

top

КОМПЬЮТЕРНАЯ
АКАДЕМИЯ

Теория Баз Данных

ЧАСТЬ 1.

Введение в теорию баз данных

Содержание

Unit 1. Введение в теорию баз данных	5
1. Введение в теорию баз данных.....	6
История и этапы развития	6
Понятия база данных и система управления базами данных.....	7
Сравнение существующих моделей баз данных	8
Понятие реляционной модели баз данных	12
Двенадцать правил Кодда.....	14
Сравнительный анализ СУБД Microsoft SQL Server с существующими системами управления базами данных.....	16
2. Основы взаимодействия с Microsoft SQL Server....	18
Версии и редакции Microsoft SQL Server	18
Инсталляция Microsoft SQL Server.....	18
Инструменты управления и утилиты MS SQL Server	32

Управление базой данных	36
Изменение размера базы данных	42
Домашнее задание	48
Unit 2. Таблицы, типы данных, запросы, основы взаимодействия с MS SQL Server	49
1. Таблицы.....	50
Первичный ключ	52
Значение по умолчанию.....	56
Уникальность.....	57
2. Типы данных	64
Целочисленные типы данных.....	64
Типы данных для хранения текста.....	64
Вещественные типы данных.....	66
Типы данных для хранения даты и времени	67
Типы данных с фиксированной точкой	68
Другие типы данных	68
3. Индекс.....	70
Что такое индекс?	70
Цели и задачи индекса.....	70
Внутреннее устройство индекса.....	71
4. Системные базы данных и таблицы.....	74
5. Запросы	76
Введение в язык структурированных запросов SQL	76
Язык SQL.....	76
Стандарты языка SQL.....	77

Диалекты языка SQL.....	78
Диалект Transact-SQL	79
Понятия DDL, DML, DCL.....	79
Домашнее задание	80
Unit 3. Запросы SELECT, INSERT, UPDATE, DELETE	81
1. Оператор SELECT	82
Предложение SELECT.....	82
Предложение FROM	82
Предложение WHERE	89
Предложение ORDER BY	94
2. Ключевые слова IN, BETWEEN, LIKE	96
3. Оператор INSERT	101
4. Оператор UPDATE.....	109
5. Оператор DELETE	111
6. Понятие транзакции. Использование транзакций	112
7. Домашнее задание	116

Unit 1.
Введение
в теорию
баз данных

1. Введение в теорию баз данных

Базы данных используются для хранения и обработки различной информации уже не один десяток лет и будут актуальны еще длительное время. Наверное, вам известна знаменитая фраза Наташа Ротшильда: «Кто владеет информацией, тот владеет миром», а где же сейчас хранится эта информация? Правильно, в базах данных, поэтому специалисты в этой области будут востребованы всегда.

История и этапы развития

Исследования по созданию баз данных стали возможны благодаря появлению программируемого оборудования обработки записей и начались в конце 50-х годов прошлого столетия. Основной идеей этих исследований была автоматизация офисной работы, связанная с хранением и учетом данных, которая выполнялась вручную и требовавшая больших затрат труда. Благодаря удешевлению вычислительных мощностей для решения этих задач стало возможным использование компьютеров.

Лидером в сфере исследований баз данных была компания IBM, которая в 1968 году выпустила свою первую систему управления базами данных IMS (IBM *Information Management System*), основанную на иерархической модели данных. В середине 70-х годов прошлого столетия компания IBM создала первую реляционную систему управления базами данных (РСУБД) — IBM

System R, в которой впервые был реализован язык запросов для реляционных баз данных — *Structured Query Language* (SQL).

В 1979 году компания Oracle выпустила первую коммерческую РСУБД Oracle v2, вслед за этим различные компании выпустили свои коммерческие продукты. В 1982 году появилась РСУБД DB2 от IBM, в 1988 году компании Microsoft и Ashton-Tate выпустили совместный продукт Ashton-Tate/Microsoft SQL Server 1.0, в 1992 году появилась первая версия РСУБД Access от компании Microsoft.

Понятия база данных и система управления базами данных

Вы, наверное, встречали в повседневной жизни сочетание слов «база данных», но это довольно широкое понятие, смысл которого сводится к организации хранения данных. Однако различаются способы хранения, например это может быть набор файлов.

В нашем случае, **база данных** (БД) — это коллекции связанных данных, сгруппированных в единый объект. Допустим, при создании базы данных какого-то учебного заведения, мы размещаем в ней необходимые наборы данных (информация о студентах, преподавателях, факультетах и т.д.), и впоследствии управляем полученной БД при помощи программного обеспечения как единым целым.

Специализированное программное обеспечение, позволяющее работать с базами данных (обновлять, извлекать данные и т.д.), имеет общее название система

управления базами данных (СУБД), например Microsoft SQL Server, Access, Oracle Database и т.д.

СУБД состоит из ряда серверных и клиентских средств, которые позволяют выполнять администрирование баз данных и различные действия, связанные с манипуляцией данными. Практически любая СУБД позволяет обрабатывать запросы на извлечение и изменение данных, предоставляет механизмы для резервного копирования и восстановления данных, оптимизирует производительность выполнения запросов, управляет памятью.

Сравнение существующих моделей баз данных

Модели баз данных различаются правилами взаимосвязи основных типов структур данных и операциями над ними. Каждая СУБД использует логическую структуру хранения данных, которая зависит от конкретной модели хранения данных, рассмотрим далее наиболее часто используемые модели баз данных.

Файловая модель

Файловая модель данных характеризуется определенным набором файлов, не связанных между собой, основными типами структур данных которых являются поле, запись, файл. Поле — это элементарная, неделимая единица данных. Запись — это совокупность логически связанных полей. Файл — это множество записей, одинаковых по структуре.

Файловая модель была первой моделью хранения данных, ее можно считать моделью без СУБД, ведь вну-

тренняя структура файлов была известна только разработчику данного программного обеспечения, то есть была уникальна.

Файловая модель обладает рядом недостатков, перечислим основные из них:

- алгоритм управления базой данных полностью заложен в программном обеспечении, при изменении структуры данных необходимо вносить правки в программное обеспечение;
- сложности совместимости форматов файлов, созданных при помощи различных языков программирования, из-за отличающихся друг от друга структур;
- трудности при переносе данных из одной БД в другую, по причине несовпадения структур данных;
- отсутствие централизованного хранения приводит к необходимости дублирования одних и тех же данных.

Иерархическая модель

Иерархическая модель представляет собой простую структуру, в которой отдельные записи организуются в отношения типа «родитель-потомок» и образовывают обращенное дерево. Данные в этой иерархической структуре делятся на логические категории и подкатегории, использующие записи для представления логических единиц данных.

Иерархическая модель довольно удобна для представления отношений между сущностями реального мира, однако совершенно не подходит для многих современных приложений. Иерархическая структура не

поддерживает сложные отношения между записями типа «многие ко многим», потому что дочерняя запись может ассоциироваться только с одной родительской записью. Еще одним недостатком является довольно громоздкий процесс навигации, ведь для того чтобы получить доступ к записям необходимо перемещаться вверх и вниз по уровням иерархии.

Наиболее известная СУБД, созданная на основе этой модели — это IMS от компании IBM. Иерархическая модель нашла свое применение не только в СУБД, такая модель применяется для систем управления файлами в операционных системах.

Сетевая модель

Говоря о сетевой модели баз данных нельзя не упомянуть о Чарльзе Бахмане (*Charles Bachman*), который являлся один из основоположников сетевой модели баз данных. Именно он в 1963 году разработал одну из первых сетевых систем управления базами данных IDS (*Integrated Data Store*). За свои разработки, посвященные технологиям баз данных, в 1973 году он был удостоен премии Тьюринга.

Сетевая модель появилась как результат усовершенствования иерархической модели, и в отличие от последней позволяет записям принимать участие во множестве отношений типа «родитель-потомок».

Сетевая модель была стандартизована в 1975 году организацией CODASYL (*Conference of Data System Languages*), которая определила базовые понятия этой модели и язык описания.

Хотя сетевая модель является более гибкой по сравнению с иерархической, программирование навигации по записям должны осуществлять разработчики. Реализация этой модели подразумевает использование значительных ресурсов памяти, потому что каждый элемент должен хранить ссылки на другие элементы. Кроме того, внесение любых изменений может привести к сложным операциям обновления базы данных.

Реляционная модель

Еще одним выдающимся ученым был Эдгар Кодд (*Edgar Codd*), будучи сотрудником кампании IBM, разработал реляционную модель данных, которую в 1970 году описал в статье «A Relational Model of Data for Large Shared Data Banks». В 1981 году за свои исследования в области баз данных он был награжден премией Тьюринга.

Реляционная модели данных не использует связь между родительскими и дочерними записями, а основана на взаимодействии строк и столбцов, которые образуют таблицы с данными, связанными между собой. Внутри базы данных, таблицы имеют уникальные названия и связываются между собой отношениями, которые позволяют осуществлять навигацию по записям. Такая структура данных является очень гибкой и удобной при извлечении и изменении информации.

В качестве примера РСУБД можно назвать: Microsoft SQL Server, Access, MySQL.

Объектно-ориентированная модель

Объектно-ориентированная модель данных имеет древовидную структуру, узлами которой являются объ-

екты. В этой модели технология объектно-ориентированного программирования (ООП) применяется к технологии баз данных. Каждая запись в базе данных является объектом, связи между записями осуществляются с помощью механизмов, которые используются в ООП. Поиск необходимой записи заключается в сравнении пользовательского объекта с объектами, которые хранятся в БД.

Базовые понятия объектно-ориентированной модели полностью повторяют основные понятия ООП: класс, объект, метод, инкапсуляция, наследование, полиморфизм, агрегация.

Основные недостатки модели объектно-ориентированной модели: сложность понимания ее структуры и относительно низкая скорость выполнения запросов. К достоинствам этой модели можно отнести возможность хранения и отображения информации о сложных объектах, с возможностью определения методов для работы с ними. Благодаря этим достоинствам некоторые РСУБД дополняются элементами объектно-ориентированного проектирования, например Oracle Database, которая является объектно-реляционной СУБД.

Понятие реляционной модели баз данных

Начиная с конца 70-х годов прошлого столетия наиболее популярной моделью баз данных, является реляционная модель, в основе которой лежат математические принципы обработки данных.

В реляционной модели данные представлены в виде таблиц, которые состоят из полей. Например, на рисун-

На рисунке 1.1 показаны две таблицы, в первой из них хранится информация о студентах, а во второй о группах, в которых проходят обучение студенты.

Students				Groups		
Id	FirstName	LastName	GroupId	Id	Name	FacultyId
1	Jack	Jones	2	1	29PR31	1
2	Harry	Miller	1	2	29GR32	2
3	Grace	Evans	3	3	29PPS12	1
4	Lily	Wilson	3	4	29GPS21	2
5	Joshua	Johnson	1			
6	Emily	Taylor	4			
7	Charlie	Thomas	4			
8	Oliver	Moore	2			
9	Jessica	Brown	1			

Рисунок 1.1. Пример структуры реляционной базы данных

Каждое поле предназначено для хранения определенной информации, которая характеризует конкретную сущность. В нашем случае каждая запись в таблице **Students** представляет конкретного студента, в которой указывается уникальный идентификатор студента, имя, фамилия и идентификатор той группы, где он обучается. В таблице **Groups** записи содержат уникальную информацию по каждой группе, и состоят из идентификатора группы, имени группы и идентификатора факультета.

В литературе, описывающей реляционную модель данных, вы можете встретить несколько другие определения: так таблицы называются отношениями (англ. *relation*), в общем, отсюда и пошло название самой модели — реляционная, записи в таблице называются кортежами, а поля — атрибутами. Однако какие бы вы названия

не использовали, суть реляционной модели останется прежней.

Как показано на рисунке 1.1 связь между таблицами обеспечивается благодаря наличию в одной таблице уникального идентификатора из другой таблицы, мы сейчас не будем останавливаться на этом подробно, более полную информацию вы получите в последующих уроках.

Двенадцать правил Кодда

В процессе разработки реляционной модели баз данных, Эдгар Кодд сформулировал требования, которым должна соответствовать любая реляционная СУБД, он опубликовал их в 1985 году. На самом деле основное правило имеет номер 0, поэтому всего насчитывается 13 правил. Перечислим эти правила:

1. Основное правило: СУБД должна управлять базами данных, используя исключительно свои реляционные возможности;
2. Правило информации: вся информация в любой базе данных должна быть представлена исключительно значениями в таблицах;
3. Правило гарантированного доступа: каждое значение в любой таблице можно получить с помощью комбинации имени таблицы, значения первичного ключа и имени столбца (имя таблицы позволяет найти требуемую таблицу, имя столбца позволяет найти требуемый столбец, а первичный ключ позволяет найти строку, содержащую искомый элемент данных);
4. Правило поддержки недействительных значений: в СУБД должна быть реализована возможность под-

- держки неизвестных или отсутствующих значений (отсутствующие данные должны быть представлены с помощью недействительных значений (**NULL**));
5. Правило динамического каталога: информация о базах данных должна храниться в виде таблиц, и СУБД должна обеспечивать стандартный доступ к ней и пользовательским данным при помощи одних и тех же средств (СУБД должна содержать набор системных таблиц, описывающих структуру баз данных);
 6. Правило исчерпывающего подъязыка данных: СУБД должна поддерживать хотя бы один язык, операторы которого обеспечивают все ее основные функции (создание базы данных, манипулирование данными, управление доступом и т.д.);
 7. Правило обновления представлений: каждое представление должно поддерживать те же операции с данными, что и таблицы (чтение, вставка, изменение и удаление данных);
 8. Правило добавления, обновления и удаления: операции связанные с изменением и удалением данных должны одинаково работать как с одной записью в таблице, так и с множеством записей;
 9. Правило независимости физических данных: приложения, использующие любую базу данных, не должны зависеть от способа хранения информации и аппаратного обеспечения компьютеров;
 10. Правило независимости логических данных: приложения не должны зависеть от структуры базы данных, любые изменения в структуре БД не должны влиять на работу приложения;

11. Правило независимости условий целостности: язык СУБД должен поддерживать проверку входных данных и обеспечивать их целостность;
12. Правило независимости распространения: база данных может находиться на разных компьютерах и язык СУБД должен поддерживать возможность работы с распределенными данными;
13. Правило единственности: не должно быть возможности нарушить безопасность и целостность данных в обход языка СУБД, то есть при работе с данными должен использоваться только язык СУБД.

Сравнительный анализ СУБД Microsoft SQL Server с существующими системами управления базами данных

На сегодняшний день существует большое количество систем управления базами данных, некоторые из них уже упоминались нами в текущем уроке. В качестве сравниваемых СУБД мы возьмем MS SQL Server, Oracle, PostgreSQL и MySQL, у каждой из них есть как свои преимущества, так и недостатки. Сейчас вы, наверное, скажете: «Всяк кулик своё болото хвалит», однако мы все же попытаемся быть максимально объективными.

Если рассматривать Oracle и MS SQL Server, то по производственным характеристикам они приблизительно одинаковы, может быть по ряду показателей Oracle лучше. Главное же отличие Oracle от MS SQL Server — это поддержка большого количества программно-аппаратных платформ (Linux, Windows, Mac OS и т.д.). В качестве отрицательных сторон Oracle можно назвать: высокую стоимость, высокие требования к аппаратному обеспечению.

нию и сложность администрирования. К тому же Oracle в основном предназначен для использования в больших промышленных проектах, и в относительно простых приложениях его использование будет не рационально.

MySQL — это РСУБД, предназначенная для использования в простых и средних приложениях и основное ее преимущество в том, что она бесплатная. К недостаткам можно отнести: ограниченную функциональность и некоторые проблемы с надежностью.

PostgreSQL — это свободно распространяемая объектно-реляционная СУБД, максимально соответствующая стандартам SQL, поддерживаемая ОС Windows и множеством UNIX-подобных платформ. И хотя PostgreSQL является довольно качественным программным продуктом, у нее тоже есть недостатки: производительность, сложность настройки и небольшая популярность (малое количество хостингов с поддержкой этой СУБД).

Из всего вышеперечисленного можно сделать вывод, что MS SQL Server является некой золотой серединой среди СУБД, хотя, конечно же, у нее есть и свои недостатки.

При изучении текущего предмета мы будем использовать СУБД Microsoft SQL Server, поэтому далее остановимся на ней более подробно.

2. Основы взаимодействия с Microsoft SQL Server

Версии и редакции Microsoft SQL Server

Как мы уже говорили ранее, первая версия MS SQL Server (SQL Server 1.0) появилась в 1988 году. Следующая версия — SQL Server 1.1 — вышла в 1990 году. За прошедшие двадцать с лишним лет вышло 15 версий MS SQL Server.

В 2016 году вышла последняя, на текущий момент времени, версия — SQL Server 2016, именно эту версию СУБД MS SQL Server мы и будем использовать при изучении текущего предмета, но прежде чем использовать тот или иной программный продукт его необходимо установить на компьютер.

Инсталляция Microsoft SQL Server

СУБД Microsoft SQL Server 2016 доступна в нескольких редакциях: Enterprise, Standard, Developer и Express. Для изучения материалов курса вам достаточно установить версию MS SQL Server 2016 Express, к тому же она бесплатная.

Прежде всего, вам необходимо скачать установщик программного продукта, для этого нужно перейти на сайт компании Microsoft по данному адресу: <https://www.microsoft.com/en-us/download/details.aspx?id=54284>. На этой же странице необходимо ознакомиться с системными требованиями, которым должен соответствовать

ваш компьютер, для того чтобы установка MS SQL Server была успешной. Перечислим эти требования:

- поддерживаемые операционные системы: Windows 10, Windows 8, Windows 8.1, Windows Server 2012, Windows Server 2012 R2, Windows Server 2016;
- минимальная скорость процессора: 1 ГГц;
- минимальный объем оперативной памяти: 512 МБ;
- свободное место на жестком диске: 4,2 ГБ.

После скачивания и запуска установщика, появится окно, позволяющее выбрать тип установки. Если выбрать базовый тип установки, то ядро MS SQL Server будет сконфигурировано по умолчанию. Мы же выберем пользовательский тип установки — это расширенный тип установки, требующий больше времени (рис. 2.1).

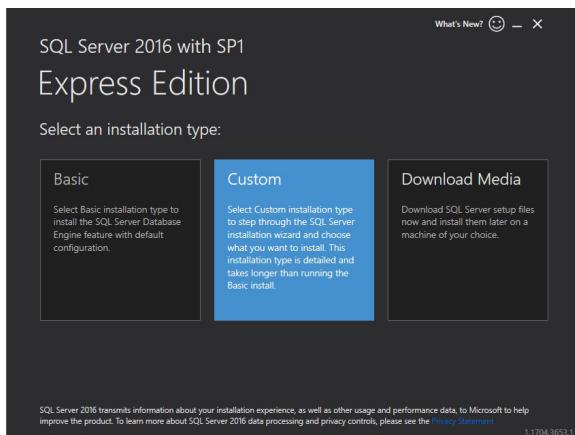


Рисунок 2.1. Выбор пользовательского типа установки

В следующем окне отображается необходимое свободное пространство на диске, размер скачивания и папка на компьютере, куда будет производиться установка,

в случае необходимости ее можно изменить. Оставим расположение установки по умолчанию и запустим установку, нажав кнопку **Install** (Рис. 2.2).

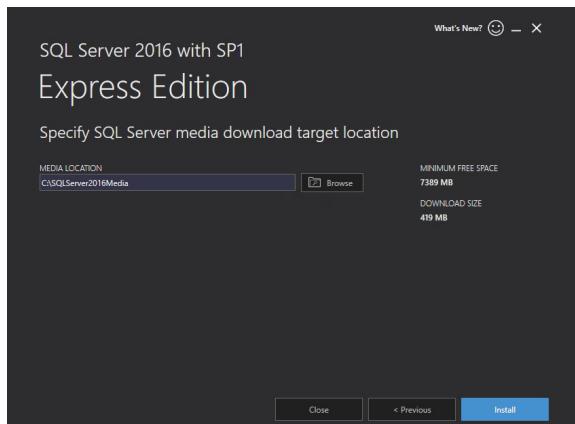


Рисунок 2.2. Начало установки

После этого начинается скачивание пакета установки, процесс скачивания отображается при помощи индикатора (рис. 2.3).

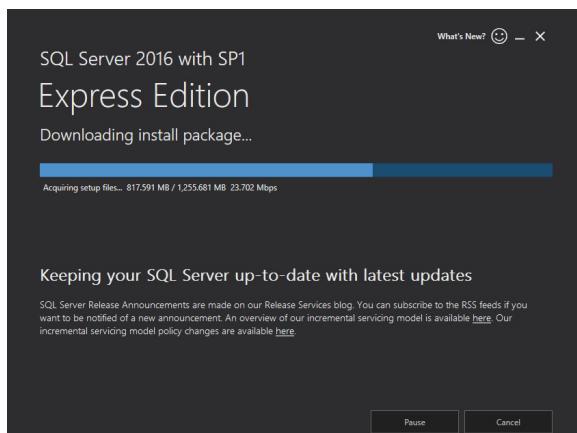


Рисунок 2.3. Скачивание пакета установки

После того как будет скачан пакет установки, появиться окно установки MS SQL Server 2016 с выбранным пунктом **Installation**. В списке возможных установок необходимо выбрать первый пункт — **New SQL Server stand-alone installation...** (рис. 2.4).

