person WHERE Pr_ID >= 150;

Отбирать строки для удаления не обязательно только на основании содержимого этих строк. Можно составить условие для удаляемых строк, опираясь на данные из других таблиц. Для составления таких условий необходимо сначала изучить оператор SELECT.

7.4.3. Обновление существующих данных

Оператор UPDATE обновляет значения одного или нескольких столбцов в выбранных строках одной таблицы. Строки для обновления указываются в предложении WHERE. Если пропустить предложение WHERE, то изменятся все строки таблицы.

UPDATE Имя_Таблицы
SET Колонка = <величина>[,
Колонка = <величина>...]
[WHERE <условие поиска>]
<величина> = { Колонка | :Переменная | <константа>
| <выражение> | <функция>
| udf([<величина> [, <величина> ...]]) | NULL | USER}
<выражение> = SQL выражение, возвращающее единичное значение
<условие поиска> = как в операторе SELECT

Примеры:

-- Увеличить зарплату всем служащим на 10%:
UPDATE Employee
SET Salary = 1.1*Salary;

/* Увеличить зарплату всем служащим, которые имеют
зарплату меньше 10000 на 15%: */
UPDATE Employee
SET Salary = 1.15*Salary;
WHERE Salary <= 10000;

Отбирать строки для изменения, как и для удаления, можно с использование подчиненного запроса SELECT, который позволит учитывать в условии поиска изменяемых строк данные из других таблиц.

Например, можно выполнить такой запрос: увеличить зарплату на 10% всем служащим, работающим в отделе продаж, которые обслужили за последний месяц клиентов больше чем в полтора раза, чем в среднем по их отделу.

7.5. Задание

Добавление новых строк

Предложение INSERT вставляет строки в таблицу базы данных. Есть три разновидности этой команды:

INSERT VALUES
INSERT SELECT
INSERT DESALT VALUES

Первая из них производит вставку в таблицу явно заданной строки, вторая разновидность – вставку группы строк, выбранных в результате выполнения запроса, а третья — вставку значений по умолчанию.

Вставка отдельных строк

Предложение INSERT... VALUES выполняет вставку в таблицу одной строки. Его удобно использовать для небольших операций, когда в таблицу нужно вставить несколько строк. Синтаксис этого предложения следующий:

INSERT INTO имя_таблицы [<имя_столбца[, имя_столбца]...)]
VALUES (значение[, значение]...);

Указание вставляемых столбцов

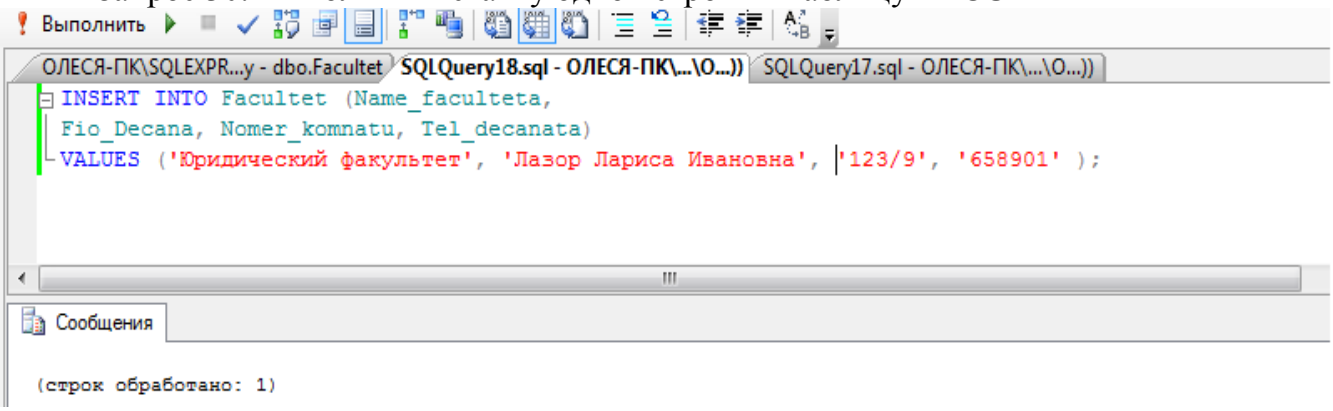
Этот формат предполагает указания имени таблицы, в которую производится вставка, списка имен столбцов, в которые будут вставляться значения, и списка собственно вставляемых в строку значений. При этом следует придерживаться следующих правил:

–вставляемые данные должны согласовываться с типами данных указанных столбцов;

–размеры данных должны соответствовать размерам столбцов;

–порядок данных во фразе VALUES должен соответствовать порядку перечисления столбцов.

Запрос 30. Выполнить вставку одной строки в таблицу FACULTET



В таблицу вставляется строка со значениями, указанными в списке фразы VALUES, причем расположение значений в списке соответствует расположению соответствующих столбцов в списке столбцов таблицы.

Примечание. Чтобы данные были добавлены, не забудьте нажать на кнопки , а затем для подтверждения внесенных изменений в таблицу. Просмотрите внесенные изменения в таблицу FACULTET.

В этом примере мы перечислили столбцы в том порядке, в каком они были определены при создании таблицы, однако это не обязательно. При желании порядок перечисления имен столбцов в команде INSERT можно изменить.

Список имен столбцов может быть не полным. Можно указывать только те из них, значения которых известны для вставляемой строки, Столбцы, отсутствующие в списке, будут принимать значения NULL для вводимой строки.

Самостоятельно создайте запрос 31. Выполнить вставку одной строки в таблицу KAFEDRA для столбцов name_kafedru, fio_zavkaf, kod_faculteta с данными 'Психологии', 'Иванова', 5.

В этом примере в водимой строке отсутствуют значения столбцов Nomer_komnatu, Num_korpusa и Tel_kafedru. В базе данных они примут значение NULL.

Поддержка ограничений целостности

Помните, некоторые из столбцов или наборов столбцов могут иметь ограничения целостности PRIMARY KEY и NOT NULL. Такие столбцы не могут принимать значения NULL.

Приведенные выше рассуждения относятся ко всем ограничениям целостности, определенным для таблиц. При попытке ввода данных (как, впрочем, и при обновлении и удалении) СУБД проверяет возможное нарушение объявленных ограничений целостности. И если это так, команда будет отклонена с выдачей соответствующего уведомления.

Использование выражений

В качестве вставляемых значений могут использоваться выражения.

Самостоятельно создайте запрос 32.Ввести в таблицу TEACHER данные (50, 10, ' Капуста Леонид Владимирович',
1271, 1271/3, ‘доцент’, GETDATE()-1)

Здесь мы указали, что надбавка равна третьей части ставки (1271 / 3),а дата приема на работу на один день меньше текущей даты (CURRENT_DATE -1).

Результат запроса в качестве вставляемого значения

Вместо вставляемого значения можно использовать запрос. Это оказывается очень удобным в том случае, когда вставляемое значение присутствует в базе данных.

Запрос 33.Например, в следующем предложении в качестве фамилии заведующего вновь вставляемой кафедры выбирается фамилия декана факультета «Компьютерных наук и технологий» .

```
INSERT INTO KAFEDRA (name_kafedru, kod_faculteta, fio_zavkaf)
VALUES ( 'Философии', 5, (SELECT fio_decana FROM FACULTET
WHERE LOWER(Name_faculteta) = 'международный'));
```

Обновление существующих данных

Целью предложения обновления является изменение значений отдельных столбцов всех или удовлетворяющих указанному условию строк таблицы. Упрощенный синтаксис предложения следующий:

```
UPDATE имя_таблицы [[AS] синоним]
SET имя_столбца = выражение[, имя_столбца = выражение]...
[WHERE условие];
```

Его элементы означают следующее:

имя_таблицы — имя обновляемой таблицы;

синоним — синоним обновляемой таблицы для ссылки на нее в подзапросе;

имя_столбца - имя обновляемого столбца;

выражение - допустимое в SQL выражение соответствующего типа, значение которого присваивается обновляемому столбцу;

условие - допустимое в SQL выражение условия, которое используется для отбора обновляемых строк.

По одному предложению UPDATE обновлению подвергаются строки только одной базовой таблицы.

Обновление всех строк

Как видно из определения синтаксиса команды UPDATE, фраза WHERE является факультативной. При ее отсутствии все строки таблицы подвергаются обновлению согласно фразе SET.

Во фразе SET можно одновременно изменять значения нескольких столбцов таблицы.

Запрос 34. Например, в следующем примере всем преподавателям увеличивается ставка на 12 % и надбавка на 7 %:

```
UPDATE TEACHER
SET    Salary = Salary + Salary * 0.12, Rise = Rise + Rise * 0.08;
```

Во фразе SET в правой части оператора присваивания может использоваться любое допустимое в SQL выражение того же типа, что и столбец, имя которого приведено слева от оператора присваивания. Если в этом выражении используется имя столбца целевой таблицы, для вычисления выражения применяется значение этого столбца в текущей строке, которое было перед обновлением.

Примечание. Чтобы данные были добавлены, не забудьте нажать на кнопки , а затем для подтверждения внесенных изменений в таблицу. Просмотрите внесенные изменения в таблицу TEACHER.

Обновление по условию

Данный вариант использует фразу WHERE. В этом случае обновляются столбцы только тех строк таблицы, на которых выполняется условие фразы WHERE. Рассмотрим несколько примеров.

Запрос 35. Увеличить всем ассистентам зарплату и надбавку на 10 %:

```
UPDATE TEACHER
SET    Salary = Salary * 1.1, Rise = Rise * 1.1
WHERE  LOWER(Dolgnost ) = 'ассистент';
```

Самостоятельно создать запрос 36. Установить, что деканат юридического факультета переместился в комнату 232 8 корпуса.

Подзапросы во фразе WHERE

Во фразе WHERE можно использовать подзапросы, как мы это делали в предложении SELECT. Это дает возможность отбирать строки для обновления на основе информации из других таблиц.

Запрос 37. Например, увеличить ставку всех преподавателей кафедры прикладной математики факультета математики и информатики в полтора раза:

```
UPDATE TEACHER
SET    Salary = Salary * 1.5
WHERE  KOD_kafedru = (SELECT KOD_kafedru FROM KAFEDRA
WHERE  LOWER(Name_kafedru) = 'прикладная математика');
```

Подзапросы во фразе SET

До сих пор новые значения представляли собой константы или выражения с использованием значений обновляемой строки. Однако если такие значения присутствуют в других строках обновляемой таблицы или вообще в других таблицах, можно воспользоваться подзапросом. В этом случае допускается использовать две формы фразы SET:

SET {имя_столбца | (список_инен_столбцов)} = (подзапрос)

В обоих вариантах подзапрос должен возвращать одну строку. В первом случае он также должен возвращать значение одного столбца, а во втором возвращаемая строка должна содержать столько значений, сколько столбцов приведено в списке имен столбцов. При этом производится присвоение значений строки подзапроса соответствующим столбцам из списка слева.

Запрос 38. Установить всем ассистентам надбавку, равную 70 % текущей средней надбавки по вузу.

```
UPDATE TEACHER
SET     Rise = (SELECT SUM(Rise) * 0.7 / COUNT(*) FROM TEACHER)
WHERE  LOWER(Dolgnost ) = 'ассистент';
```

Удаление существующих строк

Удалять строки из таблицы можно с помощью предложения DELETE. Оно удаляет только строки целиком, а не индивидуальные значения столбцов. Синтаксис команды следующий:

```
DELETE FROM имя_таблицы [[AS] синонин]
[WHERE условие];
```

При использовании предложения DELETE вы прежде всего обнаружите, что предупреждающая подсказка, как правило, не выдается. Обычно, когда пользователь удаляет какой-либо объект операционной среды, он получает сообщение типа «Вы уверены (Д/Н)?». В системах, поддерживающих SQL, строки удаляются без такого сообщения. Поэтому будьте внимательны.

В зависимости от применения фразы WHERE предложение DELETE позволяет удалить отдельную строку, несколько или все строки таблицы. Строки могут быть и не удалены. При использовании предложения DELETE помните о следующем:

нельзя удалить значение отдельного столбца (используйте для этого предложение UPDATE);

как и предложения INSERT и UPDATE, удаление строк может нарушить ограничения целостности;

сама таблица не удаляется (используйте для этого предложение DROP TABLE).

Удаление всех строк таблицы

Чтобы удалить все содержимое таблицы, не нужно использовать фразу WHERE. Помните, что вы удаляете не саму таблицу, а только все ее строки.

Запрос 39. Удалить содержимое таблицы Сотрудники базы данных Educator.

```
use Educator
DELETE FROM Сотрудники;
```

Удаление по условию

Обычно нужно удалять только некоторые строки из таблицы. Чтобы определить, какие строки будут удалены, нужно использовать условие во фразе WHERE. Приведем несколько примеров.

Самостоятельно создать запрос 40. Удалить сведения об ассистентах, которые были приняты на работу до 01.01.1986.

Удаление одной строки

Чтобы удалить одну конкретную строку, нужно сформулировать условие таким образом, чтобы оно идентифицировало эту единственную строку. Обычно для этого в условии используются первичный ключ таблицы или уникальный набор столбцов.

Самостоятельно создать запрос 41. Удалить всех преподавателей под фамилией Швец.

Задание для практической работы №7

Для созданной базы данных, согласно номеру варианта, самостоятельно создать **на языке Transact-SQL 14 многотабличных запросов:**

- 1 запрос с использованием функции **COUNT**;
- 1 запрос с использованием функции **SUM**;
- 1 запрос с использованием функций **UPPER, LOWER**;
- 1 запрос с использованием временных функций;
- 1 запрос с использованием группировки по одному столбцу;
- 1 запрос на использование группировки по нескольким столбцам;
- 1 запрос с использованием условия отбора групп **HAVING**;
- 1 запрос с использованием фразы **HAVING** без фразы **GROUP BY;**
- 1 запрос с использованием сортировки по столбцу;
- 1 запрос на добавление новых данных в таблицу;
- 1 запрос на добавление новых данных по результатам запроса в качестве вставляемого значения;
- 1 запрос на обновление существующих данных в таблице;
- 1 запрос на обновление существующих данных по результатам подзапроса во фразе **WHERE;**
- 1 запрос на удаление существующих данных.

Все программные инструкции команд SQL сохранять в файлах с расширением ***.sql** в папке ***ФИО_студента/Лаб7***.

Для каждого запроса сформулировать текстовое задание, которое должно быть выполнено к базе данных.

Создать текстовый отчет, в котором отобразить sql-команды разработанных запросов и скриншоты результатов работы из СУБД **SQL Server Management Studio**.

ПРАКТИЧЕСКАЯ РАБОТА №8. СОЗДАНИЕ И УПРАВЛЕНИЕ ПРЕДСТАВЛЕНИЯМИ

8.1. Цель работы

Изучение назначения представлений баз данных, синтаксиса и семантики команд языка Transact-SQL для их создания, изменения и удаления, системных хранимых процедур для получения информации о представлениях, а также приобретение навыков их создания с помощью графических средств утилиты Enterprise Manager и мастера Create View Wizard.

8.2. Методические рекомендации для выполнения практической работы

Представление (View) для пользователей баз данных выглядит как таблица, но при этом оно не содержит данных, а лишь представляет данные, расположенные в одной или нескольких таблицах. Таким образом, представления – это виртуальные таблицы, определяемые запросом на языке Transact-SQL. Подобно реальным таблицам представления содержат именованные столбцы и строки с данными, которые они динамически выбирают из таблиц и предлагают эти данные пользователю для просмотра. Представления часто применяются для ограничения доступа к конфеденциальным данным в таблицах баз данных. Когда в представление не включается столбец исходной таблицы, то считают, что на таблицу наложен вертикальный фильтр. Если в SQL – запросе установлено одно или несколько условий для выборки строк, то считают, что на таблицу наложен горизонтальный фильтр.

Представление может выбирать данные из других представлений, которые, в свою очередь, могут также основываться на представлениях или таблицах. Вложенность представлений не должна превышать 32. Представления можно создавать, используя базы данных одного сервера (текущего). Максимальное количество столбцов в представлении равно 1024. Представление не может ссылаться на временные таблицы. Кроме того, нельзя создавать временное представление.

Для представления нельзя определить ограничения целостности, триггеры, правила, или умолчания, а также создать обычный или полнотекстовый индекс.

В основном представления используются для выборки данных. Однако с помощью представлений можно выполнять и изменение данных в таблицах, на основе которых построено представление, при этом требуется соблюдение ряда правил: представление должно содержать, как минимум, одну таблицу в параметре FROM команды SELECT, не разрешается использование функций агрегирования и др.

Как и для таблиц, для представлений можно определить следующие права доступа:

SELECT – просмотр данных;

INSERT – добавление данных через представления;

UPDATE – изменение данных в исходных таблицах;

DELETE –удаление данных в исходных таблицах.

Чтобы иметь возможность создавать представления, надо обладать правами владельца баз данных и иметь соответствующие разрешения для любых таблиц или представлений, упомянутых "в запросе на создание этого представления.

Для создания представления используется следующая команда Transact-SQL:

CREATE VIEW [Имя базы данных.] [имя владельца.]

Имя представления
[(Имя колонки [,... n])]
[WITH{ENCRYPITION\SHEMABINDING

VIEW_METADATA}
AS Команда SELECT
[WITH CHECK OPTION]

Если в команде не заданы имена колонок представления , то они определяются по именам выбираемых колонок в команде SELECT. Параметр ENCRYPTION скрывает код создания этого представления, а параметр SHEMABINDING обеспечивает контроль

структуры исходных объектов, к которым обращается оператор SELECT. Опция WITH CHEC OPTION не позволяет изменять строки таким образом, чтобы они исчезли при отборе командой SELECT.

8.3. Задания для выполнения практической работы №8

Задание 1. Создать представление auth, ссылающегося на таблицу authors базы данных Pubs и содержащего идентификационный номер автора au_lname и телефон phone, при этом отобразить только авторов из Калифорнии ‘CA’ или авторов, не подписавших контракт с издательством, выполнив следующую команду:

```
CREATE VIEW auth
WITH SHEMABINDING
AS SELECT au_id, au_lname, au_fname, phone
FROM dbo. Authors
WHERE state = ‘CA’ OR contract = 0
WITH CHECK OPTION.
```

Задание 2. Создать представление report, которое ссылается на представление auth и таблицы titleauthor и titles и в котором выводятся имя автора au_fname, фамилия автора au_lname и сокращенные названия написанных им книг, выполнив команду:

```
CREATE VIEW report
AS SELECT [Фамилия] = CAST (au_lname aschar(10)),
[Имя] = CAST(au_fname aschar(10)),
[Название книги] =
CAST (title as char(30)) +
CASE WHEN LEN (title) >30 THEN ‘...’ END
FROM auth a, titleauthor ta, titles t
WHERE ta.au_id = a.au_id AND
t.title_id = ta .title_id.
```

Задание 3. Создать представление auth, рассмотренное в первом задании, с помощью графических средств утилиты Enterprise Manager.

Задание 4. Создать представление report, рассмотренное во втором задании, с помощью мастера Create View Wizard.

Задание 5. Сопоставить запросы, полученные автоматически в заданиях 3и 4, с запросами соответственно в первом и втором заданиях. Модифицировать запросы с помощью команды ALTER VIEW и получить справочную информацию об этих представлениях с помощью процедур sp_help, sp_helptexst и sp_depends.

