Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

БАЗЫ ДАННЫХ

сборник учебно-методических материалов для направлений подготовки

01.03.02 , 09.03.01, 09.03.02, 38.03.05

Благовещенск, 2017

Печатается по решению
редакционно-издательского совета
факультета математики и информатики
Амурского государственного
Университета

Составитель: Соловцова Л.А.

Базы данных: сборник учебно-методических материалов для направления подготовки 01.03.02 , 09.03.01, 09.03.02, 38.03.05. – Благовещенск: Амурский гос. ун-т, 2017.

Содержание

Краткое изложение лекционного материала.

Тема 1. Введение в базы данных. Основные понятия и определения

Цель лекции. Уяснить разницу между базой данных и системой управления базой данных. Ознакомиться с основными требованиями, которые предъявляются к банку данных и основными определениями, относящимися к БД и СУБД.

План

1. Основные понятия БД.
2. Требования, предъявляемые к банкам данных.
3. Компоненты банка данных

Краткое содержание

База данных (БД) – совокупность взаимосвязанных, хранящихся вместе данных при наличии такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений.

Создание базы данных, ее поддержка и обеспечение доступа пользователей к ней осуществляется централизованно с помощью специального программного инструментария – системы управления базами данных.

Система управления базами данных (СУБД) – это комплекс программных и языковых средств, необходимых для создания баз данных, поддержания их в актуальном состоянии и организации поиска в них необходимой информации.

Концептуальная модель БД описывает сущности, их свойства и связи между ними; не зависит от конкретной СУБД.

Сущность (entity) – это реальный или представляемый тип объекта, информация о котором должна сохраняться и быть доступна. В диаграммах сущность представляется в виде прямоугольника, содержащего имя сущности. При этом имя сущности – это имя типа, а не некоторого конкретного экземпляра этого типа.

Связь (relationship) – это графически изображаемая ассоциация, устанавливаемая между двумя сущностями. Связь может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь). Возможны связи на основе отношений:

- один-к-одному;
- один-ко-многим;
- многие-ко-многим.

В структуре банка данных выделяют следующие компоненты:

- Информационная база;

– Лингвистические средства;

– Программные средства;

– Технические средства;

– Организационно-административные подсистемы и нормативно-методическое обеспечение.

Тема 2. Модели данных

Цель лекции. Уяснить разницу между моделями организации *БД*. Ознакомиться с их достоинствами и недостатками. Понять, как организовываются связи в этих моделях, как применяются *операции* изменения в той или иной модели.

План

1. Виды моделей и их характеристики.
2. Достоинства и недостатки моделей.
3. Примеры

Краткое содержание

Иерархические базы данных - самая ранняя модель представления сложной структуры данных. Информация в иерархической базе организована по принципу древовидной структуры, в виде отношений "предок-потомок". Каждая запись может иметь не более одной родительской записи и несколько подчиненных. Связи записей реализуются в виде физических указателей с одной записи на другую. Основной недостаток иерархической структуры базы данных - невозможность реализовать отношения "многие-ко-многим", а также ситуации, когда запись имеет несколько предков.

Иерархические базы данных. Иерархические базы данных графически могут быть представлены как перевернутое дерево, состоящее из объектов различных уровней. Верхний уровень (корень дерева) занимает один объект, второй - объекты второго уровня и так далее.

Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка (объект, более близкий к корню) к потомку (объект более низкого уровня), при этом объект-предок может не иметь потомков или иметь их несколько, тогда как объект-потомок обязательно имеет только одного предка. Объекты, имеющие общего предка, называются близнецами.

Сетевая модель данных определяется в тех же терминах, что и иерархическая. Она состоит из множества записей, которые могут быть владельцами или членами групповых отношений. Связь между записью-владельцем и записью-членом также имеет вид 1:N.

Появление объектно-ориентированных СУБД вызвано потребностями программистов на ОО-языках, которым были необходимы средства для хранения объектов, не помещавших-

ся в оперативной памяти компьютера. Также важна была задача сохранения состояния объектов между повторными запусками прикладной программы. Поэтому, большинство ООСУБД представляют собой библиотеку, процедуры управления данными которой включаются в прикладную программу. Примеры реализации ООСУБД как выделеного сервера базы данных крайне редки.

Сразу же необходимо заметить, что общепринятого определения "объектно-ориентированной модели данных" не существует. Сейчас можно говорить лишь о неком "объектном" подходе к логическому представлению данных и о различных объектно-ориентированных способах его реализации.

Тема 3. Этапы проектирования базы данных

Цель лекции. Рассмотрение этапов проектирования базы данных.

План

1. Инфологическое проектирование.
2. Определение требований к операционной обстановке, в которой будет функционировать информационная система.
3. Выбор системы управления базой данных (СУБД) и других инструментальных программных средств.
4. Логическое проектирование БД.
5. Физическое проектирование БД.

Краткое содержание

Инфологическая модель предметной области (ПО) представляет собой описание структуры и динамики ПО, характера информационных потребностей пользователей в терминах, понятных пользователю и не зависимых от реализации БД. Это описание выражается в терминах не отдельных объектов ПО и связей между ними, а их типов, связанных с ними ограничений целостности и тех процессов, которые приводят к переходу предметной области из одного состояния в другое.

В настоящее время применяют проектирование с использованием метода "Сущность-связь"(entity–relation, ER–method), который является комбинацией предметного и прикладного методов и обладает достоинствами обоих.

На этом этапе производится оценка требований к вычислительным ресурсам, необходимым для функционирования системы, определение типа и конфигурации конкретной ЭВМ, выбор типа и версии операционной системы. Объём вычислительных ресурсов зависит от предполагаемого объёма проектируемой базы данных и от интенсивности их использова-

ния. Если БД будет работать в многопользовательском режиме, то требуется подключение её к сети и наличие соответствующей многозадачной операционной системы.

На весь процесс проектирования БД и реализацию информационной системы принципиальным образом влияет выбор СУБД. Теоретически при выборе СУБД нужно принимать во внимание десятки факторов. Но практически разработчики руководствуются лишь собственной интуицией и несколькими наиболее важными критериями, к которым, в частности, относятся:

– тип модели данных, которую поддерживает данная СУБД, её адекватность потребностям рассматриваемой предметной области;

– характеристики производительности системы;

– запас функциональных возможностей для дальнейшего развития ИС;

– степень оснащённости системы инструментарием для персонала администрирования данными;

– удобство и надежность СУБД в эксплуатации;

– стоимость СУБД и дополнительного программного обеспечения.

На этапе логического проектирования разрабатывается логическая структура БД, соответствующая логической модели ПО. Решение этой задачи существенно зависит от модели данных, поддерживаемой выбранной СУБД.

База данных создаётся на основании схемы базы данных. Инфологическую модель данных, построенную в виде ER–диаграммы, следует преобразовать в схему БД. Преобразование ER–диаграммы в схему БД выполняется путем сопоставления каждой сущности и каждой связи, имеющей атрибуты, отношения (таблицы БД).

Для этого необходимо выполнить следующие шаги процедуры проектирования модели.

1. Представить каждый стержень (независимую сущность) таблицей базы данных (базовой таблицей) и специфицировать первичный ключ этой базовой таблицы.

2. Представить каждую ассоциацию (связь вида "многие-ко-многим" или "многие-ко-многим-ко-многим" и т.д. между сущностями) как базовую таблицу. Использовать в этой таблице внешние ключи для идентификации участников ассоциации и специфицировать ограничения, связанные с каждым из этих внешних ключей.

3. Представить каждую характеристику как базовую таблицу с внешним ключом, идентифицирующим сущность, описываемую этой характеристикой. Специфицировать ограничения на внешний ключ этой таблицы и ее первичный ключ – по всей вероятности,

комбинации этого внешнего ключа и свойства, которое гарантирует "уникальность в рамках описываемой сущности".

4. Представить каждое обозначение, которое не рассматривалось в предыдущем пункте, как базовую таблицу с внешним ключом, идентифицирующим обозначаемую сущность. Специфицировать связанные с каждым таким внешним ключом ограничения.

5. Представить каждое свойство как поле в базовой таблице, представляющей сущность, которая непосредственно описывается этим свойством.

6. Для того чтобы исключить в проекте непреднамеренные нарушения каких-либо принципов нормализации, выполнить процедуру нормализации.

7. Если в процессе нормализации было произведено разделение каких-либо таблиц, то следует модифицировать инфологическую модель базы данных и повторить перечисленные шаги.

8. Указать ограничения целостности проектируемой базы данных и дать (если это необходимо) краткое описание полученных таблиц и их полей.

Этап физического проектирования заключается в увязке логической структуры БД и физической среды хранения с целью наиболее эффективного размещения данных, т.е. отображении логической структуры БД в структуру хранения. Решается вопрос размещения хранимых данных в пространстве памяти, выбора эффективных методов доступа к различным компонентам "физической" БД. Результаты этого этапа документируются в форме схемы хранения на языке определения данных (DDL). Принятые на этом этапе решения оказывают определяющее влияние на производительность системы.

Одной из важнейших составляющих проекта базы данных является разработка средств защиты БД. Защита данных имеет два аспекта: защита от сбоев и защита от несанкционированного доступа. Для защиты от сбоев разрабатывается стратегия резервного копирования. Для защиты от несанкционированного доступа каждому пользователю доступ к данным предоставляется только в соответствии с его правами доступа.

Тема 4. Семантический анализ предметной области

Цель лекции. Рассмотрение основных понятий и определений реляционной базы данных.

План

1. Системный анализ предметной области
2. Инфологическое (семантическое) моделирование предметной области.

Краткое содержание

Существуют два подхода к выбору состава и структуры предметной области:

Функциональный подход – реализует принцип движения «от задач» и применяется тогда, когда заранее известны функции некоторой группы лиц и комплексов задач, для обслуживания информационных потребностей которых создается рассматриваемая БД. В этом случае мы можем четко выделить минимальный необходимый набор объектов предметной области, которые должны быть описаны.

Предметный подход – когда информационные потребности будущих пользователей БД жестко не фиксируются. Они могут быть многоаспектными и весьма динамичными. Мы не можем точно выделить минимальный набор объектов предметной области, которые необходимо описывать. В описание предметной области в этом случае включаются такие объекты и взаимосвязи, которые наиболее характерны и наиболее существенны для нее. БД, конструируемая при этом, называется предметной, то есть она может быть использована при решении множества разнообразных, заранее не определенных задач.

Инфологическое моделирование (иногда используется термин семантическое моделирование) применяется на втором этапе проектирования БД, то есть после системного анализа предметной области. На этапе системного анализа были сформированы понятия о предметах, фактах и событиях, которыми будет оперировать БД. Инфологическое проектирование связано с представлением семантики предметной области в модели БД, т.е. моделирование структур данных, опираясь на смысл этих данных.

Наибольшее распространение получила модель "сущность-связь" (entity-relationship model, ER-модель).

Модель «сущность-связь» является концептуальной моделью, т.е. не учитывает особенности конкретной СУБД. Из модели "сущность-связь" могут быть получены все основные модели данных (иерархическая, сетевая, реляционная).

Основными понятиями модели "сущность-связь" являются: сущность, связь и атрибут. Любой фрагмент предметной области может быть представлен как множество сущностей, между которыми существует некоторое множество связей.

Тема 5. Основные понятия реляционных баз данных

Цель лекции. Рассмотрение основных понятий и определений реляционной базы данных.

План

1. Понятие реляционной базы данных..
2. Основные элементы реляционной базы данных.

Краткое содержание

Реляционной базой данных называется совокупность отношений, которые содержат всю информацию, хранящуюся в БД. Пользователи воспринимают эту базу данных как совокупность таблиц.

Особенности таких таблиц:

– Таблица имеет уникальное имя и состоит из однотипных строк.

– Существование фиксированного числа полей (столбцов) и значений

– (множественные поля и повторяющиеся группы недопустимы). То есть, каждая позиция отличается от другой хотя бы единственным значением.

– Возможность однозначной идентификации любой строки таблицы.

– Присвоение столбцам таблицы однозначного имени, причем в каждом из них размещаются однородные значения данных (даты, фамилии, целые числа или денежные суммы).

– Полное информационное содержание базы данных представляется в виде явных значений данных и такой метод представления является единственным. В частности, не существует каких-либо специальных "связей" или указателей, соединяющих одну таблицу с другой. При выполнении операций с таблицей ее строки и столбцы можно обрабатывать в любом порядке безотносительно к их информационному содержанию. Этому способствует наличие имен таблиц и их столбцов, а также возможность выделения любой их строки или любого набора строк с указанными признаками.

Механизмы реляционной алгебры и реляционного исчисления эквивалентны. Это означает, что для любого допустимого выражения реляционной алгебры можно построить эквивалентную формулу реляционного исчисления и наоборот. Учитывая эквивалентность механизмов реляционной алгебры и реляционного исчисления, можно пользоваться любым из этих механизмов для проверки степени реляционности некоторого языка БД.