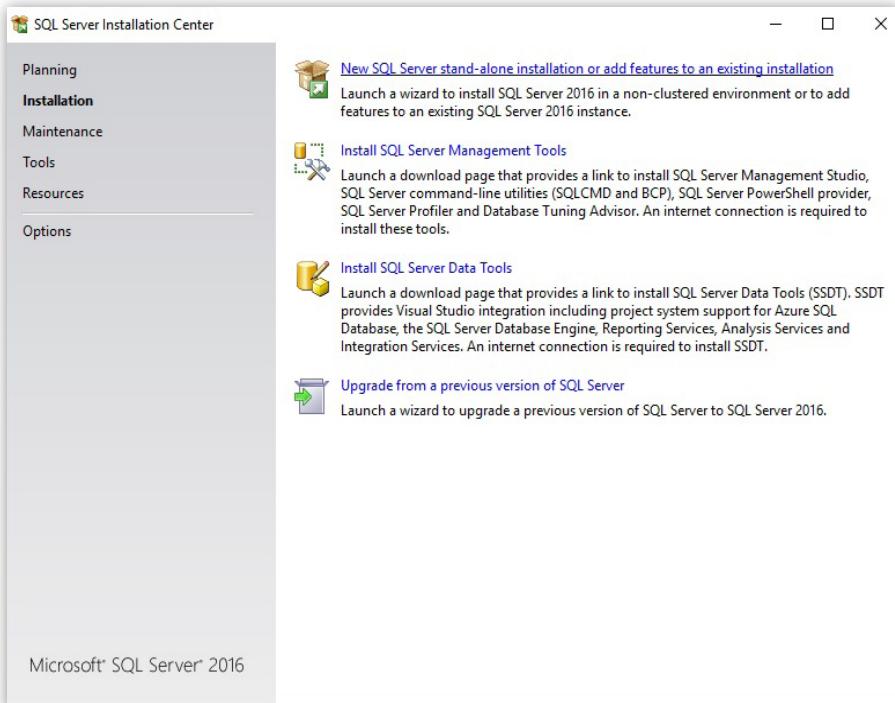


Рисунок 2.4. Начало новой установки экземпляра
MS SQL Server

В следующем окне предлагается ознакомиться с лицензионным соглашением от корпорации Microsoft по использованию Microsoft SQL Server 2016 Express. После внимательного ознакомления, его следует принять и нажать кнопку **Next** (рис. 2.5).

Unit 1. Введение в теорию баз данных

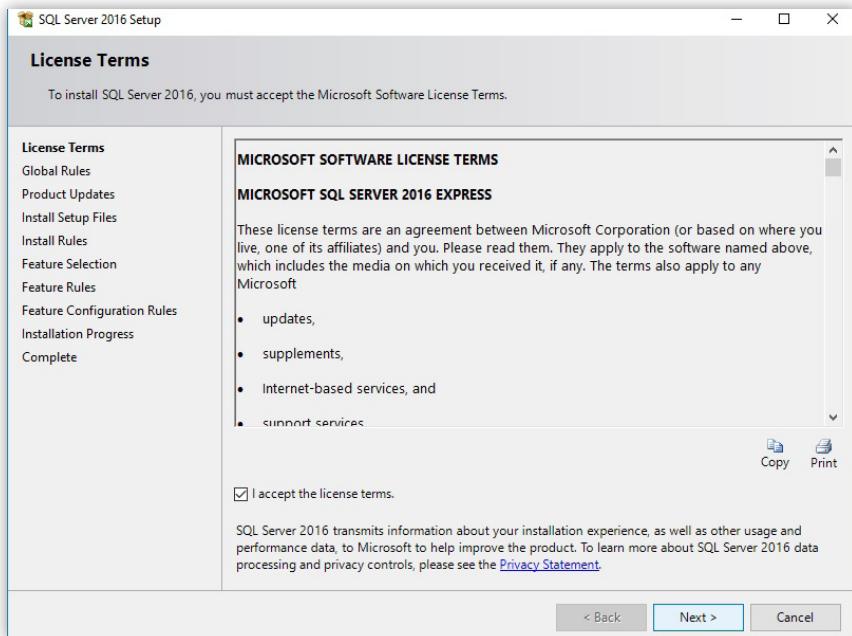


Рисунок 2.5. Лицензионное соглашение от корпорации Microsoft

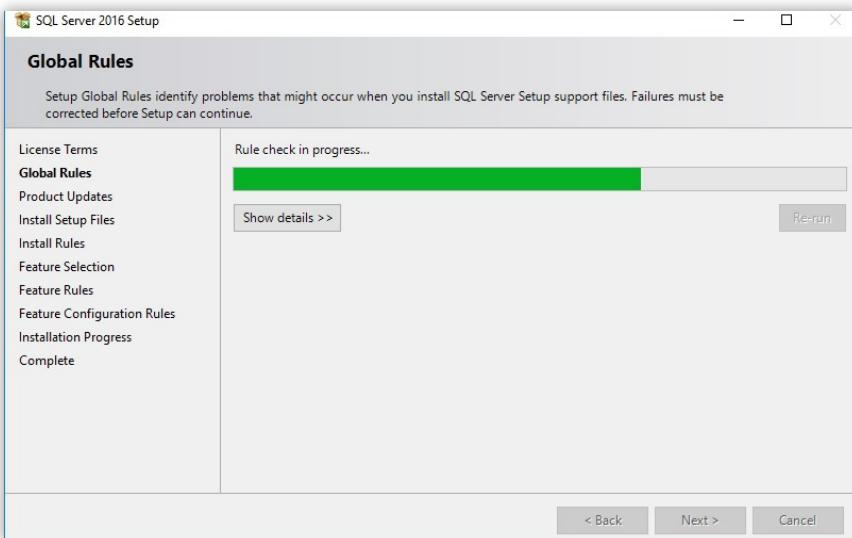


Рисунок 2.6. Установка файлов

После этого осуществляется поиск обновлений, загрузка и установка файлов установки (рис. 2.6).

Затем программа установки определяет потенциальные проблемы, которые могут возникнуть при выполнении установки MS SQL Server. В нашем случае появилось предупреждение, в котором говорится о необходимости открытия соответствующих портов для обеспечения удаленного доступа. Мы учтем это предупреждение при настройке удаленного доступа, а пока что нажмем кнопку **Next** (рис. 2.7).

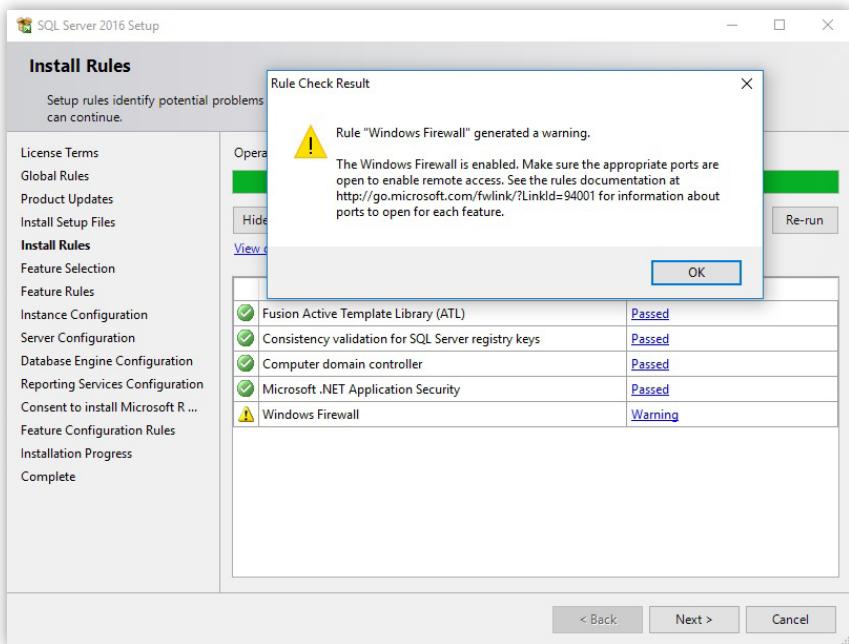


Рисунок 2.7. Правила установки

В следующем окне установки необходимо выбрать компоненты для установки и перейти на следующее окно (рис. 2.8).

Unit 1. Введение в теорию баз данных

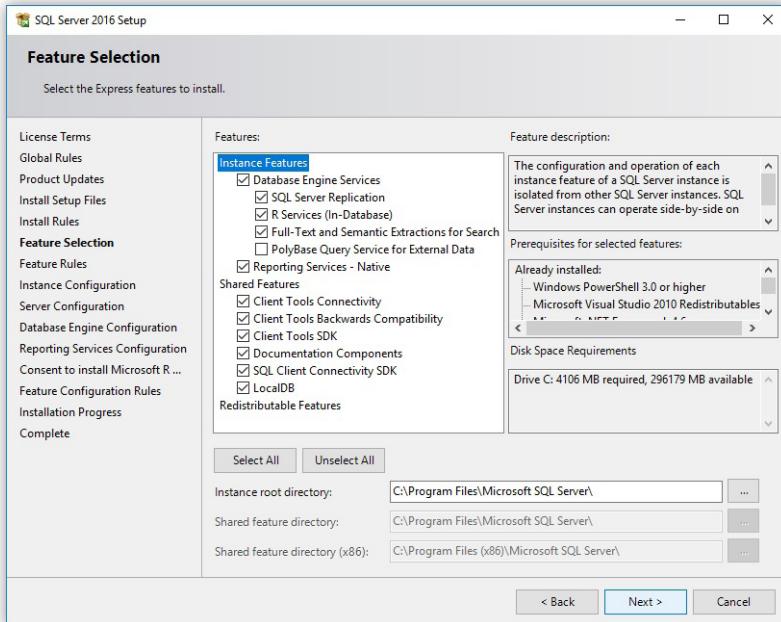


Рисунок 2.8. Выбор компонентов

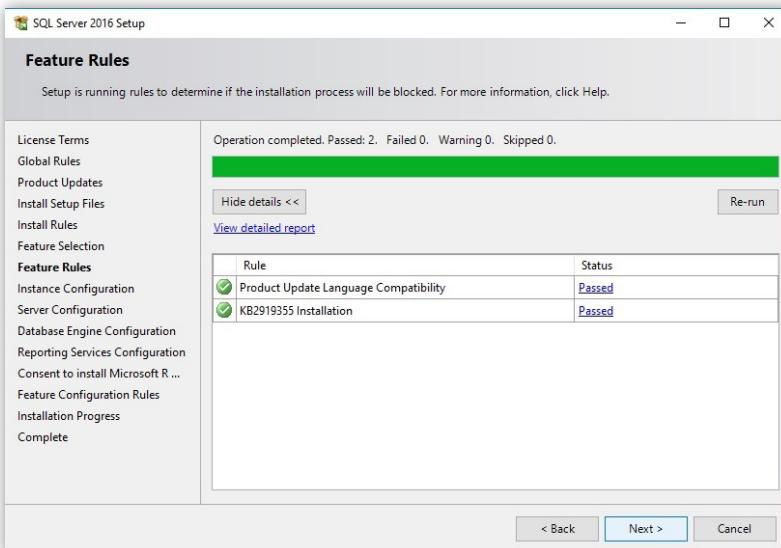


Рисунок 2.9. Установка компонентов

В следующем окне демонстрируется процесс и результат установки выбранных ранее компонентов, убедившись, что установка прошла успешно нужно нажать кнопку **Next** (рис. 2.9).

Далее вы увидите окно, в котором необходимо указать имя и идентификатор для экземпляра SQL Server, выберем пункт **Named instance** (*Именованный экземпляр*) и перейдем далее (рис. 2.10).

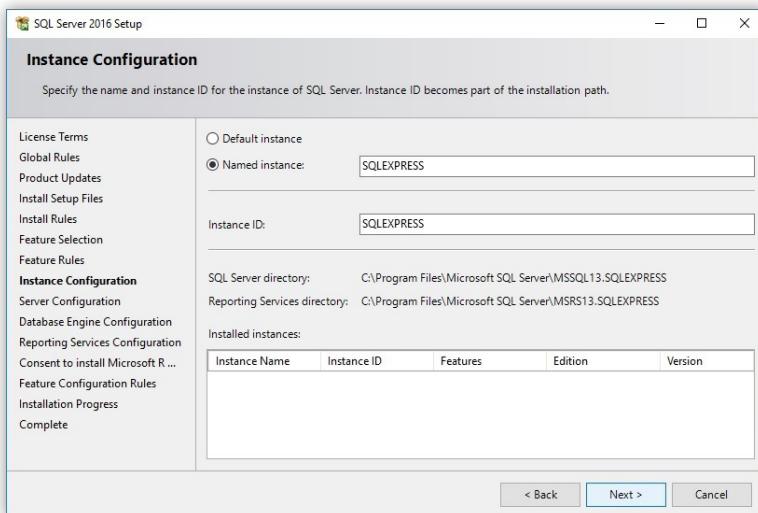


Рисунок 2.10. Настройка экземпляра

Следующее окно предназначено для настройки учетных записей и параметров сортировки, оставим эти пункты без изменений и нажмем кнопку **Next** (рис. 2.11).

Следующее окно позволяет настроить компонент **Database Engine**. На вкладке **Server Configuration** нужно выбрать один из двух режимов проверки подлинности, либо **Windows authentication mode** (*режим проверки подлинности Windows*), либо **Mixed Mode** (*смешанный*

Unit 1. Введение в теорию баз данных

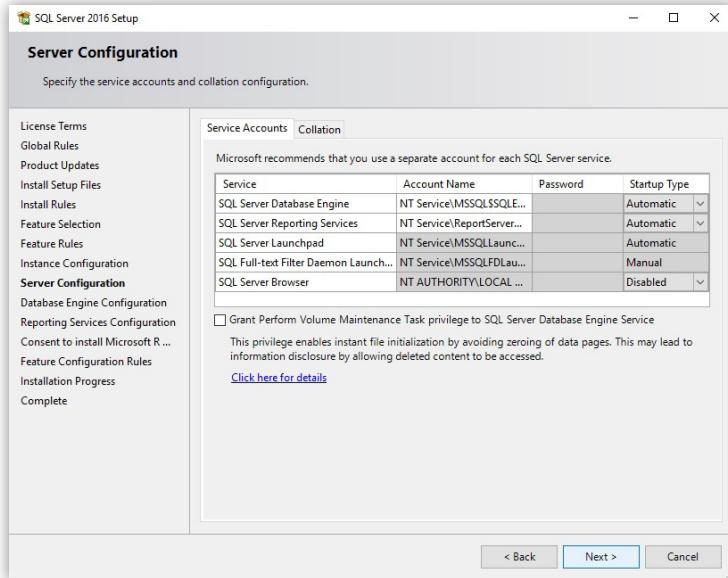


Рисунок 2.11. Конфигурация сервера

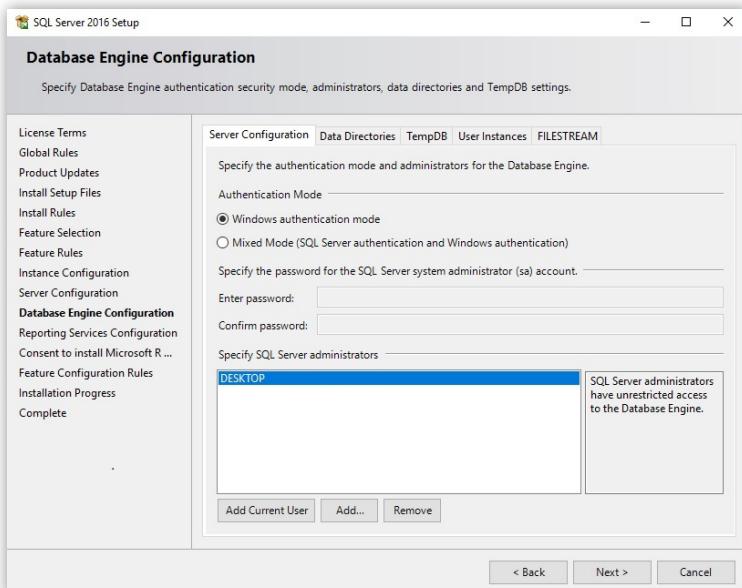


Рисунок 2.12. Вкладка «Конфигурация сервера»

режим), при выборе которого необходимо указать пароль для встроенной учетной записи администратора SQL Server (sa), оставим здесь значение по умолчанию. Также необходимо указать администраторов SQL Server, по умолчанию в списке будет находиться имя текущего пользователя. При помощи соответствующих кнопок вы можете, как добавить пользователя или текущего пользователя, так и удалить пользователя из списка (рис. 2.12).

На вкладке **Data Directories** задим корневой каталог данных. Можно оставить значение по умолчанию, однако рекомендуется указывать каталог на другом физическом диске, в крайнем случае, на другом логическом диске, что повышает вероятность сохранения данных в случае отказа системы. В нашем случае, за неимением еще одного физического диска, укажем заранее созданный каталог SQLDB на логическом диске D, для чего нажмем кнопку с тремя точками в пункте **Data root directory** (рис. 2.13).

После нажатия кнопки **OK** в окне **Browse For Folder** (рис. 2.13), пути ко всем каталогам данных обновятся автоматически. Можно изменить расположение каталогов пользовательской базы данных и каталога резервного копирования, проделав аналогичные операции, но мы остановимся на полученном результате (рис. 2.14).

Вкладка **TempDB** предназначена для настройки базы данных, хранящей временные данные, на следующей вкладке **User Interfaces** существует возможность установить разрешение обычным пользователям запускать экземпляр компонента **Database Engine** (по умолчанию установлено) и последняя вкладка позволяет включить **FILESTREAM** для интеграции Database Engine с файловой

Unit 1. Введение в теорию баз данных

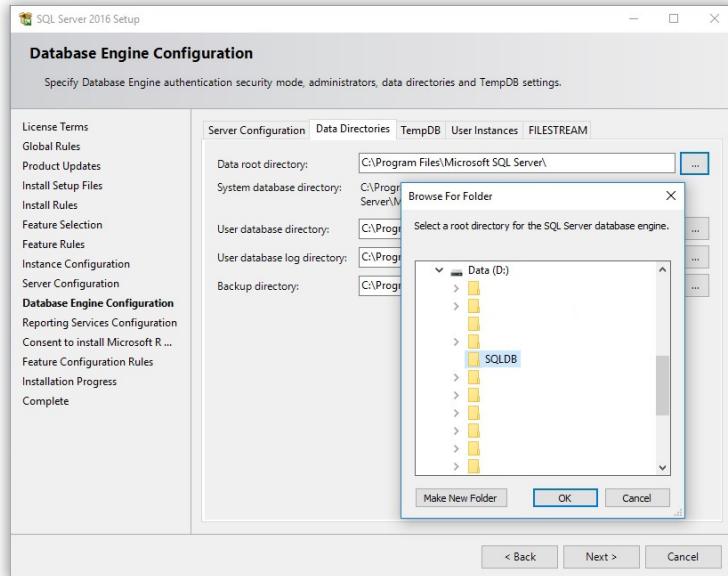


Рисунок 2.13. Установка корневого каталога данных

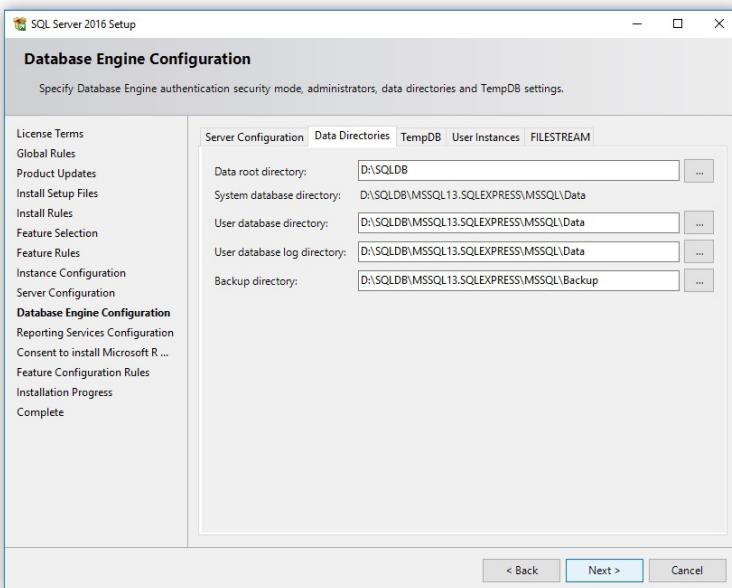


Рисунок 2.14. Вкладка «Каталоги данных»

системой NTFS (по умолчанию отключено). Оставим эти три вкладки без изменений и перейдем далее.

Следующее окно позволяет задать настройку служб **Reporting Services**, выберем пункт **Install and configure** и нажмем кнопку **Next** (рис. 2.15).