ПРАКТИЧЕСКАЯ РАБОТА №9. ОСНОВЫ ПРОГРАММИРОВАНИЯ С ПОМОЩЬЮ ВСТРОЕННОГО ЯЗЫКА TRANSACT-SQL В MICROSOFT SQL SERVER

9.1. Цель работы

Изучить используемый в реляционных СУБД встроенный язык программирования Transact-SQL для написания программ в MS SQL Server. Изучить правила построения идентификаторов, правила объявления переменных и их типов. Изучить принципы работы с циклами и ветвлениями. Изучить работу с переменными типа Table. Изучить синтаксис и семантику функций и хранимых процедур Transact–SQL: способов их идентификации, методов задания и спецификации параметров и возвращаемых значений и вызовов функций и хранимых процедур.

9.2. Исходные данные

Исходными данными является индивидуальное задание и результат предыдущих практических работ.

9.3. Используемые программы

Программа Microsoft SQL Server Managemant Studio.

9.4. Задание

Практическую работу следует выполнять в следующем порядке:

1. Знакомство с правилами обозначения синтаксиса команд в справочной системе MS SQL Server (утилита Books Online).
2. Изучение правил написания программ на Transact SQL.
3. Изучение правил построения идентификаторов, правил объявления переменных и их типов.
4. Изучение работы с циклами и ветвлениями.
5. Изучение работы с переменными типа Table.
6. Изучение правил написания хранимых процедур и функций.
7. Проработка всех примеров, анализ результатов их выполнения.
8. Выполнение индивидуальных заданий по вариантам.

Для освоения программирования используем пример базы данных **University,** которая была создана в предыдущих практических работах. При выполнении примеров и заданий обращайте внимание на соответствие названий БД, таблиц и других объектов проекта.

9.5. Теоретические сведения

Специальные знаки и простейшие операторы в Transact SQL

Знак	Назначение	Знак	Назначение
*	Знак умножения	" "	В них заключают строковые значения, если SET QUOTED_IDENTIFIER OFF
-	Знак вычитания	‘ ’	В них заключают строковые значения
%	Остаток от деления двух чисел	<>	Не равно
+	Знак сложения или конкатенации (объединение двух строк в одну)	[ ]	Аналог кавычек, в них можно заключать названия идентификаторов, если в их названиях встречаются пробелы
=	Знак равенства или сравнения	!<	Не менее чем
<=	Меньше или равно	!>	Не более чем
>=	Больше или равно	>	Больше
!=	Не равно	<	Меньше
@	Ставится перед именем переменной	.	Разделяет родительские и подчиненные объекты
@@	Указывает на системные функции	/	Знак деления
--	Однострочный комментарий или комментарий с текущей позиции и до конца строки	/* */	Многострочный комментарий

Идентификаторы - это имена объектов, на которые можно ссылаться в программе, написанной на языке Transact SQL. Первый символ может состоять из букв английского алфавита или "**@**", "**#**". Остальные дополнительно из цифр и «$».

Имя идентификатора не должно совпадать с зарезервированным словом.

Для ограничителей идентификаторов при установленном параметре **SET QUOTED_IDENTIFIER ON** можно использовать как квадратные скобки, так и одинарные кавычки, а строковые значения только в одинарных кавычках (режим по умолчанию).

Если использовать установленный параметр в режиме **SET QUOTED_IDENTIFIER OFF**, то в качестве ограничителей идентификаторов можно использовать только квадратные скобки, а строковые значения указываются в одинарных или двойных кавычках.

Переменные используются для сохранения промежуточных данных в хранимых процедурах и функциях. Все переменные считаются локальными. Имя переменной должно начинаться с **@**.

Объявление переменных

Синтаксис в обозначениях MS SQL Server:

DECLARE @имя_переменной1 тип_переменной, @имя_переменнойN тип_переменной

Если тип переменной предполагает указание размера, то используется следующий синтаксис для объявления переменных:

DECLARE @ имя_переменной1 тип_переменной (размер), .,
@ имя_переменнойN тип_переменной(размер)

Пример 1: Введите в редактор SQL запросов следующее объявление переменных
DECLARE @a INT, @b numeric(10,2)
DECLARE @str CHAR(20)

Присвоение значений переменным и вывод значений на экран

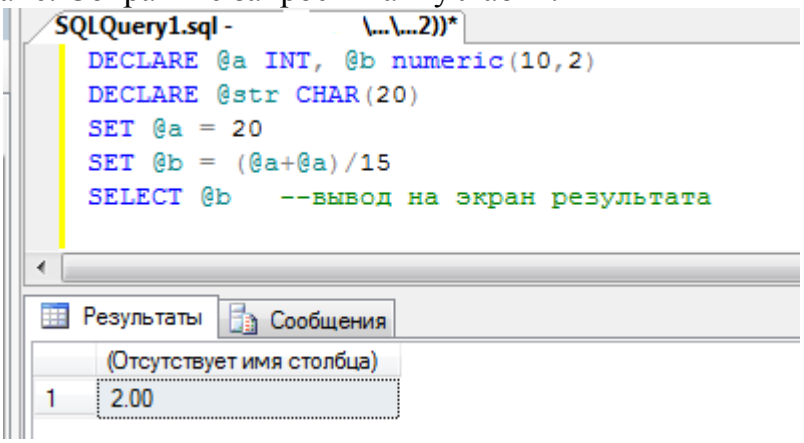
Присвоение с помощью **SET** - обычное присвоение, синтаксис:

SET @имя_переменной = значение

Допишите в редакторе SQL объявление переменных с вводом их значений и выводом результата

```
DECLARE @a INT, @b numeric(10,2)
SET @a = 20
SET @b = (@a+@a)/15
SELECT @b   --вывод на экран результата
```

Запустить эту конструкцию на выполнение. У вас появиться следующий результат на экране. Сохраните запрос в папку Лаб12.



Присвоение с помощью **SELECT** - помещение результата запроса в переменную. Если в результате выполнения запроса не будет возвращено ни одной строки, то значение переменной не меняется, т.е. остается старым.

Рассмотрим пример, в котором переменной присвоим результат выполнения агрегированной функции над таблицами базы данных

Пример 2: Вывести количество строк в таблице **Студенты** и присвоить это значение переменной

```
DECLARE @a INT
SELECT @a = COUNT(*) FROM Student
SELECT @a
```

Пример 3:

```
DECLARE @str CHAR(30)
SELECT @str = Sutname FROM Student
SELECT @str
```

В данном примере в переменную поместится последнее значение из результата запроса.

Сочетание ключевых слов SET и SELECT

Измените код запроса из примера 2 на следующий:

DECLARE @a INT
SET @a = (SELECT COUNT(*) FROM Authors)
SELECT @a

Создание временной таблицы через переменную типа TABLE

Объявляется через DECLARE с указанием в скобках столбцов таблицы, их типов, размеров, значений по умолчанию, а также индексов типа PRIMARY KEY или UNIQUE.

Пример 4: Создание временной таблицы с двумя полями, в которую будет добавлены 2 строки с данными

DECLARE @mytable TABLE (id INT, myname CHAR(20) DEFAULT 'Иванов Иван')
INSERT INTO @mytable(id) VALUES (1)
INSERT INTO @mytable(id, myname) VALUES (2,'Игорь Троцкий')
SELECT * FROM @mytable
Выполните и сохраните запрос

Пример 5: Создание временной таблицы с двумя полями, в которую будет добавлены строки, как результат выполнения запроса выборки данных из таблицы **факультет**

DECLARE @mytable TABLE(id INT, myname CHAR(255) DEFAULT 'Введите название')
INSERT @mytable SELECT kod_faculteta, name_faculteta FROM facultet
SELECT * FROM @mytable
Выполните и сохраните запрос.

Самостоятельно создать на языке P-SQL запросы, с помощью которых

Запрос 1. Подсчитать среднюю зарплату преподавателей (с помощью запроса **SELECT**) и умножить ее на значение 123,34, которое необходимо сохранить в отдельной переменной, вывести значение переменной на экран.

Запрос 2. Подсчитать суммарное значение всех стипендий у студентов, результат поместить в переменную, вывести значение переменной на экран.

Запрос 3. Подсчитать количество кафедр, результат поместить в переменную, вывести значение переменной на экран.

Запрос 4. Создать локальную таблицу с названием **TEMP** и полями типа дата/время, длинное целое, строка. Добавить в нее две записи с данными и вывести результат на экран.

Операторские скобки
BEGIN
/* в них нельзя помещать команды, изменяющие структуры объектов БД. Операторские скобки должны содержать хотя бы один оператор. Требуются для конструкций поливариантных ветвлений, условных и циклических конструкций
*/
END
Условная конструкция IF
Синтаксис:
IF условие
Набор операторов**1 ELSE**
Набор операторов**2**

Пример 6: Использование операторских скобок и условных конструкций для поиска ответа – количество кафедр больше 10, да или нет

```
DECLARE @a INT
DECLARE @str CHAR(30)
SET @a = (SELECT COUNT(*) FROM kafedra)
IF @a >10 BEGIN
        SET @str = 'Количество кафедр больше 10'
        SELECT @str
END ELSE
BEGIN
        SET @str = 'Количество кафедр = ' + str(@a)
        SELECT @str
END
```

Выполните и сохраните запрос.

Самостоятельно создать на языке P-SQL запросы, с помощью которых

Запрос 5. Подсчитать количество факультетов. Если их в таблице от 2 до 4, то ничего не сообщать, в противном случае вывести сообщение вида "В таблице ... факультетов" (вместо многоточия поставить точное количество факультетов).

Запрос 6. Подсчитать средний год рождения студентов. Если полученный год в диапозоне от 1980 до 1999, то ничего не сообщать, в противном случае вывести сообщение вида "Средний год рождения = ." (вместо многоточия поставить точный средний год).

Цикл WHILE

Синтаксис:

WHILE Условие
Набор операторов1
BREAK
Набор опреторов2
CONTINUE

Конструкции **BREAK** и **CONTINUE** являются необязательными.

Цикл можно принудительно остановить, если в его теле выполнить команду **BREAK**. Если же нужно начать цикл заново, не дожидаясь выполнения всех команд в теле, необходимо выполнить команду **CONTINUE.**

Пример 7: Объявление переменной a, проверка в цикле, чтобы значение a не превосходило 100. Переменная a увеличивает свое значение в случайном порядке. Также организуется проверка на условие.

```
DECLARE @a INT
SET @a = 1
WHILE @a <100
    BEGIN
        PRINT @a -- вывод на экран значения переменной
        IF (@a>40) AND (@a<50)
            BREAK    --выход и выполнение 1-й команды за циклом
        ELSE
```

```
        SET @a = @a+rand()*10
CONTINUE
END
PRINT @a
```

Выполните и сохраните запрос.

Самостоятельно создать на языке P-SQL запросы, с помощью которых

Запрос 7. Определить количество записей в таблице кафедра. Пока записей меньше 10, делать в цикле добавление записи в временную таблицу с автоматическим наращиванием значения ключевого поля, а вместо названия кафедры ставить значение 'Имя не известено'.

Функции и хранимые процедуры

Функции и хранимые процедуры используются в SQL Server для реализации на языке Transact-SQL сложных часто используемых алгоритмов обработки данных или различных административных действий создания учетных записей, получения информации об объектах базы данных, управления свойствами сервера и баз данных, управления подсистемой репликации и автоматизации и т.д.

Они хранятся в виде исходного текста и являются программными модулями, существующими независимо от таблиц или каких либо других объектов баз данных.

Исключением являются **расширенные хранимые процедуры**, которые хранятся в двоичном формате в виде динамически подключаемых библиотек типа ***.dll** и создаются с помощью других языков программирования с использованием интерфейса **SQL Server Open Data Services API.** Такие процедуры подключаются, отключаются и выгружаются соответственно командами **sp_addextendedproc, sp_dropextendedproc** и **DBCC dlname (FREE),** где dllno me_имя dll_библиотеки.

Хранение функций и хранимых процедур в виде исходных модулей языка Transact – SQL на сервере и в соответствующих базах данных **позволяет уменьшить размер запроса**, посылаемого по сети от клиента на сервер, а следовательно и **нагрузку на сеть**, что повышает общую производительность системы. Это также позволяет **упростить сопровождение программных комплексов** и внесение изменений в исходный текст модулей, причем большинство изменений не отразится на работоспособности клиентских приложений.

Значительная часть функций и хранимых процедур поставляются в составе SQL Server. Они называются **системными**, или **встроенными (built - in)**.

Кроме того, пользователю предоставляется возможность разрабатывать и включать в свою базу данных собственные, или **пользовательские (user-defined) функции и хранимые процедуры**, реализующие специальные алгоритмы обработки данных.

Таким образом, пользовательские функции и хранимые процедуры становятся объектами той базы данных, в которой они создавались. Поэтому при их создании, если необходимо, требуемую базу данных следует сделать текущей с помощью команды **USE** имя базы данных. Системные же функции хранятся на экземпляре сервера, а системные хранимые процедуры – в базе **MASTER** этого же экземпляра сервера.

В SQL Server можно создавать и так называемые **временные хранимые процедуры** в базе данных **tempdb** экземпляра сервера, которые существуют лишь некоторое время, после чего автоматически уничтожаются сервером. Они бывают **локальными** и **глобальными.**

Функции и хранимые процедуры могут быть вызваны клиентскими программами, другими функциями или хранимыми процедурами, а также **триггерам**и.

В любом случае необходимо указать имя функции или хранимой процедуры и список аргументов, которые сопоставляются параметрам соответствующей функции или хранимой процедуры при этом типы аргументов и параметров должны совпадать или допускать автоматические преобразования типов. Если для некоторого параметра задано значение по умолчанию и это значение подходит для данного вызова, то соответствующий аргумент может быть опущен.

Поскольку функция возвращает значение, она используется в качестве операнда некоторого выражения в виде вызова функций, состоящего из имени этой функции и списка аргументов, заключенного в круглые скобки, при этом в качестве аргументов могут быть любые выражения языка Transact – SQL, дающие в результате значения требуемых типов.

Аргументы в вызове функции отделяются запятыми.

Если список аргументов пуст, то круглые скобки после имени функции, как правило, задаются.

Исключения составляют некоторые системные функции, для которых круглые скобки не задаются, когда нет аргументов.

Хранимые процедуры могут вызываться только командой **EXECUTE**, или сокращенно **EXEC**. За этой командой должны быть указаны имя процедуры и через пробел список аргументов, если вызывается процедура с параметрами. Аргументы разделяются запятой. Если для параметра задано значение по умолчанию, то аргумент либо совсем не задается (в конце списка), либо используется слово **DEFAULT**(в середине списка).

Процедура может возвращать результаты только через параметры с ключевым словом **OUTPUT**, при этом и аргумент должен быть задан с таким же ключевым словом.

Создание, изменение и удаление функций и хранимых процедур производится соответственно командами:

- **CREATE FUNCTION,**
- **CREATE PROCEDURE ,**
- **ALTER FUNCTION,**
- **ALTER PROCEDURE,**
- **DROP FUNCTION,**
- **DROP PROCEDURE**.

При создании функции указывается тип возвращаемого значения и в теле функции обязательно задается команда **RETURN**, за которой следует выражения для вычисления возвращаемого значения.

В теле процедуры использование команды **RETURN** (конечно, без последующего выражения) вовсе не обязательно. Когда этой команды нет, выход из процедуры будет происходить после исполнения последней команды процедуры.

Тело, как функции, так и хранимой процедуры начинается ключевым словом **AS.**

Поскольку каждая из них храниться как отдельный объект, то для указания конца тела не требуется записывать какое-либо специальное ключевое слово или знак.

За командами создания функции или хранимой процедуры перечисляются имена параметров, начинающиеся с символа **@**, и их типы, а также важно значение по умолчанию.

Для функции этот список заключается в круглые скобки, после которых записывается ключевое слово **RETURNS** (возвращает) и тип возвращаемого значения.

Для хранимой процедуры круглые скобки не используются, и задавать тип возвращаемого значение не требуется.

Для тела функции часто используют ключевое слово **begin** после ключевого слова **as** и ключевое слово **end** в конце тела.

Дополнительные опции функции или хранимой процедуры задаются ключевым словом **with** до начала тела.

В SQL Server 2003 можно создавать **функции трех классов:**

- Scalar – возвращают обычное скалярное значение;

- Inline – возвращают таблицу динамической структуры, создаваемую единственной командой тела функции SELECT;

- Multi – statement – возвращает обычную таблицу заданной структуры, при этом количество команд в теле функции не ограничивается.

1. Создание функций

Команда:

CREATE FUNCTION (Transact-SQL)

создает **определяемую пользователем функцию** в SQL Server. Определяемая пользователем функция представляет собой подпрограмму Transact-SQL или среды CLR, которая принимает параметры, выполняет действия, такие как сложные вычисления, а затем возвращает результат этих действий в виде значения. Возвращаемое значение может быть скалярным значением или таблицей. При помощи этой инструкции можно создать подпрограмму, которую можно повторно использовать следующими способами:

- В инструкциях Transact-SQL, например **SELECT.**
- В приложениях, вызывающих функцию.
- В определении другой пользовательской функции.
- Для параметризации представления или улучшения функциональности индексированного представления.
- Для определения столбца таблицы.
- Для определения ограничения **CHECK** на столбец.
- Для замены хранимой процедуры.

Синтаксис создания скалярной функции.

```
--Transact-SQL Scalar Function Syntax
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type
    [ = default ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS return_data_type
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN scalar_expression
    END
[ ; ]
```

Аргументы

schema_name - Имя схемы, к которой принадлежит определяемая пользователем функция.

function_name - Имя определяемой пользователем функции. Имена функций должны удовлетворять правилам построения идентификаторов и должны быть уникальными в пределах базы данных и схемы.

@parameter_name - Аргумент пользовательской функции. Может быть объявлен один или несколько аргументов.

Для функций допускается не более 2 100 параметров. При выполнении функции значение каждого из объявленных параметров должно быть указано пользователем, если для них не определены значения по умолчанию.

[**type_schema_name.**] **parameter_data_type -** Тип данных параметра

[=default] - Значение по умолчанию для аргумента. Если определено значение default, то функция выполняется даже в том случае, если для данного аргумента значение не указано.

READONLY - Указывает, что параметр не может быть обновлен или изменен при определении функции. Если тип параметра является определяемым пользователем табличным типом, то должно быть указано ключевое слово READONLY.

return_data_type - Возвращаемое значение скалярной функции, определяемой пользователем.

function_body - Указывает серию инструкций Transact-SQL

scalar_expression - Указывает скалярное значение, возвращаемое скалярной функцией.

Предложение EXECUTE AS - Указывает контекст безопасности, в котором выполняется определяемая пользователем функция. Иными словами, есть возможность управлять тем, какую учетную запись пользователя SQL Server использует при определении разрешений на объекты базы данных, на которые ссылается функция.

Синтаксис функции, которая возвращает табличное значение:

```
--Transact-SQL Inline Table-Valued Function Syntax
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type
    [ = default ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS TABLE
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    RETURN [ ( ] select_stmt [ ) ]
[ ; ]
```

Аргументы:

TABLE - Указывает, что возвращаемым значением функции с табличным значением, является таблица. Функциям с табличным значением, могут передаваться только константы и @local_variables.