Отметим, что хотя и редко алгебру или исчисление принимаю в качестве полной основы какого-либо языка БД. Обычно (как, например, в случае языка SQL) язык основывается на некоторой смеси алгебраических и логических конструкций. Тем не менее, знание алгебраических и логических основ языков баз данных часто бывает полезно на практике.

Различные проблемы при обновлении таблиц связаны, прежде всего, со стремлением минимизировать число таблиц. Поэтому будут даны рекомендации по разбиению некоторых больших таблиц на несколько маленьких. Возникает вопрос: Как правильно сформировать требуемый ответ, если необходимые данные хранятся в разных таблицах?

Э. Ф. Кодд предложил реляционную модель данных. Он так же стал создателем инструмента для удобной работы с отношениями - реляционной алгебры. Каждой операцией этой алгебры используется одна или несколько таблиц (отношений) в качестве ее операндов и продуцирует в результате новую таблицу, т.е. позволяет "разрезать" или "склеивать" таблицы.

Для реализации всех операций реляционной алгебры и почти всех их сочетаний были созданы языки манипулирования данными. Назовем самые распространенные среди них:

. SQL: Structured Query Language - структуризованный язык запросов;

2. QBE: Quere-By-Example - запросы по образцу.

Они относятся к языкам очень высокого уровня. Они применяются для указания пользователем тех данных, которые необходимо получить, но при этом процедура получ

Наименьшей единицей данных реляционной модели было принято отдельное атомарное (неразложимое) для данной модели значение данных не уточняется.

Домен - это множество атомарных значений одного и того же типа. Их смысл в следующем, например, если взять значения двух атрибутов из одного и того же домена, то тогда необходимо производить сравнение, которое использует оба атрибута, если же значения двух атрибутов берутся из различных доменов, то смысла в их сравнении нет.

Отношение на доменах D1, D2, ..., Dn (не обязательно, чтобы все они были различны) состоит из заголовка и тела.

Индекс представляет собой указатель на данные, размещенные в реляционной таблице. При создании индекса в нем сохраняется информация о местонахождении записей, относящихся к индексируемому столбцу таблицы. При добавлении в таблицу новых записей или удалении существующих индекс также модифицируется. Различают несколько типов индексов. Наиболее часто выделяют три типа: простые; составные; уникальные.

Простые индексы представляют собой простейший и вместе с тем наиболее распространенный тип индекса. Простой индекс строится на основе только одного столбца реляционной таблицы (индекс, приведенный на Рисунке 3 является простым).

Составные индексы строятся по двум и более столбцам реляционной таблицы. При создании составного индекса необходимо принимать во внимание, что последовательность столбцов, по которым создается индекс, влияет на скорость поиска данных

Тема 6. Нормальные формы. Нормализация таблиц

Цель лекции. Изучение методов нормализации баз данных.

План

1. Характеристика нормальных форм.
2. Методы нормализации баз данных.
3. Денормализация баз данных. .

Краткое содержание

Нормализация отношений (таблиц) — одна из основополагающих частей теории реляционных баз данных. Нормализация имеет своей целью избавиться от избыточности в отношениях и модифицировать их структуру таким образом, чтобы процесс работы с ними не был обременён различными посторонними сложностями. При игнорировании такого подхода эффективность проектирования стремительно снижается, что вкупе с прочими подобными вольностями может привести к критическим последствиям. Отношение находится в первой нормальной форме (сокращённо 1НФ), если все его атрибуты атомарны, то есть если ни один из его атрибутов нельзя разделить на более простые атрибуты, которые соответствуют каким-то другим свойствам описываемой сущности.

Отношение находится во второй нормальной форме (сокращённо 2НФ) тогда и только тогда, когда оно находится в первой нормальной форме и каждый его неключевой атрибут неприводимо зависим от первичного ключа.

Отношение находится в третьей нормальной форме, если оно находится во второй нормальной форме и все неключевые атрибуты не зависят друг от друга.

Отношение находится в четвертой нормальной форме , если оно находится в третьей нормальной форме и если в нем не содержатся независимые группы атрибутов, между которыми существует отношение «многие-ко-многим».

Атрибут B функционально зависит от атрибута A, если каждому значению A соответствует в точности одно значение B. Математически функциональная зависимость B от A обозначается записью $A \rightarrow B$. Это означает, что во всех кортежах с одинаковым значением атрибута A атрибут B будет иметь также одно и то же значение. Отметим, что A и B могут быть составными - состоять из двух и более атрибутов.

Метод нормальных форм (НФ) состоит в сборе информации о объектах решения задачи в рамках одного отношения и последующей декомпозиции этого отношения на несколько взаимосвязанных отношений на основе процедур нормализации отношений.

В основе классического процесса проектирования лежит последовательность переходов от предыдущей нормальной формы к последующей. Однако в процессе декомпозиции мы сталкиваемся с проблемой обратимости, то есть возможности восстановления исходной схемы. Таким образом, декомпозиция должна сохранять эквивалентность схем БД при замене одной схемы на другую.

Основные определения.

Эквивалентные схемы. Схемы БД называются эквивалентными, если содержание исходной БД может быть получено путем естественного соединения отношений, входящих в результирующую схему, и при этом не появляется новых кортежей в исходной БД.

Функциональная зависимость. В отношении R набор атрибутов Y функционально зависит от набора атрибутов X или набор атрибутов X функционально определяет набор атрибутов Y (обозначается R.X◊R.Y) тогда и только тогда, когда каждому значению X соответствует в точности одно значение Y.

Полная функциональная зависимость. Функциональная зависимость называется полной, если набор атрибутов Y функционально зависит от X и не зависит функционально от любого подмножества X.

Транзитивная зависимость. Функциональная зависимость R.X◊ R.Y называется транзитивной, если существует набор атрибутов Z такой, что: Z не является подмножеством X; Z не включает в себя Y;

существует функциональная зависимость R.X ◊ R.Z; существует функциональная зависимость R.Z ◊ R.Y и не существует функциональной зависимости R.Z ◊ R.X.

Замечание: Транзитивная зависимость появляется, когда неключевой атрибут имеет свои собственные свойства.

Возможный ключ. Возможный ключ — это набор атрибутов, однозначно определяющий кортеж отношения, и при этом при удалении любого атрибута из этого набора его свойство однозначной идентификации кортежа теряется.

Среди всех возможных ключей отношения обычно выбирают одни, который считается главным и который называют первичным ключом отношения.

Неключевой атрибут. Неключевой атрибут — атрибут, не входящий в состав ни одного возможного ключа.

Взаимно-независимые атрибуты. Взаимно-независимые атрибуты — атрибуты, которые не зависят функционально один от другого.

Детерминант отношения. Если в отношении существует несколько функциональных зависимостей, то каждый атрибут или набор атрибутов, от которого зависит другой атрибут, называется детерминантом отношения.

Для функциональных зависимостей как фундаментальной основы проекта БД были проведены исследования, позволяющие избежать избыточного их представления. Ряд зависимостей могут быть выведены из других путем применения правил, названных аксиомами Армстронга, по имени исследователя, впервые сформулировавшего их.

Это три основных аксиомы:

Рефлективность: если X является подмножеством Y, то Y◊X

Пополнение: если X◊Y , то XC◊YC

Транзитивность: если X◊Y и Y◊Z , то X◊Z. Доказано, что данные правила являются полными и исчерпывающими, то есть, применяя их, из заданного множества функциональных зависимостей можно вывести все возможные функциональные зависимости.

Существует несколько других правил вывода, которые следуют из аксиом Армстронга.

Правило самоопределения. X◊X.

Правило объединения. Если X◊Y и X◊Z, то X◊Y∪Z.

Правило псевдотранзитивности. Если X◊Y и W∪Y◊Z, то X∪W◊Z.

Правило композиции. Если X◊Y и Z ◊W, то X∪W◊Y∪W.

Правило декомпозиции. Если X◊Y и Z входит в Y, то X◊Z.

Множество всех возможных функциональных зависимостей, выводимое из заданного набора исходных функциональных зависимостей, называется его замыканием.

При выполнении эквивалентных преобразований сохраняется множество исходных фундаментальных функциональных зависимостей между атрибутами отношений.

Функциональные зависимости определяют не текущее состояние БД, а все возможные ее состояния, то есть они отражают те связи между атрибутами, которые присущи реальному объекту, который моделируется с помощью БД. Поэтому определить функциональные зависимости по текущему состоянию БД можно только в том случае, если экземпляр БД содержит абсолютно полную информацию (то есть никаких добавлений и модификации БД не предполагается). В реальной жизни это требование невыполнимо, поэтому набор функциональных зависимостей задает разработчик, системный аналитик, исходя из глубокого системного анализа ПО.

Декомпозиция схемы отношения. Декомпозицией схемы отношения $R(A_1, A_2, A_n)$ называется замена ее совокупностью подмножеств R, таких, что их естественное соединение дает R. При этом допускается, чтобы подмножества были пересекающимися.

Алгоритм декомпозиции основан на следующей теореме.

Теорема о декомпозиции. Пусть R(A, B, C)–отношение, A, B, C–атрибуты.

Если R удовлетворяет зависимости A◊B, то R равно соединению его проекций A,B и A, C

При нормализации необходимо выбирать такие декомпозиции, которые обладают свойством соединения без потерь. Вторым важнейшим желательным свойством декомпозиции является свойство сохранения функциональных зависимостей.

Тема 7. Проектирование связей между таблицами

Цель лекции. Рассмотрение способов построения связей между сущностями.

План

1. Понятие связи между сущностями.
2. Виды связей.
3. Характеристика связей

Краткое содержание

Между двумя или более таблицами базы данных могут существовать отношения подчиненности. Отношения подчиненности определяют, что для каждой записи главной таблицы *{master,* называемой *еще родительской}* может существовать одна или несколько записей в подчиненной таблице *{detail,* называемой еще *дочерней}*.

Существует три разновидности связей между таблицами базы данных:

- «один-ко-многим»,
- «один-к-одному»,
- «многие-ко-многим».

Отношение «один-ко-многим» имеет место, когда одной записи родительской таблицы может соответствовать несколько записей в дочерней таблице.

Связь "один-ко-многим" является самой распространенной для реляционных баз данных.

В широко распространенной нотации структуры баз данных IDEF1X отношение «один-ко-многим» изображается путем соединения таблиц линией, которая на стороне дочерней таблицы оканчивается кружком или иным символом. Поля, входящие в первичный ключ для данной ТБД, всегда расположены вверху и отчеркнуты от прочих полей линией.

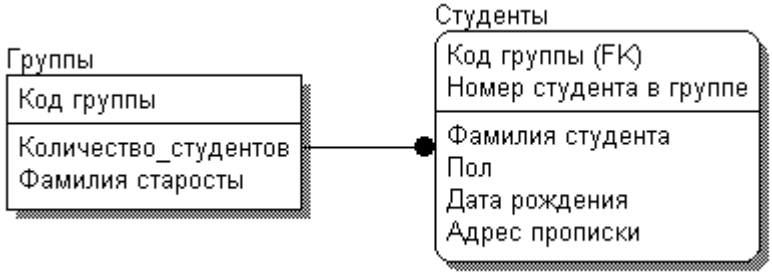


Рисунок 1 – Связь между сущностями

Отношение «один-к-одному» имеет место, когда одной записи в родительской таблице соответствует одна запись в дочерней таблице.

№ сотрудника	ФИО	Должность	Отдел
1	Иванов П.П.	лаборант	2
2	Сидорова А.М.	бухгалтер	1
3	Петров А.Н.	инженер	2

№ сотрудника	Год рождения	Число детей
1	1967	1
2	1955	1
3	1943	2

Рисунок 2 – Связь «один-к-одному»

Данное отношение используют, если не хотят, чтобы таблица БД «не распухала» от второстепенной информации.

Отношение «многие-ко-многим» имеет место, когда:

а) записи в родительской таблице может соответствовать больше одной записи в дочерней таблице;

б) записи в дочерней таблице может соответствовать больше одной записи в родительской таблице.

Например, каждой студент изучает несколько дисциплин. Каждая дисциплина изучается несколькими студентами.

Группа	№ студента	ФИО студента
ИДМ-107	1	Иванов
ИДМ-107	2	Петров
ИДМ-107	3	Сидоров

Код дисциплины	Название дисциплины
1	Информатика
2	Математика
3	История

Рисунок 3 – Связь «один-ко-многим»

Многие СУБД не поддерживают связи «многие-ко-многим» на уровне индексов и ссылочной целостности. Считается, что всякую связь «многие-ко-многим» можно заменить на одну или более связей «один-ко-многим».

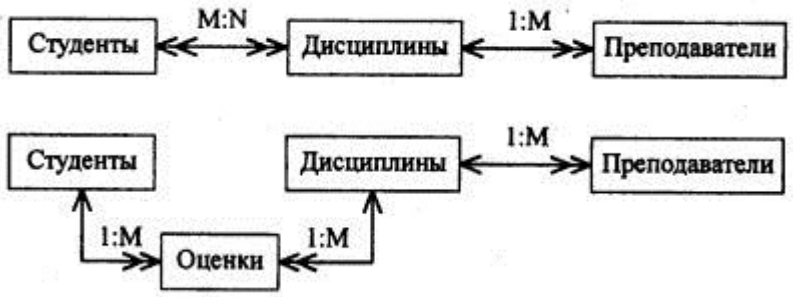


Рисунок 4 – Связь «многие-ко-многим»

Тема 8. Концептуальная модель данных

Цель лекции. Рассмотрение способов построения концептуальных моделей баз данных.