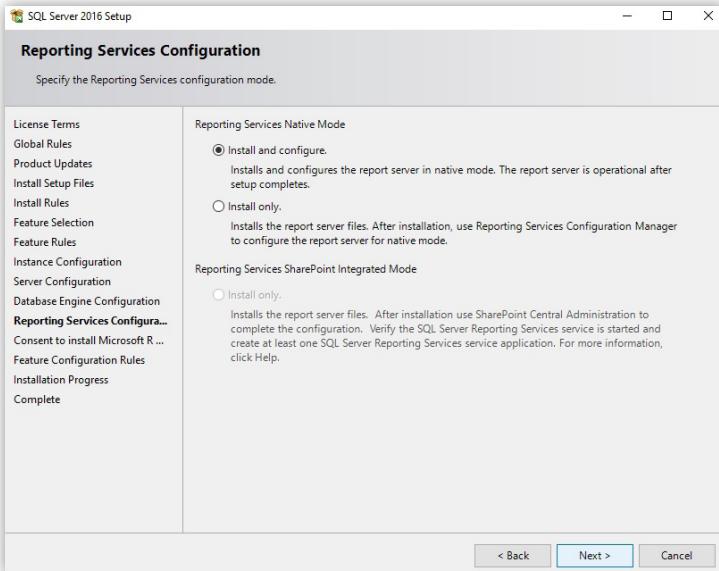


Рисунок 2.15. Настройка служб Reporting Services

Для продолжения установки SQL Server, в следующем окне необходимо нажать кнопку **Accept**, чтобы установить на ваш компьютер расширенное распределение R от корпорации Microsoft — Microsoft R Open, которое поддерживает интеллектуальное моделирование и работу со статистическими данными (рис. 2.16).

Нажав кнопку **Next**, после успешной установки Microsoft R Open начнется процесс установки SQL Server на ваш компьютер, ход выполнения которой вы увидите в следующем окне (рис. 2.17).

Unit 1. Введение в теорию баз данных

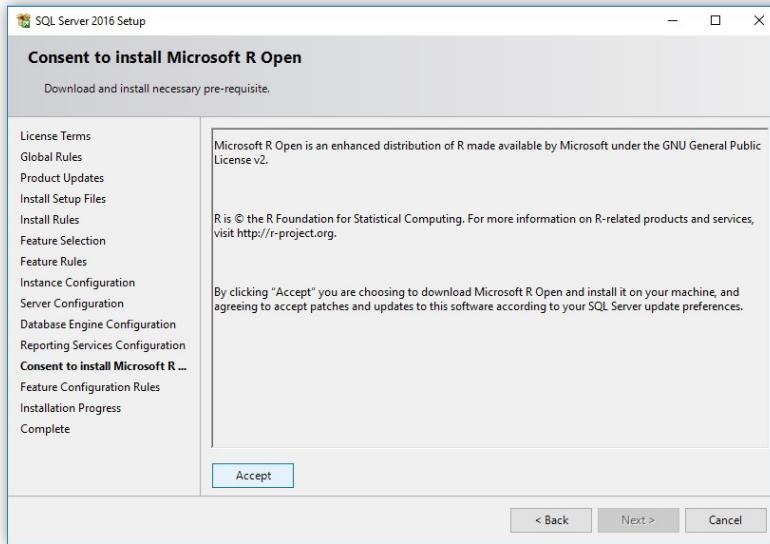


Рисунок 2.16. Установка Microsoft R Open

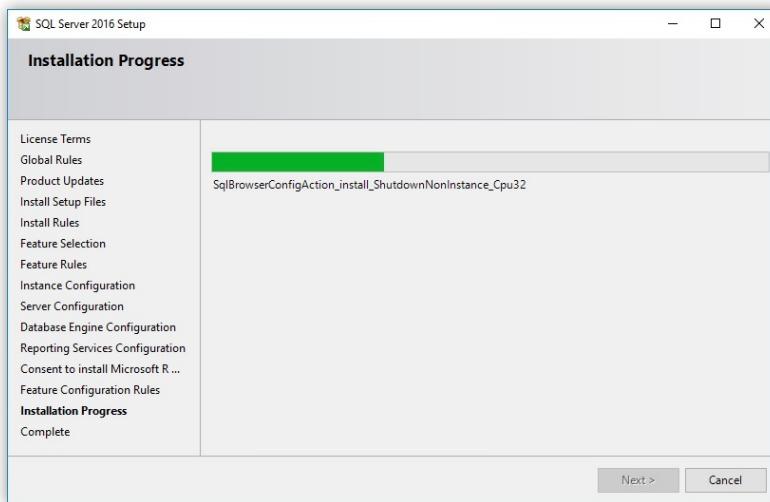


Рисунок 2.17. Ход выполнения установки

По прошествии определенного времени, процесс установки завершится, и вы увидите окно завершения

2. Основы взаимодействия с Microsoft SQL Server

установки SQL Server, которое содержит сведения об установленных компонентах (рис. 2.18).

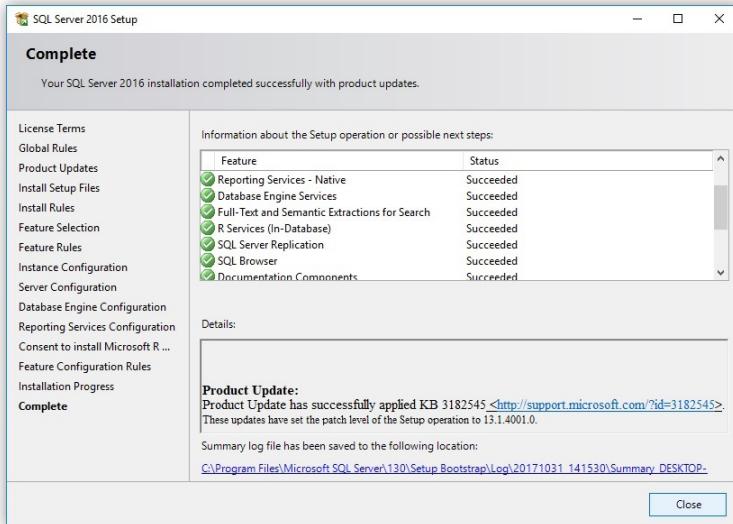


Рисунок 2.18. Завершение установки

По завершению процесса установки, откроем список установленных программ и убедимся, что SQL Server 2016 установлен на ваш компьютер (рис. 2.19).

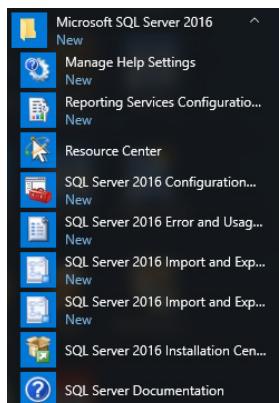


Рисунок 2.19. Программа Microsoft SQL Server 2016

Инструменты управления и утилиты MS SQL Server

В списке установленных компонентов Microsoft SQL Server 2016 вы заметили ряд утилит, кратко остановимся на каждой из них.

Утилита **Reporting Services Configuration Manager** используется для настройки сервера отчетов: учетной записи, базы данных, электронной почты и т.д.

Компонент **SQL Server 2016 Configuration Manager** предназначен для управления службами, связанными с SQL Server (запуск, приостановка, возобновление и остановка служб). При помощи этой утилиты осуществляется управление конфигурацией сетевых подключений с клиентских компьютеров и настройки сетевых протоколов, используемых SQL Server.

Для того чтобы улучшить функциональность SQL Server, корпорация Microsoft использует отчеты об ошибках, за сбор и отправку этой информации отвечает утилита **SQL Server 2016 Error and Usage Reporting**.

Утилита **SQL Server 2016 Import and Export Data** обеспечивает удобный способ копирования данных между источниками данных.

В списке установленных компонентов SQL Server 2016 отсутствует один очень важный инструмент управления — **Microsoft SQL Server Management Studio**, именно этот компонент обеспечивает наличие визуальной среды для работы с SQL Server. Следующее наши действия будут направлены на установку этого компонента, но перед этим необходимо перезагрузить ваш компьютер.

Для установки Management Studio в списке компонентов SQL Server 2016 (рис. 2.19) необходимо выбрать

утилиту **SQL Server 2016 Installation Center**, которая обеспечивает установку необходимых компонентов в SQL Server, нажав соответствующий пункт. После этого откроется одноименное окно (рис. 2.20).

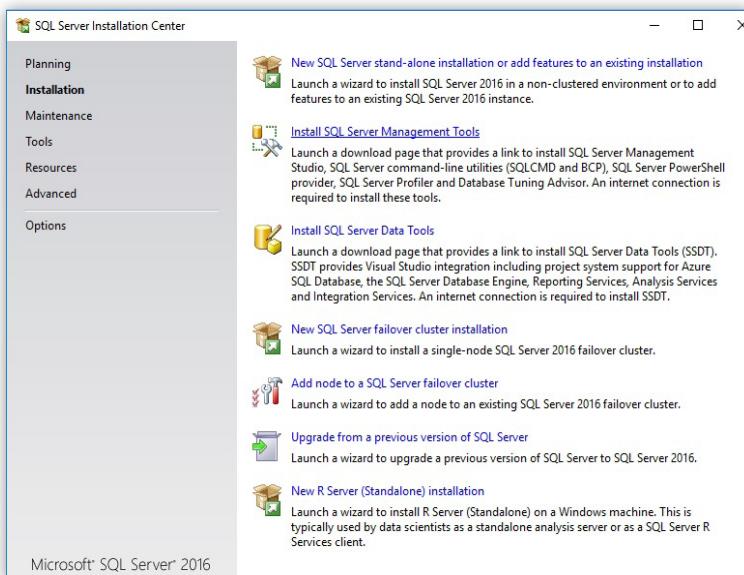


Рисунок 2.20. Центр установки SQL Server

В открывшемся окне необходимо выбрать пункт **Install SQL Server Management Tools**, после чего в окне браузера откроется страница по следующему пути <https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms>, на этой странице необходимо выбрать пункт **Download SQL Server Management Studio 17.3**, после чего начнется загрузка установщика Management Studio на ваш компьютер.

После запуска установщика, появится начальное окно установки, чтобы начать установку нужно нажать кнопку **Install** (рис. 2.21).

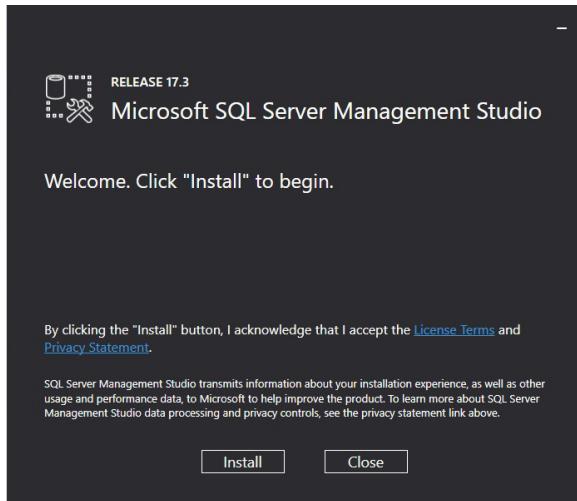


Рисунок 2.21. Начало установки SQL Server Management Studio

Первоначально появится окно загрузки пакетов установки (рис. 2.22).

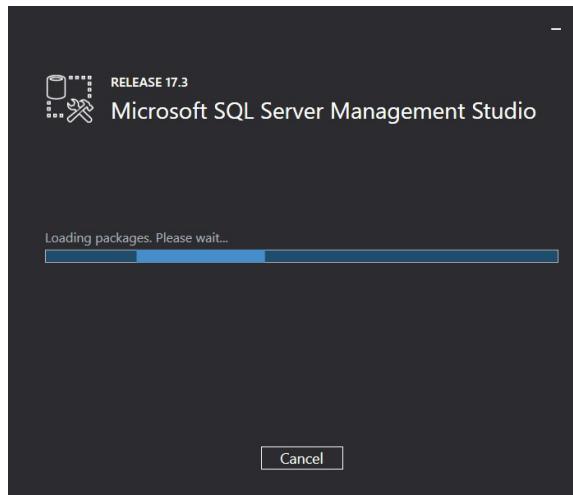


Рисунок 2.22. Загрузка пакетов установки

После того как пакеты будут загружены начнется сам процесс установки SQL Server Management Studio (рис. 2.23).

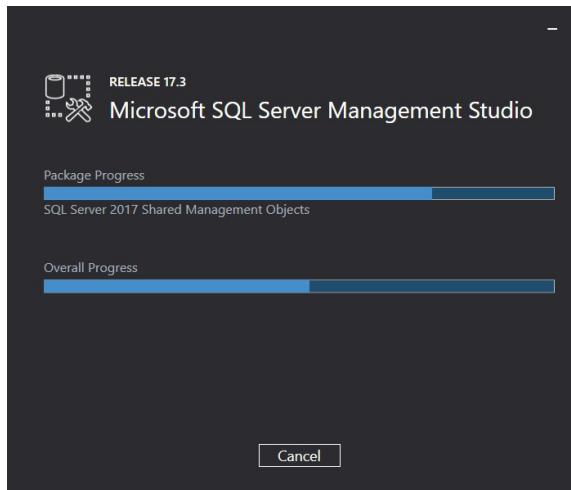


Рисунок 2.23. Процесс установки SQL Server Management Studio

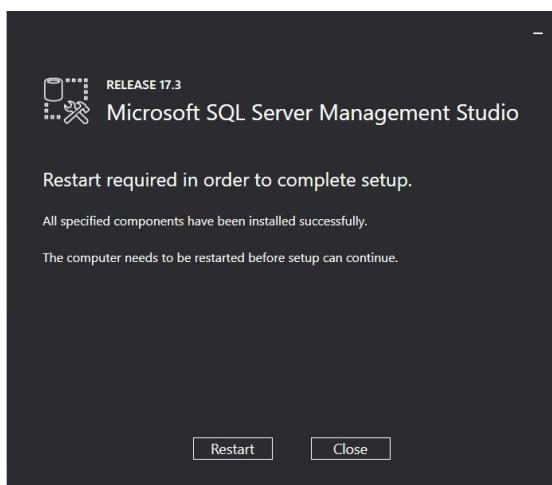


Рисунок 2.24. Окончание установки SQL Server Management Studio

По окончании установки SQL Server Management Studio появится окно, в котором предлагается перезагрузить ваш компьютер, для чего необходимо нажать кнопку **Restart** (рис. 2.24).

После перезагрузки компьютера в списке установленных программ появится пункт **Microsoft SQL Server Tools 17**, раскрыв который вы увидите пункт **Microsoft SQL Server Management Studio 17** (рис. 2.25).



Рисунок 2.25. Microsoft SQL Server Management Studio

Управление базой данных

Чтобы создать базу данных, запускаем Management Studio, нажав соответствующий пункт (рис. 2.25).

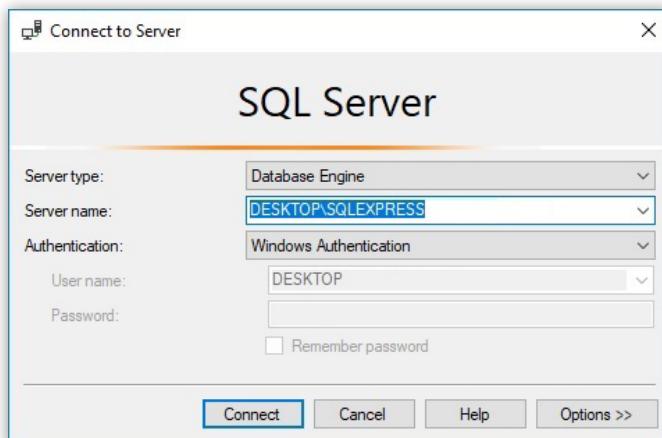


Рисунок 2.26. Подключение к серверу

При первом запуске вы увидите окно загрузки пользовательских настроек, после чего запустится оболочка и появится окно подключения к серверу (рис. 2.26).

В открывшемся окне можно сделать несколько дополнительных настроек, кратко остановимся на каждой из них.

При помощи первой настройки (*Server type*) можно задать одну из технологий управления и анализа данных:

- **Database Engine** — основная служба, обеспечивающая безопасность данных, их хранение и обработку (значение по умолчанию);
- **Analysis Services** — служба, предназначенная для обработки аналитических данных и включает набор средств для бизнес-аналитики;
- **Reporting Services** — служба, предоставляющая средства для создания различного рода отчетов, с возможностью их публикации в разнообразных форматах;
- **Integration Services** — служба, которая включает различного рода решения по интеграции данных (извлечение, преобразование, загрузка) (рис. 2.27).

Имя сервера (*Server name*) прописывается автоматически, но в случаях, когда требуется указать его явно, необходимо открыть выпадающий список и выбрать требуемый сервер (рис. 2.28).

При помощи последней настройки (*Authentication*) можно задать режим проверки подлинности пользователя:

- **Windows Authentication** — этот режим задан по умолчанию и позволяет пользователю при подключении использовать учетную запись ОС Windows;

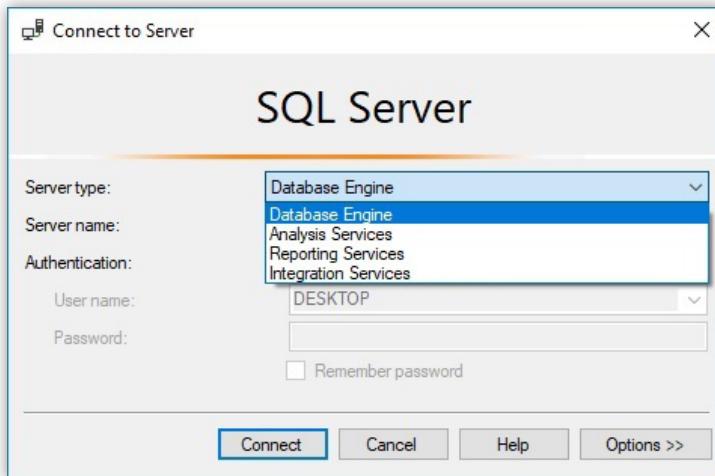


Рисунок 2.27. Технологии управления и анализа данных

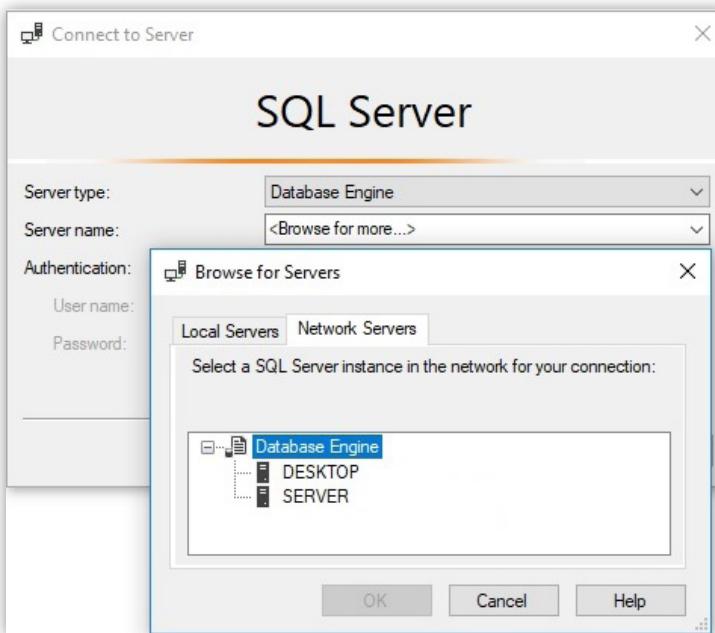


Рисунок 2.28. Выбор сервера

- **SQL Server Authentication** — режим, при котором подлинность пользователя определяется средствами SQL Server, в этом случае проверяется соответствие имени пользователя и пароля;
- **Active Directory – Universal with MFA support** — режим, который обеспечивает аутентификацию пользователя при помощи различных вариантов проверки (телефонный звонок, текстовое сообщение и т.д.);
- **Active Directory – Password** — режим аутентификации, который позволяет подключиться к базе данных Microsoft Azure SQL, используя идентификаторы в Azure Active Directory, если текущая учетная запись не связана с Azure;
- **Active Directory – Integrated** — режим аутентификации, который подобен предыдущему, но применяется, если при входе в OC Windows использовалась учетная запись Azure Active Directory (рис. 2.29).

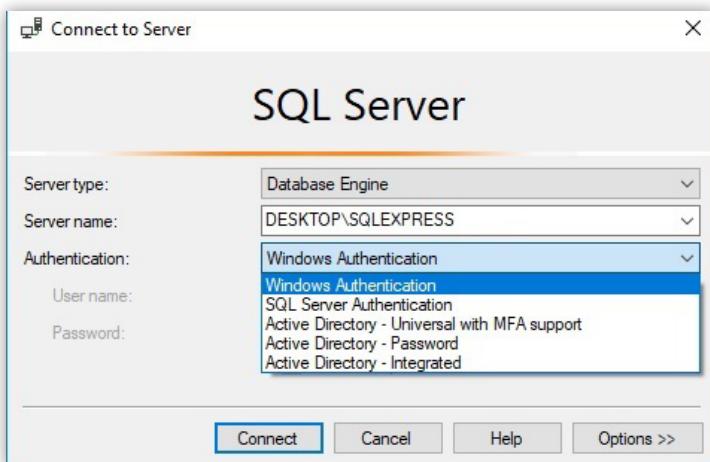


Рисунок 2.29. Выбор режима аутентификации

Оставьте все настройки по умолчанию (рис. 2.26) и нажмите кнопку **Connect**. После этого откроется Management Studio, в левой части которой вы увидите окно **Object Explorer**. В этом окне находится список папок, в которых содержатся различные настройки SQL Server, сейчас нас интересует папка **Databases**, с остальными папками ознакомимся по мере необходимости (рис. 2.30).