Во встроенных функциях с табличным значением возвращаемое значение TABLE определяется при использовании единственной инструкции **SELECT**. Встроенные функции не имеют соответствующих возвращаемых переменных.

select_stmt - Одиночная инструкция **SELECT**, определяющая возвращаемое значение встроенной функции с табличным значением.

В функциях допустимы следующие инструкции.

- Инструкции присваивания.
- Инструкции управления потоком, за исключением инструкций **TRY...CATCH**.
- Инструкции **DECLARE**, объявляющие локальные переменные и локальные курсоры.
- Инструкции **SELECT**, которые содержат списки выбора с выражениями, присваивающими значения локальным переменным.
- Операции над локальными курсорами, которые объявляются, открываются, закрываются и освобождаются в теле функции. Допустимы только те инструкции **FETCH,** которые предложением **INTO** присваивают значения локальным переменным. Инструкции **FETCH**, возвращающие данные клиенту, недопустимы.
- Инструкции **INSERT, UPDATE** и **DELETE**, которые изменяют локальные табличные переменные.
- Инструкции **EXECUTE**, вызывающие расширенные хранимые процедуры.

Ограничения

Определяемые пользователем функция не может выполнять действия, изменяющие состояние базы данных.

Определяемые пользователем функции не могут содержать предложение **OUTPUT INTO**, целью которого является таблица.

Определяемые пользователем функции могут быть **вложенными**, то есть из одной функции может быть вызвана другая. Вложенность определяемых пользователем функций не может превышать 32 уровней.

Пример 1. Применение скалярной определяемой пользователем функции, вычисляющей месяц.

В следующем примере показано создание определяемой пользовательской функции **ISOweek,** которая получает в качестве аргумента дату и вычисляет номер месяца, а затем по номеру определяет название месяца. Для правильной работы этой функции перед ее вызовом должна быть выполнена инструкция **SET DATEFIRST 1 .**

Следующий пример также показывает использование предложения **EXECUTE AS** для указания контекста безопасности, в котором может быть выполнена хранимая процедура. В этом примере параметр **CALLER** указывает, что процедура будет выполнена в контексте пользователя, который ее вызывает.

В начале, прежде чем создавать функцию, будет выполнена проверка на наличие в базе данных функции под таким же именем. Если функция с таким именем уже была создана ранее, то она будет удалена с помощью команды **DROP FUNCTION** и создана заново **CREATE FUNCTION**.

Откройте sql-редактор. Создайте новый запрос.

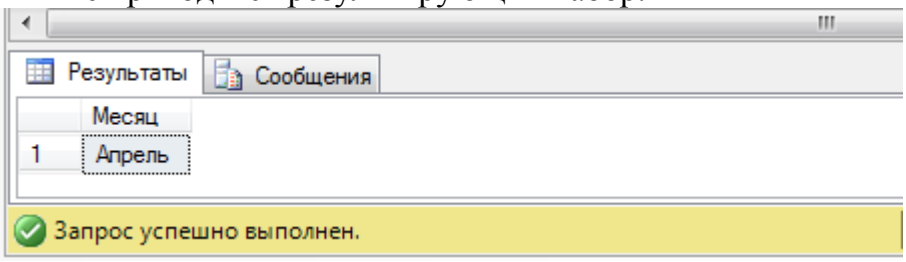
```
USE University;
GO
IF OBJECT_ID (N'dbo.ISOweek', N'FN') IS NOT NULL
   DROP FUNCTION dbo.ISOweek;

GO
CREATE FUNCTION dbo.ISOweek (@DATE date)
RETURNS CHAR(15)
WITH EXECUTE AS CALLER
AS
BEGIN
    DECLARE @man int;
    DECLARE @ISOweek char(15);
    SET @man= MONTH(@DATE)

    IF (@man=1) SET @ISOweek='Январь';
    IF (@man=2) SET @ISOweek='Февраль';
    IF (@man=3) SET @ISOweek='Март';
    IF (@man=4) SET @ISOweek='Апрель';
    IF (@man=5) SET @ISOweek='Май';
    IF (@man=6) SET @ISOweek='Июнь';
    IF (@man=7) SET @ISOweek='Июль';
    IF (@man=8) SET @ISOweek='Август';
    IF (@man=9) SET @ISOweek='Сентябрь';
    IF (@man=10) SET @ISOweek='Октябрь';
    IF (@man=11) SET @ISOweek='Ноябрь';
    IF (@man=12) SET @ISOweek='Декабрь';


    RETURN(@ISOweek);
END;
```

Для того чтобы увидеть результат, выведем на экран вычисление значение функции от произвольной даты, например '12.04.2004'. для этого ниже в sql-редакторе пишем:

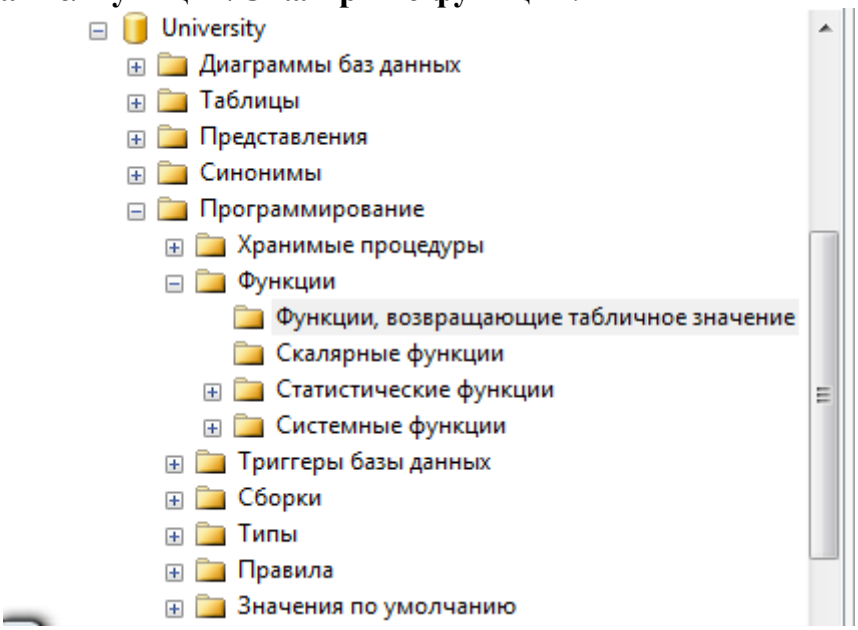
```
GO
SET DATEFIRST 1;
```

SELECT dbo.ISOweek('12.04.2004') AS 'Месяц';

Ниже приводится результирующий набор.



Просмотреть все функции, которые пользователь может создавать, откройте в окне обозревателя объектов в базе данных **University** группу **Программирование/Функции/Скалярные функции**.



Пример 2. Создание пользовательской функции с табличным значением.

Например, создадим пользовательскую функцию, которая будет разрешена к использованию всеми пользователями с правами роли «**dekan**» (созданной в лаб.работе №2). Функция будет возвращать результат в виде **таблицы** – вывод на всех кафедрах суммирующей зарплаты по каждой должности всех преподавателей. При этом функция имеет один параметр **@storeid**, с помощью которого введем ограничение на вычисление, а именно зарплата должна быть больше 100.

Создайте новый запрос:

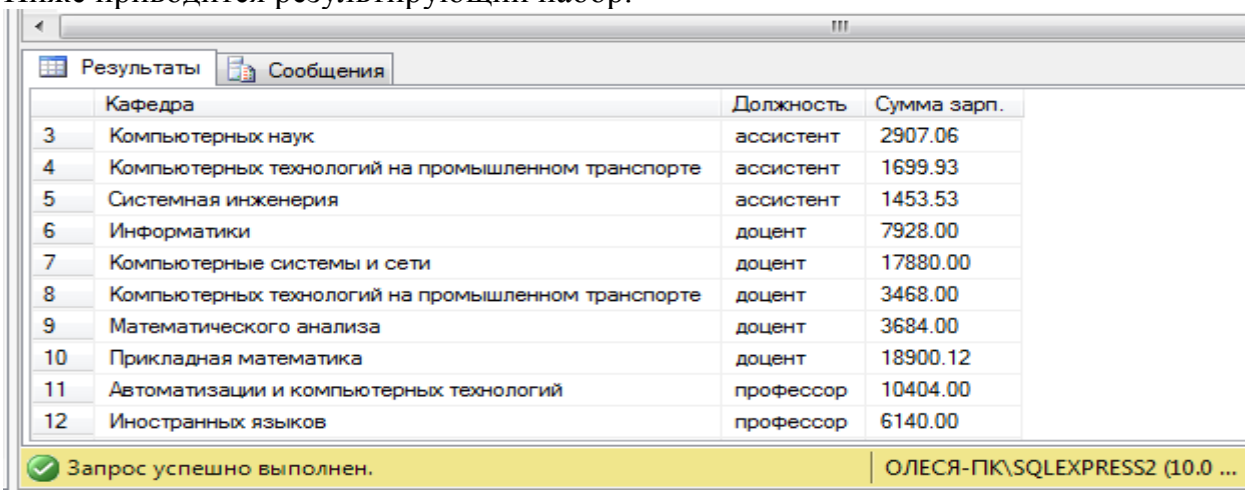
```
USE University;
GO
IF OBJECT_ID (N'ufn_SalesByStore', N'IF') IS NOT NULL
   DROP FUNCTION dekan.ufn_SalesByStore;
GO
CREATE FUNCTION dekan.ufn_SalesByStore(@storeid int)
RETURNS TABLE
AS
RETURN
(
  SELECT d.Name_kafedru AS "Кафедра",
  t.Dolgnost  AS "Должность",
```

```
    SUM(t.Salary + t.RISE) AS "Сумма зарплаты"
    FROM    KAFEDRA d, TEACHER t
    WHERE d.KOD_kafedru =t.KOD_kafedru
        and t.salary>@storeid
    GROUP BY d.Name_kafedru, t.Dolgnost

);
```

Для вызова этой функции нужно выполнить следующий запрос:

```
GO
SELECT * from dekan.ufn_SalesByStore(100);
```

Ниже приводится результирующий набор.



	Кафедра	Должность	Сумма зарп.
3	Компьютерных наук	ассистент	2907.06
4	Компьютерных технологий на промышленном транспорте	ассистент	1699.93
5	Системная инженерия	ассистент	1453.53
6	Информатики	доцент	7928.00
7	Компьютерные системы и сети	доцент	17880.00
8	Компьютерных технологий на промышленном транспорте	доцент	3468.00
9	Математического анализа	доцент	3684.00
10	Прикладная математика	доцент	18900.12
11	Автоматизации и компьютерных технологий	профессор	10404.00
12	Иностранных языков	профессор	6140.00

Самостоятельно создайте запрос:

Запрос 8. Создать пользовательскую функцию, которая будет возвращать результат в виде **таблицы, а именно** выводить всех учащихся студентов по кафедрам с указанием курса. При этом функция имеет один параметр **@city**, с помощью которого введем ограничение на вычисление, а именно город проживания должен быть Ростов-на-дону.

2. Создание хранимых процедур в Microsoft SQL Server

Хранимые процедуры представляют собой набор команд, состоящий из одного или нескольких операторов SQL или функций и сохраняемый в базе данных в откомпилированном виде.

Типы хранимых процедур

Системные хранимые процедуры предназначены для выполнения различных административных действий. Практически все действия по администрированию сервера выполняются с их помощью.

Пользовательские хранимые процедуры реализуют те или иные действия. Хранимые процедуры - полноценный объект базы данных. Вследствие этого каждая хранимая процедура располагается в конкретной базе данных, где и выполняется.

Временные хранимые процедуры существуют лишь некоторое время, после чего автоматически уничтожаются сервером. Они делятся на локальные и глобальные..

Создание, изменение хранимых процедур

Создание хранимой процедуры предполагает решение следующих задач: планирование прав доступа. При создании хранимой процедуры следует учитывать, что

она будет иметь те же права доступа к объектам базы данных, что и создавший ее пользователь; определение параметров хранимой процедуры, хранимые процедуры могут обладать входными и выходными параметрами; разработка кода хранимой процедуры. Код процедуры может содержать последовательность любых команд SQL, включая вызов других хранимых процедур.

Синтаксис оператора создания новой или изменения имеющейся хранимой процедуры в обозначениях MS SQL Server:

```
--Transact-SQL Stored Procedure Syntax
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
    [ { @parameter [ type_schema_name. ] data_type }
        [ VARYING ] [ = default ] [ OUT | OUTPUT | [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]

<procedure_option> ::=
    [ ENCRYPTION ]
    [ RECOMPILE ]
    [ EXECUTE AS Clause ]
```

Рассмотрим параметры данной команды.

Используя префиксы **sp_, #, ##,** создаваемую процедуру можно определить в качестве **системной** или **временной**. Как видно из синтаксиса команды, не допускается указывать имя владельца, которому будет принадлежать создаваемая процедура, а также имя базы данных, где она должна быть размещена.

Таким образом, чтобы разместить создаваемую хранимую процедуру в конкретной базе данных, необходимо выполнить команду **CREATE PROCEDURE** в контексте этой базы данных. При обращении из тела хранимой процедуры к объектам той же базы данных можно использовать укороченные имена, т. е. без указания имени базы данных. Когда же требуется обратиться к объектам, расположенным в других базах данных, указание имени базы данных обязательно.

Для передачи входных и выходных данных в создаваемой хранимой процедуре имена параметров должны начинаться с символа **@**. В одной хранимой процедуре можно задать множество параметров **@parameter**, разделенных запятыми. В теле процедуры не должны применяться локальные переменные, чьи имена совпадают с именами параметров этой процедуры.

Наличие ключевого слова **OUTPUT** означает, что соответствующий параметр предназначен для **возвращения данных из хранимой процедуры**. Указание ключевого слова **OUTPUT** предписывает серверу при выходе из хранимой процедуры присвоить текущее значение параметра локальной переменной, которая была указана при вызове процедуры в качестве значения параметра.

Ключевое слово **VARYING** применяется совместно с параметром OUTPUT, имеющим тип CURSOR. Оно определяет, что выходным параметром будет результирующее множество.

Ключевое слово **DEFAULT** представляет собой значение, которое будет принимать соответствующий параметр по умолчанию. Таким образом, при вызове процедуры можно не указывать явно значение соответствующего параметра.

Ключевое слово **AS** размещается в начале собственно тела хранимой процедуры.

В теле процедуры могут применяться практически все команды SQL, объявляться транзакции, устанавливаться блокировки и вызываться другие хранимые процедуры.

Выход из хранимой процедуры можно осуществить посредством команды **RETURN.**

Удаление хранимой процедуры

DROP PROCEDURE {имя_процедуры} [,...n]

Выполнение хранимой процедуры

Для выполнения хранимой процедуры используется команда:

[[EXEC [UTE] имя_процедуры [;номер]
[[@имя_параметра=]{ значение | @имя_переменной}
[OUTPUT]|[DEFAULT]][,...n]

Если вызов хранимой процедуры не является единственной командой в пакете, то присутствие команды **EXECUTE** обязательно. Более того, эта команда требуется для вызова процедуры из тела другой процедуры или триггера.

Использование ключевого слова **OUTPUT** при вызове процедуры разрешается только для параметров, которые были объявлены при создании процедуры с ключевым словом **OUTPUT.**

Когда же при вызове процедуры для параметра указывается ключевое слово **DEFAULT**, то будет использовано значение по умолчанию. Естественно, указанное слово **DEFAULT** разрешается только для тех параметров, для которых определено значение по умолчанию.

Из синтаксиса команды **EXECUTE** видно, что имена параметров могут быть опущены при вызове процедуры. Однако в этом случае пользователь должен указывать значения для параметров в том же порядке, в каком они перечислялись при создании процедуры. Присвоить параметру значение по умолчанию, просто пропустив его при перечислении, нельзя. Если же требуется опустить параметры, для которых определено значение по умолчанию, достаточно явного указания имен параметров при вызове хранимой процедуры. Более того, таким способом можно перечислять параметры и их значения в произвольном порядке.

Отметим, что при вызове процедуры указываются либо имена параметров со значениями, либо только значения без имени параметра. Их комбинирование не допускается.

Использование RETURN в хранимой процедуре

Позволяет выйти из процедуры в любой точке по указанному условию, а также позволяет передать результат выполнения процедуры числом, по которому можно судить о качестве и правильности выполнения процедуры.

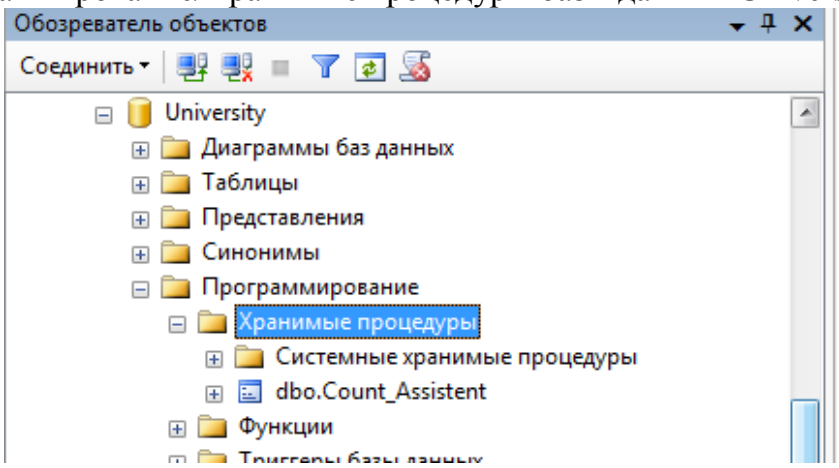
Примеры создания процедур

Пример 3. Создание процедуры без параметров. Процедура вычисляет кол-во всех ассистентов:

CREATE PROCEDURE Count_Assistent
AS

```
Select count(DOLGNOST)
from TEACHER
where DOLGNOST='Ассистент'
Go
```

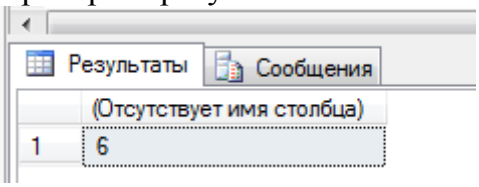
Создайте данную процедуру к базе данных University через утилиту SQL server Management Studio. Выполните запрос. Посмотрите созданную процедуру в разделе Программирование/Хранимые процедуры базы данных University.



Запустите ее с помощью команды:

EXECUTE Count_Assistent

Проверьте результат. Количество ассистентов = 6.



Пример 4. Создание процедуры с входным параметром. Например, нужно посчитать кол-во ассистентов, зарплата у которых не более заданного параметра @Sum_salary:

```
CREATE PROCEDURE Count_Assistent_Salary
@Sum_salary as Int
AS
Select count(DOLGNOST)
from TEACHER
WHERE DOLGNOST='Ассистент' and SALARY>=@Sum_salary
Go
```

Создайте данную процедуру. Запустите ее с помощью команды

EXEC **Count_Assistent_Salary 1500**

Проверьте результат.

Самостоятельно создать запрос:

Запрос 8. Создать процедуру Count_Assistent_Salary_Title с входными параметрами @Sum_salary целого типа, @Title строка с длиной 15 символов, которая определяет кол-во преподавателей, должность которых совпадает с параметром @Title и зарплата у которых не менее заданного параметра @Sum_salary.

Создайте данную процедуру. Запустите ее с помощью команды
EXECUTE Count_Assistent_Salary_Title 1300, '%нт%'
Проверьте результат.