План

1. Понятие концептуальной модели данных.
2. Элементы концептуальной модели данных.
3. Диаграммы для представления концептуальной модели данных. ER-диаграмма

Краткое содержание

Концептуальное проектирование начинается с анализа предметной области, включает анализ концептуальных требований и информационных потребностей, выявление информационных объектов и связей между ними, построение концептуальной модели (схемы) данных.

Главными элементами концептуальной модели данных являются объекты и отношения.

Объекты представляют собой любой конкретный (реальный) объект в рассматриваемой области.

Исходя из спецификации требования, определим основные типы сущностей.

Сущностьюназывается некоторая принятая в конкретной постановке задачи абстракция реального мира, процесса или явления, о котором необходимо хранить информацию в системе. В качестве синонима термина «сущность» используется также термин «информационный объект».

Объекты в каждый момент времени характеризуются определенным состоянием, которое описывается набором свойств и отношений (или связей) с другими объектами.

Характеристика, описывающая какое-либо свойство сущности, которое можно сформулировать и записать, называется атрибутом. Атрибут, который однозначно определяет сущность, называется идентификатором.

Сущность - объект любой природы данные, о котором хранятся в отношении (таблице, в которой содержатся данные).

концептуальная модель представляется в виде диаграммы сущностей – связей (entity – relationship) или ER-диаграммы. Процесс построения ER-диаграммы называется ER-моделированием.

Введем основные понятия, с помощью которых описывается предметная область.

Сущность (Entity) или объект – то, о чем будет накапливаться информация в информационной системе (нечто такое, за чем пользователь хотел бы наблюдать).

Если в системе обрабатывается информация о факультетах, сущностью будет являться факультет, если о студентах, сущность – студент и т.п.

Имя сущности при ER-моделировании, как правило, записывается заглавными буквами. Каждая сущность обладает определенным набором свойств (рассматриваем только свойства, представляющие интерес для пользователей в рамках проводимого исследования), которые запоминаются в информационной системе.

Для информационного описания сущности вводится понятие атрибута.

Атрибут – поименованное свойство (характеристика) сущности. Атрибут представляет собой информационное отображение свойства сущности и принимает конкретное значение из множества допустимых значений.

Тема 9. Логическая модель данных

Цель лекции. Рассмотрение способов построения логических моделей баз данных.

План

1. Понятие логической модели данных.
2. Переход от концептуальной модели к логической модели.
3. Методологии построения логической модели данных.

Краткое содержание

При реализации связи *многие-ко-многим*, допустимой в инфологической модели, производится ее преобразование к связям *один-ко-многим*, например, через промежуточное отношение.Промежуточное отношение будет иметь первичный ключ, состоящий из первичных ключей связываемых отношений.

После выполнения преобразований необходимо убедиться в корректности полученной схемы БД, в противном случае в таблицах могут оказаться нежелательные функциональные зависимости, которые впоследствии могут привести к возникновению различных аномалий. Проверить полученную схему БД, на отсутствие нежелательных функциональных зависимостей, можно используя правила нормализации.

Инфологическая модель позволяет понять суть разрабатываемой базы данных, но она не подходит для непосредственной реализации структуры БД. Необходимо преобразование инфологической модели в логическую, с учетом особенностей выбранной логической модели.

Алгоритм перехода от инфологической-модели к реляционной модели данных обычно сводится к следующим шагам:

1. Каждой сущности ставится в соответствие отношение РМД. Имена отношений могут быть ограничены требованиями конкретной СУБД, они ограничены по длине и не должны содержать пробелов и некоторых специальных символов.

2. Каждый атрибут сущности становится атрибутом соответствующего отношения, на их имена также могут накладываться некоторые ограничения (например, многие СУБД не поддерживают кириллицу). Для каждого атрибута задается конкретный допустимый в СУБД тип данных и обязательность или необязательность данного атрибута. На рисунке 11.1, упрощенно, показан процесс преобразования, с учетом выбранной СУБД (в правой части отношения указаны типы данных принятые в СУБД MS Access для соответствующих полей).

Тема 10. Физическая модель данных

Цель лекции. Рассмотрение физических моделей баз данных, которые определяют способы размещения данных в среде хранения.

План

1. Файловые структуры, используемые для хранения информации в базах данных
2. Классификация файлов, используемых в системах баз данных
3. Стратегия разрешения коллизий с областью переполнения
4. Организация стратегии свободного замещения

Краткое содержание

Физические модели баз данных определяют способы размещения данных в среде хранения и способы доступа к этим данным, которые поддерживаются на физическом уровне. Исторически первыми системами хранения и доступа были файловые структуры и системы управления файлами (СУФ), которые фактически являлись частью операционных систем. *СУБД* создавала над этими файловыми моделями свою надстройку, которая позволяла организовать всю совокупность файлов таким образом, чтобы она работала как единое целое и получала *централизованное управление* от *СУБД*. Однако непосредственный *доступ* осуществлялся на уровне файловых команд, которые *СУБД*использовала при манипулировании всеми файлами, составляющими хранимые данные одной или нескольких баз данных.

В каждой *СУБД по*-разному организованы хранение и *доступ* к данным, однако существуют некоторые файловые структуры, которые имеют общепринятые способы организации и широко применяются практически во всех *СУБД*.

В системах баз данных файлы и файловые структуры, которые используются для хранения информации во внешней памяти, можно классифицировать следующим образом

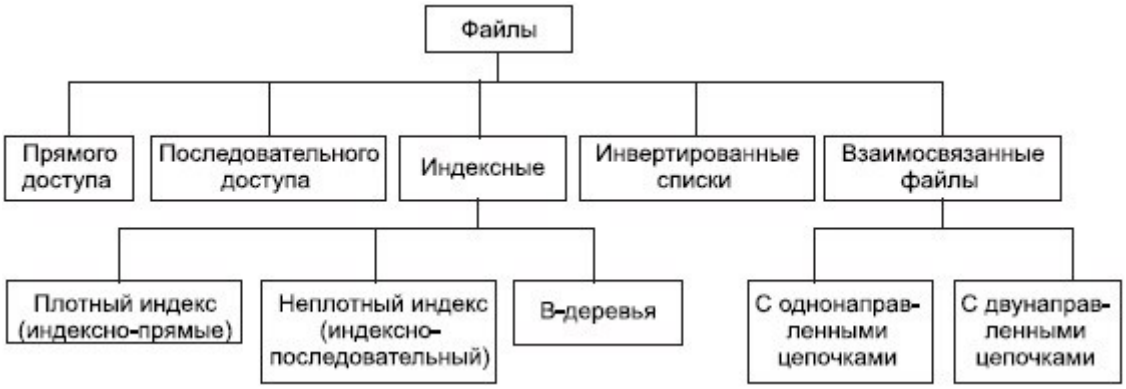


Рисунок 5 – Классификация файлов, используемых в системах баз данных

Первая стратегия условно может быть названа стратегией с областью переполнения. При выборе этой стратегии область хранения разбивается на 2 части:

- основную область;
- область переполнения.

Для каждой новой записи вычисляется значение хэш-функции, которое определяет адрес ее расположения, и запись заносится в основную область в соответствии с полученным значением хэш-функции.

При этой стратегии файловое пространство не разделяется на области, но для каждой записи добавляется 2 указателя: указатель на предыдущую запись в цепочке синонимов и указатель на следующую запись в цепочке синонимов. Отсутствие соответствующей ссылки обозначается специальным символом, например нулем. Для каждой новой записи вычисляется значение хэш-функции, и если данный адрес свободен, то запись попадает на заданное место и становится первой в цепочке синонимов. Если адрес, соответствующий полученному значению хэш-функции, занят, то по наличию ссылок определяется, является ли запись, расположенная по указанному адресу, первой в цепочке синонимов. Если да, то новая запись располагается на первом свободном месте и для нее устанавливаются соответствующие ссылки: она становится второй в цепочке синонимов, на нее ссылается первая запись, а она ссылается на следующую, если таковая есть.

Тема 11. Способы организации памяти для хранения данных

Цель лекции. Изучение способов организации памяти для хранения данных реляционной базы данных.

План

1. Организация хранения
2. Методы доступа
3. Проблемы управления внешней памятью

Краткое содержание

Основными единицами физического хранения являются блок данных, экстент, файл (либо раздел жесткого диска). Логический уровень представления информации включает пространства (либо табличные пространства). Блок данных (block) или страница (page) является единицей обмена с внешней памятью. Размер страницы фиксирован для базы данных (Oracle) или для ее различных структур (DB2, Informix, Sybase) и устанавливается при создании. Очень важно сразу правильно выбрать размер блока: в работающей базе изменить его практически невозможно (для этого часто проводят ряд испытаний базы данных-прототипа).

Размер блока оказывает большое влияние на производительность базы данных — при больших размерах скорость операций чтения/записи растет (особенно это характерно для полных просмотров таблиц и операций интенсивной загрузки данных), однако возрастают накладные расходы на хранение (база увеличивается) и снижается эффективность индексных просмотров. Меньший размер блока позволяет более экономно расходовать память, но вместе с тем относительно дорог. Длинные блоки (16, 32 или 64 Кбайт) лучше использовать для больших объектов данных: полнотекстовые фрагменты, мультимедиа-объекты, длинные строки и т.п. Короткие блоки (2 или 4 Кбайт) лучше подходят для значений числовых типов, недлинных строк, значений даты и времени. Следует также учитывать размер блока ОС, он должен быть кратен размеру блока базы данных. Малый размер блока лучше подходит для систем оперативной обработки транзакций, потому что, если сервер блокирует данные на уровне блоков, то это позволяет большему числу пользователей работать, не мешая друг другу.

Методы доступа.

Физическо-последовательный. Значения ключей физических записей находятся в логической последовательности. Применяется в основном для дампа и восстановления данных и для выборки данных.

Индексно-последовательный. До осуществления доступа к собственно записям БД проверяются значения ключей.

Индексно-произвольный. При индексно-произвольном методе доступа записи хранятся в произвольном порядке.

Метод прямого доступа. Между ключом записи и ее физическим адресом существует взаимно однозначное соответствие. Физическое местоположение записи определяется непо-

средственно из значения ключа. Не требует упорядоченности значений ключей физических записей.

Хешированный. Разновидность метода прямого доступа, обеспечивающего быструю выборку и обновление записей. Сущность метода хеширования заключается в том, что все адресное пространство делится на несколько областей фиксированного размера, которые называются бакетами. Если при занесении нового значения индекса все бакеты заняты, то для него выделяется дополнительная область памяти, называемая областью переполнения. Не требуется логическая упорядоченность значений ключей физических записей. Значениям нескольких ключей может соответствовать один и тот же физический адрес (блок).

Инвертированный (метод вторичного индексирования). Значения ключей физических записей необязательно находятся в логической последовательности. Метод применяется только для выборки данных.

Использование структуры данных типа «дерево»: очевидно и аналогично вопросу бинарные деревья.

Современные СУБД предоставляют достаточно широкий набор различных методов доступа, которые чаще всего являются теми или иными видами индексирования — способа отображения ключа индексирования в адрес хранимой записи. Используются следующие типы индексных структур: на основе B-дерева (B-tree); на основе хэш-функции или хеширование (hashing); на базе битовых шкал или индексов (bitmap). Индекс может служить различным целям: для ускорения доступа к записям одной таблицы и для ускорения операций соединения, тогда он называется индексом соединения. Если в качестве ключа индексирования используется некоторая функция атрибутов таблицы, такой индекс называют «основанным на функции» (function-based). Скажем, можно создать такой индекс для ускорения поиска с учетом регистра символов для таблицы Famous (таблица 1): Распространенной проблемой для администраторов при управлении физическим хранением является фрагментация различных структур внешней памяти, которая чаще обусловлена операциями удаления. В отличие от преднамеренной фрагментации, такая «фрагментация» обычно ухудшает производительность. Встречается фрагментация на уровне блоков, когда в блоке остается свободное пространство после удаления строк из таблицы, на уровне экстентов, когда заполненные экстенты чередуются с незаполненными, появившимися после операций удаления таблиц, и т.п. Укажем лишь часть из проблем, которые могут быть обусловлены фрагментацией.

1. Выделение свободного пространства. При вставке строк в таблицу СУБД выдает сообщение о нехватке свободного пространства, хотя на первый взгляд пространства достаточно. Пространство выделяется экстентами и, если в базе данных нет непрерывного свободного блока нужного размера, то выдается это сообщение.

2. Замедление операций вставки в таблицу. В [2] приводится пример с таблицей базы данных, в которую в ходе выполнения ночного пакетного задания происходит вставка большого количества строк. Для того чтобы уменьшить время его выполнения был в несколько раз увеличен размер вновь выделяемых экстентов, благодаря чему снизились накладные расходы на выделение экстентов, а скорость выполнения пакетного задания возросла на 30%.