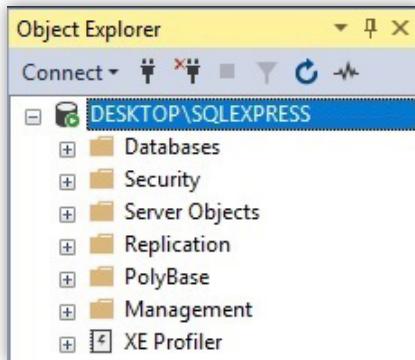


Рисунок 2.30. Окно Object Explorer

Создание базы данных

Для того чтобы создать новую базу данных необходимо на папке **Databases** нажать правой клавишей мыши и в контекстном меню выбрать пункт **New Database...** (рис. 2.31).

После этих действий появится окно **New Database**, в котором необходимо указать имя базы данных, в нашем случае **University**, при этом автоматически изменяются логические имена файлов БД (*таблица Database files*). Также можно указать владельца базы данных, здесь мы оставим значение по умолчанию (*default*) (рис. 2.32).

2. Основы взаимодействия с Microsoft SQL Server

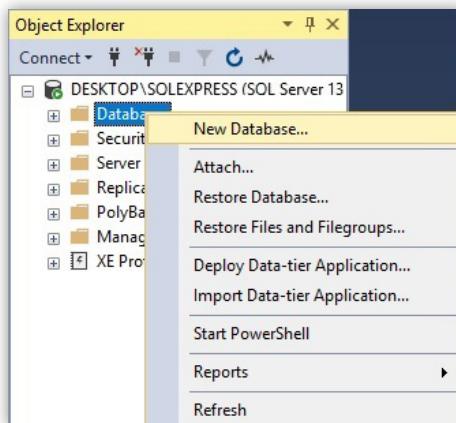


Рисунок 2.31. Начало создания новой базы данных

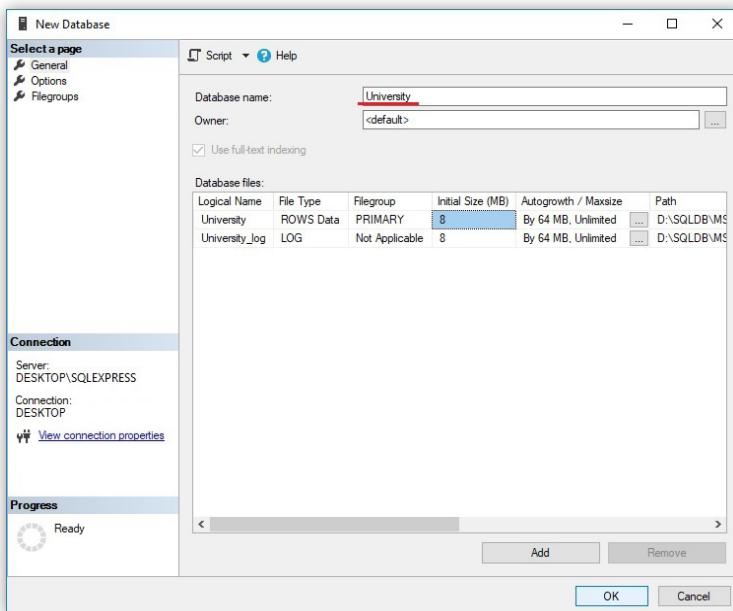


Рисунок 2.32. Имя новой базы данных

Хотя для создания базы данных изменять остальные настройки не обязательно, мы остановимся на них более подробно.

Настройка параметров базы данных

Любая база данных в Microsoft SQL Server состоит из двух физических файлов. Файл с расширением **.mdf** — это основной файл данных, в нем содержится информация обо всех объектах БД. Файл с расширением **.ldf** — это файл журнала транзакций, он играет роль некоего предварительного хранилища данных. Параметры этих файлов находятся в таблице **Database files**.

Изменение размера базы данных

Чтобы изменить размер базы данных изменяем три параметра: начальный размер, автоматическое приращение размеров и максимальный размер файлов БД.

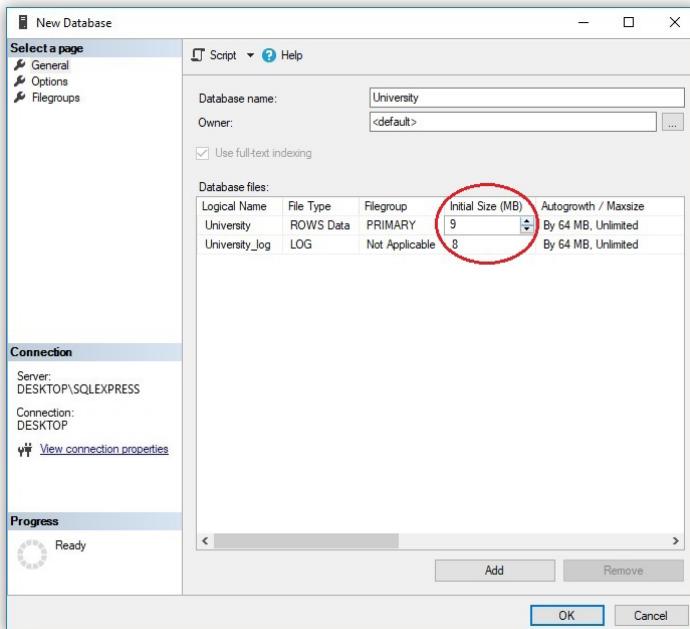


Рисунок 2.33. Изменение начального размера файлов базы данных

Начальный размер файлов базы данных изменяется указанием целочисленного значения размера в поле **Initial Size** таблицы **Database files** (рис. 2.33).

Максимальный размер и автоматическое приращение размера конкретного файла базы данных можно настроить, нажав кнопку с тремя точками в соответствующей ячейке поля **Autogrowth/Maxsize**, после чего появиться окно **Change Autogrowth for University** (рис. 2.34).

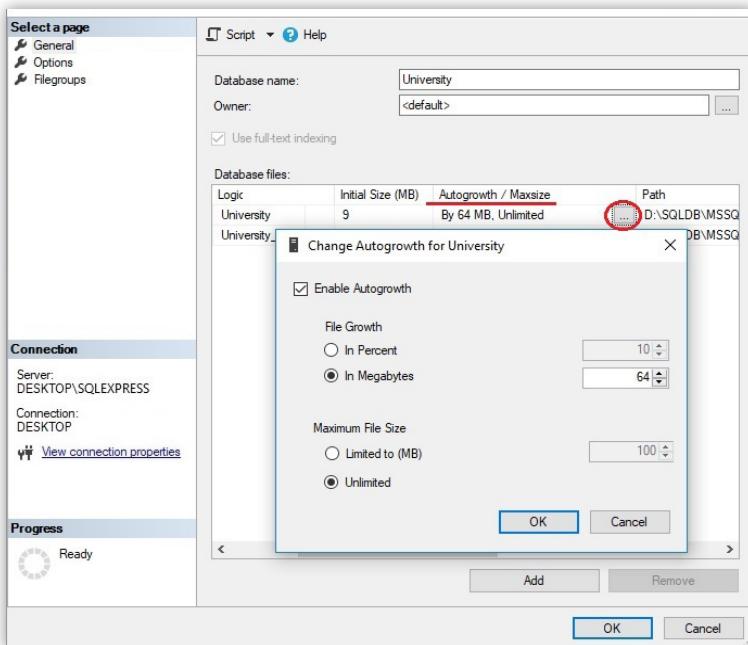


Рисунок 2.34. Изменение максимального размера и приращения размера файла БД

В окне **Change Autogrowth for University** вы можете задать автоматическое приращение размера файла либо в процентах, либо в мегабайтах, указав необходимое значение. Аналогичным образом можно задать макси-

мальный размер файла базы данных, либо указав размер в мегабайтах, либо выбрав пункт **Unlimited**.

Управление группами файлов

Файлы данных базы данных подразделяются на файловые группы, которые используются для объединения файлов с целью повышения производительности. В любой базе данных существует файловая группа **PRIMARY**, в ней находится первичный файл данных. Также существует возможность создания пользовательских файловых групп, для этого необходимо в списке [Select a page](#) выбрать пункт **Filegroups**, после чего появится соответствующий интерфейс (рис. 2.35).

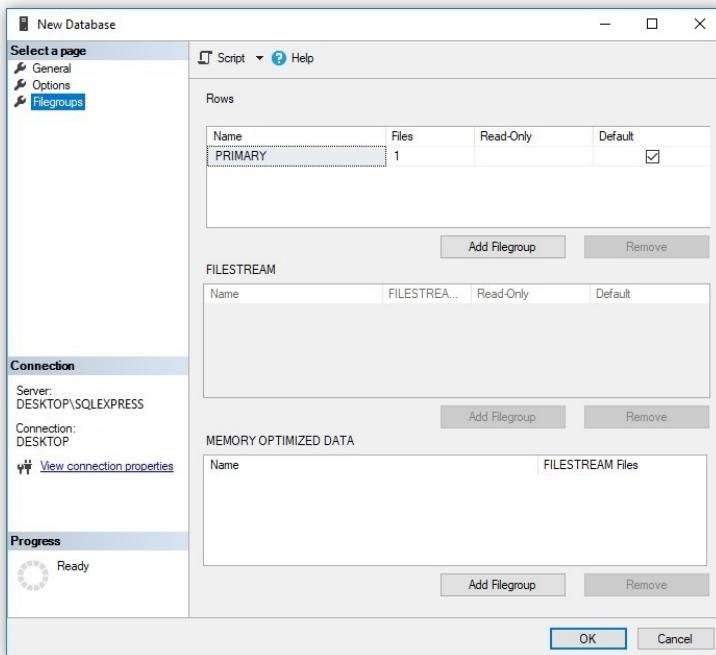


Рисунок 2.35. Управление группами файлов

2. Основы взаимодействия с Microsoft SQL Server

После того как вы сделаете все настройки вашей базы данных и нажмете кнопку **OK** в окне **New Database**, вы увидите индикатор создания БД (рис. 2.36).

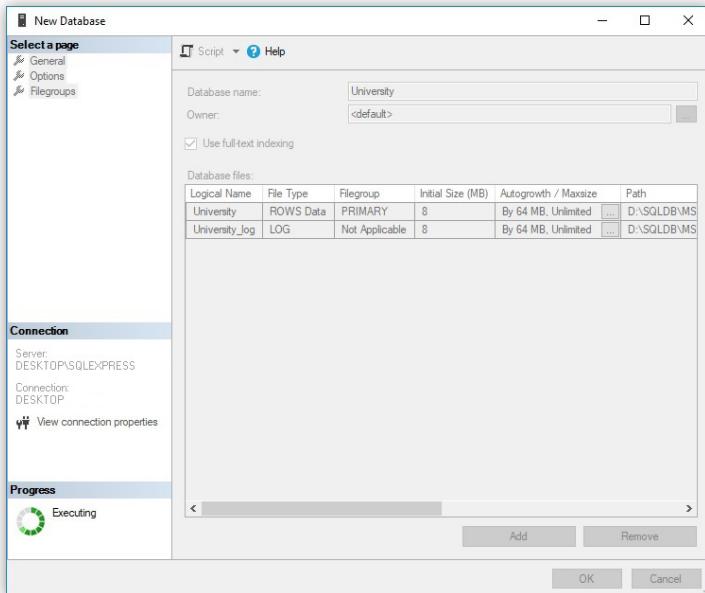


Рисунок 2.36. Процесс создания базы данных

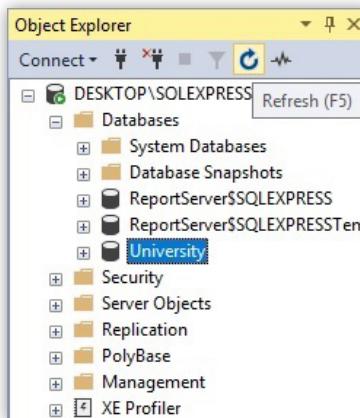


Рисунок 2.37. Результат создания новой базы данных

После закрытия окна, ваша база данных появится в списке **Databases** окна **Object Explorer** (рис. 2.37).

Если по какой-то причине, созданная вами база данных не отобразилась в списке, нажмите кнопку **Refresh** (рис. 2.37) и список баз данных обновится.

Переименование базы данных

После того как база данных создана, существует возможность внесения в нее ряда изменений, например ее можно переименовать. Для этого необходимо щелкнуть правой клавишей мыши на названии редактируемой базы данных и в контекстном меню выбрать пункт **Rename**, а затем ввести новое имя (рис. 2.38).

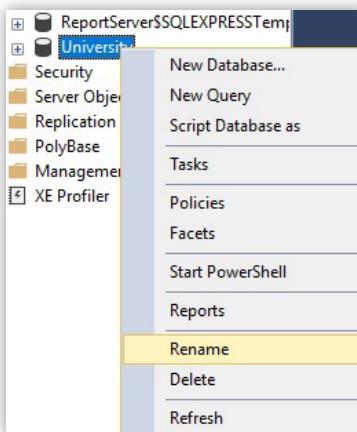


Рисунок 2.38. Переименование базы данных

Удаление базы данных

Существует также возможность удаления существующей базы данных, для этого необходимо вызвать контекстное меню на названии конкретной базы данных и выбрать пункт **Delete** (рис. 2.39).

2. Основы взаимодействия с Microsoft SQL Server

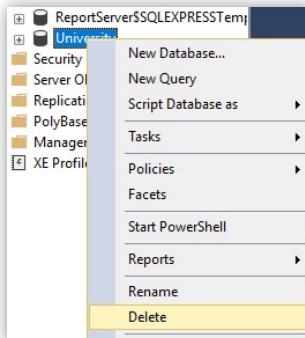


Рисунок 2.39. Начало удаления базы данных

После этого появится окно **Delete Object**, в котором после нажатия кнопки **OK** вы увидите индикатор процесса удаления (рис. 2.40).

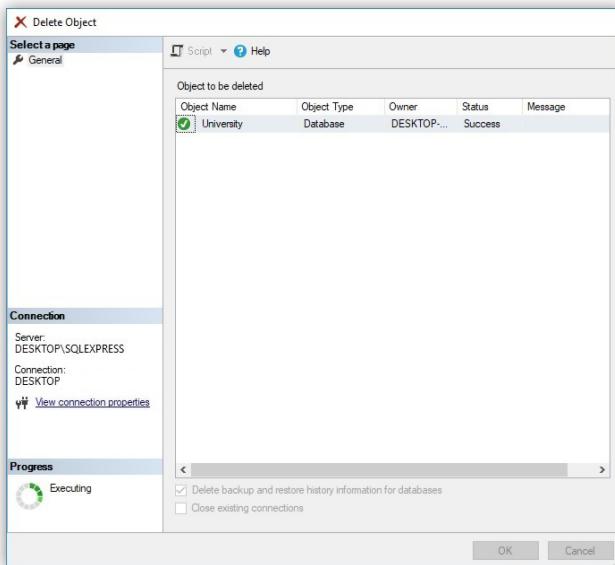


Рисунок 2.40. Завершение удаления базы данных

После того как окно закроется, ваша база данных исчезнет из списка **Databases** окна Object Explorer.

Домашнее задание

1. Установить Microsoft SQL Server 2016 Express и SQL Server Management Studio на свой компьютер.
2. Создать собственную базу данных. Задать: файлу данных начальный размер 15 МВ, приращение размера файла 15% и неограниченный максимальный размер; файлу транзакций — начальный размер 10 МВ, автоматическое приращение размера в 5 МВ и максимальный размер в 60 МВ.

Unit 2.
Таблицы,
типы данных,
запросы, основы
взаимодействия
с MS SQL Server

1. Таблицы

На прошлом занятии вы изучили порядок создания базы данных и выполнение различных действий по ее настройке. Однако сама по себе база данных особой пользы не принесет, для того чтобы ее можно было использовать в полном объеме необходимо наличие в ней ряда таблиц.

Таблицы в базе данных представляют собой множество данных, объединенных по определенному принципу. Каждая таблица описывает некую сущность при этом в столбцах хранится конкретное свойство сущности, а в строках — совокупность свойств для конкретной сущности.

Что бы было более понятно, разберем это на примере. Допустим, у нас есть таблица **Students**, которая хранит информацию о студентах некоего учебного заведения. В этой таблице столбцы **Last Name**, **First Name**, **Birth Date**, **Grants**, **Email** необходимы для хранения информации о фамилии, имени, дате рождения, стипендии и адресе электронной почты соответственно, а каждая строка в этой таблице содержит информацию о конкретном студенте. Например, студент Joshua Johnson родился 23 мая 1997 года, адрес его электронной почты jo@net.eu, и данные о его стипендии отсутствуют (рис. 1.1).

Id	Last Name	First Name	Birth Date	Grants	Email
1	Doe	John	1998-02-11	NULL	jh@net.eu
2	Jones	Jack	1997-11-05	1256.00	jj@net.eu
4	Johnson	Joshua	1997-05-23	NULL	jo@net.eu

Рисунок 1.1. Таблица с информацией о студентах

Теперь, когда предназначение таблиц стало понятным, мы можем приступить к процессу создания собственных таблиц. Для того чтобы создать таблицу в базе данных необходимо раскрыть пункт необходимой БД (в нашем случае **University**) в окне Object Explorer, на папке **Tables** нажать правой клавишей мыши и в контекстном меню выбрать пункт **Table...** (рис. 1.2).

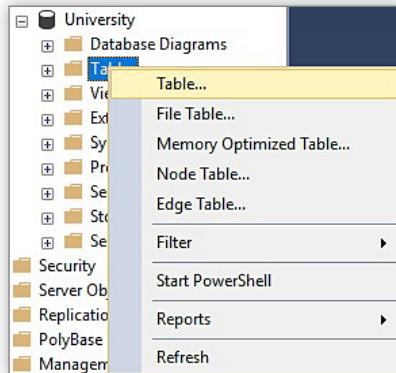


Рисунок 1.2. Создание новой таблицы (начало)

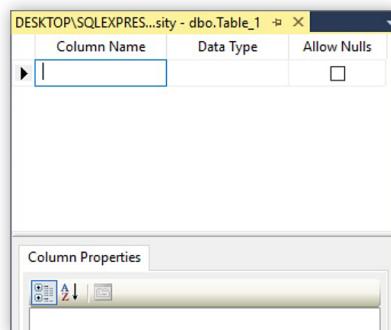


Рисунок 1.3. Создание новой таблицы (продолжение)

В открывшейся вкладке находится таблица, состоящая из трех столбцов **Column Name**, **Data Type** и **Allow**

Nulls. Эта таблица позволяет сформировать новую таблицу в текущей базе данных, путем создания необходимых полей, задав им значения в строках, соответственно, название (*Column Name*), тип данных (*Data Type*) и возможность использования неопределенного (*NULL*) значения (*Allow Nulls*) (рис. 1.3).

Первичный ключ

В примере прошлого раздела (рис. 1.1) мы нарочно проигнорировали столбец *Id* дело в том, что этот столбец не совсем обычный. В каждой таблице базы данных должно быть поле или совокупность полей, которое имеет общее название — первичный ключ. Значения первичного ключа никогда не должны повторяться, тем самым обеспечивая уникальность любой записи в таблице. В примере на рисунке 1.1 поле *Id* является первичным ключом таблицы *Students*.

UniversityName	FacultyName
Harvard	Medicine
MIT	Architecture
Stanford	Law
Stanford	Medicine
UT Austin	Architecture
UT Austin	Law

Рисунок 1.4. Пример составного первичного ключа

Существует также понятие составного первичного ключа, в этом случае первичный ключ состоит из двух и более полей, сочетание значений в которых должно быть уникальным. Например, в качестве составного первичного ключа можно использовать поля *UniversityName*

и FacultyName (рис. 1.4), что естественно, так как названия факультетов в одном университете не могут повторяться, а в различных университетах могут.

Из вышесказанного следует, что в любой таблице необходимо создать поле, которое будет являться первичным ключом. Обычно такое поле называют **Id** или в его названии присутствует слово **Id**, например, **StudId** и также обычно, но не обязательно, тип этого поля должен быть целочисленным (о типах данных будет рассказано далее).

Для того чтобы создать в нашей таблице поле с названием **Id** типа **int**, необходимо в новой строке таблицы в поле **Column Name** ввести соответствующее название, а в поле **Data Type** из выпадающего списка выбрать название требуемого типа данных (рис. 1.5).