Пример 6. Создание процедуры с входными параметрами и выходным параметром:

```
CREATE PROCEDURE Count_Assistent_Itogo
@Sum_salary Int, @Title Char(15) , @Itogo Int OUTPUT
AS
Select @Itogo = count(DOLGNOST)
from TEACHER
WHERE SALARY<=@Sum_salary
AND DOLGNOST LIKE @Title
Go
```

Создайте данную процедуру. Запустите с помощью набора команд:

```
Declare @q As int
EXEC Count_Assistent_Itogo 2000, '%тент%', @q output select @q
```

Проверьте результат.

Пример 7. Создание процедуры с входными параметрами и RETURN. Пусть процедура проверяет, если номер студента равен параметру @param и одновременно его фамилия 'Петрова', то выдать 1, иначе 2:

```
CREATE PROCEDURE checkname @param int AS
IF (SELECT SUTFNAME FROM STUDENT WHERE STUDENT_ID = @param)
= 'Петрова'
RETURN 1
ELSE
RETURN 2
```

Создайте данную процедуру. Запустите ее с помощью команд:

```
DECLARE @return_status int
EXECUTE @return_status = checkname 15
SELECT 'Return Status' = @return_status
```

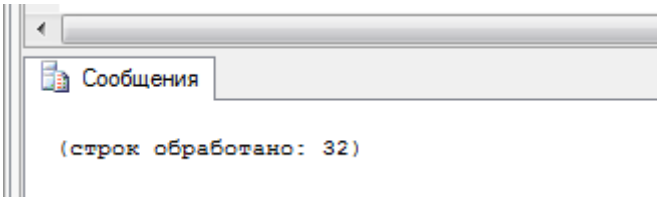
Проверьте результат.

Пример 8. Создание процедуры без параметров для уменьшения значения стипендии в таблице Student на 50 грн:

```
CREATE PROC update_proc AS
UPDATE STUDENT SET stipend = stipend-50
```

Процедура не возвращает никаких данных.

Создайте данную процедуру. Запустите ее с помощью команды
EXEC update_proc
Проверьте результат.



Пример 9. Создание процедуры с входным параметром для получения всей информации о конкретном заведующем кафедры:

```
CREATE PROC select_zavkaf @k CHAR(10)
AS
SELECT * FROM kafedra WHERE fio_zavkaf=@k
```

Создайте данную процедуру. Запустите ее с помощью команд:

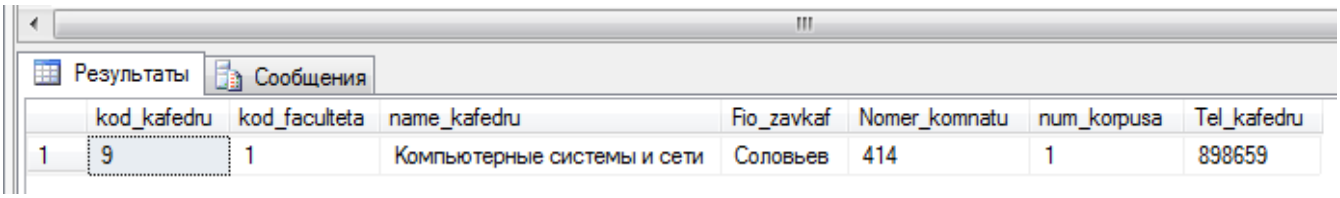
EXEC select_zavkaf 'Соловьев'

или

select_author @k= 'Соловьев'

или

EXEC select_author @k='Соловьев'



	kod_kafedru	kod_faculteta	name_kafedru	Fio_zavkaf	Nomer_komnatu	num_korpusa	Tel_kafedru
1	9	1	Компьютерные системы и сети	Соловьев	414	1	898659

Самостоятельно создайте запрос:

Запрос 9. Создать процедуру update_proc_rise с входным параметром и значением по умолчанию @p real = 0.5 для увеличения значения надбавки к зарплате в таблице TEACHER в заданное количество раз:

Процедура не возвращает никаких данных.

Создайте данную процедуру. Запустите ее с помощью команд:

EXEC update_proc_rise 1.5

или

EXEC update_proc_rise @p = 2

или

EXEC update_proc_rise --будет использовано значение по умолчанию.

Пример 11. Создание процедуры с входным и выходным параметрами. Создать процедуру для определения количества преподавателей, с датой приема на работу в указанный период:

```
CREATE PROC count_teacher
@d1 DATE, @d2 DATE, @c INT OUTPUT
AS
SELECT @c=count(KOD_TEACHER) from teacher
WHERE Data_hire BETWEEN @d1 AND @d2
SET @c = ISNULL(@c,0)
```

Создайте данную процедуру. Запустите ее с помощью команд:

```
DECLARE @c2 INT
EXEC count_teacher '01/01/2006', '31/12/2008', @c2 OUTPUT SELECT @c2
```

Задание к практической работе №9

Для созданной базы данных, согласно номеру варианта, самостоятельно создать **на языке P-SQL 10 запросов:**

- 1 запрос для создания временной таблицы через переменную типа **TABLE**;
- 1 запрос с использованием условной конструкции **IF**;
- 1 запрос с использованием цикла **WHILE**;
- 1 запрос для создания скалярной функции;
- 1 запрос для создания функции, которая возвращает табличное значение;
- 1 запрос для создания процедуры без параметров ;
- 1 запрос для создания процедуры с входным параметром;
- 1 запрос для создания процедуры с входными параметрами и **RETURN**;
- 1 запрос для создания процедуры обновления данных в таблице базы данных **UPDATE**;
- 1 запрос для создания процедуры извлечения данных из таблиц базы данных **SELECT**;

Все программные инструкции команд SQL сохранять в файлах с расширением ***.sql** в папке **ФИО_студента/Лаб8**.

Для каждого запроса сформулировать текстовое задание, которое должно быть выполнено к базе данных.

Создать текстовый отчет, в котором отобразить sql-команды разработанных запросов и скриншоты результатов работы из СУБД **SQL Server Management Studio**.

ПРАКТИЧЕСКАЯ РАБОТА №10. СОЗДАНИЕ, ИЗМЕНЕНИЕ, ПРИМЕНЕНИЕ И УДАЛЕНИЕ ФУНКЦИЙ И ХРАНИМЫХ ПРОЦЕДУР

10.1 Цель практической работы

Изучение синтаксиса и семантики функций и хранимых процедур Transact– SQL: способов их идентификации, методов задания и спецификации параметров и возвращаемых значений, кодирования тела и вызовов функций и хранимых процедур, применение команд для создания, изменения и удаления системных и пользовательских как скалярных, так и табличных (с одной Inline или несколькими Multi – statement командами в теле) функций, системных, пользовательских, временных (локальных или глобальных) и расширенных хранимых процедур, а также приобретение навыков программирования, отладки, тестирования и включения в группу или подключения библиотеки функций и хранимых процедур.

10.2 Методические рекомендации для выполнения практических работ

Функции и хранимые процедуры используются в SQL Server для реализации на языке Transact-SQL сложных часто используемых алгоритмов обработки данных или различных административных действий создания учетных записей, получения информации об объектах базы данных, управления свойствами сервера и баз данных, управления подсистемой репликации и автоматизации и т.д. Они хранятся в виде исходного текста и являются программными модулями, существующими независимо от таблиц или каких либо других объектов баз данных. Исключением являются расширенные хранимые процедуры, которые хранятся в двоичном формате в виде динамически подключаемых библиотек типа *.dll и создаются с помощью других языков программирования с использованием интерфейса SQL Server Open Data Services API. Такие процедуры подключаются, отключаются и выгружаются соответственно командами sp_addextendedproc, sp_dropextendedproc и DBCC dlname (FREE), где dllno me_имя dll_библиотеки.

Хранение функций и хранимых процедур в виде исходных модулей языка Transact – SQL на сервере и в соответствующих базах данных позволяет уменьшить размр запроса, посылаемого по сети от клиента на сервер, а следовательно и нагрузку на сеть, что повышает общую производительность системы. Это также позволяет упростить сопровождение программных комплексов и внесение изменений в исходный текст модулей, причем большинство изменений не отразится на работоспособности клиентских приложений.

Значительная часть функций и хранимых процедур поставляются в составе SQL Server. Они называются системными, или встроенными (built - in). Кроме того, пользователю предоставляется возможность разрабатывать и включать в свою базу данных собственные, или пользовательские (user-defined) функции и хранимые процедуры, реализующие специальные алгоритмы обработки данных. Таким образом, пользовательские функции и хранимые процедуры становятся объектами той базы данных, в которой они создавались. Поэтому при их создании, если необходимо, требуемую базу данных следует сделать текущей с помощью команды USE имя базы данных. Системные же функции хранятся на экземпляре сервера, а системные хранимые процедуры – в базе master этого же экземпляра сервера.

В SQL Server можно создавать и так называемые временные хранимые процедуры в базе данных tempdb экземпляра сервера, которые существуют лишь некоторое время, после чего автоматически уничтожаются сервером. Они бывают локальными и глобальными.

Локальные хранимые процедуры должны иметь имя, начинающееся с символа #, и могут быть вызваны только из того соединения, в котором они были созданы. Они автоматически удаляются при отключении пользователя, перезапуске или остановке сервера.

Глобальные хранимые процедуры должны иметь имя, начинающееся с символов ##, и доступны для любых соединений с экземпляром сервера, на котором они были созданы. Они удаляются либо при закрытии соединения, в контексте которого они были созданы, либо автоматически – при перезапуске или остановке сервера.

Функции и хранимые процедуры могут быть вызваны клиентскими программами, другими функциями или хранимыми процедурами, а также триггерами. В любом случае необходимо указать имя функции или хранимой процедуры и список аргументов, которые сопоставляются параметрам соответствующей функции или хранимой процедуры при этом типы аргументов и параметров должны совпадать или допускать автоматические преобразования типов. Если для некоторого параметра задано значение по умолчанию и это значение подходит для данного вызова, то соответствующий аргумент может быть опущен. Поскольку функция возвращает значение, она используется в качестве операнда некоторого выражения в виде вызова функций, состоящего из имени этой функции и списка аргументов, заключенного в круглые скобки, при этом в качестве аргументов могут быть любые выражения языка Transact – SQL, дающие в результате значения требуемых типов. Аргументы в вызове функции отделяются запятыми. Если список аргументов пуст, то круглые скобки после имени функции, как правило, задаются.

Исключения составляют некоторые системные функции, для которых круглые скобки не задаются, когда нет аргументов. Хранимые процедуры могут вызываться только командой EXECUTE, или сокращенно EXEC. За этой командой должны быть указаны имя процедуры и через пробел список аргументов, если вызывается процедура с параметрами. Аргументы разделяются запятой. Если для параметра задано значение по умолчанию, то аргумент либо совсем не задается (в конце списка), либо используется слово DEFAULT(в середине списка). Процедура может возвращать результаты только через параметры с ключевым словом OUTPUT, при этом и аргумент должен быть задан с таким же ключевым словом.

Создание, изменение и удаление функций и хранимых процедур производится соответственно командами:

CREATE FUNCTION,
CREATE PROCEDURE ,
ALTER FUNCTION,
ALTER PROCEDURE,
DROP FUNCTION,
DROP PROCEDURE.

При создании функции указывается тип возвращаемого значения и в теле функции обязательно задается команда RETURN, за которой следует выражения для вычисления возвращаемого значения. В теле процедуры использование команды RETURN(конечно, без последующего выражения) вовсе не обязательно. Когда этой команды нет, выход из процедуры будет происходить после исполнения последней команды процедуры.

Тело, как функции, так и хранимой процедуры начинается ключевым словом AS.

Поскольку каждая из них храниться как отдельный объект, то для указания конца тела не требуется записывать какое-либо специальное ключевое слово или знак. За командами создания функции или хранимой процедуры перечисляются имена параметров, начинающиеся с символа @, и их типы, а также важно значение по умолчанию. Для функции этот список заключается в круглые скобки, после которых

записывается ключевое слово RETURNS (возвращает) и тип возвращаемого значения. Для хранимой процедуры круглые скобки не используются, и задавать тип возвращаемого значение не требуется. Для тела функции часто используют ключевое слово begin после ключевого слова as и ключевое слово end в конце тела. Дополнительные опции функции или хранимой процедуры задаются ключевым словом with до начала тела. Например, опция encryption позволяет зашифровать исходный текст функции или хранимой процедуры и сделать его, таким образом, нечитабельным. Опция функции schemabinding запрещает производить какие-либо изменения в объектах базы данных. Опция хранимой процедуры recompile обеспечивает повторную компиляцию исходного текста процедуры при каждом её вызове. Наконец, опция FOR REPLICATION указывает, что данная хранимая процедура будет использоваться при репликации данных. При создании однотипных хранимых процедур можно использовать для них одно групповое имя. В этом случае конкретная процедура в группе идентифицируется своим номером, который должен задаваться как при создании, так и при вызове процедуры сразу же после группового имени и отделяться от него точкой с запятой.

Функции и хранимые процедуры можно создавать также с помощью Enterprise Manager, а хранимые процедуры еще и с помощью мастера Create Stored Procedure Wisard.

В SQL Server можно создавать функции трех классов:

–**Scalar** – возвращают обычное скалярное значение;

–**Inline** – возвращают таблицу динамической структуры, создаваемую единственной командой тела функции SELECT;

–**Multi – statement** – возвращает обычную таблицу заданной структуры, при этом количество команд в теле функции не ограничивается.

10.3 Задание для выполнения практической работы №10

Задание 1. Создать функцию для выполнения четырех арифметических операций “+”, “- ”, “*” и “/” над целыми операндами типа bigint, выполнив кодирование и проверку:

1. Кодирование

```
CREATE FUNCTION Calculator
(@ Opd1 bigint,
@ Opd2 bigint,
@ Oprt char(1) = “*”)
RETURNS bigint
AS
BEGINT
DECLARE @ Result bigint
SET @ Result =
CASE @ Oprt
WHEN “+” THEN @ Opd1 + @ Opd2
WHEN “-” THEN @ Opd1 - @ Opd2
WHEN “*” THEN @ Opd1 * @ Opd2
WHEN “/” THEN @ Opd1 / @ Opd2
ELSE 0
END
Return @ Result
END
```

2. Тестирование

```
SELECT dbo.Calculator(4,5, ‘+’),
dbo. Calculator(3,7, ‘*’) – dbo.Calculator(64,4,‘/’)*2.
9 -11
(1 row (s) affected)
```

Задание 2. Создать функцию, возвращающую таблицу с динамическим набором столбцов, выполнив кодирование и тестирование:

1. Кодирование

```
CREATE FUNCTION DYNTAB (@ State char(2))
RETURNS Table
AS
RETURNS SELECT au_id, au_lname, au_fname FROM authors
WHERE state = @ state
```

2. Проверка

```
SELECT * FROM DYNTAB (“CA”)
ORDER BY au_lname, au_fname
au_id au_lname au_fname
```

Задание 3. Создать функцию, разбивающую входную строку на подстроки, используя в качестве разделителя пробелы, выполнив кодирование и тестирование:

1. Кодирование

```
CREATE FUNCTION Parse (@ String nvarchar (500))
RETURNS @ tabl TABLE
(Number int IDENTITY (1,1) NOT NULL,
Substr nvarchar (30))
AS
BEGIN
DECLARE @ Str1 nvarchar (500), @ Pos int
SET@Str1 = @String
WHILE 1>0
BEGIN
SET@Pos = CHARINDEX(“ ”, @Str1)
IF @POS>0
BEGIN
INSERT INTO @tab1
VALUES (SUBSTRING (@Str1,1,@Pos))
END
ELSE
BEGIN
INSERT INTO @tab1e
VALUES (@Str1)
BREAK
END
END
RETURN
END
```

2. Тестирование

```
DECLARE @ Test String nvchar (500)
```

```
Set @ TestString = ‘”SQL Server 2000”
SELECT * FROM Parse (@ Test String)
-----------------------------------------------
Number Substr
-----------------------------------------------
1 SQL
2 Server
3 2000
(3 row(s)) affected)
```

Задание 4. Создать указанные три функции с помощью утилиты Enterprise Manager и проверить их синтаксис, щелкнув по кнопке Check Syntax (проверить синтаксис), затем сохранить их как шаблон, щелкнув по кнопке Save as Template (сохранить как шаблон).

Задание 5. Создать и отредактировать функцию, используя шаблон, полученный в задании №4.

Задание 6. Выбрать вновь созданную функцию и, дважды щелкнув по ней, открыть окно редактирования, а затем отредактировать текст этой функции и права доступа пользователей и ролей к данной функции.

Задание 7. Используя утилиту Enterprise Manager, ознакомиться с текстами исходных модулей системных функций (если это возможно), их семантикой и способами вызова в выражения, отображая при этом результаты их работы.

Задание 8. С помощью Transact-SQL создать три хранимые процедуры, семантика которых аналогична рассмотренным функциям, введя при этом дополнительный параметр для получения результата работы соответствующей процедуры.

Задание 9. Отредактировать указанные хранимые процедуры с помощью Enterprise Manager.

Задание 10. Создать эти же процедуры с помощью мастера Create Stored Procedure Wizard и проанализировать их свойства. Протестировать вновь созданные хранимые процедуры. С помощью sp_help получить справочные данные по одной из этих процедур.

ПРАКТИЧЕСКАЯ РАБОТА №11. СОЗДАНИЕ, ПРОГРАММИРОВАНИЕ И УПРАВЛЕНИЕ ТРИГГЕРАМИ

11.1 Цель практической работы

Изучение назначения и типов триггеров, условий их активации, синтаксиса и семантики команд языка Transact – SQL для их создания, модификации, переименования, программирования и удаления, а также приобретение навыков их проектирования, кодирования и отладки с применением хранимых процедур для получения информации о триггерах базы данных.

11.2 Методические рекомендации для выполнения практической работы

Триггер SQL Server 2000 – это специальный тип хранимых процеду, которые запускаются сервером автоматически при выполнении тех или иных действий с данными таблицы. Триггеры рвзличаются по типу команд, на которые они реагируют:

INSERT TRIGGER – запускаются при попытке вставить данные с помощью команды INSERT;

UPDATE TRIGGER – запускаются при попытке изменения данных с помощью команды UPDATE;

authsmall TRIGGER – запускаются при попытке удаления данных с помощью команды DELETE.

Параметры FOR, AFTER и INSTEAD OF, указываемые при создании триггера, определяют его поведение следующим образом:

FOR – запуск триггера при выполнении заданной в этом списке команды;

AFTER – запуск триггера после успешного выполнеиния команд списка;

INSTEAD OF – триггеры вызывается вместо выполнения команд списка.