3. Расход свободного пространства и общее увеличение времени обработки данных. В случае сильной фрагментации обе величины могут вырасти вдвое.

Фрагментация неизбежна и поэтому является нормальным явлением, и не всегда ухудшает характеристики базы данных. Индексы также подвержены проблемам фрагментации пространства, и могут стать несбалансированными, поэтому SQL-оператор ALTER INDEX обычно имеет опцию REBUILD, позволяющую перестроить индекс. Индекс также можно удалить и создать заново даже в работающей базе данных.

Тема 12. Язык DDL. Основные объекты базы данных

Цель лекции. Изучение основных команд языка определения данных.

План

1. Характеристика языка определения данных.
2. Синтаксис основных команд языка.
3. Примеры .

Краткое содержание

Язык определения данных решает задачи создания и удаления объектов базы данных. По стандартам SQL-92к таким объектам относятся:

-схемы;

-таблицы;

-индексы;

-представления;

-курсоры.

Каждый объект в базе однозначно описывается его именем и имеет владельца. Подавляющее число операторов DDL начинается с ключевых слов CREATE (создать) илиDROP (удалить).

Создание базы данных – CREATE DATABASE

Уничтожение базы данных – DROP DATABASE

Создание схемы – CREATE SCHEMA

Удаление схемы – DROP SCHEMA

Создание таблицы – CREATE TABLE

Удаление таблицы – DROP TABLE

Изменение структуры таблицы – ALTER TABLE

Тема 13. Команды DDL для работы с таблицами.

Цель лекции. Изучение команд DDL для работы с таблицами.

План

1. Перечень команд
2. Синтаксис команд
3. Примеры

Краткое содержание

После создания базы данных можно переходить к созданию таблиц. Ниже представлен общий формат оператора CREATE TABLE:

CREATE [{GLOBAL | LOCAL} TEMPORARY]TABLE имя_таблицы

({определение поля | [ограничение таблицы]}., … [ON COM-
MIT {DELETE |PRESERVE}ROWS]):

определение поля ::= имя_поля {имя домена | тип данных [размер]} [ограничение поля…]

[DEFAULT значение по умолчанию] [COLLATE имя сравнения]

Оператор предназначен для удаления таблиц.

DROP TABLE имя_таблицы CASCADE| RESTRICT

Если определён параметр RESTRICT, то на удаляемую таблицу не должно быть ни каких ссылок в ограничениях или просмотрах (представлениях). Таким образом СУБД обеспечивает целостность данных и таблица не будет удалена если она связана с другими объектами базы данных. Параметр CASCADE осуществляет каскадное удаление всех ссылающихся на таблицу объектов (таблиц, ограничений, просмотров).

Предложение ALTER TABLE вносит изменения в определение уже существующей таблицы.

ALTER TABLE имя_таблицы

{ADD[COLUMN] определение столбца}

| {ALTER [COLUMN] имя изменяющегося столбца}

| {DROP [COLUMN] имя_столбца RESTRICT | CASCADE} | {ADD определение ограничения для таблицы}

| {DROP CONSTRAINT имя ограничения RESTRICT | CASCADE}

Тема 14. Команды манипулирования данными DML

Цель лекции. Изучение команд манипулирования данными и их возможностей.

План

1. Характеристика команды манипулирования данными
2. Синтаксис команд.
3. Примеры

Краткое содержание

Язык DML содержит следующие конструкции:
SELECT – выборка данных

- INSERT – вставка новых данных
- UPDATE – обновление данных
- DELETE – удаление данных
- MERGE – слияние данны

оператор INSERT используется для вставки (добавления) строк в таблицу базы данных. Добавление можно осуществить несколькими способами:

- добавить одну полную строку
- добавить часть строки
- добавить результаты запроса.

Итак, чтобы добавить новую строку в таблицу, нам необходимо указать название таблицы, перечислить названия колонок и указать значение для каждой колонки с помощью конструкции INSERT INTO название_таблицы (поле1, поле2 ...)VALUES (значение1, значение2 ...).

Добавление части строк

Добавление отобранных данных

Копирование данных из одной таблицы в другую

Тема 15. Команды выборки данных (SELECT)

Цель лекции. Изучение методов построения и использования представлений для реляционной базы данных.

План

1. Понятие и назначение выборки
2. Формат команды SELECT
3. Примеры

Краткое содержание

Выборка – это обращение к БД с целью извлечь данные в виде, удобном для пользователя. Для выборки применяются запросы к БД. Иногда в SQL выделяют даже раздел, который называют языком запросов к данным DQL (Data Query Language). Фактически этот раздел языка ANSI SQL представлен только одной командой – SELECT. Но эта команда достаточно обширна. Она является ядром языка SQL. и используется для реализации операций проекции, ограничения, расширения.

Оператор SELECT позволяет производить:

– выбор данных (отбор записей и полей);

– вычисления и сравнения;

– упорядочение записей при выводе содержимого таблиц;

– группирование данных и применение к этим группам специальных групповых операций.

Источником данных (ИД) для запроса могут быть РТ или ранее созданные запросы.

После выполнения запроса на выборку создается набор записей в виде временной рабочей таблицы, содержащей данные, которые отбираются из ИД согласно заданным условиям.

В большинстве случаев с набором записей можно работать точно так же, как с таблицей: можно просматривать, выбирать и даже обновлять информацию. Однако в отличие от реальной таблицы, этот набор записей физически не существует в БД. Запрос с точки зрения пользователя можно рассматривать как шаблон, который создает набор записей из ИД только во время своего выполнения. При отсутствии данных результат представляет пустой набор записей.

Синтаксис инструкции SELECT определяется конструкциями, используемыми при реализации функций выборки. Инструкция в общем виде использует пять частей, которые делятся на две группы:

основная часть

SELECT [предикат] <список полей> выбрать

FROM <список ИД>из

дополнительные части

[WHERE <спецификация выбора записей>] где

[[GROUP BY <спецификация группировки>] группируя по

[HAVING <спецификация выбора групп>]] имея

[ORDER BY <спецификация сортировки>]упорядочить по.

Рассмотрим синтаксис инструкции по частям. При этом учтем следующее:

– инструкция языка SQL – это предложение (команда, оператор);

– отдельные составные части инструкции (список полей, спецификация) – это опции предложения;

– любая спецификация – это фраза, отвечающая требованиям синтаксиса предложения.

Из синтаксиса видно, что

- основная часть команды SELECT …FROM обязательна;

- опция предикат необязательна

- дополнительные части WHERE, GROUP BY, ORDER BY **н**еобязательны, они следуют за FROM;

- опция HAVING не может применяться без GROUP BY.

Тема 16. Представления (View)

Цель лекции. Изучение методов построения и использования представлений для реляционной базы данных.

План

1. Определение представления
2. Создание, изменение и использование представления
3. Обновления данных в представлениях
4. Преимущества и недостатки представлений
5. Примеры

Краткое содержание

Представления, или просмотры (VIEW), представляют собой временные, производные (иначе - виртуальные) таблицы и являются объектами базы данных, информация в которых не хранится постоянно, как в базовых таблицах, а формируется динамически при обращении к ним. Обычные таблицы относятся к базовым, т.е. содержащим данные и постоянно находящимся на устройстве хранения информации. Представление не может существовать само по себе, а определяется только в терминах одной или нескольких таблиц. Применение представлений позволяет разработчику базы данных обеспечить каждому пользователю или группе пользователей наиболее подходящие способы работы с данными, что решает проблему простоты их использования и безопасности. Содержимое представлений выбирается из других таблиц с помощью выполнения запроса, причем при изменении значений в таблицах данные в представлении автоматически меняются. Представление - это фактически тот же запрос, который выполняется всякий раз при участии в какой-либо команде. Результат выполнения этого запроса в каждый момент времени становится содержанием представления. У пользователя создается впечатление, что он работает с настоящей, реально существующей таблицей.

У СУБД есть две возможности реализации представлений. Если его определение простое, то система формирует каждую запись представленияпо мере необходимости, постепенно считывая исходные данные из базовых таблиц. В случае сложного определения СУБД приходится сначала выполнить такую операцию, как материализация представления, т.е. сохранить информацию, из которой состоит представление, во временной таблице. Затем система приступает к выполнению пользовательской команды и формированию ее результатов, после чего временная таблица удаляется.

Создания и изменения представлений в стандарте языка и реализации в MS SQL Server совпадают и представлены следующей командой:

<определение_представления> ::=

{ CREATE| ALTER} VIEW имя_представления

[(имя_столбца [,...n])]

[WITH ENCRYPTION]

AS SELECT_оператор

[WITH CHECK OPTION]

Рассмотрим назначение основных параметров.

Параметр WITH ENCRYPTION предписывает серверу шифровать SQL-код запроса, что гарантирует невозможность его несанкционированного просмотра и использования.

Параметр WITH CHECK OPTION предписывает серверу исполнять проверку изменений, производимых через представление, на соответствие критериям, определенным в операторе SELECT.

Использование аргумента WITH CHECK OPTION гарантирует, что сделанные изменения будут отображены в представлении.

Основные преимущества применения представлений:

независимость от данных

Актуальность

Повышение защищенности данных

Снижение стоимости

Дополнительные удобства при использовании представоений

Возможность настройки

Обеспечение целостности данных

Однако использование представлений в среде SQL не лишено недостатков.

Ограниченные возможности обновления

Структурные ограничения

Снижение производительности

Тема 17. Процедуры. Курсоры.

Цель лекции. Изучение методов использования курсоров для разработки приложений в среле SQL.

План

1. Понятие курсора и основные действия
2. Реализация курсоров
3. Классификация курсоров
4. Управление курсором

5. Объявление курсора
6. Работа с курсором
7. Закрытие курсора
8. Освобождение курсора

Краткое содержание

Понятие курсора

Запрос к реляционной базе данных обычно возвращает несколько рядов (записей) данных, но приложение за один раз обрабатывает лишь одну запись. Даже если оно имеет дело одновременно с несколькими рядами (например, выводит данные в форме электронных таблиц), их количество по-прежнему ограничено. Кроме того, при модификации, удалении или добавлении данных рабочей единицей является ряд. В этой ситуации на первый план выступает концепция курсора, и в таком контексте курсор – указатель на ряд.

Курсор в SQL – это область в памяти базы данных, которая предназначена для хранения последнего оператора SQL. Если текущий оператор – запрос к базе данных, в памяти сохраняется и строка данных запроса, называемая текущим значением, или текущей строкой курсора. Указанная область в памяти поименована и доступна для прикладных программ.

В соответствии со стандартом SQL при работе с курсорами можно выделить следующие основные действия:

– создание или объявление курсора ;

– открытие курсора, т.е. наполнение его данными, которые сохраняются в многоуровневой памяти ;

– выборка из курсора и изменение с его помощью строк данных;

– закрытие курсора, после чего он становится недоступным для пользовательских программ;

– освобождение курсора, т.е. удаление курсора как объекта, поскольку его закрытие необязательно освобождает ассоциированную с ним память.

SQL Server поддерживает три вида курсоров:

– курсоры SQL применяются в основном внутри триггеров, хранимых процедур и сценариев;

– курсоры сервера действуют на сервере и реализуют программный интерфейс приложений для ODBC, OLE DB, DB_Library;

– курсоры клиента реализуются на самом клиенте.

Они выбирают весь результирующий набор строк из сервера и сохраняют его локально, что позволяет ускорить операции обработки данных за счет снижения потерь времени на выполнение сетевых операций.

Курсоры делятся на две категории:

последовательные и прокручиваемые.

Последовательные позволяют выбирать данные только в одном направлении – от начала к концу. Прокручиваемые же курсоры предоставляют большую свободу действий – допускается перемещение в обоих направлениях и переход к произвольной строке результирующего набора курсора.

SQL Server поддерживает курсоры статические, динамические, последовательные и управляемые набором ключей.

Управление курсором в среде MS SQL Server

Управление курсором реализуется путем выполнения следующих команд:

- DECLARE – создание или объявление курсора ;
- OPEN – открытие курсора, т.е. наполнение его данными;
- FETCH – выборка из курсора и изменение строк данных с помощью курсора;
- CLOSE – закрытие курсора ;
- DEALLOCATE – освобождение курсора, т.е. удаление курсора как объекта.

Объявление курсора

В стандарте SQL для создания курсора предусмотрена следующая команда:

<создание_курсора>::=

```
DECLARE имя_курсора
  [INSENSITIVE][SCROLL] CURSOR
FOR SELECT_оператор
[FOR { READ_ONLY | UPDATE
  [OF имя_столбца[,...n]]}]
```

При использовании ключевого слова INSENSITIVE будет создан статический курсор. Изменения данных не разрешаются, кроме того, не отображаются изменения, сделанные другими пользователями. Если ключевое слово INSENSITIVE отсутствует, создается динамический курсор.

При указании ключевого слова SCROLL созданный курсор можно прокручивать в любом направлении, что позволяет применять любые команды выборки. Если этот аргумент опускается, то курсор окажется последовательным, т.е. его просмотр будет возможен только в одном направлении – от начала к концу.