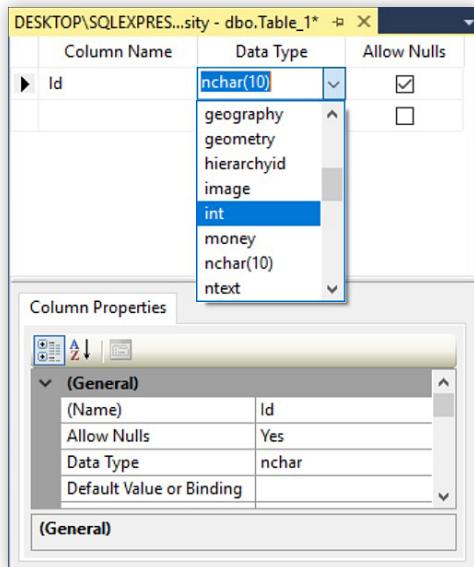


Рисунок 1.5. Выбор требуемого типа данных

Для того чтобы сделать это поле первичным ключом необходимо нажать на него правой кнопкой мыши и в появившемся контекстном меню выбрать пункт **Set Primary Key** (рис. 1.6).

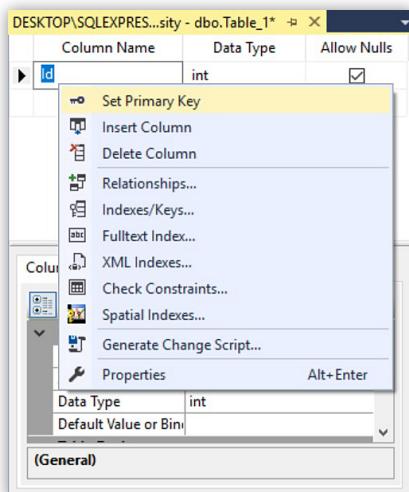


Рисунок 1.6. Создание первичного ключа

В качестве визуального результата вы заметите изображение ключа рядом с выбранным полем, также вы заметите невозможность установки первичному ключу неопределенного значения (сброшено значение в поле **Allow Nulls**).

Существует возможность настроить получение уникального значения первичного ключа путем задания автоматического инкрементирования значений в требуемом поле. Для этого необходимо в таблице **Column Properties** раскрыть свойство **Identity Specification** и в выпадающем списке поля (**Is Identity**) выбрать значение **Yes** (рис. 1.7).

1. Таблицы

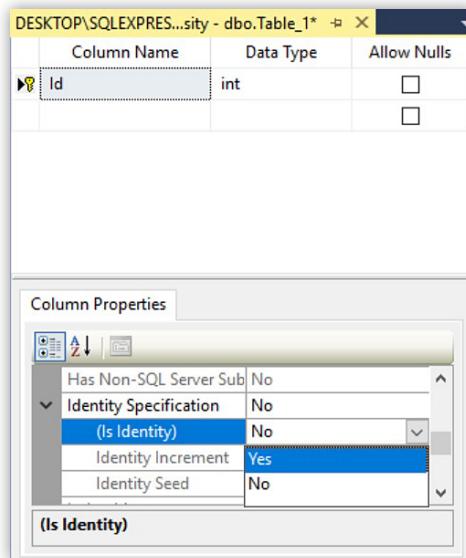


Рисунок 1.7. Настройка уникальности первичного ключа

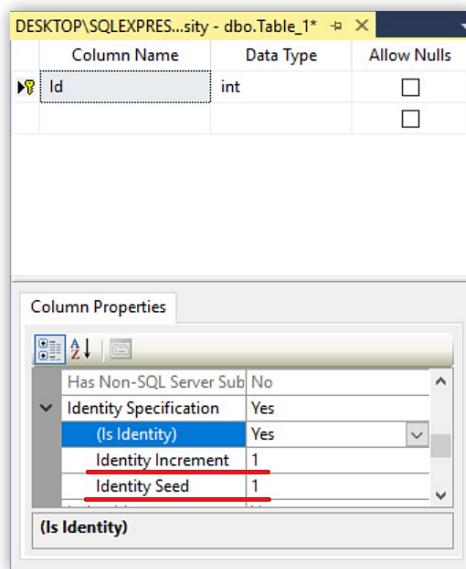


Рисунок 1.8. Настройка автоматического инкрементирования значений

После этого станут доступны поля **Identity Increment** и **Identity Seed**. Изменив значение в поле **Identity Increment**, вы можете задать величину приращения первичного ключа, при помощи поля **Identity Seed** можно задать начальное значение соответствующего поля таблицы, по умолчанию значения в этих полях равны 1 (рис. 1.8).

Первичные ключи также используются для создания связей между таблицами, но об этом вы более подробно узнаете в четвертом уроке.

Значение по умолчанию

При создании таблицы существует возможность задать значения по умолчанию для конкретных полей, то есть в том случае если при добавлении новой записи в таблицу полю не будет указано конкретное значение, то будет записано указанное значение по умолчанию.

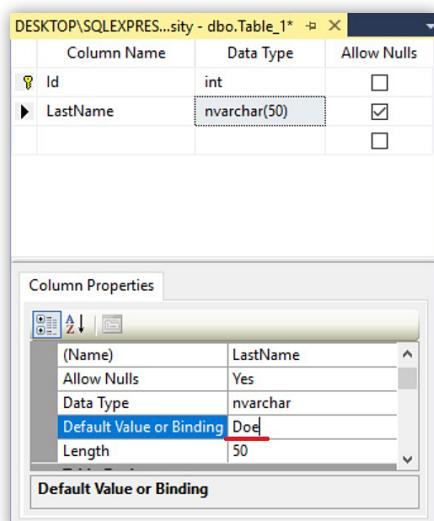


Рисунок 1.9. Настройка значения по умолчанию

Продолжим формирование полей нашей таблицы. Следующее поле имеет название `LastName`, оно предназначено для хранения фамилии студента и имеет тип данных `nvarchar(50)`.

Для того чтобы задать полю значение по умолчанию необходимо в таблице `Column Properties` задать требуемое значение свойству `Default Value or Binding`, в нашем случае — `Doe` (рис. 1.9).

Уникальность

При создании таблицы может возникнуть необходимость в использовании полей с уникальными значениями для каждой записи и `Management Studio` предоставляет такую возможность.

В нашем случае таким полем является поле для хранения электронного адреса студента (`Email`). Для того чтобы объявить это поле уникальным, необходимо нажать по нему правой клавишей мыши и в контекстном меню выбрать пункт `Indexes/Keys...` (рис. 1.10).

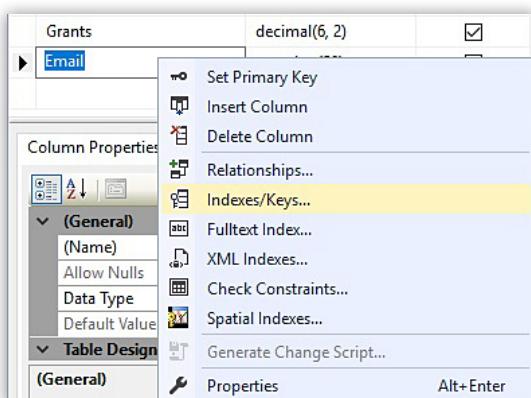


Рисунок 1.10. Создание уникального ключа (начало)

В открывшемся окне **Indexes/Keys** нужно добавить настраиваемый ключ, нажав на кнопку **Add**, при этом система сгенерирует произвольное имя этому ключу, изменить которое можно в поле (**Name**) раздела **Identity** (рис. 1.11).

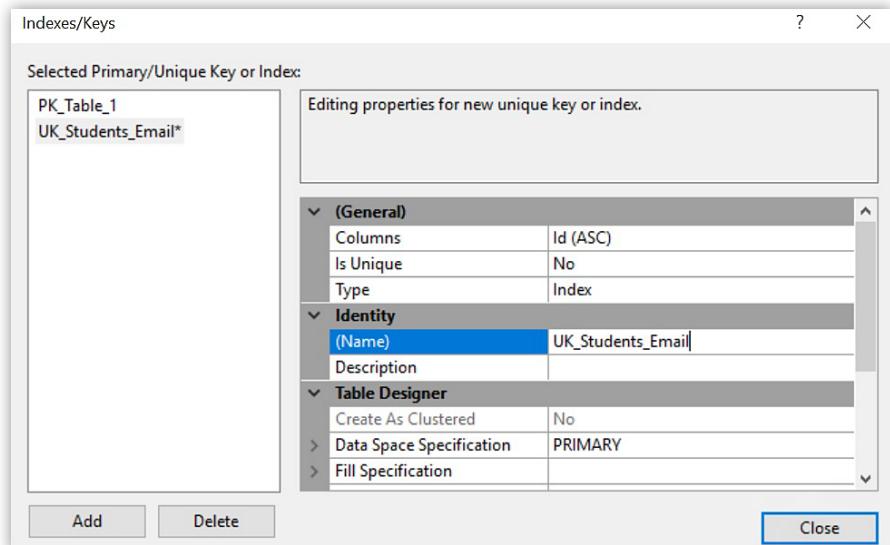


Рисунок 1.11. Изменение имени ключа

Следующее что необходимо сделать — это указать имя настраиваемого столбца в поле **Columns**, для чего нужно в этом поле нажать кнопку с тремя точками и в выпадающем списке появившегося окна **Index Columns** выбрать требуемое название, после чего нажать кнопку **OK** (рис. 1.12).

Последним шагом в настройке уникального поля является указание типа ключа в поле **Type** раздела **(General)**, в выпадающем списке которого необходимо выбрать значение **Unique Key** (рис. 1.13).

1. Таблицы

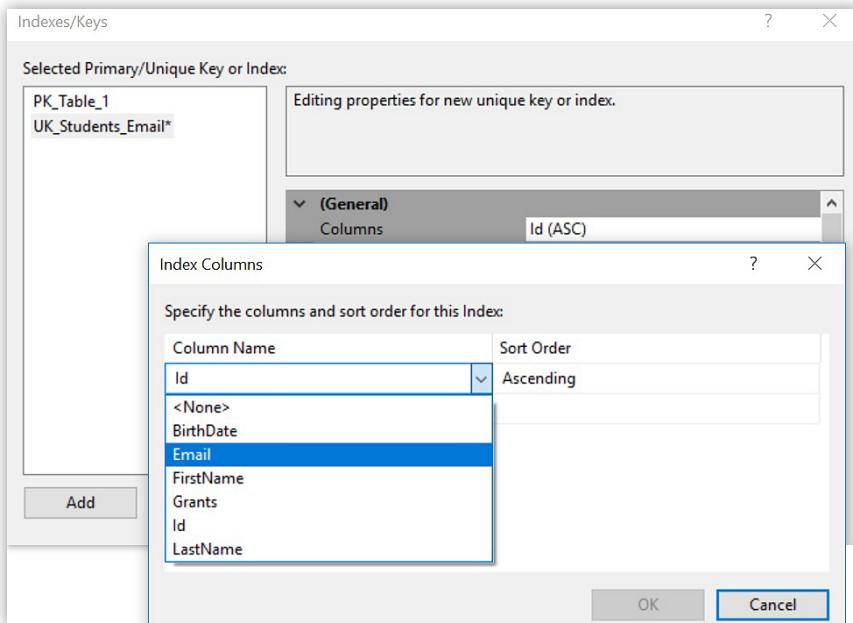


Рисунок 1.12. Выбор настраиваемого столбца

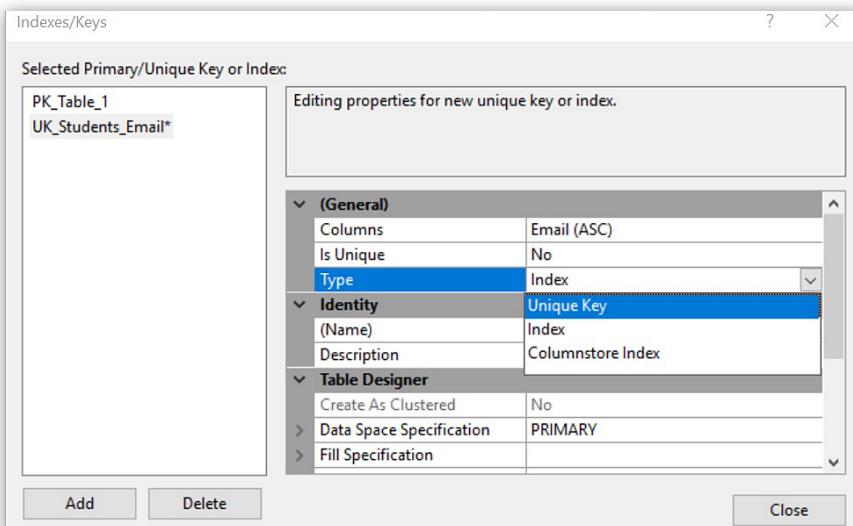


Рисунок 1.13. Указание типа ключа

Чтобы настройки уникальности поля вступили в силу нужно закрыть окно **Indexes/Keys**, нажав кнопку **Close**.

После того как вы создали и настроили все поля таблицы имеет смысл изменить имя таблицы, заданное по умолчанию. Это можно сделать в окне свойств текущей таблицы, указав в поле (**Name**) раздела **Identity** требуемое имя таблицы (рис. 1.14).

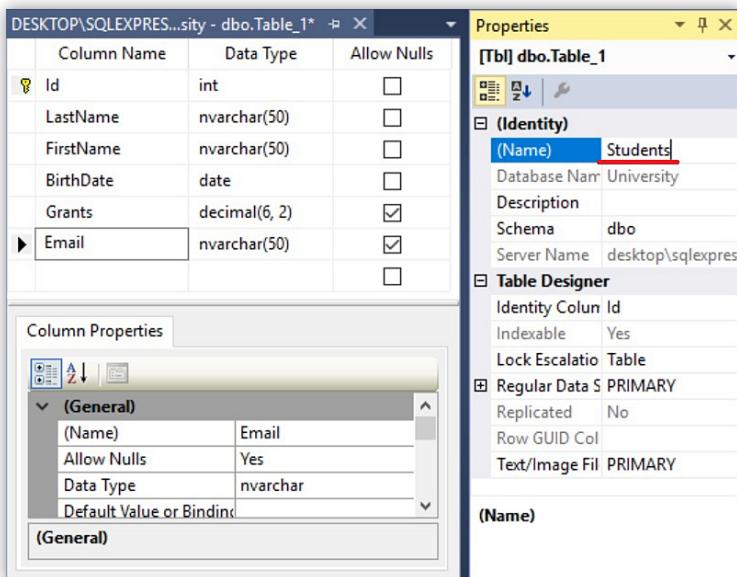


Рисунок 1.14. Переименование таблицы

Сохранение созданной нами таблицы происходит стандартным образом — нажатием кнопки с изображенной дискетой на панели инструментов или сочетанием клавиш **Ctrl+S**. После этого таблица **Students** появится в списке таблиц базы данных **University**.

Для того чтобы убедиться в правильности выполненных нами настроек, заполним таблицу **Students** зна-

1. Таблицы

чениями, для этого необходимо нажать правой клавишей мыши на имени таблицы и в контекстном меню выбрать пункт **Edit Top 200 Rows** (рис. 1.15).

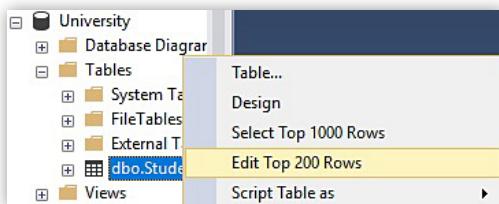


Рисунок 1.15. Заполнение таблицы значениями (начало)

В появившейся вкладке вы увидите таблицу с **NULL**-значениями в каждом поле, заполним их определенными значениями.

Вы можете заметить, что при добавлении в таблицу очередной записи значение в поле **Id** генерируется автоматически, что происходит благодаря настройке автокрементирования первичного ключа.

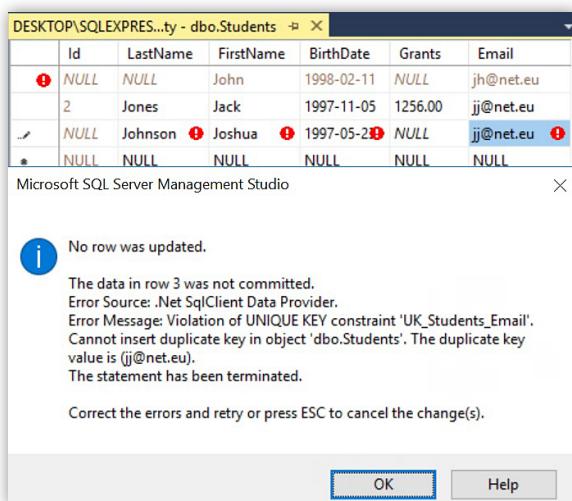


Рисунок 1.16. Ошибка: нарушение уникальности записи

При попытке указать в поле **Email** повторяющееся значение, вы увидите ошибку, в которой сообщается о нарушении уникальности и невозможности дублирования конкретной записи (рис. 1.16).

В первой записи мы не указали значение для поля **Last Name** (проверка на значение по умолчанию) и Management Studio реагирует на это как на ошибку, правда в описании указывается, что ошибки нет, и эта проблема исчезнет после сохранения таблицы (рис. 1.17).

	Id	Last Name	First Name	Birth Date	Grants	Email
1	NULL	NULL	John	1998-02-11	NULL	jh@net.eu

Рисунок 1.17. Ошибка: незаполненное поле

Для того чтобы проверить правильность заполнения таблицы **Students** значениями необходимо нажать правой клавишей мыши на имя таблицы и выбрать пункт **Select Top 1000 Rows** (рис. 1.18).

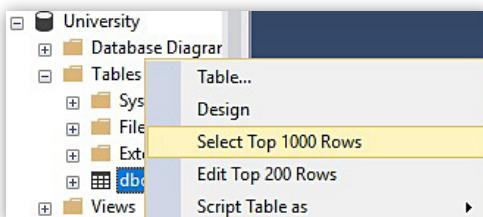
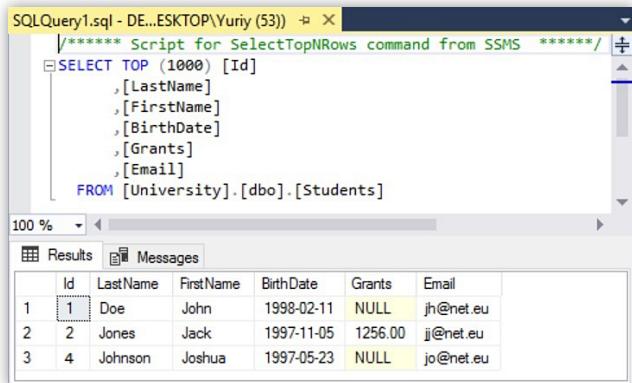


Рисунок 1.18. Выбор записей в таблице (начало)

В открывшейся вкладке вы увидите текст запроса, а в результатах — полученную таблицу с названиями полей и данными в них (рис. 1.19).

1. Таблицы



```
SQLQuery1.sql - DE..ESKTOP\Yuriy (53)  ➔ X
/******** Script for SelectTopNRows command from SSMS *****/
SELECT TOP (1000) [Id]
    ,[LastName]
    ,[FirstName]
    ,[BirthDate]
    ,[Grants]
    ,[Email]
FROM [University].[dbo].[Students]
```

100 %

	Id	LastName	FirstName	BirthDate	Grants	Email	
1	1	Doe	John	1998-02-11	NULL	jh@net.eu	
2	2	Jones	Jack	1997-11-05	1256.00	jj@net.eu	
3	4	Johnson	Joshua	1997-05-23	NULL	jo@net.eu	

Рисунок 1.19. Результат выбора записей таблицы

И хотя текст запроса вам пока непонятен, вы можете убедиться, что все записи заполнились корректно.

2. Типы данных

Как вы уже заметили, для хранения информации в таблицах используются различные типы данных, рассмотрим основные из них.

Целочисленные типы данных

Целочисленные типы данных используются для хранения точных числовых данных, например, возраста человека или количества единиц определенного товара. Ниже представлена таблица целочисленных типов данных с указанием названия, диапазона допустимых значений и количества занимаемой памяти (Таблица 2.1).

Таблица 2.1. Целочисленные типы данных

Название	Диапазон	Память
<code>bigint</code>	от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807	8 байт
<code>int</code>	от <code>-2 147 483 648</code> до <code>2 147 483 647</code>	4 байта
<code>smallint</code>	от <code>-32 768</code> до <code>32 767</code>	2 байта
<code>tinyint</code>	от <code>0</code> до <code>255</code>	1 байт

Еще одним целочисленным типом данных является `bit`, он занимает один байт памяти и может хранить значения 0, 1 или `NULL`, фактически этот тип данных является аналогом логического типа данных в таких языках программирования как C++ или C#.

Типы данных для хранения текста

Для хранения текстовой информации в базе данных используются четыре основных типа данных:

- **char(n)** — предназначен для хранения строк фиксированной длины не в кодировке Unicode, где вместо **n** указывается возможное количество символов в строке от 1 до 8000, память при этом выделяется по одному байту на символ;
- **varchar(n | max)** — также предназначен для хранения строк не в кодировке Unicode, но переменной длины, допустимое количество символов в строке можно задать либо указав значение от 1 до 8000, либо использовать для этого слово **max** и в этом случае для хранения строки выделится память до 2 ГБ;
- **nchar(n)** — предназначен для хранения строк фиксированной длины в кодировке Unicode, при помощи **n** задается максимально возможное количество символов в строке от 1 до 4000, размер выделенной памяти по два байта на символ;
- **nvarchar(n | max)** — тип данных, который предназначен для хранения строк переменной длины в кодировке Unicode, максимальное количество символов в строке можно задать от 1 до 4000 или выделить память для хранения строки до 2 ГБ, если использовать ключевое слово **max**.