Можно определить несколько AFTER – триггеров для каждой операции INSERT, UPDATE и DELETE. По умолчанию все триггеры являются AFTER – триггерами. Триггеры нельзя создавать для временных или системных таблиц. Команда создания триггера должна быть первой в пакете и применяться только к одной таблице. Ее формат следующий:

```
CREATE TRIGGER Имя триггера
ON {Имя таблицы\Имя представления}
[WITH ENCRYPTION] -- шифрование кода триггера;
{ {{FOR\AFTER\INSTEAD OF}
{[DELETE] [,] [INSERT] [,] [UPDATE]}
[WITE APPEND] -- только для версий 6.5 и ниже;
[NOT FOR REPLICATION] -- не для репликации;
AS sql_statement […n] -- тело триггера;
}
|
{{FOR\AFTER\INSTEAD OF}
{[INSERT] [,] [UPDATE]}
[WITE APPEND] -- только для версий 6.5 и ниже;
[NOT FOR REPLICATION] -- не длярепликации;
AS {IF UPDATE (column) -- при изменении столбца;
[{AND\OR}UPDATE (column) […n]]}-- тоже;
|
IF (COLUMNS_UPDATED() {bitwise_operator}
```

Update_bitmask)
{comparison_operator}column_bitmask […n]
}
sql_statement […n] -- тело
}
|
{{FOR\AFTER\INSTEAD OF}
{[INSERT] [,] [UPDATE]}
[WITE APPEND] -- только для версий 6.5 и ниже;
[NOT FOR REPLICATION] -- не дляреплликации;
AS {IF UPDATE (column) -- при изменении столбца;
[{AND\OR}UPDATE (column) […n]]}-- тоже;
|
IF (COLUMNS_UPDATED() {bitwise_operator}
Update_bitmask)
{comparison_operator}column_bitmask […n]
}
sql_statement […n] -- тело триггера.
}
}

Вторая альтернатива команды {IF UPDATE…} используется для детального анализа изменений содержимого колонок с помощью специальных функций, битовых масок, операторов побитовой обработки, оператор сравнения и логических операторов.

Команда ALTER TRIGGER позволяет изменить параметры и тело триггера. С помощью команды DROP TRIGGER можно удалть любой триггер базы данных.

Переименовать триггер можно системной хранимой процедурой sp_rename, а получить информацию о триггере можно при помощи системных хранимых процедур sp_helptext и sp_helptrigger.

Внутри триггера допускается использование любых команд языка Transact – SQL с некоторыми ограничениями. Также допускается и вызов хранимых процедур, включая системные.

Задание к практической работе №11

Задание 1. Создать таблицу authsmall из таблицы authors базы данных Pubs и для новой таблицы запрограммировать триггер auth_del, который будет выводить информацию о попытках удаления и количестве удаляемых строк, выполнив действия:

1. Создание таблицы authsmall с колонками au_id, au_fname, au_lname, phone и копирование в нее данных из таблицы authors:

SELECT au_id, au_fname, au_lname, phone
INTO authsmoll
FROM authors
PRINT ‘Содержимое таблицы authsmoll:’
SELECT * FROM authsmall

2. Создание и программирование триггера:

CREATE TRIGGER auth_del
ON authsmall
FOR DELETE
AS
PRINT ‘Попытка удаления’ + STR (@@ POWCOUNT)+

‘строк в таблице authsmall’
PRINT ‘Пользователь’ + CURRENT_USER
IF CURRENT_USER <> ‘dbo’
BEGIN
PRINT ‘Удаление запрещено’
ROLLBACK TRANSACTION
END
ELSE
PRINT ‘Удаление разрешено’
3. Тестирование триггера :
DELETE FROM authsmall WHERE au_fname = ‘Johnson’
DELETE FROM authsmall WHERE 2*2=5

Задание 2. Создать триггер auth_upd для таблицы authsmall, построенный в первом задании, который будет разрешать изменение столбца au_id этой таблицы всем, кроме владельца dbo, выполнив следующие действия:

1. Создание и программирование триггера:

CREATE TRIGGER auth_upd
ON authsmall
FOR UPDATE
AS
SET NOCOUNT ON -- не сообщать о завершении команд;
PRINT ‘Попытка изменения данных в таблице authsmall’
IF (COLUMNS_UPDATE () &1)! = 0 -- 1-й столбец;
PRINT ‘Изменение столбца au_id’
IF (COLUMNS_UPDATE () &2)! = 0 -- 2-й столбец;
PRINT ‘Изменение столбца au_fname’
IF (COLUMNS_UPDATE () &4)! = 0 -- 3-й столбец;
PRINT ‘Изменение столбца au_lname’
IF UPDATE (Phone)
PRINT ‘Изменение столбца phone’
IF ((CURRENT_USER = ‘dbo’) AND
(COLUMNS_UPDATED()&1)! = 0 -- 1-ый стлбец;
BEGIN
PRINT ‘Пользователь dbo не может изменять’ + ‘идентификационный номер автора’
ROLLBACK TRANSACTION
END

2. Тестирование триггера:

UPDATED authsmall SET phone =‘415 986 - 7020’, au_fname = ‘John’
WHERE au_lname = ‘Green’
UPDATED authsmall SET phone =‘913 843 - 7302’, au_id = ‘748-126859’
WHERE au_lname = ‘Smith’

Задание 3. Создать триггер для команд INSERT и UPDATE, запрещающий производить изменения для автора Billy Geitsi, выполнив действия:

1. Создание и программирование триггера:

CREATE TRIGGER auth_ ins_upd ON authsmall

```
FOR INSERT, UPDATE
AS
IF EXISTS (SELECT * FROM authsmall -- inserted;
WHERE au_lname = 'Geitsi' -- фамилия;
au_fname = 'Billy') -- имя;
BEGIN
PRINT 'Недопустимо написание книги'+
'автором Billy Geitsi'
ROLLBACK TRANSACTION
END
```

2. Тестирование триггера:

```
UPDATE authsmall SET au_lname = 'Geitsi',
au_fname = 'Billy' WHERE au_lname = 'Smith'.
```

ПРАКТИЧЕСКАЯ РАБОТА №12. СОЗДАНИЕ И УПРАВЛЕНИЕ ТРАНЗАКЦИЯМИ

12.1 Цель практической работы

Изучение способов обеспечения надежной работы SQL Server с помощью механизма транзакций и контрольных точек, приобретение навыков управления локальными и распределенными транзакциями различных видов, а также ознакомление с физической и логической архитектурой журнала транзакций и способами восстановления баз данных.

12.2 Методические рекомендации для выполнения практической работы

Одним из способов повышения надежности работы системы MS SQL Server 2000 является применение встроенного в систему механизма транзакций и контрольных точек и умелое его управление.

Транзакция – это одна или несколько последовательных команд языка Transact – SQL, образующих логически завершенный пакет и выполняемых как единое целое. Если по какой-либо причине хотя бы одна из команд пакета не выполняется, то происходит откат системы к состоянию, в котором она была до начала транзакции, и транзакция считается не выполненной. По умолчанию каждая команда выполняется как самостоятельная транзакция. При необходимости в пакете можно явно указать начало и конец транзакции.

Обработка транзакций в любой системе управления базами данных должна производиться с соблюдением следующих правил – правил ASID (Atomicity, Consistency, Isolation и Durability):

Atomicity – атомарность: выполняемые в транзакции изменения либо выполняются все, либо не выполняются вовсе;

Consistency – согласованность: все данные после выполнения транзакции должны находиться в согласованном состоянии с соблюдением всех правил и ограничений целостности;

Isolation – изолированность: изменения данных, выполняемых

Durability – долговечность: после завершения транзакции ничто не может вернуть систему в состояние, в котором она была до начала транзакции (происходит фиксация транзакции).

Транзакции должны как можно меньше включать команд и изменять минимум данных.

Соблюдение этого требования позволит наиболее эффективным образом обеспечить одновременную работу с данными множества пользователей системы SQL Server. Они определяются на уровне соединения с сервером. Поэтому при закрытии соединения происходит откат невыполненной транзакции, и ее нельзя выполнить позже после восстановления соединения.

SQL Server поддерживает три вида определений транзакций: явное,автоматическое и подразумеваемое.

Для управления явными транзакциями применяют команды:

BEGIN TRANSACTION [Имя транзакции] – начало транзакции;

COMMIT TRANSACTION [Имя транзакции] – конец транзакции;

ROLLBACK TRANSACTION[Имя транзакции] – откат транзакции;

В последних двух командах слово TRANSACTION можно либо опускать, либо заменять словом WORK. Во всех трех командах допускается использование сокращения

TRAN вместо слова TRANSACTION и переменной строкового типа, которой присваивается имя транзакции, вместо непосредственного указания этого имени. Дополнительный аргумент WITH MARK ‘Описание’ позволяет специальным образом маркировать транзакцию в журнале транзакций, что используется при восстановлении базы данных.

Если команды явного определения транзакций не используются, то сервер работает в одном из двух режимов:

а) в режиме автоматического начала транзакций, в котором каждая рассматривается как отдельная транзакция, при этом если команда выполнена успешно, то сделанные ей изменения фиксируются, и выполняется следующая команда, в противном случае производится откат транзакции и выполнение команды повторяется;

б) в режиме неявного начала транзакции, когда начала транзакции не указывается, а ее завершение задается явно командой COMMIT или инициируется командами:

ALTER TABLE,
CREATE,
DELETE,
DROP,
FETCH,
GRANT,
INSERT,
OPEN,
REVOKE,
SELECT,
TRANCATE
TABLE и UPDATE;

в этом режиме можно использовать команды COMMIT и ROLLBACK; после завершения текущей транзакции, начинается выполнение следующей, если не был задан откат транзакции.

Режим автоматического начала транзакций устанавливается по умолчанию или командой.

SET IMPLICIT – TRANSACTION OFF

Режим неявного (или подразумевающегося) начала транзакций задается только командой **SET IMPLICIT – TRANSACTION ON.**

Когда в запросах используются разные базы данных, даже если они расположены на одном физическом сервере, необходимо использовать распределенные транзакции, которые на самом деле представляют собой несколько отдельных транзакций, выполняемых локально в каждой базе данных, используемой в запросе. Если при этом одна из локальных транзакций не выполняется, то происходит откат распределенной транзакции.

Для управления распределенными транзакциями в MS SQL Server 2000 используется координатор DTC (Distribution Transaction Coordinator), удовлетворяющий спецификации “X/OPUN XA for Distributed Transaction Processing”. Координатор MS DTC начинает и заканчивает локальные транзакции, а также откатывает их назад, если одна из них закончилась с ошибкой. При выполнении распределенных транзакций пользователь может обращаться не только к серверам SQL Server, но и к другим источникам данных: Oracle, Access, источники ODBC и другие.

Для клиентского приложения работа с определенными транзакциями практически ничем не отличается от работы с локальными транзакциями, так как все согласование транзакций в различных источниках данных выполняется автоматически и невидимо для пользователя.

Распределенная транзакция может быть начата несколькими способами.

1. Если приложение в локальной транзакции использует распределенный запрос, то сервер автоматически начинает выполнение распределенной транзакции.

2. Если приложение начинает локальную транзакцию и из нее вызывает удаленную хранимую процедуру при установленном параметре REMOTE_PROC_TRANSACTION, то эта транзакция автоматически расширяется до распределенной транзакции (см.sp_configure).

3. Приложение может начать распределенную транзакцию, используя методы OLE DB или ODBC.

4. Сервер начинает выполнение распределенной транзакции, если встречает команду BEGIN DISTRIBUTED TRANSACTION Имя транзакции.

В этом случае для завершения и отката транзакций используются команды COMMIT TRANSACTION и ROLLBACK TRANSACTION.

При работе с явными транзакциями можно использовать вложенные транзакции, выполнение которых инициируется из тела уже активной транзакции. Для управления вложенными транзакциями используются те же команды, при этом каждая команда COMMIT работает только с последней начатой транзакцией. Если в команде ROLLBACK не задано имя транзакции, то откатываются все вложенные транзакции и транзакция самого высокого уровня.

Если же имеется необходимость откатить лишь часть транзакций, то предварительно надо создать точку сохранения с помощью команды SAVE TRANSACTION, которую следует указывать при откате. Функция @@TRANSACTION предназначена для определения количества активных транзакций, начатых в активном соединении.

Во всех транзакциях нельзя использовать следующие команды:

ALTER DATABASE,
BACKUP LOG,
CREATE DATABASE,
DISK INIT,
DROP DATABASE,
DUMP TRANSACTION,
LOAD DATABASE,
LOAD TRANSACTION,
RECONFIGURE,
RESTORE DATABASE,
RESTORE
LOG, UPDATE STATISTICS,

а также системной хранимой процедуры sp_droption и любой другой хранимой процедуры, изменяющей значения в системной базе master. Для отката таких действий необходимо использовать архивирование базы данных и последующее ее восстановление.

MS SQL Server для обеспечения целостности данных использует журнал транзакции, который имеет сложную физическую и логическую архитектуру и в котором производятся все промежуточные модификации до завершения транзакции, после которой происходит фиксация изменений в самой базе данных. Контрольные точки минимизируют данные в журнале транзакций.

Задание к практической работе №12

Задание 1. Проверить режимы автоматического начала транзакций и неявного начала транзакций, используя переключатель IMPLICIT_TRANSACTION и команду COMMIT.

Задание 2. Создать несколькими способами распределенные транзакции и убедиться в корректности их выполнения.

Задание 3. Создать вложенные транзакции, выполнив следующие команды:

```
CREATE TABLE #aaa (cola int) -- 0-й уровень
BEGIN TRAN -- 1-й уровень
INSERT INTO #aaaVALUES (111)
BEGIN TRAN -- 2-й уровень
INSERT INTO #aaaVALUES (222)
BEGIN TRAN -- 3-й уровень
INSERT INTO #aaaVALUES (333)
SELECT * FROM #aaa
SELECT ‘Вложенность транзкций’, @@TRANCOUNT
ROLLBACK TRAN
SELECT * FROM #aaa -- откат на 0-й уровень
SELECT ‘Вложенность транзакций’, @@TRANCOUNT
```

Проанализировать полученные результаты.

Задание 4. Написать пример пакета запросов с использованием команд COMMIT и ROLLBACK для автоматических, неявных и явных транзакций.

Задание 5. Написать пример пакета команд, иллюстрирующих использование средств оптимизации при откате транзакций.

Задание 6. Используя средства MS SQL Server 2000, изучить физическую и логическую архитектуру журнала транзакций.

Задание 7. С помощью системной хранимой процедуры sp_configure изменить интервал контрольных точек для базы данных Pubs.

Задание 8. Уточнить синтаксис команд управления транзакциями и написать пример пакета с использованием всех вариантов этих команд.

ПРАКТИЧЕСКАЯ РАБОТА №13. СОЗДАНИЕ, ПРИМЕНЕНИЕ И УПРАВЛЕНИЕ КУРСОРАМИ

13.1 Цель практической работы

Изучение назначения и типов курсоров, синтаксиса и семантики команд языка Transact – SQL для создания и открытия курсоров, выборки данных из курсора и изменения строк таблиц с помощью курсоров, удаления данных, закрытия и освобождения курсоров, а также приобретения навыков их применения и управления с помощью команд и системных хранимых процедур SQL Server.

13.2 Методические рекомендации для выполнения практической работы

Набор данных, имеющийся в таблице базы данных, называется полным набором строк таблицы (complete set of rows). Набор строк, возвращаемый команду SELECT, называется результирующим набором данных (result set). Он является частью полного набора, отфильтрованного горизонтально с помощью условий, заданных в разделе WHERE. Можно в результирующий набор не включать те или иные столбцы, применяя вертикальную фильтрацию.

Результирующие наборы могут содержать сотни тысяч строк, и клиентские приложения не всегда справляются с таким объемом данных. Для решения этой проблемы используются курсоры, которые представляют собой окна, налагаемые на результирующие набором данных и выделяющие требуемую часть данных. Перемещая созданный курсор, можно получить доступ ко всем результирующим данным. Таким образом, курсоры SQL Server представляют собой механизм обмена данными между сервером и клиентом, который минимизирует ресурсы клиентского приложения. Однако всегда, когда это возможно, следует избегать использования курсоров и применять команды SELECT, UPDATE, DELETE и INSERT.

MS SQL Server поддерживает три вида курсоров:

1. Курсоры Transact – SQL, которые применяются внутри триггеров, хранимых процедур и сценариев;

2. Курсоры сервера, которые действуют на сервере и реализуют программный интерфейс приложений для ODBC, OLE DB и DB_Library;

3. Курсоры клиента, которые реализуются на клиенте и выбирают весь результирующий набор для ускорения обработки данных.

Один курсор может базироваться на нескольких таблицах, расположенных в разных базах данных. Операция считывания определенных в курсоре данных называется выборкой (fetch). Если за одну операцию курсор позволяет выбрать несколько строк таблицы, то такой курсор называется блочным. По способу просмотра данных курсоры бывают последовательные (forward only), которые обеспечивают просмотр строк только в одном направлении – от начала к концу, и прокручиваемые, которые допускают просмотр в обоих направлениях и переход к произвольной строке.

По представляемым возможностям курсоры делятся на четыре типа:

статические,
динамические,
последовательные
ключевые.

Тип курсора определяется на стадии его создания и не может быть изменен.

Статический курсор (static cursor) называют также курсорами моментального снимка (snapshot cursor). При открытии такого курсора сервер выбирает все данные,

соответствующие заданным критериям, и сохраняет результирующий набор строк в системной базе данных tempdb без изменения, если даже исходные строки и изменяются. Поэтому статический курсор всегда открывается в режиме “только для чтения”.

Динамические курсоры (dynamic cursor) противоположны статическим. При их использовании не создается полная копия исходных данных, а выполняется динамическая выборка данным из исходных таблиц только при обращении пользователя к тем или иным данным, при этом на время выборки соответствующие строки блокируются сервером. После выборки строк исходные строки могут изменяться пользователями, но эти изменения уже не отражаются в выбранных строках. С другой стороны, изменения в выбранных строках не будут видны другим пользователям, пока они не будут подтверждены (committed).

Последовательный курсор выбирает данные только от начала к его концу. Он не хранит результирующий набор. Строки считываются из базы данных, как только они выбираются в курсоре. Это позволяет отображать все изменения в базе данных. В курсоре видно самое последнее состояние данных.

Курсоры, зависящие от набора ключей (keyset-driven cursor), или ключевые курсоры, построены на основе уникальных идентификаторов. Множество всех уникальных идентификаторов (ключей) строк таблиц базы данных называется набором ключей. Сервер блокирует строки исходных таблиц только на время составления таблицы ключей.

Ключевой курсор представляет набор ключей, идентифицирующих строки полного результирующего набора курсора. Набор ключей строится в системной базе данных tempdb.

При работе с курсорами можно выделить пять основных операций: создание курсора, открытие курсора, выборка из курсора и изменение строк данных с помощью курсора, закрытие курсора и освобождение курсора.

Формат команды языка Transact – SQL для создания курсора следующий:
DECLARE Имя курсора CURSOR [LOCAL/GLOBAL]
[FORWARD_ONLY\SCROLL]
[STATIC\KEYSET\DINAMIC\FAST_FORWARD]
[READ_ONLY\SCROLL_LOCKS\OPTIMISTIC]
[TYPE_WARNING]
FOR select_statement
[FOR UPDATE [OF column_name [,…n]]].

Для открытия курсора используется команда
OPEN {{[GLOBAL] Имя курсора}\Имя переменной}.
Для выборки данных необходимо применять команду
FETCH [[NEXT\PRIOR\FIRST\LAST\ABSOLUTE {n\@nvar}
RELATIVE {n\@nvar}
]
FROM
]
{{[GLOBAL] Имя курсора}\Имя переменной}
[]INTO @ Имя переменной [,…n]]

Команды **UPDATE, DELETE, CLOSE и DEALLOCATE** позволят соответственно произвести изменение данных, удаление данных, закрытие и освобождение курсора.