SELECT-оператор задает тело запроса SELECT, с помощью которого определяется результирующий набор строк курсора.

При указании аргумента FOR READ_ONLY создается курсор "только для чтения", и никакие модификации данных не разрешаются. Он отличается от статического, хотя последний также не позволяет менять данные. В качестве курсора "только для чтения" может быть объявлен динамический курсор, что позволит отображать изменения, сделанные другим пользователем.

Создание курсора с аргументом FOR UPDATE позволяет выполнять в курсоре изменение данных либо в указанных столбцах, либо, при отсутствии аргумента OF имя_столбца, во всех столбцах.

В среде MS SQL Server принят следующий синтаксис команды создания курсора:

<создание_курсора>::=

DECLARE имя_курсора CURSOR [LOCAL | GLOBAL]
[FORWARD_ONLY | SCROLL]
[STATIC | KEYSET | DYNAMIC | FAST_FORWARD]
[READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
[TYPE_WARNING]
FOR SELECT_оператор
[FOR UPDATE [OF имя_столбца[,...n]]]

Для открытия курсора и наполнения его данными из указанного при создании курсора запроса SELECT используется следующая команда:

OPEN {{[GLOBAL]имя_курсора }
|@имя_переменной_курсора}

Выборка данных из курсора

Сразу после открытия курсора можно выбрать его содержимое (результат выполнения соответствующего запроса) посредством следующей команды:

FETCH [[NEXT | PRIOR | FIRST | LAST
| ABSOLUTE {номер_строки
| @переменная_номера_строки}
| RELATIVE {номер_строки |
@переменная_номера_строки}]
FROM]{{[GLOBAL]имя_курсора }|
@имя_переменной_курсора }
[INTO @имя_переменной [,...n]]

Изменение и удаление данных

Для выполнения изменений с помощью курсора необходимо выполнить команду UPDATE в следующем формате:

UPDATE имя_таблицы SET {имя_столбца={

DEFAULT | NULL | выражение}}[,...n]

WHERE CURRENT OF {{[GLOBAL] имя_курсора}

|@имя_переменной_курсора}

За одну операцию могут быть изменены несколько столбцов текущей строки курсора, но все они должны принадлежать одной таблице.

Для удаления данных посредством курсора используется команда DELETE в следующем формате:

DELETE имя_таблицы

WHERE CURRENT OF {{[GLOBAL] имя_курсора}

|@имя_переменной_курсора}

В результате будет удалена строка, установленная текущей в курсоре.

Закрытие курсора

CLOSE {имя_курсора | @имя_переменной_курсора}

После закрытия курсор становится недоступным для пользователей программы. При закрытии снимаются все блокировки, установленные в процессе его работы. Закрытие может применяться только к открытым курсорам. Закрытый, но не освобожденный курсор может быть повторно открыт. Не допускается закрывать неоткрытый курсор.

Освобождение курсора

Закрытие курсора необязательно освобождает ассоциированную с ним память. В некоторых реализациях нужно явным образом освободить ее с помощью оператора DEALLOCATE. После освобождения курсора освобождается и память, при этом становится возможным повторное использование имени курсора.

DEALLOCATE { имя_курсора |

@имя_переменной_курсора }

Для контроля достижения конца курсора рекомендуется применять функцию: @@FETCH_STATUS

Функция @@FETCH_STATUS возвращает:

0, если выборка завершилась успешно;

-1, если выборка завершилась неудачно вследствие попытки выборки строки, находящейся за пределами курсора ;

-2, если выборка завершилась неудачно вследствие попытки обращения к удаленной или измененной строке.

Тема 18. Хранимый код. Триггеры.

Цель лекции. Изучение методов разработки и использования триггеров.

План

1. Определение триггера
2. Назначение триггер
3. Основной формат команды CREATE TRIGGER
4. Реализация триггеров в среде MS SQL Server
5. Типы триггеров
6. Программирование триггера

Краткое содержание

Определение триггера в стандарте языка SQL

Триггеры являются одной из разновидностей хранимых процедур. Их исполнение происходит при выполнении для таблицы какого-либо оператора языка манипулирования данными (DML). Триггеры используются для проверки целостности данных, а также для отката транзакций.

Триггер – это откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определенных событий внутри реляционной базы данных. Применение триггеров большей частью весьма удобно для пользователей базы данных. И все же их использование часто связано с дополнительными затратами ресурсов на операции ввода/вывода. В том случае, когда тех же результатов (с гораздо меньшими непроизводительными затратами ресурсов) можно добиться с помощью хранимых процедур или прикладных программ, применение триггеров нецелесообразно.

Основной формат команды CREATE TRIGGER:

<Определение_триггера>::=

```
CREATE TRIGGER имя_триггера
BEFORE | AFTER <триггерное_событие>
ON <имя_таблицы>
[REFERENCING
  <список_старых_или_новых_псевдонимов>]
[FOR EACH { ROW | STATEMENT}]
```

[WHEN(условие_триггера)]

<тело_триггера>

триггерные события состоят из вставки, удаления и обновления строк в таблице. В последнем случае для триггерного события можно указать конкретные имена столбцов таблицы. Время запуска триггера определяется с помощью ключевых слов BEFORE (триггерзапускается до выполнения связанных с ним событий) или AFTER (после их выполнения).

Выполняемые триггером действия задаются для каждой строки (FOR EACH ROW), охваченной данным событием, или только один раз для каждого события (FOR EACH STATEMENT).

Обозначение <список_старых_или_новых_псевдонимов> относится к таким компонентам, как старая или новая строка (OLD / NEW) либо старая или новая таблица (OLD TABLE / NEW TABLE). Ясно, что старые значения не применимы для событий вставки, а новые – для событий удаления.

Реализация триггеров в среде MS SQL Server

В реализации СУБД MS SQL Server используется следующий оператор создания или изменения триггера:

<Определение_триггера>::=

{CREATE | ALTER} TRIGGER имя_триггера

ON {имя_таблицы | имя_представления }

[WITH ENCRYPTION]

{

{ { FOR | AFTER | INSTEAD OF }

{ [DELETE] [,] [INSERT] [,] [UPDATE] }

[WITH APPEND]

[NOT FOR REPLICATION]

AS

sql_оператор[...n]

} |

{ {FOR | AFTER | INSTEAD OF } { [INSERT] [,]

[UPDATE] }

[WITH APPEND]

[NOT FOR REPLICATION]

AS

{ IF UPDATE(имя_столбца)

[{AND | OR} UPDATE(имя_столбца)] [...n]

|

IF (COLUMNS_UPDATES(){оператор_бит_обработки}

бит_маска_изменения)

{оператор_бит_сравнения }бит_маска [...n]}

sql_оператор [...n]

}

}

Триггер может быть создан только в текущей базе данных, но допускается обращение внутри триггера к другим базам данных, в том числе и расположенным на удаленном сервере.

Рассмотрим назначение аргументов из команды CREATE | ALTER TRIGGER.

При указанииаргумента WITH ENCRYPTION сервер выполняет шифрование кода триггера, чтобы никто, включая администратора, не мог получить к нему доступ и прочитать его. Шифрование часто используется для скрытия авторских алгоритмов обработки данных, являющихся интеллектуальной собственностью программиста или коммерческой тайной.

Типы триггеров

В SQL Server существует два параметра, определяющих поведение триггеров:

– AFTER. Триггер выполняется после успешного выполнения вызвавших его команд. Если же команды по какой-либо причине не могут быть успешно завершены, триггер не выполняется. Следует отметить, что изменения данных в результате выполнения запроса пользователя и выполнение триггера осуществляется в теле одной транзакции: если произойдет откат триггера, то будут отклонены и пользовательские изменения. Можно определить несколько AFTER -триггеров для каждой операции (INSERT, UPDATE, DELETE). Если для таблицы предусмотрено выполнение нескольких AFTER -триггеров, то с помощью системной хранимой процедуры sp_settriggerorder можно указать, какой из них будет выполняться первым, а какой последним. По умолчанию в SQL Server все триггеры являются AFTER -триггерами.

– INSTEAD OF. Триггер вызывается вместо выполнения команд. В отличие от AFTER -триггера INSTEAD OF -триггер может быть определен как для таблицы, так и для представления. Для каждой операции INSERT, UPDATE, DELETE можно определить только один INSTEAD OF -триггер.

Триггеры различают по типу команд, на которые они реагируют.

Существует три типа триггеров:

– INSERT TRIGGER – запускаются при попытке вставки данных с помощью команды INSERT.

– UPDATE TRIGGER – запускаются при попытке изменения данных с помощью команды UPDATE.

– DELETE TRIGGER – запускаются при попытке удаления данных с помощью команды DELETE.

Конструкции [DELETE] [,] [INSERT] [,] [UPDATE] и FOR | AFTER | INSTEAD OF } { [INSERT] [,] [UPDATE] определяют, на какую команду будет реагировать триггер. При его создании должна быть указана хотя бы одна команда. Допускается создание триггера, реагирующего на две или на все три команды.

Аргумент WITH APPEND позволяет создавать несколько триггеров каждого типа.

При создании триггера с аргументом NOT FOR REPLICATION запрещается его запуск во время выполнения модификации таблиц механизмами репликации.

Конструкция AS sql_оператор[...n] определяет набор SQL- операторов и команд, которые будут выполнены при запуске триггера.

Отметим, что внутри триггера не допускается выполнение ряда операций, таких, например, как:

– создание, изменение и удаление базы данных;

– восстановление резервной копии базы данных или журнала транзакций.

Программирование триггера

При выполнении команд добавления, изменения и удаления записей сервер создает две специальные таблицы: inserted и deleted . В них содержатся списки строк, которые будут вставлены или удалены по завершении транзакции. Структура таблиц inserted и deletedидентична структуре таблиц, для которой определяется триггер. Для каждого триггера создается свой комплект таблиц inserted и deleted, поэтому никакой другой триггер не сможет получить к ним доступ. В зависимости от типа операции, вызвавшей выполнение триггера, содержимое таблиц inserted и deleted может быть разным:

– команда INSERT – в таблице inserted содержатся все строки, которые пользователь пытается вставить в таблицу; в таблице deleted не будет ни одной строки; после завершения триггера все строки из таблицы inserted переместятся в исходную таблицу;

– команда DELETE – в таблице deleted будут содержаться все строки, которые пользователь попытается удалить; триггер может проверить каждую строку и определить, разрешено ли ее удаление; в таблице inserted не окажется ни одной строки;

– команда UPDATE – при ее выполнении в таблице deleted находятся старые значения строк, которые будут удалены при успешном завершении триггера. Новые значения

строк содержатся в таблице inserted. Эти строки добавятся в исходную таблицу после успешного выполнения триггера.

Для получения информации о количестве строк, которое будет изменено при успешном завершении триггера, можно использовать функцию @@ROWCOUNT; она возвращает количество строк, обработанных последней командой. Следует подчеркнуть, что триггер запускается не при попытке изменить конкретную строку, а в момент выполнения команды изменения. Одна такая команда воздействует на множество строк, поэтому триггер должен обрабатывать все эти строки.

Для получения списка столбцов, измененных при выполнении команд INSERT или UPDATE, вызвавших выполнение триггера, можно использовать функцию COLUMNS_UPDATED(). Она возвращает двоичное число, каждый бит которого, начиная с младшего, соответствует одному столбцу таблицы (в порядке следования столбцов при создании таблицы). Если бит установлен в значение "1", то соответствующий столбец был изменен. Кроме того, факт изменения столбца определяет и функция UPDATE (имя_столбца).

Для удаления триггера используется команда

DROP TRIGGER {имя_триггера} [,...n]

Тема 19. Характеристика технологии ADO.NET

Цель лекции. Рассмотрение технологии доступа к данным ADO.NET, характеристики ее объектной модели..

План

1. Характеристика Microsoft .NET
2. Определение технологии ADO.NET
3. Преимущества ADO.NET
4. Компоненты ADO.NET
5. Объекты ADO.NET
6. Схема доступа к данным

Краткое содержание

В состав Microsoft .NET входят следующие основные компоненты: библиотека классов .NET: языки .NET; общая языковая среда исполнения (CLR), интегрированная среда разработки приложений, например, Visia) Studio .NET.

ADO.NET (ActiveX Data Object.NET) - набор классов, используемый для доступа к источникам данны$_{л}$ в платформе .NET. ADO.NET представляет собой ноьую объектную > модель, которая использует стандарт XM L для передачи данных. В ADO.NET реализована

идея использования *orrn oedime.ltiux* наборов данных, причем такой способ работы является основным. По сравнению с ADO, ADO.NET предполагает более легкое программирование, .'(учшую производительность и масштабирование, меньшую зависимость от источников данш іх.

ADO.NET можно использовать для доступа к реляционным СУБД, таким как SQL Server 2000, и ко многим дополнительным источникам данных, для работы с которыми предназначен провайдер OLE DB.

Преимущества ADO.NET. *Масштабируемость*. Независимость от источника данных. Способность к взаимодействию.