В чем же состоит разница между строкой переменной длины и фиксированной строкой?

Если, например, для хранения строки указан тип данных **char** или **nchar** с размером 10 символов (**char(10)** или **nchar(10)**), а сохраняется строка из шести символов, то количество выделенной памяти будет соответствовать заявленному размеру — 10 или 20 байт, соответственно.

Если для хранения строки указан тип данных `varchar` или `nvarchar` с размером 10 символов (`varchar(10)` или `nvarchar(10)`), а сохраняется строка из шести символов, то размер выделенной памяти будет соответствовать реальному количеству символов — 6 или 12 байт, соответственно.

Вы также можете встретить типы данных `text` и `ntext`, но эти типы данных являются устаревшими и их использование не рекомендовано.

Вещественные типы данных

Для хранения числовых типов данных с плавающей точкой используются вещественные типы данных `float` и `real`.

При помощи типа данных `float` можно хранить числа в диапазоне от $-1,79\text{E}+308$ до $-2,23\text{E}-308$, 0 и от $2,23\text{E}-308$ до $1,79\text{E}+308$.

При использовании типа данных `float(n)`, указывая значение `n`, вы можете устанавливать точность числа и естественно изменять размер выделяемой памяти. Если значение `n` с 1 по 24, то SQL Server трактует его как 24 и выделяет память в 4 байта для хранения числа с точностью до 7 знаков. Если в `n` указывается значение с 25 по 53, то SQL Server трактует его как 53 и выделяет 8 байт памяти для хранения числа с точностью до 15 знаков. По умолчанию значение `n` равно 53.

Типа данных `real` позволяет хранить числа в диапазоне от $-3,40\text{E}+38$ до $-1,18\text{E}-38$, 0 и от $1,18\text{E}-38$ до $3,40\text{E}+38$.

Синонимом для типа данных `real` является `float(24)`.

Типы данных для хранения даты и времени

Для хранения в таблицах даты и времени SQL Server предоставляет ряд типов данных, которыми вы можете воспользоваться в зависимости от поставленной задачи.

Тип данных **datetime** позволяет хранить дату и время в 24-часовом формате с указанием долей секунды, в диапазоне с 1 января 1753 года по 31 декабря 9999 года, например, **2017-12-28 15:20:35.693**.

Тип данных **datetime2** позволяет хранить дату и время в 24-часовом формате, но имеет большую точность долей секунды по сравнению с типом **datetime** и диапазон дат с 1 января 0001 года по 31 декабря 9999 года, например, **2017-12-28 15:20:35.6930000**.

Тип данных **datetimeoffset** позволяет хранить дату и время в 24-часовом формате с указанием долей секунды с учетом часового пояса в диапазоне с 1 января 0001 года по 31 декабря 9999 года, например, **2017-12-28 15:20:35.6930000 +00:00**.

Предыдущие типы данных выводят довольно точные значения времени, но, если вам достаточно получить только дату и время, например, время покупки товара, тогда вы можете использовать тип данных **smalldatetime**. Этот тип данных позволяет хранить дату и время в 24-часовом формате с секундами, всегда равными нулю, без долей секунды в диапазоне с 1 января 1900 года по 6 июня 2079 года, например, **2017-12-28 15:20:00**.

Если поле вашей таблицы предназначено для хранения только даты, например, даты рождения человека, то вам больше подойдет тип данных **date**, который позво-

ляет хранить дату в диапазоне с 1 января 0001 года по 31 декабря 9999 года, например, `2017-12-28`.

Тип данных `time` позволяет хранить только время в 24-часовом формате с указанием долей секунды без учета часового пояса в диапазоне от `00:00:00.0000000` до `23:59:59.9999999`, например, `15:20:35.6930000`.

Типы данных с фиксированной точкой

Для хранения вещественных значений в более точном формате используются типы данных `decimal(p, s)` и `numeric(p, s)`. Эти типы данных являются взаимозаменяемыми и позволяют хранить данные в диапазоне от $-10^{38}+1$ до $10^{38}-1$.

При помощи параметра `p` задается общее количество цифр в числе, как целой, так и дробной части. Диапазон значений от 1 до 38 (по умолчанию 18), чем большее значение вы укажете, тем больше байт памяти будет выделено для хранения информации. Значение в параметре `s` определяет количество цифр дробной части числа в диапазоне от 0 до `p` (по умолчанию 0).

Тип `decimal(6,2)` мы использовали в качестве типа данных для поля `Grants` таблицы `Students` из раздела № 1. В этом случае общее количество цифр в числе равно 6, а цифр после запятой 2, то есть целая часть должна состоять из 4 цифр.

Другие типы данных

Для хранения денежных значений используются типы данных `money` и `smallmoney`, которые представляют собой вещественные числа, дробная часть которых

предназначена для хранения мелочи (копейки, центы и т.д.) (Таблица 2.2).

Таблица 2.2. Денежные типы данных

Название	Диапазон	Память
money	от -922 337 203 685 477,5808 до 922 337 203 685 477,5807	8 байт
smallmoney	от -214 748,3648 до 214 748,3647	4 байта

Для хранения различной информации в бинарном виде используются типы данных `binary(n)` и `varbinary(n | max)`. Параметр `n` может принимать значения от 1 до 8000, если указать ключевое слово `max`, то выделится память для хранения 2 ГБ информации.

Если у вас есть необходимость хранить данные в двумерной системе координат, тогда вы можете использовать тип данных `geometry`.

Для хранения координат широты и долготы объекта предназначен тип данных `geography`.

Хранение XML-документов обеспечивает тип данных `xml`.

Временный результирующий набор данных, полученных в результате запроса, сохраняется в типе данных `table`.

Обычно при создании таблицы полям указываются такие типы данных, максимальный размер которых гарантировано, позволит сохранить требуемый диапазон значений, такой подход используется в целях уменьшения размера базы данных.

3. Индекс

Что такое индекс?

Индекс — это физическая структура данных в БД, при помощи которой осуществляется ускоренный доступ к необходимой информации, использование подходящего индекса значительно улучшает производительность запроса.

Для того чтобы лучше понять, что такое индекс, представим ситуацию, когда вам необходимо найти определенную главу в любой книге. Существуют два способа как это сделать — последовательный перебор всех страниц либо получение номера нужной страницы из оглавления книги. Конечно же, используя второй способ, вы быстрее достигните требуемого результата. Так вот индексы, также как оглавление в книге, позволяют ускорить доступ к требуемой информации в базе данных, если подходящий индекс отсутствует, то поиск осуществляется последовательным перебором всех записей таблицы.

Цели и задачи индекса

К этому моменту вы можете сказать: «Давайте назначим индексы для всех полей таблицы, что может быть проще?» Однако такой подход принесет больше вреда, чем пользы, ведь индекс по сути представляет собой копию данных того поля, для которого он создан, следовательно, при создании индекса увеличивается размер базы данных. Также наличие большого количества индексов увеличивает время выполнения операций по измене-

нию данных (**INSERT**, **UPDATE**, **DELETE**), потому что изменение полей автоматически приводит к изменению индексов, в результате чего происходит фрагментация индекса. Поэтому создавать индексы необходимо с осторожностью и помнить о некоторых особенностях:

- индексы автоматически создаются для уникальных полей таблицы и полей, которые указаны в качестве первичных ключей;
- индексы следует создавать для полей таблицы, по которым часто производится поиск (можно также создавать один индекс для набора столбцов);
- для создания индекса наиболее подходят те поля таблицы, у которых количество повторяющихся значений минимально;
- индексы также можно создавать и для представлений (**VIEWS**), которые будут рассмотрены в курсе MS SQL Server.

Внутреннее устройство индекса

Прежде чем мы опишем внутреннее устройство индекса необходимо разобраться в способе хранения данных в самой базе данных.

Минимальной единицей распределения памяти в базе данных является страницы, размер каждой страницы 8 КБ. Существует два вида страниц — страницы данных и страницы индексов. Восемь страниц образуют экстент, который используется для эффективного управления страницами, размер экстента, соответственно, 64 КБ.

Для хранения индексов используется структура данных в виде сбалансированного дерева *B-дерево* (*Balanced*

Tree — B-Tree). Такая структура представляется в виде дерева, ветви которого направлены вниз, и обеспечивает автоматическую сбалансированность узлов, то есть количество ветвей справа от корневого узла приблизительно равно количеству ветвей слева. Такой способ хранения обеспечивает простой и быстрый способ выборки необходимой информации (рис. 3.1).

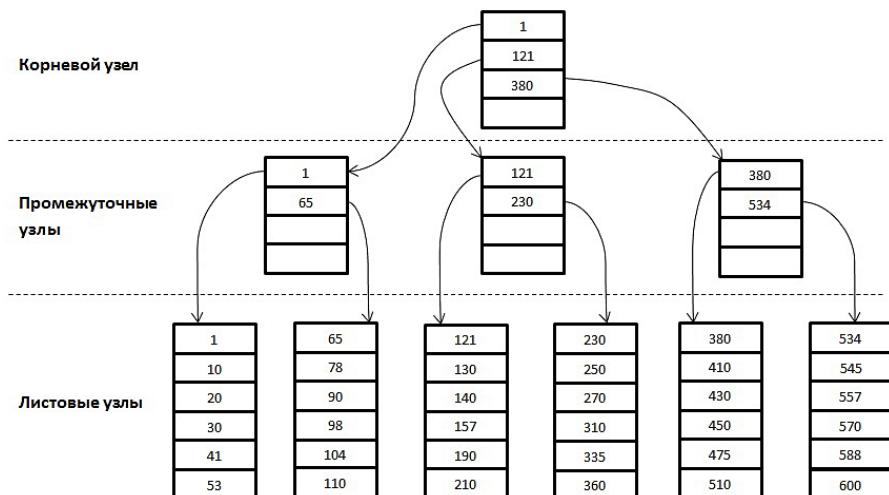


Рисунок 3.1. Пример B-дерева

Индексы бывают кластерные и некластерные.

Кластерный индекс отличается тем, что на его листовом уровне содержатся действительные данные, то есть после достижения листового уровня в таком индексе система получает требуемые данные. В таблице может быть только один кластерный индекс и создается автоматически на поле, которое указано в качестве первичного ключа.

На листовом уровне некластерного индекса находится либо так называемый идентификатор строки (Row

ID — RID), в котором содержится информация о требуемой записи в виде номера: экстента, страницы и смещении строки на странице, либо кластеризованный ключ, который содержит информацию о кластеризованном индексе. После получения этой информации осуществляется дальнейший поиск данных. Количество некластерных индексов для одной таблицы неограничено.

Использование индексов будет подробно рассмотрено вами в курсе MS SQL Server.

4. Системные базы данных и таблицы

При установке SQL Server 2016 автоматически создаются четыре системных базы данных, наличие которых необходимо для корректного функционирования SQL Server. Вы можете их увидеть, если в окне Object Explorer Management Studio развернете папку Databases, а затем подпапку System Databases (рис. 4.1).

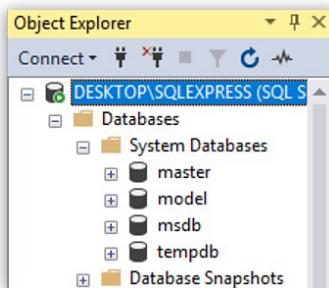


Рисунок 4.1. Системные базы данных

В этой папке находятся следующие системные базы данных:

- **master** — содержит всю информацию о самом SQL Server, а также обо всех базах данных, без этой базы данных запуск SQL Server не возможен;
- **model** — служит шаблоном при создании всех пользовательских баз данных, наличие этой базы данных обязательно;
- **msdb** — эта база данных предназначена для создания расписания заданий, например, создания резервных копий и восстановления любой базы данных;

- **tempdb** — используется для хранения различных временных объектов.

Помимо системных БД также создаются и системные таблицы, которые являются общими для всех баз данных. Системных таблиц довольно много поэтому мы рассмотрим только некоторые из них, за более подробной информацией рекомендуем обратиться к справочной документации:

- **backupfile** — содержит информацию обо всех файлах баз данных на момент создания резервных копий, находится в БД msdb;
- **restorefile** — содержит информацию по каждому восстановленному файлу БД, расположена в базе данных msdb;
- **log_shipping_primary_secondaries** — содержит информацию, которая связывает БД-источник с БД-получателем, находится в базе данных msdb;
- **cdc.lsn_time_mapping** — в этой таблице хранятся данные о всех транзакциях из таблицы изменений;
- **MSdbms** — в этой таблице находится полный список СУБД, отличных от MS SQL Server, которые поддерживают совместимую с MS SQL Server репликацию баз данных, находится в БД msdb;
- **MSreplication_options** — содержит данные, использующиеся при репликациях, расположена в базе данных master;
- **sys.sysoledbusers** — содержит данные по всем пользователям текущего сервера, находится в базе данных master.

5. Запросы

Введение в язык структурированных запросов SQL

Основным предназначением любой базы данных является накопление требуемой информации и предоставление ее при необходимости. Необходимые данные мы можем получить, запросив их у БД, то есть, написав некоторый код — запрос.

Чтобы написать соответствующий запрос нужен специализированный язык программирования. Традиционные языки программирования, существовавшие на момент появления реляционной модели данных, такие как COBOL или Fortran, на эту роль не подошли. Поэтому был разработан новый язык — **SQL** (*Structured Query Language* — язык структурированных запросов).

Язык SQL

Первая версия языка реляционных баз данных появилась в начале 70-х годов прошлого столетия и называлась SEQUEL (*Structured English Query Language* — язык структурированных запросов на основе английского). Этот язык был разработан компанией IBM как часть проекта System/R, направленного на реализацию реляционной СУБД. Через несколько лет была выпущена вторая версия — SEQUEL/2, позже язык был переименован в SQL.

Язык SQL является декларативным языком, при помощи которого вы обращаетесь к базе данных, запрашивая требуемую информацию, то есть вы указываете что вам необходимо, а задачу как это получить решает сама СУБД.

Основное предназначение языка SQL — обеспечение взаимодействия с базами данных, путем формирования запросов, состоящих из специальных операторов. Операторы — это инструкции, при помощи которых вы запрашиваете информацию из БД, указывая какие данные вас интересуют, откуда их нужно получить и, при необходимости, ограничивая полученный результат. Например, запрос может звучать следующим образом: «Покажите мне фамилии только тех студентов, которые родились в ноябре». В данном случае часть запроса «Покажите мне» указывает, что требуется предоставить, следующая часть запроса «студентов» — информирует откуда взять данные, условие «родились в ноябре» ограничивает количество записей.

В виде SQL-запроса это выглядит таким образом:

```
SELECT LastName  
FROM Students  
WHERE MONTH(BirthDate) = 11;
```

Не вдаваясь в подробности выполнения запроса, о которых вы узнаете из следующего урока, можнонести некоторую ясность. При помощи оператора **SELECT** вы прописываете **что** вам нужно, в операторе **FROM** вы указываете **откуда** необходимо взять требуемую информацию, а в операторе **WHERE** находится **условие**, которое конкретизирует конечный результат.

Стандарты языка SQL

Несмотря на то, что язык SQL был принят в качестве основного языка по работе с базами данных, существование нескольких производителей СУБД привело к появлению нескольких реализаций (диалектов) языка

SQL от разных производителей. Такое положение вещей исключало возможность переноса программного обеспечения с одной СУБД на другую, поэтому стал вопрос о стандартизации языка SQL.

В 1986 году Американский национальный институт стандартов (ANSI) разработал первый стандарт языка SQL, который был утвержден Международной организацией стандартов (ISO) в 1987 году, получивший название SQL-86. После внесения изменения в оригинальный стандарт в 1989 году вышел следующий стандарт языка — SQL-89.

Последующие расширения языка SQL привели к появлению нескольких стандартов в 1992, 1999, 2003, 2006, 2008, 2011 и 2016 годах. В настоящее время действует, соответственно стандарт SQL-2016.

Диалекты языка SQL

Несмотря на наличие большого количества стандартов в настоящее время не существует единого диалекта языка SQL. Введение производителями СУБД новых функциональных средств вносит в существующие диалекты новые изменения, тем самым диалекты все больше отличаются друг от друга.

Приведем список диалектов языка SQL в СУБД наиболее известных производителей:

- **T-SQL (Transact-SQL)** — этот диалект используется в СУБД Microsoft SQL Server и Sybase ASE;
- **PL/SQL (Procedural Language/SQL)** — данный диалект используется в СУБД Oracle;
- **SQL/PSM (SQL/Persistent Stored Module)** — этот диалект используется в СУБД MySQL;

- **PLpg/SQL** (*Procedural Language/postgreSQL*) — диалект реализован в СУБД PostgreSQL;
- **SQLPL** (*SQL Procedural Language*) — этот диалект реализован в СУБД DB2;
- **Jet SQL** — данный диалект реализован в СУБД Microsoft Access.

Диалект Transact-SQL

Transact-SQL — реализация языка SQL, разработанная корпорацией Microsoft для СУБД Microsoft SQL Server. Знакомиться с этим диалектом вы будете на протяжении текущего и последующего курсов, а пока приведем его краткое описание.

T-SQL позволяет использовать различные операторы: арифметические (+, -, *, /, %), логические (AND, OR, NOT), сравнения (=, >, <, >=, <=, <>) и операторы для работы со множествами (IN). При использовании T-SQL существует возможность создавать переменные при помощи команды **DECLARE**, используя для этого специальные символы — идентификаторы (@). В T-SQL содержится условный оператор (IF) и цикл (WHILE). При написании запросов существует возможность вызова встроенных функций (COUNT, SUM, MIN, MAX, DATEDIFF, ABS и т.д.). Для комментирования кода в T-SQL используется либо строчный (--), либо блочный /* */ комментарий.

Понятия DDL, DML, DCL

SQL-операторы делятся на три категории: операторы **DDL** (*Data Definition Language* — язык описания данных), **DML** (*Data Manipulation Language* — язык управления

данными) и **DCL** (*Data Control Language* — язык управления доступом к данным). Рассмотрим эти операторы более подробно.

Операторы DDL позволяют работать со структурой данных в БД:

- создание объекта (**CREATE**);
- изменение объекта (**ALTER**);
- удаление объекта (**DROP**).

Операторы DML используются при работе с данными в БД:

- запрос определенной информации (**SELECT**);
- вставка необходимых данных в таблицу (**INSERT**);
- обновление существующих данных в таблицы (**UPDATE**);
- удаление данных из таблицы (**DELETE**).

Операторы DCL позволяют управлять доступом к базе данных:

- предоставление доступа пользователя для работы с объектом (**GRANT**);
- запрет на доступ пользователя к объекту (**DENY**);
- отмена привилегий доступа пользователя к объекту (**REVOKE**).

Домашнее задание

3. Создайте базу данных, которая содержит таблицу с информацией о книгах. Названия необходимых полей продумайте самостоятельно. Используйте все знания, полученные вами к этому моменту.

Unit 3.
Запросы
SELECT, INSERT,
UPDATE, DELETE

1. Оператор SELECT

На прошлом занятии мы продемонстрировали вам пример SQL-запроса, который вернул фамилии студентов, родившихся в ноябре. На текущем занятии вы более подробно узнаете, какие операторы используются при написании запросов, и начнем мы с оператора **SELECT**.

Предложение SELECT

Предложение **SELECT** позволяет считывать необходимую информацию из базы данных. В самом простом варианте использования после инструкции **SELECT** указываются имена столбцов, значения которых требуется получить из одной или нескольких таблиц, названия этих столбцов разделяются между собой запятыми. Также после оператора **SELECT** можно указывать вычисляемые результаты и функции агрегирования, о которых вы узнаете в одном из последующих уроков.

Написание SQL-запроса с применением только одного оператора **SELECT** приведет к синтаксической ошибке, потому что необходимо использовать его в сочетании еще как минимум с одним оператором — **FROM**.

Предложение FROM

Предложение **FROM** используется для указания источника получения данных, после него прописывается название таблицы или список таблиц, разделенных между собой запятыми. Общая форма записи SQL-запроса

1. Оператор SELECT

с использованием операторов **SELECT** и **FROM** выглядит следующим образом:

```
SELECT columnName1, columnName2, ...
FROM tableName;
```

Продемонстрируем простой запрос, позволяющий получить всю информацию из таблицы **Students** базы данных **University**, которая была создана нами на предыдущем уроке. Для этого откроем MS SQL Server Management Studio 17 и на панели инструментов нажмем кнопку **New Query** (рис. 1.1) или сочетание клавиш **Ctrl+N**, после чего в выпадающем списке необходимо выбрать требуемую БД (рис. 1.1).