Задание к практической работе №13

Задание 1. Создать курсор curs для таблицы authors базы данных Pubs, выполнил следующие действия:

1. Создание курсора:

```
DECLARE curs cursor
GLOBAL SCROLL KEYSET TYPE_WARNING
FOR SELECT au_lname, au_fname, phone, title,
price, advance, sales = ytd_sales
FROM titleauthor INNER JOIN authors
ON titleauthor. au_id = authors. au_id
INNER JOIN titles
ON titleauthor. title_id = titles. titles_id
WHERE authors. state <> ‘CA’
FOR UPDATE.
```

2. Открытие курсора:

```
OPEN curs
```

3. Выборка данных:

```
DECLARE @@Str1 char (5),
@@VFName varchar (20),
@@VLName varchar (40),
@@VPhone char (12),
@@ VTitle varchar (80),
@@VPrice money,
@@VAdrance money,
@@VSales int,
@@Count1 timyint,
@@Var1 money
SET @@Count = 1
SET @@Var1 = 0
WHILE @@Count1 <@@CURSOR_ROWS
BEGIN
IF @@Count = 1
FETCH ABSOLUTE 1 FROM CURS INTO @@VFName,
@@VLName, @@VPhone, @@VTitle, @@VPrice,
@@VAdrance, @@VSales
ELSE
FETCH curs INTO @@VFName,
@@VLName, @@VPhone, @@VTitle, @@VPrice,
@@VAdrance, @@VSales
SET @@Count1 = @@Count1+1
SET @@Var1 = @@Var1+@@Vprice*@@VSales-@@VAdrance
END
SELECT ‘Итого прибыли:’, @@Var1
```

4. Закрытие и освобождение курсора:

```
CLOSE curs
DEALLOCATE curs.
```

ПРАКТИЧЕСКАЯ РАБОТА №14. СИСТЕМА БЕЗОПАСНОСТИ SQL SERVER

14.1 Цель практической работы

Изучение общих правил разграничения и предоставление прав доступа пользователям баз данных, архитектуры и компонент системы безопасности SQL Server и режимов аутентификации пользователей, а также приобретение навыков администрирования системы безопасности: создания и управления учетными записями, управления пользователями, ролями и группами.

14.2 Методические рекомендации для выполнения практической работы

Система хранения информации должна быть максимально защищена как от случайного, так и от злонамеренного повреждения, искажения или утечки информации. С этой целью, прежде всего, надо определить круг пользователей, которые будут иметь доступ к базам данных. Далее, для этих пользователей необходимо создать учетные записи в домене Windows, а также в соответствующем экземпляре сервера SQL Server, чтобы разрешить им обращаться к этому серверу. Разрешение доступа к серверу не дает автоматически доступа к его базам данных и их объектам.

Второй этап планирования безопасности использования баз данных сервера заключается в определении действий, который может выполнить конкретный пользователь. Полный доступ к базам данных и всем их объектам имеет администратор – ему позволено все. Второе лицо после администратора – это владелец объекта. При создании любого объекта в любой базе данных ему назначается владелец, который может устанавливать права доступа к этому объекту, модифицировать и удалять его. Все остальные пользователи, составляющую третью, основную группу, имеют права доступа, выданные администратором или владельцем объекта. Эти права должны быть тщательно спланированы в соответствии с занимаемой должностью и необходимостью выполнения конкретных действий.

Для работы с базами данных пользователи любой категории проходят следующие два этапа проверки системой безопасности. На первом этапе пользователь идентифицируется по имени учетной записи и паролю, т. е. проходит аутентификацию. Если данные введены правильно, то пользователь подключается к требуемому серверу, выбрав его из списка серверов и исполнив команду подключения либо с помощью Enterprise Manager, либо исполнив команды Transact –SQL в Query Analyzer. Пользователю при этом будут предоставлены те права доступа к серверу, которые имеют роль сервера, содержащая данного пользователя. Подключение к выбранному серверу, или регистрация, не дает автоматического доступа к базам данных.

На втором этапе по регистрационному имени находится имя пользователя базы данных, которое было создано администратором, и пользователь получает права доступа к выбираемой базе данных в соответствии с той ролью базы данных, в которую был включен этот пользователь администратором базы на этапе конфигурирования системы без опасности. В разных базах данных один и тот же пользователь может иметь одинаковые или разные имена пользователя базы данных с разными правами доступа, как правило. Таким образом, пользователь имеет одно имя, которое он задает при входе в систему, и, возможно, несколько имен для доступа к базам данных и их объектам.

Для доступа приложений к базам данных им также понадобятся права. Чаще всего приложениям выдаются те же права, которые предоставлены пользователям, запускающим эти приложения. Однако для работы некоторых приложений необходимо

иметь фиксированный набор прав доступа, не зависящих от прав доступа пользователя. Это обеспечивается использованием специальных ролей приложения.

Итак, компонентами системы безопасности SQL Server 2000 на уровне сервера являются: система аутентификации средствами Windows и средствами SQL Server, учетные записи пользователей и встроенные роли сервера. На уровне базы данных компонентами системы безопасности являются: идентификация пользователей баз данных, фиксированные и пользовательские роли баз данных, а также роли приложений.

Фиксированными ролями сервера являются:

Sysadmin – для выполнения любых действий в сервере;

Sereradmin– для конфигурирования и выключения сервера;

Setupadmin– для управления связанными серверами и процедурами, автоматически запускающимися при старте сервера;

Securityadmin– для управления учетными записями и правами на создание базы данных, а также для контроля журнала ошибок;

Processadmin - для управления процессами, запущенными на сервере__

Dbcreator – для создания и модификации баз данных;

Diskadmin– для управления файлами сервера;

Bulcadmin– для массивного копирования баз данных.

Фиксированную роль сервера нельзя удалить или модифицировать. Нельзя также создать новую фиксированную роль. Предоставить права доступа к серверу можно только путем включения пользователя в требуемую роль сервера. Таким образом, роли сервера позволяют объединять пользователей, выполняющих одинаковые функции, для упрощения администрирования системы безопасности SQL Server. В предыдущих версиях SQL Server можно было использовать только учетную запись sa, которая предоставляла все права доступа к серверу.

При создании базы данных сервер автоматически создает для нее фиксированные роли, которые, как и фиксированные роли сервера, нельзя удалить или модифицировать:

Db_owner – для выполнения любых действий в базе данных;

Db_accessadmin – для добавления и удаления пользователей;

Db_securityadmin – для управления всеми разрешениями, объектами, ролями и именами ролей;

Db_ddladmin – для выполнения любых команд DDL, кроме GRANT, DENY и REVOKE;

Db_backupoperator– для выполнения команд DBCC, CHECK, POINT и BACKUP;

Db_datareader – для контроля данных во всех таблицах базы данных и и чтения;

Db_datawriter – для модификации данных в любых таблицах базы данных;

Db_denydatareader – для запрета просмотра данных в любой таблице базы данных;

Db_denydatawriter – для запрета модификации данных во всех таблицах базы данных.

Кроме этих фиксированных ролей любой базы данных есть еще одна роль public, членами которой автоматически становятся все пользователи, имеющие тот или иной доступ к базе данных. Эта роль имеет специальное назначение и обеспечивает минимальные права доступа к базе данных тем пользователям, для которых их права не определены явно. Эта роль имеется во всех базах данных, включая системные master, tempdb, msdbu model и не может быть удалена.

Если в базе данных разрешен пользователь quest, то установленный для public доступ будут иметь все пользователи, получившие доступ к SQL Server.

В отличие от сервера базы данных могут иметь пользовательские роли и роли приложения, которые создает администратор с помощью Enterprise Manager или Transact_SQL индивидуально для групп пользователей и групп приложений, наделяя их необходимыми правами доступа к конкретной базе данных.

В любую роль базы данных можно включать:

а) пользователей сервера;

б) роли сервера;

в) пользователей Windows;

г) группы пользователей Windows.

Средствами Enterprise Manager можно включать только пользователей сервера. Процедура SQL Server sp_addrolemember ‘role’, ‘security_account’ позволяет включать как роли сервера, так и пользователей Windows, в том числе и групп пользователей с помощью задания их учетной записи в SQL Server или Windows ‘security_account’ и указания требуемой роли ‘role’.

Работа с данными и выполнение хранимых процедур требуют наличия класса доступа, называемого правами на доступ к объектам баз данных: таблицам и ее столбцам, представлениям и хранимых процедурам.

Такими правами являются:

SELECT, INSERT, UPDETE, DELETE, RFERENCES – для таблиц и представлений, а SELECT и для столбца (тоже и для UPDETE), SELECT и UPDETE – для столбца таблицы или представления;

EXECUTE – для хранимых процедур и функций.

Здесь право RFERENCES разрешает создавать внешние ключи и представления для таблиц.

Командой GRANT можно разрешать пользователям определенные права доступа к объектам, командой **DENI** – запрещать их.

Помимо прав доступа к объектам имеются еще и права, на создание объектов базы данных и самой базы данных:

CREATE DATABASE – на создание базы данных ;

CREATE TABLE – на создание таблиц;

CREATE VIEW – на создание представлений;

CREATE DEFAULT – на создание умолчаний;

CREATE RULE – на создание правил;

CREATE PROCEDURE – на создание хранимых процедур;

BACKUP DATABASE – на резервное копирование баз данных;

BACKUP LOG – на резервное копирование журнала транзакций;

ALL – на создание любых объектов.

При установке SQL Server имеется возможность выбрать один из двух режимов аутентификации:

а) средствами Windows ;

б) средствами Windows и/или средствами SQL Server.

В первом случае после успешной аутентификации с помощью Windows SQL Server автоматически обеспечивает доступ пользователя к требуемому экземпляру сервера и к нужной базе данных. Этот метод подключения называется методом установления доверительного подключения. В этом случае член стандартной роли sysadmin или securityadmin должен указать серверу, какие группы или пользователи Windows имеют

доступ к серверу. Во время подключения пользователя его имя и пароль запрашиваются только один раз при входе в систему Windows.

Во втором режиме системным администратором, входящим в роль sysadmin или securityadmin, должна быть создана и сконфигурирована учетная запись в SQL Server для каждого пользователя. Эта запись будет содержать собственное имя, имя экземпляра сервера и пароль. Две такие записи с именем BUILTIN\Administrators и sa создаются автоматически при установке сервера. Обе эти записи автоматически включаются также во встроенную роль сервера sysadmin, в результате системные администраторы получают полный доступ ко всем базам данных с именем пользователя dbo (DataBase Owner). Если функции системного администратора и администратора баз данных выполняют разные люди, следует исключить учетную запись BUILTIN\Administrators. Запись sa не следует использовать, так как она предназначена для совместимости со старыми версиями SQL Server и для входа в сервер, если администратор баз данных забыл пароль. Как правило, для администратора баз данных создается отдельная учетная запись с ролью сервера sysadmin.

После того как пользователь прошел аутентификацию и получил идентификатор учетной записи (LoginID), он считается зарегистрированным и ему предоставляется доступ к серверу.

Учетная запись при создании была связана с конкретной базой данных, а пользователь – с конкретным именем пользователя базы данных. Именно пользователи баз данных являются специальными объектами, которым предоставляются права доступа к данным. Если учетная запись не связывается с конкретным пользователем, то такому пользователю предоставляется неявный доступ с использованием гостевого имени quest, которому даются минимальные права владельцами баз данных. Все учетные записи связаны с quest.

Если в базе данных разрешен пользователь quest, то установленный для роли public доступ будут иметь все пользователи, получившие доступ к SQL Server.

Для управления системой безопасности сервера SQL Server можно использовать следующие хранимые процедуры и команды языка Transact_SQL:

sp_addapprole – создать роль для приложения ;

sp_addlogin – создать новую учетную запись сервера;

sp_addrole – создать новую роль в базе данных;

sp_addrolemember – добавить члена в роль базы данных;

sp_addsrvrolemember – добавить члена в фиксированную роль сервера;

sp_approlepassword – изменить пароль для роли приложения;

sp_defaultldb – изменить базу данных по умолчанию для учетной записи;

sp_defaultlanguage – изменить язык по умолчанию для учетной записи;

p_denylogin – запретить доступ пользователю или группе Windows;

sp_dropapprole – удалить роль приложения;

sp_droplinkedsrvlogin – удалить отображения учетной записи с другого сервера;

sp_droplogin – удалить учетную запись сервера;

sp_droprole – удалить роль базы данных;

sp_droprolemember – удалить пользователя из роли базы данных;

sp_dropsrordemember – удалить члена из роли сервера;

sp_grantdbaccess – разрешить доступ к базе данных учетной записи сервера, пользователям и группам пользователей Windows;

sp_grantlogin – разрешить доступ к серверу;

sp_helpdbfixedrole – выдать список фиксированных ролей в базе данных;

sp_helplogins – посмотреть учетную запись;

sp_helpntgraep – посмотреть группы NT в сервере;

sp_helprole – посмотреть роли, определенные в базе данных;
sp_helpsrvrole – выдать список фиксированных ролей сервера;
sp_helpsrvrolemember – выдать информацию о члене роли сервера;
sp_helpuser – просмотреть информацию о пользователе;
sp_password – изменить пароль учетной записи сервера;
sp_setapprole – инициализировать роль приложения;
GRANT – предоставить доступ;
DENI – запретить доступ;
REVOKE – неявно отключить доступ.

Задание к практической работе №14

Задание 1. Создать учетную запись SQL сервера, используя графическую утилиту Enterprise Manager, выполнив следующие действия:

1. Выбрать нужный сервер.
2. Открыть папку Security этого сервера.
3. Выбрать объект Logins, щелкнув по соответствующему значку.
4. В правом окне просмотреть список учетных записей данного сервера:

Name – имя учетной записи сервера;
Type – происхождение учетной записи:
User W– пользователь Windows;
Group W– группа пользователей Windows;
Standard – пользователь SQL сервера;
Server Access – доступ к серверу SQL:
Permit – разрешен;
Deny – запрещен;
Default Database – база данных по умолчанию, к которой подключен пользователь(обязательный параметр)
User – имя пользователя базы данных;
Default Language – язык по умолчанию для данной учетной записи.

5. Для создания новой учетной записи сервера открыть контекстное меню объекта Logins, щелкнув по нему правой клавишей мыши или по значку на панели инструментов левой клавишей мыши.
6. В появившемся диалоговом окне на вкладе General (общие) ввести имя учетной записи в поле Name.
7. Выбрать тип аутентификации: Windows Authentication или SQL Server Authentication.
8. Если выбрана аутентификация Windows, то задать в поле Domain имя домена, в котором учтен пользователь или группа Windows. Имя заданного домена добавиться впереди имени пользователя также и в поле Name (для выбора домена использовать кнопку”…”).
9. В группе Security Access (безопасный доступ) установить переключатель Grant Access (доступ разрешен). Установка переключателя Deny Access навсегда запретит\ регистрацию пользователя или домена Windows.
10. Если выбрана аутентификация SQL Server, то задать только пароль для учетной записи.
11. Задав параметры аутентификации Windows или SQL Server, указать в группе Defaults (умолчания) имя базы данных в поле Database, к которой пользователь будет

подключаться автоматически, и язык Language (Russian). Если имя базы данных не задать, то сервер будет автоматически подключать к базе master.

12. Включить создаваемую учетную запись в требуемую встроенную роль сервера:

Sysadmin, Serveradmin, Setupadmin, Securityadmin, Processadmin, Dbereator, Diskadmin, Bulkadmin, установив флажок против этой роли на вкладке Server Role.

13. На вкладке Database Access выбрать требуемую базу данных, установив флажок слева от ее имени, и задать имя пользователя, в которое будет отображаться рассматриваемая учетная запись, а в нижней части вкладки с помощью флажка включить пользователя в ту или иную роль в зависимости от его работы с базой данных.

14. Щелкнув по кнопке Properties (свойства) и просмотреть список пользователей, включенных в выбранную роль рассматриваемой базы данных.

15. Щелкнув по кнопке Permissions (права) и просмотреть список прав, предоставленных выбранной роли базы данных.

16. Закрыть все окна.

17. Вновь открыть список учетных записей сервера, дважды щелкнуть по вновь созданной записи и проверить правильность введенных параметров.

18. Закрыть все окна.

19. Приступить к работе с базами данных, используя новую учетную запись.

Задание 2. Включить учетную запись пользователя или группы пользователей Windows в фиксированную роль сервера SQL с помощью Enterprise Manager, выполнив следующие действия:

1 Выбрать требуемый сервер в левом окне Tree.

2. Открыть объекты сервера, щелкнув по его кнопке”+”.

3. Открыть объекты Security этого сервера, щелкнув по кнопке “+” для Security.

4. Выбрать объект Logins (записи) и щелкнуть по нему, при этом в правом окне откроется список учетных записей сервера.

5. Дважды щелкнуть по требуемой учетной записи сервера.

6. В открывшемся окне на Server Login Properties открыть вкладку Server role.

7. Установить флажки возле тех ролей сервера, в которые требуется включить конфигурируемую запись.

8. Закрыть все открытые окна, щелкая по кнопкам “OK” этих окон.

9. Повторить задания, используя следующий набор команд:

а) Security/Server Rolees;

б) Щелкнуть левой клавишей;

в) В правом окне выбрать требуемую фиксированную роль;

г) Два раза щелкнуть по выбранной роли;

д) В открывшемся окне Server Role Properties щелкнуть по кнопке Add вкладки General;

е) Добавить учетные записи в заданную роль;

ж) Закрыть окно со списком учетных записей;

з) На вкладке Permission окна Server Role Properties просмотреть предоставляемые права для рассматриваемой роли.

10. Закрыть все открытые диалоговые окна, щелкая по кнопкам ОК.

11. Проверить правильность выполненных действий.

Задание 3. Создать нового пользователя базы данных для учетной записи Windows с помощью Enterprise Manager, выполнив следующие действия:

1. Выбрать требуемый сервер и требуемую базу данных в левом окне Tree.

2. Открыть объекты выбранной базы данных, щелкнув по значку ”+” этой базы.

3. Выбрать в раскрывшемся списке объектов рассматриваемой базы данных объект Users (пользователи).

4. Щелкнуть правой клавишей мыши и открыть контекстное меню объекта Users (пользователи).

5. В контекстном меню исполнить команду New Database User (новый пользователь базы данных).

6. В открывшемся диалоговом окне ввести:

а) в поле Login Name – имя учетной записи пользователя или группы пользователей Windows;

б) в поле User Name – имя нового пользователя рассматриваемой базы данных.

7. Включить нового пользователя в необходимые роли базы данных:

public, db – owner, db – denydatareader и т.д. Для этого требуемые роли надо отметить флажками в списке фиксированных ролей базы данных, расположенном в правой части окна.

8. Щелкнуть по кнопке Properties и, просмотрев список всех пользователей базы данных, убедиться, что новый пользователь включен этот список.

9. Щелкнуть по кнопке Permission и задать права доступа пользователя к объектам базы данных: SELECT, INSERT, UPDATE, DELET, EXEC, DRI. В окне находится полный список объектов базы данных.

10. Щелкнуть по кнопке Columns (столбцы) для выбранной базы данных и задать права доступа к конкретным столбцам таблицы: SELECT и/или UPDATE.

11. Закрыть все открытые диалоговые окна, щелкая по кнопкам ОК.

12. Проверить работу нового пользователя с рассматриваемой базой данных и его права.

Задание 4. Создать учетную запись SQL сервера, используя мастер Create Login Wizard, выполнив следующие действия:

1. Выполнить команду в Enterprise Manager Run an Wizard/Create Login Wizard.