Основные пространства имен, которые используются в ADO.NET:

– System.Data: определяет классы, интерфейсы, делегаты, которые реализуют архитектуру ADO.NET

– System.Data.Common: содержит классы, общие для всех провайдеров ADO.NET

– System.Data.Design: определяет классы, которые используются для создания своих собственных наборов данных

– System.Data.Odbc: определяет функциональность провайдера данных для ODBC

– System.Data.OleDb: определяет функциональность провайдера данных для OLE DB

– System.Data.Sql: хранит классы, которые поддерживают специфичную для SQL Server функциональность

– System.Data.OracleClient: определяет функциональность провайдера для баз данных Oracle

– System.Data.SqlClient: определяет функциональность провайдера для баз данных MS SQL Server

– System.Data.SqlServerCe: определяет функциональность провайдера для SQL Server Compact 4.0

– System.Data.SqlTypes: содержит классы для типов данных MS SQL Servera

– Microsoft.SqlServer.Server: хранит компоненты для взаимодействия SQL Server и среды CLR

Функционально классы ADO.NET можно разбить на два уровня: подключенный и отключенный. Каждый провайдер данных .NET реализует свои версии объектов Connection, Command, DataReader, DataAdapter и ряда других, который составляют подключенный уровень. То есть с помощью них устанавливается подключение к БД и выполняется с ней взаимодействие.

Другие классы, такие как DataSet, DataTable, DataRow, DataColumn и ряд других составляют отключенный уровень, так как после извлечения данных в DataSet мы можем рабо-

тать с этими данными независимо от того, установлено ли подключение или нет. То есть после получения данных из БД приложение может быть отключено от источника данных.

Тема 20. Объекты ADO.NET

Цель. Рассмотрение объектов ADO.NET и приемов работы с ними

План

Понятие объектной модели ADO .NET

Характеристика объектов

Краткое содержание

Объектная модель ADO .NET предполагает существование (при написании приложения для работы с базой данных — использование) двух множеств классов, выполняющих четко определенные задачи при работе с базой данных:

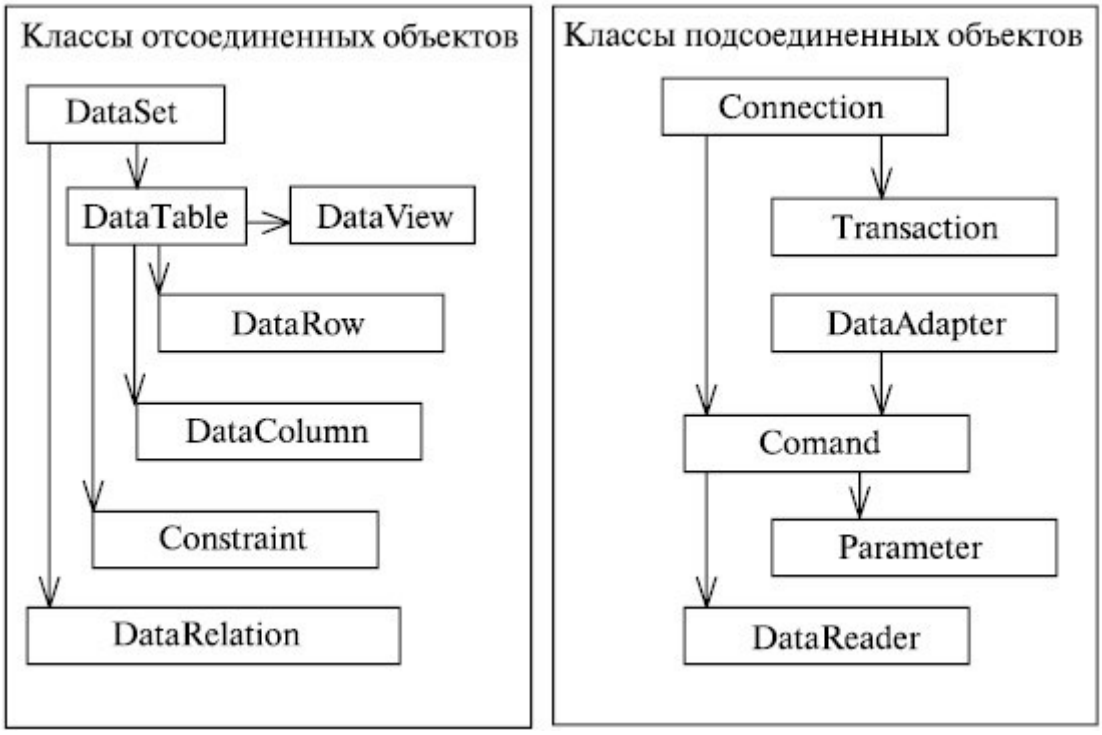


Рисунок 6 – Объектная модель ADO .NET

Классы подсоединенных объектов обеспечивают установление соединения с базой данных и управление базой со стороны приложения; классы отсоединенных объектов обеспечивают сохранение, использование и преобразование полученной от базы данных информации на стороне приложения.

Таблицы и поля (объекты DataTable и DataColumn)

Объекты DataTable используются для представления таблиц в DataSet. DataTable представляет одну таблицу из базы данных. В свою очередь, DataTable составляется из объектов DataColumn.

DataColumn - это блок для создания схемы DataTable. Каждый объект DataColumn имеет свойство DataType, которое определяет тип данных, содержащихся в каждом объекте DataColumn. Например, вы можете ограничить тип данных до целых, строковых и десятичных чисел. Поскольку данные, содержащиеся в DataTable, обычно переносятся обратно в исходный источник данных, вы должны согласовывать тип данных с источником.

Объекты DataRelation

Объект DataSet имеет также свойство Relations, возвращающее коллекцию DataRelationCollection, которая в свою очередь состоит из объектов DataRelation. Каждый объект DataRelation выражает отношение между двумя таблицами (сами таблицы связаны по какому-либо полю (столбцу). Следовательно, эта связь осуществляется через объект DataColumn).

Строки (объект DataRow)

Коллекция Rows объекта DataTable возвращает набор строк (записей) заданной таблицы. Эта коллекция используется для изучения результатов запроса к базе данных. Мы можем обращаться к записям таблицы как к элементам простого массива.

DataAdapter

DataSet - это специализированный объект, содержащий образ базы данных. Для осуществления взаимодействия между DataSet и источником данных используется объект типа DataAdapter. Само название этого объекта - адаптер, преобразователь, - указывает на его природу. DataAdapter содержит метод Fill() для обновления данных из базы и заполнения DataSet.

Объекты DBConnection и DBCommand

Объект DBConnection осуществляет связь с источником данных. Эта связь может быть одновременно использована несколькими командными объектами. Объект DBCommand позволяет послать базе данных команду (как правило, команду SQL или хранимую процедуру). Объекты DBConnection и DBCommand иногда создаются неявно в момент создания объекта DataSet, но их также можно создавать явным образом.

Тема 21. Разработка приложения в C#

Цель . Рассмотреть принципы проектирования и разработки программного приложения для работы с БД

План

Структура приложения баз данных.

Базовые компоненты, используемые при разработке приложений баз данных, и их взаимосвязь;

Понятие набора данных и его участие в основных механизмах приложения баз данных;

Модуль данных;

Программная реализация частей приложения баз данных

Тема 24. Архитектура системы баз данных

Цель . Рассмотреть принципы проектирования и разработки программного приложения для работы с БД

План

1. Понятие Архитектуры
2. Характеристика уровней архитектуры

Краткое содержание

Архитектура системы баз данных. Под архитектурой системы понимают совокупность ее основных функциональных компонентов, а также средств обеспечения их взаимодействия друг с другом пользователями и системным персоналом. Одно из наиболее важных функций систем баз данных, оказавших решающее влияние на формирование сложившегося в наши дни подхода к архитектуре является обеспечение возможностей абстракции данных. Абстракции данных, предоставляемые системой служат средством поддержки независимости способ ведения БД различными группами пользователей (это свойство системы называется независимостью данных). В системах обычно имеют дело с уровнями абстракции и часто эти уровни выстраиваются в некоторую иерархию. Функциональны компонент системы, механизмы которого служат для поддержки некоторого уровня абстракции называется архитектурным уровнем. В результате простейшего анализа в системе можно выделить 2 основных уровня абстракции: логический и физический.

Внутренний уровень (называемый также *физическим)* наиболее близок к физиче скому хранилищу информации, т.е. связан со способами сохранения информации на физических устройствах.

Внешний уровень (называемый также *пользовательским логическим)* наиболее бли зок к пользователям, т.е. связан со способами представления данных для отдель ных пользователей.

Концептуальный уровень (называемый также *общим логическим* или просто *логиче ским,* без дополнительного определения) является "промежуточным" уровнем ме жду двумя первыми.

Тема 23. Распределенные системы управления базами данных

Цель . Рассмотреть основные понятия и определения распределенных баз данных

План

1. Понятие распределенных баз данных
2. Свойства распределенных баз данных
3. Целостность данных
4. Средства повышения эффективности

Краткое содержание

Под распределенной (Distributed DataBase - DDB) обычно подразумевают базу данных, включающую фрагменты из нескольких баз данных, которые располагаются на различных узлах сети компьютеров, и, возможно управляются различными СУБД. Распределенная база данных выглядит с точки зрения пользователей и прикладных программ как обычная локальная база данных. В этом смысле слово "распределенная" отражает способ организации базы данных, но не внешнюю ее характеристику. ("распределенность" базы данных невидима извне).

Определение распределенных баз данных (DDB) предложил Дэйт (C.J. Date). Он установил 12 свойств или качеств идеальной DDB:

- Локальная автономия (local autonomy)
- Независимость узлов (no reliance on central site)
- Непрерывные операции (continuous operation)
- Прозрачность расположения (location independence)
- Прозрачная фрагментация (fragmentation independence)
- Прозрачное тиражирование (replication independence)
- Обработка распределенных запросов (distributed query processing)

- Обработка распределенных транзакций (distributed transaction processing)
- Независимость от оборудования (hardware independence)
- Независимость от операционных систем (operationg system independence)
- Прозрачность сети (network independence)
- Независимость от баз данных (database independence)

В DDB поддержка целостности и согласованности данных, ввиду свойств DDB, представляет собой сложную проблему. Ее решение - синхронное и согласованное изменение данных в нескольких локальных базах данных, составляющих DDB - достигается применением протокола двухфазной фиксации транзакций. Если DDB однородна - то есть на всех узлах данные хранятся в формате одной базы и на всех узлах функционирует одна и та же СУБД, то используется механизм двухфазной фиксации транзакций данной СУБД.

Высокая степень эффективности системы является одним из наиболее ключевых требований к распределенным системам управления базами данных вообще и к System R* в частности. Для достижения этой цели используются два основных приема.

Во-первых, как и в System R, в System R* выполнению запроса предшествует его компиляция. В ходе этого процесса производится поиск употребляемых в запросе имен объектов баз данных в распределенном каталоге и замена имен на внутренние идентификаторы; проверка прав доступа пользователя, от имени которого производится компиляция, на выполнение соответствующих операций над базами данных и выбор наиболее оптимального глобального плана выполнения запроса, который затем подвергается декомпозиции и по частям рассылается в соответствующие узлы сети, где производится выбор оптимальных локальных планов выполнения компонентов запроса и происходит генерация модулей доступа в машинных кодах.

Вторым средством повышения эффективности системы является возможность перемещения удаленных отношений в локальную базу данных.

Тема 24. Методы поддержки распределенных данных.

Цель. Рассмотреть основные понятия и определения распределенных баз данных

План

1. Фрагментация.
2. Репликация.
3. Распределенные запросы.
4. Распределенные транзакции.
5. Распределенные ограничения целостности.

Краткое содержание

Фрагментация – это разбиение базы данных на фрагменты и размещение их по разным узлам сети. Фрагментация является основным способом организации РБД. Она позволяет хранить данные на том узле, где они наиболее часто используются. Основные проблемы, которые при этом возникают – это прозрачность написания запросов к данным и, возможно, поддержка распределенных ограничений целостности. Фрагментации могут подвергаться не только база данных в целом, но и отдельные отношения БД.

Схема фрагментации отношения должна удовлетворять трем условиям:

– Полнота: если отношение R разбивается на фрагменты R_1, R_2,…, R_n, то $UR_i = R$

– (Каждый кортеж должен входить хотя бы в один фрагмент).

– Восстановимость: должна существовать операция реляционной алгебры, позволяющая восстановить отношение R из его фрагментов. Это правило гарантирует сохранение функциональных зависимостей.

– Непересекаемость: если элемент данных $d_j \subset R_i$, то он не должен присутствовать одновременно в других фрагментах. Исключение составляет первичный ключ при вертикальной фрагментации. Это правило гарантирует минимальную избыточность данных.

Существуют два основных типа фрагментации отношений: горизонтальная и вертикальная. Горизонтальный фрагмент является результатом селекции (selection), а вертикальный – результатом проекции (projection).