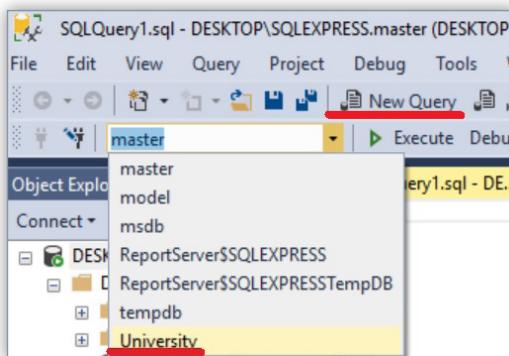


Рисунок 1.1. Создание нового запроса

В появившемся окне напишем SQL-запрос:

```
SELECT LastName, FirstName, BirthDate,
Grants, Email
FROM Students;
```

При помощи этого запроса мы хотим получить значения столбцов **Last Name**, **First Name**, **Birth Date**, **Grants** и **Email** для всех записей из таблицы **Students**.

Для того чтобы, написанный нами, запрос выполнился необходимо нажать кнопку Execute или клавишу F5 на клавиатуре (рис. 1.2).

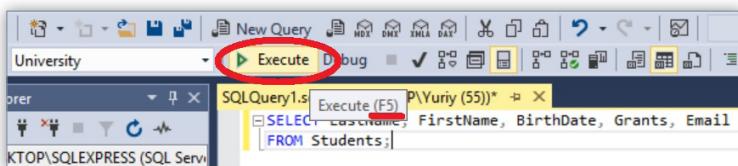


Рисунок 1.2. Запуск запроса на выполнение

В результате SQL-запроса будет создана виртуальная таблица, в которой содержится вся запрошенная информация из физической таблицы `Students` базы данных `University` (рис. 1.3).

	LastName	FirstName	BirthDate	Grants	Email
1	Doe	John	1998-02-11	NULL	jh@net.eu
2	Jones	Jack	1997-11-05	1256.00	jj@net.eu
3	Johnson	Joshua	1997-05-23	NULL	jo@net.eu

Рисунок 1.3. Получение всех записей из таблицы `Students`

В языке SQL существует возможность написать запрос, в котором вместо списка столбцов таблицы можно указать символ *:

```
SELECT *
FROM Students;
```

И хотя результат мы получим такой же (рис. 1.3) данный SQL-запрос будет выполняться дольше. Это происходит, потому что требуемые столбцы не указаны явным образом, и СУБД приходиться тратить время на определение и получение всех столбцов в таблице. Разница

по времени при небольшом количестве записей будет не существенной, однако при больших объемах данных будет весьма заметна, поэтому настоятельно рекомендуется в SQL-запросе указывать все столбцы, информацию по которым необходимо получить.

При написании SQL-запросов можно формировать новые столбцы в виртуальной таблице, соединяя значения нескольких реальных столбцов при помощи символа `+`, в этом случае заголовок такого столбца будет **(No column name)**, если такое положение вещей вас не устраивает, то вы можете задать псевдоним (*alias*) этому столбцу при помощи оператора `AS`. Например, соединив в SQL-запросе имя и фамилию студента через пробел, укажем этому столбцу псевдоним `FullName`:

```
SELECT FirstName +' '+ LastName AS FullName,
BirthDate, Grants, Email
FROM Students;
```

Результат запроса представлен на рисунке 1.4.

	FullName	BirthDate	Grants	Email
1	John Doe	1998-02-11	NULL	jh@net.eu
2	Jack Jones	1997-11-05	1256.00	jj@net.eu
3	Joshua Johnson	1997-05-23	NULL	jo@net.eu

Рисунок 1.4. Использование псевдонима при формировании нового столбца

При написании SQL-запроса вы можете изменять данные в виртуальной таблице, применяя различные арифметические действия, и это никак не скажется на реальных данных. Допустим «ректор нашего университета» захотел узнать: «Как изменяются стипендии студентов, если их

увеличить на 20 процентов?». Для того чтобы ответить на этот вопрос мы можем написать следующий запрос:

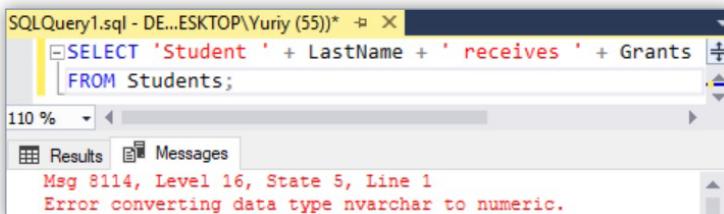
```
SELECT FirstName + ' ' + LastName AS FullName,
Grants*1.2 AS [Plus 20 percent]
FROM Students;
```

В предыдущем запросе продемонстрирована одна особенность при написании имен столбцов: если в названии столбца присутствуют пробелы, то это название необходимо брать в квадратные скобки. Результат этого SQL-запроса представлен на рисунке 1.5.

	FullName	Plus 20 percent
1	John Doe	NULL
2	Jack Jones	1507.200
3	Joshua Johnson	NULL

Рисунок 1.5. Использование арифметических операций

Оператор **SELECT** позволяет возвращать информацию в виде строки, которая формируется путем конкатенации (сцепления) определенной текстовой информации и значений из различных столбцов. Однако при этом указанные столбцы должны хранить текстовые данные, в противном случае при выполнении SQL-запроса мы получим ошибку (рис. 1.6).



The screenshot shows a SQL query window titled 'SQLQuery1.sql - DE...ESKTOP\Yuriy (55)*'. The query is:

```
SELECT 'Student ' + LastName + ' receives ' + Grants
FROM Students;
```

The results pane shows an error message:

```
Msg 8114, Level 16, State 5, Line 1
Error converting data type nvarchar to numeric.
```

Рисунок 1.6. Ошибка: недопустимое приведение типов данных

Данная ошибка произошла из-за невозможности преобразовать вещественные значения столбца **Grant** к строковому типу данных. Чтобы решить эту проблему в языке T-SQL существуют две стандартные функции **CAST()** и **CONVERT()**, которые позволяют осуществлять приведение данных к требуемому типу. Эти функции отличаются принимаемыми параметрами, но сравнив следующие запросы, вы разберетесь, как их использовать.

```
SELECT 'Student ' + LastName + ' receives ' +
CAST(Grants AS nvarchar(10))
FROM Students;
SELECT 'Student ' + LastName + ' receives ' +
CONVERT(nvarchar(10), Grants)
FROM Students;
```

В результате выполнения двух этих запросов мы получим одинаковые результаты (рис. 1.7).

(No column name)	
1	NULL
2	Student Jones receives 1256.00
3	NULL

Рисунок 1.7. Формирование SQL-запроса в виде строки

Наличие **NULL**-значений объясняется, тем, что у некоторых студентов стипендия имеет неопределенное значение, как убрать такие записи из результирующей таблицы вы узнаете в следующем подразделе.

В языке T-SQL совместно с предложением **SELECT** можно использовать оператор **TOP**, который позволяет получить первые записи либо в количественном, либо процентном отношении.

Допустим нам надо получить первые две записи в таблице **Students**, тогда запрос будет выглядеть следующим образом:

```
SELECT TOP 2 LastName, FirstName, BirthDate
FROM Students;
```

Результат SQL-запроса представлен на рисунке 1.8.

	LastName	FirstName	BirthDate
1	Doe	John	1998-02-11
2	Jones	Jack	1997-11-05

Рисунок 1.8. Использование оператора TOP

Если нам понадобится получить, например, 20 % первых записей из таблицы **Students**, тогда после числового значения необходимо добавить ключевое слово **PERCENT**, что демонстрируется в следующем SQL-запросе (результат на рисунке 1.9).

```
SELECT TOP 20 PERCENT LastName, FirstName,
BirthDate
FROM Students;
```

	LastName	FirstName	BirthDate
1	Doe	John	1998-02-11

Рисунок 1.9. Использование оператора TOP

Если в таблице вашей базы данных определенное поле содержит большое количество повторяющихся значений, а ваша задача избавиться от них в результирующей таблице, то вы можете использовать для этого оператор **DISTINCT**, который следует написать перед именем требуемого столбца, и тем самым вы исключите из результата все повторения. Например, нам надо определить какие

имена у студентов в «нашем учебном заведении», при этом следует исключить повторения:

```
SELECT DISTINCT FirstName
FROM Students;
```

Результат SQL-запроса представлен на рисунке 1.10.

	FirstName
1	Jack
2	John
3	Joshua

Рисунок 1.10. Использование оператора DISTINCT

Предложение WHERE

В тех случаях, когда необходимо отфильтровать получаемые данные, задав определенные условия, в SQL-запросе предусмотрено использование оператора **WHERE**, после которого прописываются требуемые условия. Общая форма записи запроса с использованием оператора **WHERE** выглядит следующим образом:

```
SELECT columnName1, columnName2, ...
FROM tableName
WHERE condition;
```

Для того чтобы в SQL-запросе указать необходимые ограничения, в языке T-SQL используются следующие операторы сравнения:

- = — равно;
- <> — не равно;
- > — больше;
- >= — больше или равно;

- $>$ — не больше чем;
- $<$ — меньше;
- \leq — меньше или равно;
- \neq — не меньше чем.

Например, необходимо вывести стипендию и фамилии всех студентов, у которых размер стипендии превышает 1000. SQL-запрос в этом случае будет выглядеть образом:

```
SELECT LastName, Grants
FROM Students
WHERE Grants > 1000;
```

В результате выполнения этого SQL-запроса вы получите следующую таблицу (рис. 1.11).

	Last Name	Grants
1	Jones	1256.00

Рисунок 1.11. Использование операторов сравнения

Следующий SQL-запрос позволяет получить информацию о студентах, в имени которых не больше чем четыре символа:

```
SELECT Id, LastName, FirstName, BirthDate, Grants, Email
FROM Students
WHERE LEN(FirstName) !> 4;
```

Для того чтобы определить количество символов (длину строки) в имени студента, мы используем функцию **LEN()**, которая принимает строковые данные. Эта функция является встроенной в MS SQL Server и не входит в стандарт языка T-SQL, то есть ее выполнение обеспечивает сама СУБД. В результате выполнения этого SQL-запроса вы получите следующую таблицу (рис. 1.12).

1. Оператор SELECT

	Id	LastName	FirstName	BirthDate	Grants	Email
1	1	Doe	John	1998-02-11	NULL	jh@net.eu
2	2	Jones	Jack	1997-11-05	1256.00	jj@net.eu

Рисунок 1.12. Использование операторов сравнения

Для того чтобы в операторе **WHERE** задать несколько условий, необходимо использовать операторы объединения **AND** и **OR**.

Оператор **AND** объединяет два условия и возвращает истину, если они оба верны. Например, требуется получить информацию о студентах, которые родились осенью, такой SQL-запрос может выглядеть следующим образом:

```
SELECT LastName, FirstName, BirthDate  
FROM Students  
WHERE MONTH(BirthDate) >= 9 AND MONTH(BirthDate) <= 11;
```

Для того чтобы определить месяц рождения студентов мы воспользовались стандартной функцией языка T-SQL — **MONTH()**, которая позволяет получить номер месяца из переданной даты. Для того чтобы получить год и день, как часть даты, вы можете воспользоваться стандартными функциями **YEAR()** и **DAY()** соответственно.

В результате выполнения предыдущего SQL-запроса вы получите следующую таблицу (рис. 1.13).

	LastName	FirstName	BirthDate
1	Jones	Jack	1997-11-05

Рисунок 1.13. Использование оператора AND

Оператор **OR** объединяет два условия и возвращает истину, если хотя бы одно из условий верно. Например,

необходимо отобразить информацию о студентах, у которых либо четный год рождения, либо нечетный день рождения. Для того чтобы получить требуемый результат необходимо осуществить проверку соответствующих значений на кратность числу два, используя операцию деление по модулю (%).

```
SELECT LastName, FirstName, BirthDate
FROM Students
WHERE YEAR(BirthDate) % 2 = 0 OR DAY(BirthDate) % 2 <> 0;
```

В результате выполнения предыдущего SQL-запроса вы получите следующую таблицу (рис. 1.14).

	LastName	FirstName	BirthDate
1	Doe	John	1998-02-11
2	Jones	Jack	1997-11-05
3	Johnson	Joshua	1997-05-23

Рисунок 1.14. Использование оператора OR

В тех случаях, когда вам требуется проверить значения столбцов на неопределенность (значение `NULL`), использование операторов сравнения (`=`, `<>`, `>`, `<`, и т.д.) невозможно. Вместо них вам необходимо использовать оператор `IS NULL`. Например, для того чтобы получить список всех студентов, которые не получают стипендию, необходимо написать следующий запрос к базе данных:

```
SELECT LastName, FirstName, Grants
FROM Students
WHERE Grants IS NULL;
```

Результат SQL-запроса представлен на рисунке 1.15.

1. Оператор SELECT

	Last Name	First Name	Grants
1	Doe	John	NULL
2	Johnson	Joshua	NULL

Рисунок 1.15. Использование оператора IS NULL

Существует еще один оператор — **NOT**, который используется, когда необходимо задать противоположное условие, допустим, требуется получить информацию о студентах, фамилии которых указывались при заполнении таблицы **Students**, то есть не равны **Doe**.

```
SELECT Id, LastName, FirstName, BirthDate, Grants, Email  
FROM Students  
WHERE NOT LastName = 'Doe';
```

Результат SQL-запроса представлен на рисунке 1.16.

	Id	Last Name	First Name	Birth Date	Grants	Email
1	2	Jones	Jack	1997-11-05	1256.00	jj@net.eu
2	7	Johnson	Joshua	1997-05-23	NULL	jo@net.eu

Рисунок 1.16. Использование оператора NOT

Синтаксис использования оператора **NOT** с **NULL**-значениями несколько отличается. Допустим, нам необходима информация обо всех студентах, которые получают стипендию, в этом случае SQL-запрос будет выглядеть следующим образом:

```
SELECT LastName, FirstName, Grants  
FROM Students  
WHERE Grants IS NOT NULL;
```

Результаты запроса будут, естественно, диаметрально противоположны запросу показанному на рисунке 1.15 (рис. 1.17).

	Last Name	First Name	Grants
1	Jones	Jack	1256.00

Рисунок 1.17. Использование оператора NOT с NULL-значениями

Предложение ORDER BY

При написании некоторых SQL-запросов существует необходимость упорядочить полученную информацию, тогда при написании запросов следует использовать предложение **ORDER BY**. Запись запроса с использованием оператора **ORDER BY** выглядит следующим образом:

```
SELECT columnName1, columnName2, ...
FROM tableName
ORDER BY columnName1 ASC | DESC, ...;
```

При помощи оператора **ORDER BY** можно указать количество столбцов, по которым производится сортировка и направление этой сортировки. Чтобы полученные данные были отсортированы по убыванию необходимо указать ключевое слово **DESC** (от *descending*) после соответствующего поля. Сортировка по возрастанию является сортировкой по умолчанию и ее можно не указывать, но если вы хотите сделать это явным образом, то следует воспользоваться ключевым словом **ASC** (от *ascending*).

В качестве примера использования оператора **ORDER BY** приведем SQL-запрос, возвращающий список студентов, отсортированный по дате рождения (рис. 1.18):

```
SELECT LastName, FirstName, BirthDate
FROM Students
ORDER BY BirthDate;
```

1. Оператор SELECT

	LastName	FirstName	BirthDate
1	Johnson	Joshua	1997-05-23
2	Jones	Jack	1997-11-05
3	Doe	John	1998-02-11

Рисунок 1.18. Использование сортировки в SQL-запросе

При написании SQL-запроса существует возможность указать в операторе **ORDER BY** несколько полей с различным направлением сортировки. Допустим, полученный в результате выполнения запроса, список студентов надо отсортировать как по убыванию (столбец **LastName**), так и по возрастанию (столбец **FirstName**), для этого необходимо написать следующий SQL-запрос:

```
SELECT LastName, FirstName, BirthDate  
FROM Students  
ORDER BY LastName DESC, FirstName ASC;
```

Сортировка данных будет осуществляться следующим образом: все результаты будут отсортированы по фамилии и только в том случае, когда значение столбца **LastName** совпадет более чем в одной результирующей строке, запустится сортировка по имени студента (рис. 1.19).

	LastName	FirstName	BirthDate
1	Jones	Jack	1997-11-05
2	Johnson	Joshua	1997-05-23
3	Doe	John	1998-02-11

Рисунок 1.19. Использование сортировки по нескольким столбцам

2. Ключевые слова IN, BETWEEN, LIKE

При написании различных SQL-запросов существует потребность в указании сложного условия фильтрации полученных результатов с использованием нескольких операторов **OR**. Например, необходимо получить информацию о студентах, которых зовут **John**, **David** или **Jack**:

```
SELECT LastName, FirstName, BirthDate, Grants, Email
FROM Students
WHERE FirstName='John' OR FirstName='David'
OR FirstName='Jack';
```

Данный SQL-запрос вернет правильный результат (рис. 2.1), но в нем довольно часто используется оператор **OR**.

	LastName	FirstName	BirthDate	Grants	Email
1	Doe	John	1998-02-11	NULL	jh@net.eu
2	Jones	Jack	1997-11-05	1256.00	jj@net.eu

Рисунок 2.1. Запрос, использующий
несколько операторов **OR**

Для того чтобы улучшить читабельность подобных запросов можно использовать ключевое слово **IN**, которое обеспечивает сравнение значения указанного столбца с любым значением из указанного множества. Переписав предыдущий запрос, мы получим тот же результат (рис. 2.1).

```
SELECT LastName, FirstName, BirthDate, Grants, Email
FROM Students
WHERE FirstName IN ('John', 'David', 'Jack');
```

При написании SQL-запросов, в которых требуется проверка на попадание значения определенного столбца в некий диапазон можно использовать оператор **AND**, а можно, в качестве альтернативы, применить ключевое слово **BETWEEN**. Ключевое слово **BETWEEN** прописывается после имени столбца, а после него необходимо указать нижнюю и верхнюю границу диапазона, разделенных оператором **AND**, границы диапазона также учитываются.

Допустим, необходимо определить студентов, которые родились в 1997 году. Следующие два запроса вернут одинаковый результат (рис. 2.2).

SQL-запрос, использующий оператор **AND**:

```
SELECT LastName, FirstName, BirthDate
FROM Students
WHERE BirthDate >= '1997-01-01'
AND BirthDate <= '1997-12-31';
```

SQL-запрос с использованием ключевого слова **BETWEEN**:

```
SELECT LastName, FirstName, BirthDate
FROM Students
WHERE BirthDate BETWEEN '1997-01-01'
AND '1997-12-31';
```

	Last Name	First Name	Birth Date
1	Jones	Jack	1997-11-05
2	Johnson	Joshua	1997-05-23

Рисунок 2.2. Результат попадания информации в указанный диапазон

Следует обратить ваше внимание на синтаксис записи даты рождения: дата указывается в последовательно-

сти год-месяц-день и берется в одинарные кавычки, как строковые данные. Такое представление даты определено драйвером ODBC (*Open Database Connectivity*), который обеспечивает взаимодействие приложений, написанных на различных языках программирования, с данными SQL Server.

Ключевое слово **BETWEEN** можно применять в SQL-запросах, требующих сравнения строковых значений, при этом сравнение будет осуществляться в соответствии с кодами символов. Также с ключевым словом **BETWEEN** можно использовать оператор **NOT**, например, необходимо вывести всех студентов, фамилии которых не начинаются на буквы **E**, **F** и **G**.

```
SELECT LastName, FirstName  
FROM Students  
WHERE LastName NOT BETWEEN 'E' AND 'G';
```

Результат, полученный после выполнения этого SQL-запроса, отображен на рисунке 2.3.

	Last Name	First Name
1	Doe	John
2	Jones	Jack
3	Johnson	Joshua

Рисунок 2.3. Использование ключевого слова **BETWEEN** с оператором **NOT**

В тех случаях, когда необходимо осуществить поиск по текстовым полям таблиц, существует возможность использовать шаблон, который можно задать, применив ключевое слово **LIKE**. Для того чтобы сформировать ша-

блон необходимо указывать определенные служебные символы или комбинации этих символов:

- % — соответствует любой последовательности символов от 0 и более;
- _ — представляет любой одиночный символ;
- [] — задает последовательность или диапазон возможных символов;
- [^] — задает последовательность или диапазон символов, которые должны отсутствовать.

Продемонстрируем использование ключевого слова **LIKE** на следующих примерах. При помощи первого SQL-запроса мы хотим получить всех студентов, фамилия которых начинается на букву **J** или **M**, а имя заканчивается на букву **k**:

```
SELECT LastName, FirstName, BirthDate, Grants, Email
FROM Students
WHERE LastName LIKE '[JM]%' AND FirstName LIKE '%k';
```

Результат выполнения данного SQL-запроса вы можете увидеть на рисунке 2.4.

	LastName	FirstName	BirthDate	Grants	Email
1	Jones	Jack	1997-11-05	1256.00	jj@net.eu

Рисунок 2.4. Использование ключевого слова **LIKE**

Следующий запрос вернет информацию о студентах, у которых в адресе электронной почты вторая буква не совпадает ни с одной буквой из диапазона от **f** до **m**:

```
SELECT LastName, FirstName, BirthDate, Grants, Email
FROM Students
WHERE Email LIKE '_[^f-m]%';
```

Результат выполнения предыдущего SQL-запроса (рис. 2.5).