2. В открывшемся окне мастера ознакомиться с этапами создания учетной записи сервера:

а) Select an authentication mode – выбрать режим аутентификации;

б) Grant access to security roles – представить доступ секретным ролям;

в) Grant access to databases – предоставить доступ к базам данных.

3. Щелкнуть по кнопке Next.

4. Выбрать режим аутентификации: Windows или SQL Server.

5. Щелкнуть по кнопке Next.

6. Если выбран режим аутентификации Windows, то в открывшемся окне в поле Windows account задать имя учетной записи или группы Windows и домена и указать тип доступа: Grant access to the server (предоставить доступ к серверу) или Deny access to the server (заретить доступ к серверу).

7. Если выбран режим аутентификации SQL Server, то помимо имени учетной записи, задаваемой в поле Login I указать пароль в поле Password (пароль) и в поле Confirm Password (подтвердить пароль). Этот пароль пользователь будет использовать при подключении к SQL серверу.

8. Щелкнуть по кнопке Next в том или в другом случае.

9. Включить учетную запись в требуемые фиксированные роли сервера, устанавливая против роли флажок.

10. Щелкнуть по кнопке Next.

11. Разрешить для учетной записи доступ к базам данных, отмечая их флажком.

12. Щелкнуть по кнопке Next.

13. Мастер автоматически создаст имена пользователей баз данных.

14. Проверить сделанные установки.

Задание5. Создать новую пользовательскую роль баз данных с помощью Enterprise Manager, выполнив следующие действия:

1. Выбрать требуемую базу данных.

2. Открыть объекты выбранной базы данных, щелкнув по значку “+” этой базы.

3. Выбрать в открывшемся списке объект Role (роль).

4. Открыть контекстное меню объекта Role (роль).

5. Исполнить команду меню New Database Role (новая роль базы данных).

6. В открывшемся диалоговом окне ввести имя роли в поле Name, которое должно быть уникальным в пределах базы данных.

7. Выбрать тип роли: стандартная роль Standart role или роль приложения Application Role.

8. Если выбрана стандартная роль, то с помощью кнопки “добавить ” Add включить в нее нужных пользователей.

9. Если выбрана роль приложения, то ввести в поле Password пароль, который будет использоваться для инициализации данной роли приложения. Нельзя добавлять пользователей в роль приложения.

10. Для созданной стандартной роли или роли приложения (для пользовательской роли) задать права доступа к объектам базы данных на вкладке Permissions (права), выполнив действия:

а) выделить очередной объект базы данных;

б) для каждого права: SELECT, INSERT, UPDATE, DELETE, EXEC и DRI – установить одно из трех состояний доступа:

V – GRANT – предоставить,

X – DENI – запретить,

– REVOKE – неявное отклонение (т.е. может иметь доступ через членство роли).

Закрыть все окна окна, щелкая по кнопке “OK” каждого окна.

Проверьте правильность выполненных действий.

ПРАКТИЧЕСКАЯ РАБОТА №15. АДМИНИСТРИРОВАНИЕ СЕРВЕРА БАЗ ДАННЫХ MS SQL SERVER

15.1 Цель практической работы

Приобретение навыков регистрации удаленных серверов с помощью утилиты Enterprise Manager, мастера Register Server Wizard, а также команд Transact-SQL и системной хранимой процедуры sp-addserver. Приобретение навыков управления основной службой MSSQLServer и вспомогательными службами сервера MS SQL Server: задание режима автоматического запуска службы, ручной запуск службы, запуск сервера в однопользовательском режиме, с минимальными требованиями и нестандартной конфигурации, приостановка службы и остановка служб и сервера.

15.2. Методические рекомендации для выполнения практической работы

15.2.1. Регистрация удаленных серверов

Перед использованием локального или удаленного сервера в среде Enterprise Manager его необходимо зарегистрировать.

При запуске Enterprise Manager первый раз регистрация локального сервера происходит автоматически. После установки дополнительных копий они также регистрируются в Enterprise Manager автоматически. И только при работе с удаленными серверами их необходимо регистрировать, используя среду Enterprise Manager для выполнения команд меню, запуска Register Server Wizard или интерпретации команд Transact-SQL и системной хранимой процедуры sp-addserver.

При регистрации сервера необходимо указать следующую информацию:

1. Имя сервера.
2. Тип безопасности, используемый для входа на сервер.
3. Имя учетной записи и пароль, используемые для входа на сервер.
4. Имя группы в иерархии групп, в которой необходимо зарегистрировать сервер.

Утилита Enterprise Manager представляет собой всего лишь графический интерфейс для выполнения системных хранимых процедур SQL Server 2000. Таким образом, она является клиентским средством, устанавливающим соединение с SQL Server и выполняющим те или иные процедуры. Поэтому, прежде чем эта программа сможет выполнить какие-либо операции с сервером, она должна получить соответствующие права доступа, т.е. пройти аутентификацию.

15.2.2 Запуск, остановка и приостановка служб сервера

До выполнения каких-либо работ по администрированию сервера MS SQL Server или баз данных, а также манипулированию данными необходимо запустить сервер. Точнее говоря, запустить его основную службу MSSQLServer. Только после запуска этой службы и проверки прав доступа пользователя, пользователь сможет выполнять функции, определенные его правами и разрешениями. Остальные службы являются вспомогательными, и их работа строится на фундаменте, обеспечиваемом службой MSSQLServer. Например, служба SQLServerAgent запускается лишь тогда, когда требуется автоматическое администрирование и управление системой на базе SQL Server. Служба MSSearch используется для работы с электронными документами, обеспечивает полнотекстовый поиск информации и, как правило, используется автономно. Служба MSDTC позволяет организовать доступ к распределенным источникам информации и управлять распределенными транзакциями.

Дополнительные службы запускаются отдельно и устанавливают соединение с сервером, подобно обычным клиентам. Каждая такая служба самостоятельно подключается к основной службе MSSQLServer, используя определенные учетные записи с соответствующими правами доступа.

Для сетевого варианта установки управлять службами можно как локально, так и удаленно даже средствами операционной системы. Для операционной системы Windows 98 можно запустить только один экземпляр сервера в качестве приложения, так как в Windows 98 нет служб, и управлять этим приложением локально. Запускать, останавливать и приостанавливать сервер можно также при отсутствии сети.

Задание к практической работе 15

Задание 1. Произвести регистрацию удаленного сервера с помощью окна параметров регистрации сервера Register SQL Server Properties утилиты Enterprise Manager, выполнив действия:

1. На дереве объектов консоли выбрать одну из групп серверов, где будет зарегистрирован удаленный сервер.

2. Открыть контекстное меню группы серверов и выполнить команду New SQL Server Registration.

3. В открывшемся окне Register SQL Server Properties задать следующие параметры:

a) Имя удаленного сервера в виде следующей записи: сетевое имя NetBios соответствующего компьютера, косая черта «\», имя копии сервера (для сервера по умолчанию это имя копии можно не задавать);

b) Учетную запись, которая будет использоваться для установления соединения с соответствующим сервером: либо учетная запись домена Windows NT и ее набор прав в SQL Server, либо учетная запись сервера, созданная на регистрируемом сервера и включающая входное имя пользователя Login Name и его пароль Password, с указанием режима подключения с вводом пароля при каждом соединении или без ввода пароля;

c) Имя группы серверов из числа имеющихся или имя новой группы, которую можно создать, щелкнув по кнопке с многоточием в том же окне в области Options;

d) Установитьc2k, если необходимо, следующие переключатели: Display SQL Server state in console – показывать состояние сервера в окне объектов Enterprise Manager; Automatically start SQL Server when connecting – автоматически запускать сервер при соединении; Show system database and system table – отображать системные базы данных и таблиц.

Задание 2. Произвести регистрацию удаленного сервера с помощью мастера Register Server Wizard, выполнив следующие действия:

1. Щелкнуть на кнопке Run a Wizard панели инструментов Enterprise Manager.

2. В открывшемся окне, вид которого зависит от левого окна Enterprise Manager (выбран или не выбран сервер или папка группы серверов), выбрать мастер регистрации сервера Register Server Wizard.

3. Щелкнуть по кнопке ОК.

4. В первом окне мастера предлагается следующий порядок работы:

a) выбрать SQL сервер;

b) выбрать режим аутентификации;

c) определить группу SQL серверов.

5. В этом же окне сбросить флажок From now on, I want to perform this task without using a wizard (теперь я хотел бы выполнить задачу без использования мастера), иначе мастер закончит свою работу, открыв окно Register SQL Server Properties для ручной регистрации сервера

6. Щелкнуть по кнопке Next.

7. Во втором окне выбрать или ввести имя регистрируемого сервера в левой части окна.

8. Щелкнуть по кнопке добавить Add. В случае ошибки использовать кнопку удалить Remove. Если регистрируется сразу несколько серверов, то они будут включены в одну и ту же группу с одинаковыми параметрами и с одной и той же учетной записью для установления соединения.

9. Щелкнуть по кнопке Next.

10. В третьем окне необходимо выбрать режим аутентификации при подключении к регистрируемому серверу.

11. Если использовать аутентификацию Windows NT, то возможность подключения к серверу будет зависеть от учетной записи, под которой работает SQL Server.

12. Если выбрать аутентификацию SQL Server, то для установления соединения потребуется имя и пароль учетной записи, предварительно созданной на регистрируемом сервере SQL Server. В этом случае открывается окно, в котором надо сделать выбор режим подключения к регистрируемому серверу:

a) с использованием сохраняемой одной и той же учетной записи, для которой надо в этом же окне ввести имя и пароль;

b) с использованием учетной записи, имя и пароль который надо вводить каждый раз при подключении к удаленному серверу.

13. Щелкнуть по кнопке Next и перейти к следующему окну мастера.

14. Выбрать или создать новую группу, в которою включить регистрируемый сервер.

15. Щелкнуть по кнопке Next и перейти в последнее окно мастера со списком регистрируемых серверов.

16. Щелкнуть по кнопке Finish.

17. Если регистрируемый сервер найден, то произойдет подключение к нему.

18. Если регистрируемый сервер не найден. То Enterprise Manager выдаст запрос на повторную регистрацию сервера.

19. Возможные ошибки при регистрации:

a) регистрируемый сервер был остановлен;

b) на компьютере, с которого выполняется регистрация используется иные сетевые библиотеки и протоколы, чем на регистрируемом сервера;

c) если сервер зарегистрирован с использованием аутентификации Windows NT, а для пользователей не создано соответствующей учетной записи на SQL Server (Login failed);

d) если используется аутентификация SQL Server и имя пароль заданы неверно.

Задание 3. Произвести регистрацию удаленного сервера, выполняя команду:

EXEC sp.addserver @server = ‘server’,
@local = ‘local’,
@duplicate.ok = ‘duplicate.ok’

Здесь параметры имеют следующее назначение:

Server – имя регистрируемого сервера, которое состоит из сетевого имени NetBios

соответствующего компьютера и имени копии сервера, между которыми ставиться разделитель ”\”, при этом для копии сервера по умолчанию второе имя задавать не требуется;

Local – сервер для регистрации является локальным, иначе – сервер удаленный;

duplicate.ok – разрешает дублирование имен серверов, что приводит к тому, что информация о новом сервере будет записана поверх старой информации.

Задание 4. Произвести настройку конфигурации утилиты Enterprise Manager, выполнив действия:

- исполнить команду Tools/Options утилиты;
- в открывшемся окне SQL Server Enterprise Manager Properties выбрать вкладку General.
- в группе Server state pooling (опрос состояния сервера) выбрать службу Service и задать количество секунд, через которое будет проводиться опрос состояния соответствующей службы, и отображаться это состояние в виде соответствующего значка.
- для конфигурирования одного из серверов в качестве центрального хранилища информации необходимо снять флажок Read/Store User Independent (независимое считывание/хранение пользователей), а на локальном сервере установить переключатель Read from remote (считывать с удаленного сервера) и указать имя удаленного сервера, с которого будет считываться информация о конфигурации.
- убедиться, что установленный флажок Read/Store User Independent означает коллективное использование информации о конфигурации, а сброшенный – личное, когда информация для каждого пользователя сохраняется отдельно.

Задание 5. Установить режим автоматического запуска служб SQL Server 2000, который производится автоматически операционной системой при ее запуске, выполнив следующие действия:

1. При установке сервера MS SQL Server 2000 задать режим автоматического запуска служб сервера. В этом случае сразу же после установки и каждый раз при запуске операционной системы все установленные на компьютере службы сервера будут запускаться автоматически.

2. Если режим автоматического запуска не был задан при установке или по каким-либо причинам был отключен в дальнейшем, то его можно задать следующими действиями (три варианта):

2.1 Войти в Enterprise Manager и выполнить команды:

a) в его левом окне выбрать требуемый сервер, так как для каждого экземпляра, или копии сервера имеются отдельные экземпляры, или копии служб MSSQLServer, SQLServerAgent и MSDTC;

b) щелкнуть правой клавишей мыши, чтобы открылось контекстное меню сервера;

c) щелкнуть левой клавишей по элементу Properties (свойства);

d) в открывшемся окне SQL Server Properties (свойства SQL Server) на вкладке General (общие) установить флажок для требуемых служб:

Autostart SQL Server;

Autostart SQL Server Agent;

Autostart MSDTC;

e) щелкнуть по кнопке ОК;

f) перезагрузить операционную систему и убедиться, что нужные службы запущены.

2.2 Войти 89 в • утилиту Services (Службы) операционной системы Windows NT или Windows, исполнив команду Пуск/Настройка/Панель управления/Службы (Start/…/Control panel/Services) и выполнить команды:

a) в открывшемся окне служб Services выбрать требуемую службу;

b) дважды щелкнуть по выбранной службе;

c) в открывшемся окне свойств выбранной службы Properties на вкладке General (общие0 раскрыть список Start type (тип запуска);

d) в списке выбрать режим Automatic и щелкнуть по нему;

e) щелкнуть по кнопке ОК;

f) закрыть все окна операционной системы;

g) перезагрузить операционную систему и убедиться, что все нужные службы запущены.

2.3 Войти в утилиту SQL Server Services Manager и в открывшемся окне с таким же названием выполнить команды:

a) раскрыть список Server (сервер);

b) щелкнуть по требуемому серверу;

c) раскрыть список Services (службы) для этого сервера;

d) щелкнуть по рассматриваемой службе;

e) в открывшемся окне установить флажок Autostart service when OS start (автоматический старт при запуске операционной системы);

f) закрыть окна утилиты Services Manager;

g) перезагрузить операционную систему и убедиться, что все нужные службы запущены.

Задание 6. Произвести ручной запуск службы SQL Server 2000 одним из следующих четырёх способов:

1. Войти в Enterprise Manager и выполнить действия:

a) выбрать требуемый сервер;

b) открыть его контекстное меню;

c) щелкнуть по команде Start для запуска службы MSSQLServer;

d) для запуска службы SQLServerAgent надо открыть папу Management сервера и щелкнуть по команде Start;

e) для запуска служб MSDTC и SQLMail надо открыть папку Support Services и щелкнуть по команде Start для соответствующей службы.

2. Войти в утилиту SQL Server Service Manager, выбрать требуемый сервер и службу и щелкнуть по кнопке Start.

3. В командной строке запустить утилиту командной строки net start, указав в качестве параметра имя требуемой службы или экземпляра сервера:

net start mssqlserver

net start sqlserveragent

net start MSSQL$TRELON

net start SQLAgent$TRELON

net start для выдачи списка запущенных в ОС служб

4. Установить режим работы операционной системы сеанс DOS и в командной строке исполнить команду sqlserver для запуска сервера, как отдельного приложения операционной системы. В этом случае все средства администрирования система SQL Server такие, как Service Manager, Enterprise Manager, Service (для панели управления) будут показывать, что сервер остановлен, и все системные сообщения будут появляться в консольном окне, в котором выполнена команда sqlserver. Сервер будет запущен под

учетной записью пользователя, и если необходимо завершить сеанс работы ОС, то сначала надо завершить работу SQL Server.

Задание 7. Запустить SQL Server в однопользовательском режиме, выполнив действия:

1. Убедиться, что все службы рассматриваемого сервера остановлены.
2. В командной строке исполнить команду: sqlserver.exe –m.
3. Приступить к конфигурированию характеристик сервера или восстановлению поврежденной системной базы, учитывая, что:

a) модифицированные страницы сразу записываются на диск, а не остаются, как обычно в кэш-памяти;

b) разрешен прямой доступ к системным таблицам с помощью команд INSERT< DELETE и UPDATE.

Задание 8. Произвести аварийный запуск сервера с минимальными требованиями для проведения восстановительных работ из-за неправильного конфигурирования:

1. Для запуска SQL Server 2000 как службы с минимальными требованиями исполнить команду в командной строке: sqlserver.exe –f.
2. Для запуска SQL Server 2000 как приложения с минимальными требованиями исполнить команду в командной строке: sqlserver.exe –f –c.
3. Для первого случая убедиться, что:

a) количество открытых баз данных, таблиц, открытых объектов, размер КЭШа процедур минимальны;

b) запрещено исполнение хранимых процедур;

c) установлен однопользовательский режим;

d) удаленный доступ запрещен;

e) разрешен прямой доступ к таблицам.

Задание 9. Приостановите, а затем и остановите работу служб сервера. Запустите их вновь.

Просмотрите параметры запуска в реестре по адресу

HKEY_LOCAL_MASHINE\SOFTWARE\MICROSOFT\MSSQLSERVER \PARAMETERS.

ПРАКТИЧЕСКАЯ РАБОТА №16. АДМИНИСТРИРОВАНИЕ СЕРВЕРА БАЗ ДАННЫХ MS SQL SERVER (ПРОДОЛЖЕНИЕ)

16.1 Цель практической работы

Ознакомление с основными концепциями и технологиями, лежащими в основе функционирования сервера, и реализующими и их компонентами: средствами администрирования, сетевыми библиотеками, службами, интерфейсами для создания клиентских приложений

16.2 Методические рекомендации для выполнения практической работы

SQL Server имеет множество инструментов для импорта и экспорта данных. Лучшим является служба преобразования данных Data Transformation Services (DTS), которая предоставляет набор инструментальных средств. Она также позволяет извлекать, преобразовывать и объединять данные из источников данных разной природы, расположенных как в одном, так и в разных местах. Можно управлять данными, используя инструментальные средства DTS, для графического построения пакетов DTS или создавая объектно-ориентированные пакеты DTS.

Пакет DTS – это объект, в котором хранится описание выполняемых в ходе импорта, экспорта и трансформации данных действий. Каждый пакет реализует один или большее количество шагов, которые выполняются последовательно или параллельно. С помощью пакета может выполняться копирование и преобразование данных и объектов баз данных. Пакеты можно редактировать, защищать паролем, конфигурировать для автоматического выполнения по расписанию.