Репликация – это поддержание двух и более идентичных копий (реплик) данных на разных узлах РБД. Реплика может включать всю базу данных (полная репликация), одно или несколько взаимосвязанных отношений или фрагмент отношения. Также возможен вариант *с консолидацией данных,*при котором каждый узел владеет своей частью данных (например, отношения) и может ее обновлять, а на одном из узлов РБД эти части соединяются (или объединяются) вместе для образования консолидированной копии "только для чтения" (readonly).

К основным достоинствам механизма репликации можно отнести повышение доступности и надежности данных и повышение локализации ссылок на реплицируемые данные. Основным недостатком репликации является сложность поддержания идентичности реплик: если в одну копию данных вносятся изменения, то эти изменения также должны быть внесены в другие копии. Это называется распространение изменений и реализуется службой тиражирования.

Служба тиражирования должна выполнять следующие функции:

1. Обеспечение масштабируемости, т.е. эффективная обработки больших и малых объемов данных.

2. Преобразование типов и моделей данных (для гетерогенных РБД).

3. Репликация объектов БД, например, индексов, триггеров и т.п.

4. Инициализация вновь создаваемой реплики.

5. Обеспечение возможности "подписаться" на существующие реплики, чтобы получать их в определенной периодичностью.

6. Распределенным называется запрос, который обращается к двум и более узлам РБД, но не обновляет на них данные. Запрашивающий узел должен определить, что в запросе идет обращение к данным на другом узле, выделить подзапрос к удаленному узлу и перенаправить его этому узлу.

7. Самой сложной проблемой выполнения распределенных запросов является оптимизация, т.е. поиск оптимального плана выполнения запроса. Информация, которая требуется для оптимизации запроса, распределена по узлам. Если выбрать центральный узел, который соберет эту информацию, построит оптимальный план и отправит его на выполнение, то теряется свойство локальной автономности. Поэтому обычно распределенный запрос выполняется так: запрашивающий узел собирает все данные, полученные в результате выполнения подзапросов, у себя, и выполняет их соединение (или объединение), что может занять очень много времени.

Распределенные транзакции обращаются к двум и более узлам и обновляют на них данные. Основная проблема распределенных транзакций – соблюдение логической целостности данных. Транзакция на всех узлах должна завершиться одинаково: или фиксацией, или откатом. Выполнение распределенных транзакций осуществляется с помощью специального алгоритма, который называется двухфазная фиксация (подробнее об управлении распределенными транзакциями см. ниже).

Распределенные ограничения целостности возникают тогда, когда для проверки соблюдения какого-либо ограничения целостности системе необходимо обратиться к другому узлу. Например, база данных разделена на фрагменты таким образом, что родительская таблица находится на одном узле, а дочерняя, связанная с ней по внешнему ключу, – на другом. При добавлении записи в дочернюю таблицу система обратится к узлу, на котором расположена родительская таблица, для проверки наличия соответствующего значения ключа. Или случай разбиения одной таблицы на фрагменты и размещение этих фрагментов по разным узлам сети. Здесь будет необходима проверка соблюдения ограничения первичного ключа.

Тема 25. Распределенные запросы

Цель . Рассмотреть принципы разработки распределенных запросов к DDB

План

1. Понятие распределенного запроса.
2. Проектирование распределенного запроса

Краткое содержание

Можно использовать сервер баз данных, чтобы запрашивать и обновлять несколько баз данных на нескольких серверах баз данных одного и того же экземпляра сервера баз данных (распределенные запросы для разных серверов) и на нескольких экземплярах сервера (распределенные запросы для разных серверов). Запросы такого типа называются *распределенными запросами*. Этот термин не ограничивается операторами SELECT и часто используется для более общего обозначения любой операции DML или выполнения подпрограммы, которая возвращает объекты или ссылается на объекты вне локальной базы данных. В распределенных запросах для разных серверов серверы баз данных могут находиться на одном хост-компьютере, в одной и той же сети или на шлюзе.

Утилита Management Studio не поддерживает графические методы инициализации распределенных запросов. Не существует методов перетаскивания подключенного сервера или удаленной таблицы в конструктор запросов. Однако распределенный запрос можно ввести вручную на панели SQL (рис. 15.5), а затем выполнить его.

Рисунок 7 – Распределенный запрос можно выполнить в Management Studio, если ввести его вручную на панели SQL ([XPS].Family.dbo.Person)

Распределенные представления

Представления — это сохраненные инструкции SQL. Хотя я не рекомендую основывать приложения клиент/сервер на представлениях, они могут оказаться полезными для текущих запросов. Так как большая часть пользователей (в том числе и разработчики) не знакома с различными методами выполнения распределенных запросов, помещение такового в представление может оказаться удачным решением проблемы.

Локальные распределенные запросы

Хотя термин локальный распределенный запрос звучит странно, все не так уж сложно. Это запрос, который собирает внешние данные в SQL Server, а затем выполняет запрос на локальном сервере. Так как обработка таких запросов выполняется на локальном сервере, в них используется синтаксис T-SQL, и поэтому их иногда называют локальными запросами T-SQL.

Использование четырехкомпонентного имени

Если данные находятся на другом экземпляре SQL Server, то полный синтаксис четырехкомпонентного имени следующий:

Сервер. База_данных. Схема . Имя_Объекта

Четырехкомпонентное имя может использоваться в любых запросах извлечения или модификации данных. На моем компьютере существует второй экземпляр SQL Server с именем [XPS\Yukon]. Имя владельца объекта является обязательным, если обращение осуществляется к внешнему серверу.

Следующий запрос извлекает таблицу Person из экземпляра SQL2:

```
SELECT LastName, FirstName
FROM [XPS\Yukon].Family.dbo.person
```

Результат запроса следующий:

```
LastName FirstName
Halloway Kelly Halloway James
```

При выполнении инструкций INSERT, UPDATE и DELETE в качестве распределенных запросов для имени таблицы можно использовать либо четырехкомпонентную форму, либо функцию распределенного запроса. В качестве примера приведем следующий код, который можно взять из файла CHA2_Convert. sql и который заполняет учебную базу данных CHA2. В этом примере в качестве источника данных для инструкции INSERT использовано четырехкомпонентное имя таблицы. Этот запрос извлекает названия базовых лагерей из электронной таблицы Excel и вставляет их в SQL Server:

```
INSERT BaseCamp(Name)
SELECT DISTINCT [Base Camp]
FROM CHAl_Schedule…[Base_Camp]
WHERE [Base Camp] IS NOT NULL
```

Совет Если вы уже выполняли сценарий CHA2_Convert. sql и заполнили свою копию базы CHA2, еще раз запустите сценарий CHA2_Create. sql, чтобы начать работу с пустой базы данных.

Методические указания к практическим занятиям;

Практическое занятие №1

Концептуальная модель данных

Цель занятия. Изучение методологий построения концептуально инфологической модели.

Нотации построения модели «сущность - связь» по методу Ричарда Баркера.

Одной из наиболее распространенных разновидностей нотации ERD является нотация, предложенная Ричардом Баркером, автором методов, используемых в технологии создания ПО фирмы Oracle.

Сущность - реальный либо воображаемый объект, имеющий существенное значение для рассматриваемой предметной области, информация о котором подлежит хранению. Каждая сущность должна обладать уникальным идентификатором. Каждый экземпляр сущности должен однозначно идентифицироваться и отличаться от всех других экземпляров данного типа сущности. Каждая сущность должна обладать некоторыми свойствами:

-каждая сущность должна иметь уникальное имя, и к одному и тому же имени должна всегда применяться одна и та же интерпретация. Одна и та же интерпретация не может применяться к различным именам, если только они не являются псевдонимами;

-сущность обладает одним или несколькими атрибутами, которые либо принадлежат сущности, либо наследуются через связь;

-сущность обладает одним или несколькими атрибутами, которые однозначно идентифицируют каждый экземпляр сущности;

-каждая сущность может обладать любым количеством связей с другими сущностями модели.

Связь - это ассоциация между сущностями, при которой, как правило, каждый экземпляр одной сущности, называемой родительской сущностью, ассоциирован с произвольным (в том числе нулевым) количеством экземпляров второй сущности, называемой сущностью-потомком, а каждый экземпляр сущности-потомка ассоциирован в точности с одним экземпляром сущности-родителя. Таким образом, экземпляр сущности-потомка может существовать только при существовании сущности родителя.

Связи может даваться имя, выражаемое грамматическим оборотом глагола и помещаемое возле линии связи. Имя каждой связи между двумя данными сущностями должно быть уникальным, но имена связей в модели не обязаны быть уникальными. Имя связи всегда формируется с точки зрения родителя, так что предложение может быть образовано соединением имени сущности-родителя, имени связи, выражения степени и имени сущности-потомка.

Степень связи и обязательность графически изображаются как на рис. 8.

Рисунок 8 – Степени связи

Атрибут - любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибут может быть либо обязательным, либо необязательным. Обязательность означает, что атрибут не может принимать неопределенных значений (null values). Атрибут может быть либо описательным (т.е.

обычным дескриптором сущности), либо входить в состав уникального идентификатора (первичного ключа).

Уникальный идентификатор - это атрибут или совокупность атрибутов и/или связей, предназначенная для уникальной идентификации каждого экземпляра данного типа сущности. В случае полной идентификации каждый экземпляр данного типа сущности полностью идентифицируется своими собственными ключевыми атрибутами, в противном случае в его идентификации участвуют также атрибуты другой сущности-родителя .

Каждый атрибут идентифицируется уникальным именем, выражаемым грамматическим оборотом существительного, описывающим представляемую атрибутом характеристику. Атрибуты изображаются в виде списка имен внутри блока ассоциированной сущности, причем каждый атрибут занимает отдельную строку. Атрибуты, определяющие первичный ключ, размещаются наверху списка и выделяются знаком "#".

Каждая сущность должна обладать хотя бы одним возможным ключом. Возможный ключ сущности - это один или несколько атрибутов, чьи значения однозначно определяют каждый экземпляр сущности. При существовании нескольких возможных ключей один из них обозначается в качестве первичного ключа, а остальные - как альтернативные ключи. Помимо перечисленных основных конструкций модель данных может содержать ряд дополнительных.

Пример:

Метод Баркера будет излагаться на примере моделирования деятельности компании по торговле автомобилями. Выделяются 4 сущности (автомашина, продавец, покупатель, контракт).

Связь продавца с контрактом может быть выражена следующим образом:

продавец может получить вознаграждение за 1 или более контрактов;

контракт должен быть инициирован ровно одним продавцом.

2 предложения, описывающие связь продавца с контрактом, графически будут выражены как на рис.9.

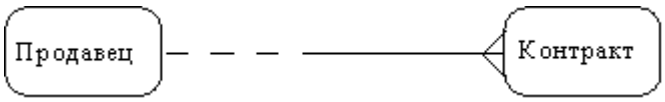


Рисунок 9– Связь продавца с контрактом

Описав также связи остальных сущностей, получим схему, представленную на рис.10.

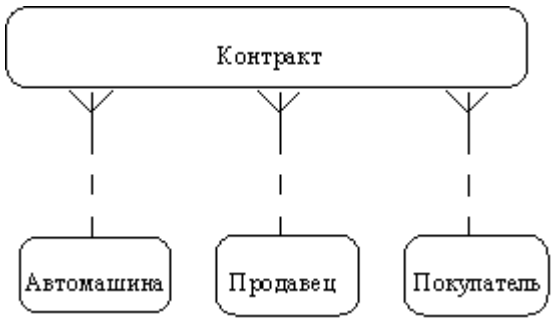


Рисунок 10– Пример нескольких связей

Добавим атрибуты и идентификаторы (рис.11)

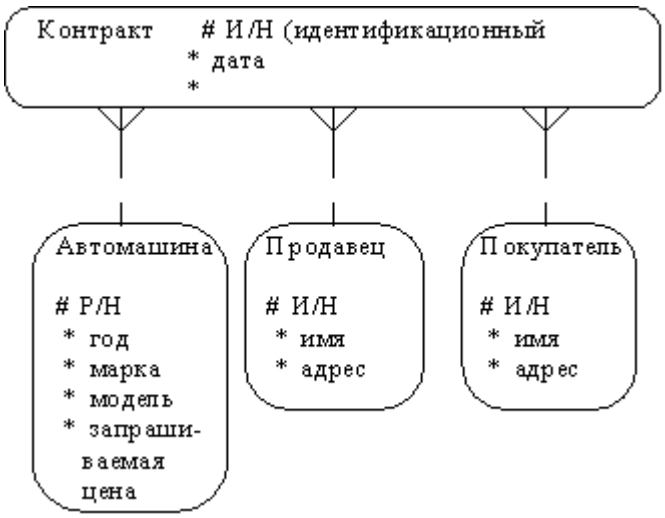


Рисунок 11– Диаграмма сущность - связь.

Задание

Предприятие имеет ремонтную мастерскую, в которой производится различного вида ремонт вычислительной техники и оргтехники. Необходимо автоматизировано вести учет, сданной техники. Для каждой единицы техники необходимо хранить ее инвентарный номер, нахвание устройства, модель и год выпуска. Необходимо также хранить дату сдачи единицы техники, вид ремонта, срок ремонта, перечень необходимых запасных частей, их стоимость. Система должна позволять получать отчетность за произвольный период о том, какая техника была отремонтирована и сколько стоили эти ремонты.