	LastName	FirstName	BirthDate	Grants	Email
1	Johnson	Joshua	1997-05-23	NULL	jo@net.eu

Рисунок 2.5. Использование ключевого слова LIKE

3. Оператор INSERT

Во втором уроке, при создании таблицы мы заполняли ее значениями, используя средства MS SQL Server Management Studio. Однако довольно часто будет возникать необходимость заполнения таблиц программным путем и в этом вам поможет оператор **INSERT**.

Оператор **INSERT** может использоваться двумя способами. В первом случае необходимо указывать имена тех столбцов таблицы, в которые вы будете записывать соответствующие значения, указанные после ключевого слова **VALUES**, общая форма записи выглядит следующим образом:

```
INSERT INTO tableName (columnName1,  
                      columnName2, ...)  
VALUES (value1, value2, ...);
```

Если использовать такую форму оператора **INSERT**, то вам необходимо указывать, как минимум, все столбцы таблицы, которые не могут принимать неопределенное (**NULL**) значение. В противном случае при попытке вставить данные Management Studio выдаст ошибку (рис. 3.1).

При написании такого вида запросов не обязательно соблюдать правильный порядок следования имен столбцов в таблице. В следующем SQL-запросе мы укажем имена требуемых столбцов в произвольной последовательности, и не будем указывать столбцы, которые могут хранить неопределенные значения (рис. 3.2).

SQLQuery1.sql - DE..ESKTOP\Yuriy (55)*

```
INSERT INTO Students(LastName, FirstName)
VALUES ('Wilson', 'Lily');
```

110 %

Messages

Msg 515, Level 16, State 2, Line 1
Cannot insert the value NULL into column 'BirthDate',
table 'University.dbo.Students'; column does not allow nulls.
INSERT fails.
The statement has been terminated.

Рисунок 3.1. Ошибка: невозможно вставить NULL-значение в столбец BirthDate

SQLQuery1.sql - DE..ESKTOP\Yuriy (55)*

```
INSERT INTO Students(BirthDate, FirstName, LastName)
VALUES ('1998-05-25', 'Lily', 'Wilson');
```

110 %

Messages

(1 row affected)

Рисунок 3.2. Использование оператора INSERT

Хотя вы видите уведомление MS SQL о корректном выполнении запроса (**1 row affected**), убедимся в правильности добавления данных самостоятельно, выполнив следующий запрос, результат на рисунке 3.3.

```
SELECT LastName, FirstName, BirthDate, Grants, Email
FROM Students;
```

	LastName	FirstName	BirthDate	Grants	Email
1	Doe	John	1998-02-11	NULL	jh@net.eu
2	Jones	Jack	1997-11-05	1256.00	jj@net.eu
3	Johnson	Joshua	1997-05-23	NULL	jo@net.eu
4	Wilson	Lily	1998-05-25	NULL	NULL

Рисунок 3.3. Проверка правильности добавления данных

При использовании оператора **INSERT** вторым способом названия столбцов таблицы не указываются, общая форма записи выглядит следующим образом:

```
INSERT INTO tableName  
VALUES (value1, value2, ...);
```

Если вы будете использовать такую форму оператора **INSERT**, то вам необходимо указывать значения для всех столбцов, которые не могут принимать неопределенное значение, обязательно соблюдая порядок их следования в таблице.

Допустим нам требуется добавить в таблицу студенку, которая не получает стипендию, но адрес ее электронной почты нам известен. В этом случае нам необходимо явно указать **NULL**-значение для стипендии, иначе в таблицу будут записаны неправильные данные:

```
INSERT INTO Students  
VALUES ('Evans', 'Grace', '1998-08-30',  
NULL, 'eg@net.eu');
```

Убедиться в правильности добавления записи вы можете самостоятельно, выполнив уже знакомый SQL-запрос (см. выше).

В языке T-SQL существует возможность заполнения существующей таблицы значениями, полученными в результате выполнения SQL-запроса. Для этих целей используется оператор **INSERT INTO SELECT**, применение которого осуществляется двумя способами.

В первом случае мы копируем все данные из таблицы, являющейся источником данных, и выглядит это следующим образом:

```
INSERT INTO destinationTable  
SELECT * FROM sourceTable  
WHERE condition;
```

Вторая форма записи предусматривает явное указание, как названий столбцов таблицы-источника, подлежащих копированию, так и названий столбцов таблицы-приемника данных:

```
INSERT INTO destinationTable (columnName1, columnName2, ...)  
SELECT columnName1, columnName2, ...  
FROM sourceTable  
WHERE condition;
```

Соответствующие столбцы этих таблиц не обязательно должны иметь одинаковые названия, необходимым требованием при использовании оператора **INSERT INTO SELECT** является соответствие типов данных в соответствующих столбцах исходной и целевой таблиц.

Для того чтобы проверить как это работает нам необходимо, для начала, создать в нашей базе данных **University** временную таблицу, которую мы назовем, например, **Temp**. Создать такую таблицу вы можете самостоятельно, освежив свои знания из урока №2. Название полей и их количество в таблице **Temp** не обязательно должно повторять все поля существующей таблицы **Students**, ограничимся только несколькими из них (рис. 3.4).

После того как мы создали временную таблицу **Temp**, заполним ее только теми сведениями о студентах (из таблицы **Students**), которые родились во второй половине года, требуемый SQL-запрос будет выглядеть следующим образом (рис. 3.5).

3. Оператор INSERT

	Column Name	Data Type	Allow Nulls
1	Id	int	<input type="checkbox"/>
	LName	nvarchar(50)	<input checked="" type="checkbox"/>
	FName	nvarchar(50)	<input checked="" type="checkbox"/>
▶	BirthDate	nchar(10)	<input checked="" type="checkbox"/>

Рисунок 3.4. Создание временной таблицы

The screenshot shows the SQL Query window of SSMS. The query is:

```
INSERT INTO Temp (LName, FName, BirthDate)
SELECT LastName, FirstName, BirthDate
FROM Students
WHERE MONTH(BirthDate) > 6;
```

The results pane shows: (2 rows affected).

Рисунок 3.5. Заполнение временной таблицы требуемыми данными

Выполнив следующий запрос к таблице Temp, мы убедимся в правильности добавления информации (рис. 3.6).

```
SELECT LName, FName, BirthDate
FROM Temp;
```

	LName	FName	BirthDate
1	Jones	Jack	1997-11-05
2	Evans	Grace	1998-08-30

Рисунок 3.6. Проверка правильности добавления данных

СУБД MS SQL Server предоставляет возможность создания новой таблицы на основе существующей, путем копирования значений всех указанных столбцов с ис-

пользованием оператора **SELECT INTO**, общая форма записи выглядит следующим образом:

```
SELECT columnName1, columnName2, ...
INTO newTable
FROM existingTable
WHERE condition;
```

Создадим в нашей базе данных **University** новую таблицу с информацией о студентах, электронный адрес которых известен, выполнив следующий запрос (рис. 3.7).

```
SQLQuery1.sql - DE...ESKTOP\Yuriy (57)*
SELECT LastName, FirstName, BirthDate, Email
INTO Temp1
FROM Students
WHERE Email IS NOT NULL;
110 %
Messages
(4 rows affected)
```

Рисунок 3.7. Создание новой таблицы на основе существующей

Для того чтобы проверить полученный результат, в следующем SQL-запросе мы специально не будем перечислять названия столбцов, результат на рисунке 3.8.

```
SELECT *
FROM Temp1;
```

	Last Name	First Name	Birth Date	Email
1	Doe	John	1998-02-11	jh@net.eu
2	Jones	Jack	1997-11-05	jj@net.eu
3	Johnson	Joshua	1997-05-23	jo@net.eu
4	Evans	Grace	1998-08-30	eg@net.eu

Рисунок 3.8. Проверка информации, записанной в новую таблицу

После того как мы продемонстрировали использование операторов **INSERT INTO SELECT** и **SELECT INTO**, удалим ненужные нам таблицы **Temp** и **Temp1** из базы данных **University**. Для этого необходимо нажать правой клавишей мыши на имени соответствующей таблицы в списке **Tables** базы данных **University** окна Object Explorer и в контекстном меню выбрать пункт **Delete** (рис. 3.9)

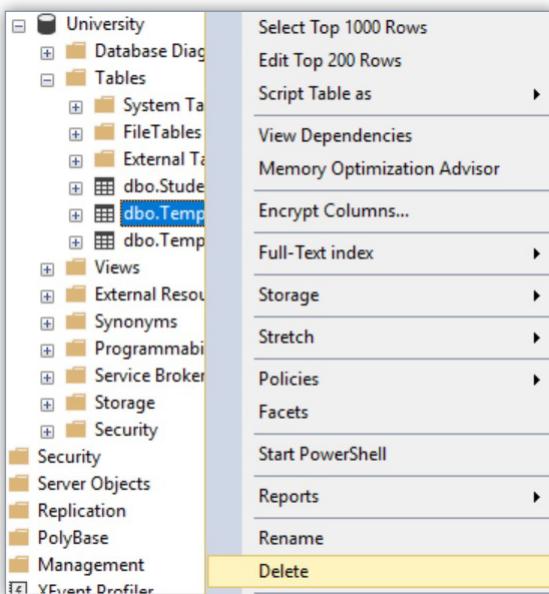


Рисунок 3.9. Удаление таблицы из базы данных (начало)

После этого появится окно Delete Object, в котором после нажатия кнопки **OK** вы увидите индикатор процесса удаления таблицы (рис. 3.10).

После того как окно закроется, таблица исчезнет из списка **Tables** вашей базы данных.

Unit 3. Запросы SELECT, INSERT, UPDATE, DELETE

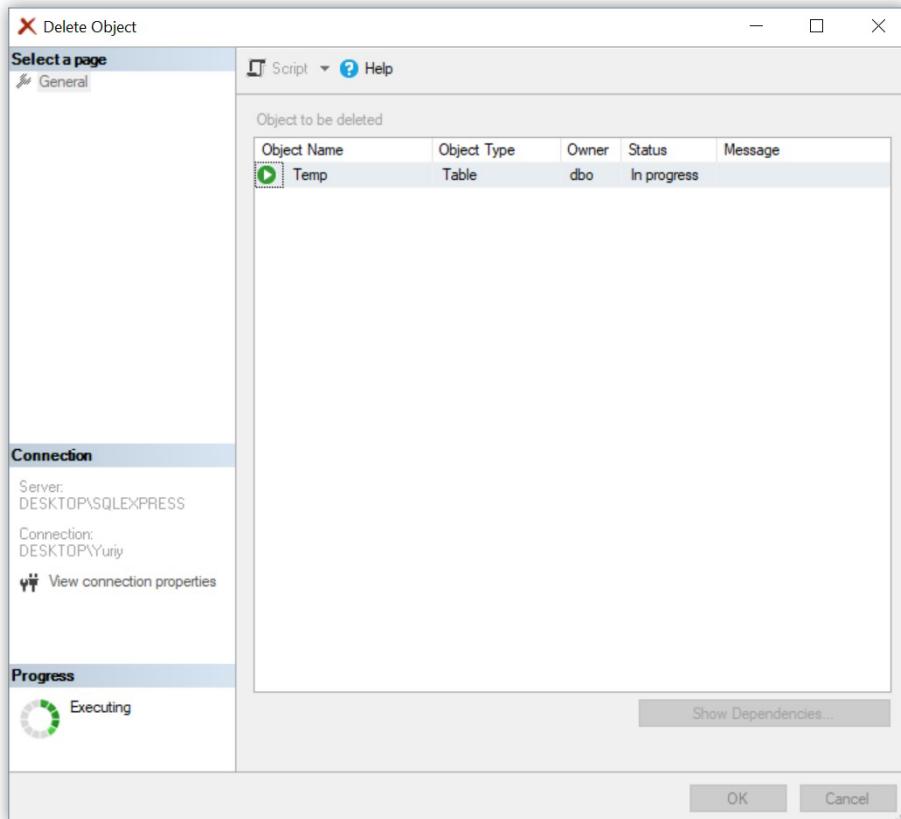


Рисунок 3.10. Удаление таблицы из базы данных (завершение)

4. Оператор UPDATE

Для того чтобы изменить любую запись при помощи SQL-запроса используется оператор **UPDATE**, общая форма записи которого выглядит следующим образом:

```
UPDATE tableName  
SET columnName1 = value1, columnName2 = value2, .  
WHERE condition;
```

Использование оператора **WHERE** можно считать обязательным, ведь если вы не укажете условие, то изменятся значения в соответствующем столбце по всей таблице.

Смоделируем определенную фантастическую ситуацию: «ректор нашего университета» решил выплачивать минимальную стипендию всем студентам независимо от их успеваемости, так сказать, на покупку тетрадок, естественно нельзя оставить без внимания студентов, которые хорошо учатся — им надо повысить стипендию. Такая инициатива должна привести к изменению информации в базе данных, то есть нам необходимо выполнить запросы с использованием оператора **UPDATE**, которые будут выглядеть следующим образом (рис. 4.1).

В данном примере очень важен порядок выполнения SQL-запросов, если вначале установим стипендию отстающим студентам, тогда после увеличения стипендии всем студентам, которые учатся хорошо, у плохо успевающих студентов стипендия увеличится в два раза.

The screenshot shows a SQL query window titled "SQLQuery1.sql - DE...ESKTOP\Yuriy (56)*". It contains the following SQL code:

```
-- increase of the scholarship  
UPDATE Students SET Grants += 500 WHERE Grants IS NOT NULL;  
  
-- appointment of a scholarship  
UPDATE Students SET Grants = 500 WHERE Grants IS NULL;
```

Below the code, the "Messages" pane displays the results of the execution:

```
(1 row affected)  
(4 rows affected)
```

Рисунок 4.1. Изменение информации в таблице

Посмотрим, какие данные сейчас хранятся в столбце Grants таблицы Students (рис. 4.2).

```
SELECT LastName, FirstName, Grants  
FROM Students;
```

	LastName	FirstName	Grants
1	Doe	John	500.00
2	Jones	Jack	1756.00
3	Johnson	Joshua	500.00
4	Wilson	Lily	500.00
5	Evans	Grace	500.00

Рисунок 4.2. Проверка правильности изменения данных

5. Operator DELETE

Для того чтобы удалить определенную информацию из таблиц базы данных программным путем необходимо использовать оператор **DELETE**, общая форма записи выглядит следующим образом:

```
DELETE FROM tableName  
WHERE condition;
```

Использовать оператор **DELETE** необходимо с особой осторожностью, так как можно случайно удалить очень важную информацию. Если вы не будете устанавливать граничное условие при помощи оператора **WHERE**, то ваш SQL-запрос удалит всю информацию из текущей таблицы.

Приведем пример использования оператора **DELETE**, допустим, нам необходимо удалить все записи из таблицы **Students**, у которых уникальный идентификатор (**Id**) больше 9 (рис. 5.1).

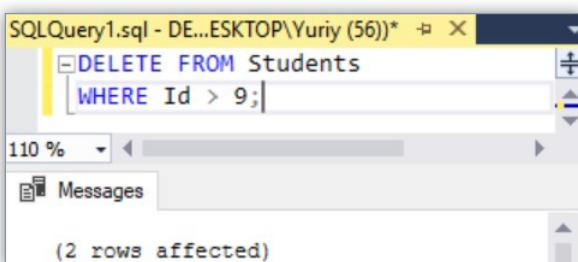


Рисунок 5.1. Удаление информации из таблицы

Как вы заметили, в данном случае были удалены две записи (`2 row affected`), при этом результат выполнения SQL-запроса в вашей базе данных может отличаться.

6. Понятие транзакции. Использование транзакций

Классический пример транзакции из повседневной жизни — перевод денежных средств с одного счета на другой. Естественно все хотят, чтобы деньги после снятия со счета-отправителя попали на счет-получателя, но если в момент осуществления этой операции произойдет сбой в системе, то необходимо чтобы деньги вернулись на счет-отправителя. Вообще транзакции окружают нас повсюду, и вы можете сами привести их примеры, подумайте.

Под транзакцией понимается последовательность действий, выполняемых как единое целое с возможность отмены каждого из них в случае ошибки.

СУБД MS SQL Server обеспечивает поддержку транзакций, которая характеризуется четырьмя важными свойствами известными под акронимом ACID:

- **Atomicity (атомарность)** — определяет целостность транзакции, то есть для успешного завершения транзакции должны выполниться либо все операции, составляющие данную транзакцию, либо ни одна из них;
- **Consistency (согласованность)** — это свойство обеспечивает целостность информации независимо от того успешно завершилась транзакция или нет;
- **Isolation (изолированность)** — означает, что все транзакции выполняются параллельно и не могут оказывать влияния друг на друга;

- **Durability (надежность)** — обеспечивает сохранность всех данных после фиксации успешно выполненной транзакции, независимо от возможных сбоев системы.

Существует три типа транзакций:

- **явные транзакции** — характеризуются явным указанием начала транзакции (**BEGIN TRAN** или **BEGIN TRANSACTION**), ее фиксации в случае успешного завершения (**COMMIT TRAN** или **COMMIT TRANSACTION**) и с возможностью прерывания (отката) транзакции при возникновении определенного условия (**ROLLBACK TRAN** или **ROLLBACK TRANSACTION**);
- **неявные транзакции** — транзакции, для которых начало и конец не указываются явным образом, такие транзакции происходят при выполнении следующих инструкций — **ALTER TABLE**, **CREATE**, **DROP**, **SELECT**, **INSERT**, **DELETE**, **UPDATE**, **GRANT**, **REVOKE**, **OPEN**, **FETCH**, **TRUNCATE TABLE** — каждая из них выполняется как отдельная транзакция;
- **автоматические транзакции** характерны тем что каждая успешно выполненная операция фиксируется, в противном случае откатывается, для того чтобы перейти в этот режим необходимо выполнить команду **SET IMPLICIT_TRANSACTION** с параметром **OFF**, для выхода **SET IMPLICIT_TRANSACTION ON**.

СУБД MS SQL Server создана как многопользовательская система, то есть система, которая позволяет одновременно использовать объекты баз данных нескольким

пользователями. В связи с этим существуют четыре возможные нарушения при работе с базами данных:

- ▷ чтение незафиксированных данных;
- ▷ неповторяющее чтение;
- ▷ фантомы;
- ▷ потерянные обновления.

Для того чтобы предотвратить конфликты, которые могут возникнуть при выполнении различных операций с одним и тем же объектом различными пользователями используются блокировки. MS SQL Server обеспечивает поддержку шести типов блокируемых ресурсов: база данных, таблица, экстент, страница, ключ и строка.

Также существует довольно большое количество типов самих блокировок, перечислим основные из них: разделяемые, исключительные, намеченные, блокировки обновления, блокировки схемы, блокировки массового обновления.

Блокировки, применяемые при выполнении транзакций, препятствуют взаимодействию с объектами другим процессам, тем самым могут служить причиной взаимной изоляции транзакций. Для управления степенью взаимной изоляции транзакций существуют четыре уровня изолированности транзакций:

- **READ UNCOMMITTED** — позволяет считывать данные, которые были изменены, но не были зафиксированы (не предотвращает ни одно из возможных нарушений);
- **READ COMMITTED** — не позволяет считывать данные, которые были изменены, но не были зафик-

сированы (используется по умолчанию) (предотвращает чтение незафиксированных данных);

- **REPEATABLE READ** — никакая транзакция не может изменять данные, считанные текущей транзакцией до ее завершения (предотвращает чтение незафиксированных данных и неповторяемое чтение);
- **SERIALIZABLE** — любые другие транзакции не могут вставлять/изменять/удалять данные, которые попадают в диапазон записей, удовлетворяющих условию, которое задано в операторе **WHERE** текущей транзакции (предотвращает чтение незафиксированных данных, неповторяемое чтение и фантомы).

Более подробная информация о транзакциях и их практическое использование будут рассмотрены в курсе MS SQL Server.

7. Домашнее задание

В домашнем задании к уроку №2 необходимо было создать базу данных «Больница» с таблицей «Patients», заполните эту таблицу произвольными данными с использованием оператора **INSERT** (не менее десяти записей) и напишите следующие SQL-запросы:

- вывести информацию обо всех пациентах, находящихся в больнице;
- показать данные о пациентах, которые лежат в определенном отделении;
- вывести названия всех отделений больницы;
- получить данные о пациентах, которые лежат в больнице больше месяца, отсортировав их по возрастанию даты поступления;
- вывести информацию о пациентах, которые были выписаны в прошлом месяце (вам поможет стандартная функция **GETDATE()** языка T-SQL, которая позволяет получить текущую дату);
- показать информацию о пациентах, которые лежали в больнице с октября по декабрь прошлого года в определенном отделении;
- вывести информацию о самом молодом пациенте и его возраст (для написания этого запроса вам следует самостоятельно изучить стандартную функцию **DATEDIFF()**, информацию о которой вы можете получить по следующей ссылке <https://docs.microsoft.com/en-us/sql/t-sql/functions/datediff-transact-sql>);

7. Домашнее задание

- получить информацию о пациентах, которые лежат в любых трех отделениях;
- показать всех пациентов, фамилия которых начинается на букву P;
- вывести данные о пациентах, которых лечит определенный врач с одинаковыми заболеваниями;
- получить данные о пациентах, пользующихся услугами определенного мобильного оператора;
- переименовать название определенного отделения;
- удалить всех пациентов, которые были выписаны больше чем полгода назад.