Задание к практической работе №16

Задание1. Осуществить передачу данных с помощью мастера Data Transformation Services(DTS), используя способ Copy table(s) and view(s) from the source database(копировать таблицу(таблицы) и представление(представления) из источника), выполнив следующие действия:

1. Запустить мастер: Пуск \ Программы \ Microsoft SQL Server \ Import and Export Data.
2. В первом открывшемся окне, которое содержит общую информацию о работе мастера, щёлкнуть по кнопке Next.
3. Во втором окне в раскрывающемся списке Source(источник) необходимо выбрать тип источника данных; в списке Server(сервер) выбрать сервера-источника; указать список аутентификации; в списке Database выбрать базу данных, в которую будет осуществляться взаимодействие. После этого щёлкнуть по кнопке Next.
4. Для редактирования, по необходимости, параметров конфигурации щёлкнуть на кнопке Advanced(дополнительно).
5. В третьем окне сконфигурировать получатель: в раскрывающемся списке Database(база данных) выбрать пункт New(создать) и создать новую базу данных.
6. В четвёртом окне DTS Wizard выбрать способ передачи данных Copy table(s) and view(s) from the source database.
7. В пятом окне в столбце Source Table(таблица источник) выбрать одну или более таблиц или представлений для копирования.

8. Для того, чтобы увидеть содержание исходной таблицы, щёлкните на кнопке Preview(просмотр).

9. В столбце Destination(получатель) указать имя таблицы-получателя.

10. Если необходимо выполнить преобразование данных, то в столбце Transform(преобразование) для соответствующей таблицы щёлкните на кнопке с многоточием. В открывшемся окне можно настроить процесс трансформации не только самих данных, но и их типов.

11. Следующее окно мастера DTS Wizard (рис. 24.22) будет общим для всех способов переноса. В этом окне для созданного пакета DTS указать способ его сохранения.

12. Если выбрали вариант SQL Server, то необходимо установить параметры:

- в поле Name(имя) указывается имя, под которым пакет DTS будет сохранен в системной базе данных msdb;
- в поле Description (описание) можно ввести описание объекта в произвольной форме;
- в поле Owner Password (пароль владельца), чтобы скрыть информацию, указанную при создании пакета, от просмотра неавторизированными пользователями, можно установить пароль владельца;
- установив в поле User Password (пароль пользователя) пароль пользователя, можно запретить выполнение пакета пользователями, которые не имеют на это права. Только те пользователи, которые знают пароль, смогут выполнить пакет DTS.
- в списке Server name (имя сервера) выбирается имя сервера, на котором будет сохранен пакет DTS.

13. При выборе режима хранения SQL Server Meta Data Services (службы метаданных SQL Server) мастер выведет окно, во многом напоминающее окно при режиме хранения SQL Server. Добавлена лишь кнопка Scanning (сканирование), с помощью которой можно установить взаимосвязи между объектами в источнике и получателе данных, сохраняемые в хранилище(первичный и внешний ключи, индексы, столбцы, типы данных и т. д.).

14. При выборе двух оставшихся режимов хранения мастер выведет окно, в котором помимо имени, описания, пароля пользователя и пароля владельца необходимо указать только имя файла, в который будет записан пакет.

На этом работа с мастером DTS Wizard по созданию пакетов для импорта экспорта данных заканчивается. В последнем окне (рис. 24.28) приведена сводная информация о созданном пакете. После щелчка на кнопке Finish(готово) будет создан сам пакет.

15. Если при создании пакета было задано его незамедлительное выполнение, то мастер откроет окно Executing Package (выполнение пакета), позволяющее следить за процессом выполнения пакета.

Задание 2. Осуществить передачу данных с помощью мастера Data Transformation Services(DTS), используя способ Use a query to specify the data to transfer(использовать запрос для выборки данных), выполнив следующие действия:

1. Выполнить с первого по пятый пункты задания1.

2. В четвёртом окне мастера DTS Wizard установить переключатель Use a query to specify the data to transfer.

3. В открывшемся окне ввести SQL-код запроса; если имеется готовый код, сохранённый на диске, его можно подключить, воспользовавшись кнопкой Browse(обзор).

4. Если необходимо написать сложный запрос с перечислением множества таблиц и столбцов и при этом гарантировать, что указаны правильные имена объектов, можно

воспользоваться встроенным в мастер конструктором запросов. Для вызова конструктора запроса щелкнуть на кнопке Query Builder (конструктор запросов).

Откроется окно, в котором нужно выбрать, какие столбцы, из каких таблиц будут включены в запрос.

5. Щёлкнуть по кнопке Next.

6. В открывшемся окне, перенося имена столбцов из левой части окна в правую, задать порядок сортировки, которая ведётся по столбцам, указанным в самом верху списка.

7. Щёлкнуть по кнопке Next.

8. В следующем окне указать критерии для выборки данных: установить указатель Only Rows meeting criteria(только строки, соответствующие критерию).

9. Если необходимости в фильтрации нет, установите переключатель All rows(все строки). Нажмите на кнопку OK.

10. После того как редактирование запроса закончено, мастер откроет окно, в котором можно настроить трансформацию данных. Работа с этим окном практически ничем не отличается от работы по настройке трансформации данных при копировании между таблицами, описанной в задании1.

11. После настройки трансформации данных необходимо сохранить пакет DTS одним из способов, которые указаны в предыдущем задании.

Задание 3. Осуществить передачу данных с помощью мастера Data Transformation Services(DTS), используя способ Copy objects and data between SQL Server databases(копировать объекты и данные между базами данных SQL Server), выполнив следующие действия:

1. Выполнить с первого по пятый пункты задания1.

2. В четвёртом окне мастера DTS Wizard установить переключатель Copy objects and data between SQL Server databases.

3. Щёлкнуть на кнопке Next.

4. В открывшемся окне указать, какие объекты и данные будут копироваться: установка флажка Create destination objects – создание переносимых объектов; установка флажка Drop destination objects first – удаление всех одноимённых объектов из конечной базы данных; установка флажка Include all dependent objects – включение всех зависимых объектов; установка флажка Copy data – копирование только структуры объектов.

5. Установив флажок Copy all objects, выполняется копирование всех объектов.

6. Если необходимо скопировать только часть объектов, сбросьте флажок Copy all objects и выберите нужные объекты, щёлкнув на кнопке Select Objects(выбор объектов).

7. Чтобы выбрать только некоторые их них, в окне мастера сбросьте флажок Use default options(использовать параметры по умолчанию). После щелчка на кнопке Options(параметры) в открывшемся окне укажите объекты, которые необходимо скопировать.

8. После указания объектов необходимо сохранить пакет DTS одним из способов, которые указаны в первом задании. После будет создан сам пакет.

Контрольные вопросы:

1. На каких критериях следует основываться при выборе метода импорта или экспорта данных.

2. Что собой представляет служба преобразования данных Data Transformation Serviced(DTS).

3. Какие способы передачи данных можно выделить, используя мастер Data Transformation Serviced(DTS) Import and Export Wizard.

4. Какими способами можно осуществить хранение пакета DTS.

5. Что собой представляет внутренняя структура пакета DTS.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. **Кузин, А.В**. Базы данных : учеб. пособие для вузов/ А. В. Кузин, С. В. Левонисова. -5-е изд., испр.. -М.: Академия, 2012. - 315 с.
2. **Евдокимов, А. В**. Системы управления реляционными базами данных [Текст] : учебное пособие / А. В. Евдокимов. - [Б. м. : б. и.], 2013. – 276 с.
3. Базы данных [электронный университет]: учебно-методическое пособие по выполнению практических работ / О.В. Игнатьева; ФГБОУ ВО РГУПС. – Ростов н/Д, 2016. – 255 с.
4. Базы данных: учебное пособие / Медведкова И.Е., Бугаев Ю.В., Чикунов С.В. ВГУИТ 2014 . - 105 с. ЭБС "КнигаФонд"
5. Проектирование информационных систем в Microsoft SQL Server 2008 и Visual Studio 2008. Бурков А.В. ИНТУИТ 2010 - 457 с. ЭБС "КнигаФонд"
6. **Казак, А.А.** Разработка реляционных баз данных: учеб.-метод. пособие; РГУПС. – 2011. 40с.
7. **Швецов, В.И.** Базы данных. - Интернет-Университет информационных технологий, 2009. ЭБС "КнигаФонд".
8. **Жилякова, Л.Ю**. Введение в проектирование реляционных баз данных : учеб. пособие [для студентов эконом. специальностей по дисциплине "Базы и банки данных"]/ Л.Ю. Жилякова; РГУПС. -Ростов н/Д, 2007. -99 с.:а-ил.
9. **Ильичева, В.В.** Лабораторный практикум по курсу "Базы данных" : практикум для студентов изучающих дисциплину "Базы и банки данных"/ В.В. Ильичева; РГУПС. -Ростов н/Д, 2008. -59 с.:а-ил. 100 экз.

Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

НЕВИННОМЫССКИЙ ТЕХНОЛОГИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)

Методические указания к самостоятельным работам

студентов направления

15.03.04 Автоматизация технологических процессов и производств

по дисциплине

«Управление данными»

Невинномысск 2026

Методические указания предназначены для студентов направления 15.03.04 Автоматизация технологических процессов и производств и других технических специальностей. Они содержат рекомендации по организации самостоятельных работ студента на правления 15.03.04 Автоматизация технологических процессов и производств для дисциплины «Управление данными».

Методические указания разработаны в соответствии с требованиями ФГОС ВО в части содержания и уровня подготовки выпускников направления 15.03.04 Автоматизация технологических процессов и производств.

Составитель канд. техн. наук Кочеров Ю. Н.

Ответственный редактор канд. техн. наук Д.В. Болдырев

Содержание

1 Подготовка к лекциям 4
2 Подготовка к практическим занятиям 6
3 Самостоятельное изучение темы. Конспект 8

1 Подготовка к лекциям

Главное в период подготовки к лекционным занятиям – научиться методам самостоятельного умственного труда, сознательно развивать свои творческие способности и овладевать навыками творческой работы. Для этого необходимо строго соблюдать дисциплину учебы и поведения. Четкое планирование своего рабочего времени и отдыха является необходимым условием для успешной самостоятельной работы. В основу его нужно положить рабочие программы изучаемых в семестре дисциплин.

Каждому студенту следует составлять еженедельный и семестровый планы работы, а также план на каждый рабочий день. С вечера всегда надо распределять работу на завтрашний день. В конце каждого дня целесообразно подводить итог работы: тщательно проверить, все ли выполнено по намеченному плану, не было ли каких-либо отступлений, а если были, по какой причине это произошло. Нужно осуществлять самоконтроль, который является необходимым условием успешной учебы. Если что-то осталось невыполненным, необходимо изыскать время для завершения этой части работы, не уменьшая объема недельного плана.

Слушание и запись лекций – сложный вид вузовской аудиторной работы. Внимательное слушание и конспектирование лекций предполагает интенсивную умственную деятельность студента. Краткие записи лекций, их конспектирование помогает усвоить учебный материал. Конспект является полезным тогда, когда записано самое существенное, основное и сделано это самим студентом. Не надо стремиться записать дословно всю лекцию. Такое «конспектирование» приносит больше вреда, чем пользы. Запись лекций рекомендуется вести по возможности собственными формулировками. Желательно запись осуществлять на одной странице, а следующую оставлять для проработки учебного материала самостоятельно в домашних условиях.

Конспект лекций лучше подразделять на пункты, параграфы, соблюдая красную строку. Этому в большой степени будут способствовать пункты плана лекции, предложенные преподавателям. Принципиальные места, определения, формулы и другое следует сопровождать замечаниями «важно», «особо важно», «хорошо запомнить» и т.п. Можно делать это и с помощью разноцветных маркеров или ручек.

Лучше если они будут собственными, чтобы не приходилось присить их у однокурсников и тем самым не отвлекать их во время лекции. Целесообразно разработать собственную «маркографию» (значки, символы), сокращения слов. Не лишним будет и изучение основ стенографии. Работая над конспектом лекций, всегда необходимо использовать не только учебник, но и ту литературу, которую дополнительно рекомендовал лектор. Именно такая серьезная, кропотливая работа с лекционным материалом позволит глубоко овладеть знаниями.

2 Подготовка к практическим занятиям

Подготовку к каждому практическому занятию студент должен начать с ознакомления с методическими указаниями, которые включают содержание работы. Тщательное продумывание и изучение вопросов основывается на проработке текущего материала лекции, а затем изучения обязательной и дополнительной литературы, рекомендованную к данной теме. На основе индивидуальных предпочтений студенту необходимо самостоятельно выбрать тему доклада по проблеме и по возможности подготовить по нему презентацию.

Если программой дисциплины предусмотрено выполнение практического задания, то его необходимо выполнить с учетом предложенной инструкции (устно или письменно). Все новые понятия по изучаемой теме необходимо выучить наизусть и внести в глоссарий, который целесообразно вести с самого начала изучения курса. Результат такой работы должен проявиться в способности студента свободно ответить на теоретические вопросы семинара, его выступлении и участии в коллективном обсуждении вопросов изучаемой темы, правильном выполнении практических заданий и контрольных работ.

В зависимости от содержания и количества отведенного времени на изучение каждой темы практическое занятие может состоять из четырех-пяти частей:

1. Обсуждение теоретических вопросов, определенных программой дисциплины.

2. Доклад и/ или выступление с презентациями по выбранной проблеме.

3. Обсуждение выступлений по теме – дискуссия.

4. Выполнение практического задания с последующим разбором полученных результатов или обсуждение практического задания.

5. Подведение итогов занятия.

Первая часть – обсуждение теоретических вопросов – проводится в виде фронтальной беседы со всей группой и включает выборочную проверку преподавателем теоретических знаний студентов. Примерная продолжительность — до 15 минут. Вторая часть — выступление студентов с докладами, которые должны сопровождаться презентациями с целью усиления наглядности восприятия, по одному из вопросов практического занятия. Обязательный элемент доклада – представление и

анализ статистических данных, обоснование социальных последствий любого экономического факта, явления или процесса. Примерная продолжительность — 20-25 минут. После докладов следует их обсуждение – дискуссия. В ходе этого этапа практического занятия могут быть заданы уточняющие вопросы к докладчикам. Примерная продолжительность – до 15-20 минут. Если программой предусмотрено выполнение практического задания в рамках конкретной темы, то преподавателями определяется его содержание и дается время на его выполнение, а замет идет обсуждение результатов. Подведением итогов заканчивается практическое занятие.

В процессе подготовки к практическим занятиям, студентам необходимо обратить особое внимание на самостоятельное изучение рекомендованной учебно-методической (а также научной и популярной) литературы. Самостоятельная работа с учебниками, учебными пособиями, научной, справочной и популярной литературой, материалами периодических изданий и Интернета, статистическими данными является наиболее эффективным методом получения знаний, позволяет значительно активизировать процесс овладения информацией, способствует более глубокому усвоению изучаемого материала, формирует у студентов свое отношение к конкретной проблеме. Более глубокому раскрытию вопросов способствует знакомство с дополнительной литературой, рекомендованной преподавателем по каждой теме семинарского или практического занятия, что позволяет студентам проявить свою индивидуальность в рамках выступления на данных занятиях, выявить широкий спектр мнений по изучаемой проблеме.

3 Самостоятельное изучение темы. Конспект

Конспект – наиболее совершенная и наиболее сложная форма записи. Слово «конспект» происходит от латинского «conspectus», что означает «обзор, изложение». В правильно составленном конспекте обычно выделено самое основное в изучаемом тексте, сосредоточено внимание на наиболее существенном, в кратких и четких формулировках обобщены важные теоретические положения.

Конспект представляет собой относительно подробное, последовательное изложение содержания прочитанного. На первых порах целесообразно в записях ближе держаться тексту, прибегая зачастую к прямому цитированию автора. В дальнейшем, по мере выработки навыков конспектирования, записи будут носить более свободный и сжатый характер.

Конспект книги обычно ведется в тетради. В самом начале конспекта указывается фамилия автора, полное название произведения, издательство, год и место издания. При цитировании обязательная ссылка на страницу книги. Если цитата взята из собрания сочинений, то необходимо указать соответствующий том. Следует помнить, что четкая ссылка на источник – непременное правило конспектирования. Если конспектируется статья, то указывается, где и когда она была напечатана.

Конспект подразделяется на части в соответсвтии с заранее продуманным планом. Пункты плана записываются в тексте или на полях конспекта. Писать его рекомендуется четко и разборчиво, так как небрежная запись с течением времени становиться малопонятной для ее автора. Существует правило: конспект, составленный для себя, должен быть по возможности написан так, чтобы его легко прочитал и кто-либо другой.

Формы конспекта могут быть разными и зависят от его целевого назначения (изучение материала в целом или под определенным углом зрения, подготовка к докладу, выступлению на занятии и т.д.), а также от характера произведения (монография, статья, документ и т.п.). Если речь идет просто об изложении содержания работы, текст конспекта может быть сплошным, с выделением особо важных положений подчеркиванием или различными значками.

В случае, когда не ограничиваются переложением содержания, а фиксируют в конспекте и свои собственные суждения по данному вопросу или дополняют кон-

спект соответствующими материалами их других источников, следует отводить место для такого рода записей. Рекомендуется разделить страницы тетради пополам по вертикали и в левой части вести конспект произведения, а в правой свои дополнительные записи, совмещая их по содержанию.

Конспектирование в большей мере, чем другие виды записей, помогает вырабатывать навыки правильного изложения в письменной форме важные теоретических и практических вопросов, умение четко их формулировать и ясно излагать своими словами.

Таким образом, составление конспекта требует вдумчивой работы, затраты времени и труда. Зато во время конспектирования приобретаются знания, создается фонд записей.

Конспект может быть текстуальным или тематическим. В текстуальном конспекте сохраняется логика и структура изучаемого произведения, а запись ведется в соответствии с расположением материала в книге. За основу тематического конспекта берется не план произведения, а содержание какой-либо темы или проблемы.

Текстуальный конспект желательно начинать после того, как вся книга прочитана и продумана, но это, к сожалению, не всегда возможно. В первую очередь необходимо составить план произведения письменно или мысленно, поскольку в соответствии с этим планом строится дальнейшая работа. Конспект включает в себя тезисы, которые составляют его основу. Но, в отличие от тезисов, конспект содержит краткую запись не только выводов, но и доказательств, вплоть до фактического материала. Иначе говоря, конспект – это расширенные тезисы, дополненные рассуждениями и доказательствами, мыслями и соображениями составителя записи.

Как правило, конспект включает в себя и выписки, но в него могут войти отдельные места, цитируемые дословно, а также факты, примеры, цифры, таблицы и схемы, взятые из книги. Следует помнить, что работа над конспектом только тогда будет творческой, когда она не ограничена текстом изучаемого произведения. Нужно дополнять конспект данными из другими источников.

В конспекте необходимо выделять отдельные места текста в зависимости от их значимости. Можно пользоваться различными способами: подчеркиваниями, вопросительными и восклицательными знаками, репликами, краткими оценками, пи-

сать на полях своих конспектов слова: «важно», «очень важно», «верно», «характерно».

В конспект могут помещаться диаграммы, схемы, таблицы, которые придадут ему наглядность.

Составлению тематического конспекта предшествует тщательное изучение всей литературы, подобранной для раскрытия данной темы. Бывает, что какая-либо тема рассматривается в нескольких главах или в разных местах книги. А в конспекте весь материал, относящийся к теме, будет сосредоточен в одном месте. В плане конспекта рекомендуется делать пометки, к каким источникам (вплоть до страницы) придется обратиться для раскрытия вопросов. Тематический конспект составляется обычно для того, чтобы глубже изучить определенный вопрос, подготовиться к докладу, лекции или выступлению на семинарском занятии. Такой конспект по содержанию приближается к реферату, докладу по избранной теме, особенно если включает и собственный вклад в изучение проблемы.