Контрольные вопросы.

1. Что такое концептуально инфологическая модель?
2. Перечислить нотации построения модели «сущность - связь»
3. Что относится к основным элементам концептуальной инфологической модели?
4. Чем отличается нотация Чена от нотации Баркера?
5. Как определяется атрибут, связь, сущность?

Практическое занятие

Логическая модель данных

Цель занятия. Освоить методику построения логической модели данных на основе инфологической модели.

Алгоритм построения логической модели данных.

1. Выполнить анализ связей между сущностями и построение внешних ключей и отношений.
2. Разрешить связи «многие-ко-многим».
3. Выполнить анализ связей «один-ко-одному».
4. Выполнить анализ полученных отношений с целью объединения отношений, отличающихся наличием внешних ключей.
5. Провести нормализацию отношений.
6. Построить логическую модель базы данных.

Задание

Представлена инфологическая модель БД. Выполнить логическое проектирование БД..

Таблица 1 – Спецификация атрибутов сущности «Клиент»

Название атрибута	Описание атрибута	Тип данных	Диапазон значений	Пример атрибута
1	2	3	4	5
Код	Число, однозначно определяющее каждого клиента.	Числовой	> 0	4587
Наименование	Наименование предприятия-аутсорсера	Текст	–	ООО «Автолада»
Расчётный счёт	Расчётный счёт предприятия-аутсорсера	Текст	–	90069450938598566
Название банка	Название банка, где открыт счет предприятия	Текст	-	ВТБ 24
БИК	Банковский реквизит предприятия	Текст	–	046311837
Корреспондентский счет	Банковский реквизит предприятия	Текст	–	90105810404067640002
ИНН	Индивидуальный номер налогоплательщика	Текст	–	7707284568
Код ОКВЭВ	Банковский реквизит предприятия	Текст	–	76502179
Руководитель	Фамилия и инициалы руководителя предприятия	Текст	-	Головин В.Н.
Адрес 1	Область, где располагается предприятие	Текст	–	Амурская обл.
Адрес 2	Город, где располагается предприятие	Текст	–	Благовещенск
Адрес 3	Улица и номер дома, где располагается предприятие	Текст	–	Ул. Амурская 118
Телефон	Телефон предприятия для контактов	Текст	–	8(4162)582694

Таблица 2 – Спецификация атрибутов сущности «Договор»

Название атрибута	Описание атрибута	Тип данных	Диапазон значений	Пример атрибута
1	2	3	4	5
Номер	Номер договора	Числовой	> 0	4765
Дата заключения	Дата заключения договора	Дата	>01.01.2010	16.04.2015
Начало	Дата начала действия договора	Дата	>01.01.2010	16.04.2016
Окончание	Дата окончания действия договора	Дата	>01.01.2010	16.04.2017

Абонентская плата	Абонентская плата	Денежный	>=0	3000

Таблица 3– Спецификация атрибутов сущности «Услуга»

Название атрибута	Описание атрибута	Тип данных	Диапазон значений	Пример атрибута
Номер	Номер услуги	Числовой	> 0	271
Дата принятия	Дата принятия	Дата	>01.01.2010	16.04.2015
Дата выполнения	Дата выполнения услуги	Дата	>01.01.2010	16.04.2015

Таблица 4 – Спецификация атрибутов сущности «Счет»

Название атрибута	Описание атрибута	Тип данных	Диапазон значений	Пример атрибута
Номер	Номер счета	Числовой	> 0	7118
Дата	Дата выставления счета клиенту	Дата	>01.01.2010	16.04.2015
Дата оплаты	Дата оплаты счета клиентом	Дата	>01.01.2010	16.04.2015
Сумма	Сумма для оплаты	Денежный	>0	2500.00

Таблица 5 – Спецификация атрибутов сущности «Документ»

Название атрибута	Описание атрибута	Тип данных	Диапазон значений	Пример атрибута
1	2	3	4	5
Номер	Номер документа	Числовой	> 0	918
Наименование	Наименование документа	Текст	–	Техническое
Вид	Вид документа	Текст	Исходный Результат	Результат
Характеристика	Краткое описание документа	Текст	–	Требования к системе пожарной безопасности

Таблица 6– Спецификация атрибутов сущности «Сотрудник»

Название атрибута	Описание атрибута	Тип данных	Диапазон значений	Пример атрибута
Код	Код сотрудника	Числовой	> 0	4
Фамилия	Фамилия сотрудника	Текст	–	Ванько
Имя	Имя сотрудника	Текст	–	Семен
Отчество	Отчество сотрудника	Текст	-	Викторович
Должность	Должность сотрудника	Текст	Менеджер Начальник	Менеджер

			Инженер Старший менмджер Управля-ющий	
Отдел	Отдел, где работает со-трудник	Текст	Отдел по работе с клиенами Бухгалте-рия, про-ектный.	Проектный отдел
Телефон	Телефон сотрудника	Текст	–	89145982537

Таблица 7 – Спецификация атрибутов сущности «Вид услуги»

Название атрибута	Описание атрибута	Тип дан-ных	Диапазон значений	Пример атрибута
Код	Код услуги	Числовой	> 0	918
Наименование	Наименование услуги	Текст	–	Разработка плана ме-ропр.
Стоимость	Стоимость услуги	Денежный	>=0	1500

Рисунок 12 – Связь «Клиент – Договор»

Рисунок 13 – Связь « Договор – Услуга»

Рисунок 14 – Связь «Договор – Счет»

Рисунок 15 – Связь «Договор – Счет»

Рисунок 16 – Связь «Услуга – Документ»

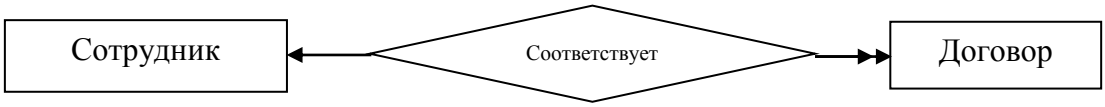


Рисунок 17 – Связь «Сотрудник – Договор»

.

Рисунок 18 – Связь «Сотрудник – Услуга»

Рисунок 19– Связь «Вид услуги – Услуга»

Контрольные вопросы

1. Как разрешается связь «многие-ко-многим»?
2. Как можно избавиться от связи «один-ко-многим»?
3. Что такое внешний ключ?
4. Как происходит миграция ключей?
5. Что такое нормальные формы?

Практическое занятие

Нормальные формы. Нормализация таблиц.

Цель занятия. Освоить метод построения нормальных форм.

Задание.

1. Построить диаграмму функциональных зависимостей для Отношения Заявка

Таблица 8 – Отношение Заявка

Номер заявки
Код клиента
Код сотрудника
Дата
Дата завершения
Договор
Марка автомобиля
Модель автомобиля
Год выпуска
Рег.номер авто
Проблема
Состояние
ФИО клиента
Телефон клиента
Адрес клиента
ФИО сотрудника
Телефон сотрудника
Образование сотрудника

Определить относится это Отношение к 3НФ.
Привести Отношение к 3НФ.

2. Построить диаграмму функциональных зависимостей для Отношения Сотрудник

Таблица 9 – Отношение Сотрудник

Ид.код
Код клиента
Код сотрудника
ФИО
Дата рождения
№ трудового договора
Дата заключения
Обязанности по дрговору
Должность
Оклад
№ диплома
Направление подготовки
Учебное заведение
Дата окончания
ФИО ребенка
Дата рождения

Определить относится это Отношение к 3НФ.
Привести Отношение к 3НФ.

Контрольные вопросы

1. Что такое нормализация и денормализация базы данных?
2. Для каких целей выполняется нормализация базы данных?
3. Охарактеризовать 1НФ, 2НФ, 3НФ.
4. Что такое метод нормальных форм?
5. Сформулировать теорему Хеза.

Практическое занятие

Команды манипулирования данными

Цель занятия. Освоить работу с командами манипулирования данных.

Задание.

1. Написать процедуру для увеличения разряда всех спортсменов определенного года на 1. Разряд не изменять, если он равен максимальному. Год рождения и величину максимального разряда передавать в качестве параметра.
2. Написать процедуру для дополнения результатов соревнований. Данные для заполнения – параметры процедуры.
3. Вывести все соревнования, мировой рекорд которых был установлен больше пяти лет назад.
4. Напишите запрос, который бы использовал оператор EXISTS для извлечения всех спортсменов, которые участвовали в соревнованиях в заданном городе.
5. Напишите запрос, который бы использовал оператор EXISTS для извлечения всех спортсменов, которые не участвовали в соревнованиях в заданном городе.

Рисунок 20 –Логическая схема данных

Практическое занятие

Команды выборки данных (SELECT)

Цель занятия. Освоить работу с командами выборки данных.

Задание.

Схема БД состоит из четырех отношений:
Company (ID_comp, name)
Trip(trip_no, ID_comp, plane, town_from, town_to, time_out, time_in)
Passenger(ID_psg, name)
Pass_in_trip(trip_no, date, ID_psg, place)

Таблица Company содержит идентификатор и название компании, осуществляющей перевозку пассажиров.

Таблица Trip содержит информацию о рейсах: номер рейса, идентификатор компании, тип самолета, город отправления, город прибытия, время отправления и время прибытия.

Таблица Passenger содержит идентификатор и имя пассажира.

Таблица Pass_in_trip содержит информацию о полетах: номер рейса, дата вылета (день), идентификатор пассажира и место, на котором он сидел во время полета. При этом следует иметь в виду, что
- рейсы выполняются ежедневно, а длительность полета любого рейса менее суток;
town_from <> town_to;
- время и дата учитывается относительно одного часового пояса;
- время отправления и прибытия указывается с точностью до минуты;

- среди пассажиров могут быть однофамильцы (одинаковые значения поля name, например, Bruce Willis);
- номер места в салоне – это число с буквой; число определяет номер ряда, буква (a – d) – место в ряду слева направо в алфавитном порядке;
- связи и ограничения показаны на схеме данных.

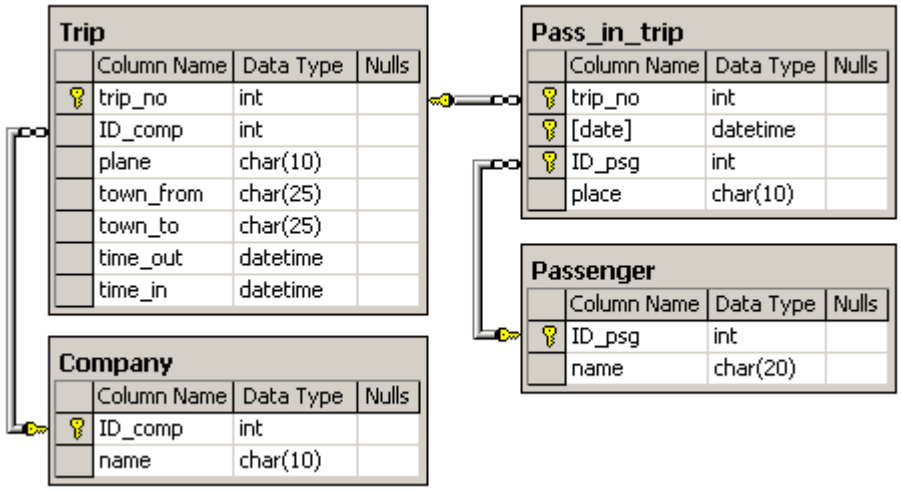


Рисунок 21 – Логическая схема данных

Запросы

1. Сформировать список пассажиров, которые летят на рейсе N.
2. Получить список рейсов, выполненных в январе 2017года
3. Получить список рейсов всех компаний, сгруппированных по названию компании.
4. Какие компании осуществляют полеты в город Владивосток.
5. Какие рейсы были совершены из Москвы в Самару 24.01.2017
6. Выдать список пассажиров с одинаковыми именами.
7. Какие типы самолетов используются в компании «Аэрофлот»
8. Получить список пассажиров прибывших в Москву 10.02.2017
9. Кто из пассажиров занимал 10 ряд на рейсе 781
10. Определить рейс с минимальным временем полета.
11. Рассчитать количество пассажиров рейса 385
12. Выявить всех пассажиров с одинаковыми именами.
13. Получить список рейсов длительность полета которых превышает 3 часа.
14. Получить список рейса 387 с указанием места
15. Сколько пассажиров было перевезено компанией «БимАвиа»
16. Вывести список компаний, которые перевезли более 100000 пассажиров
17. Для рейса 573 определить день с максимальной загрузкой
18. Получить список всех занятых мест рейса 582, выполненного 25.02.2017.
19. Получить список всех пассажиров рейса 853, выполненного 15.02.2017, в имени которых имеется сочетание «оро».
20. Получить список компаний, которые не выполняют рейсы в Якутск

Контрольные вопросы

1. Какова структура оператора SELECT?
2. Какой вид имеет оператор SELECT, если нужно вывести содержимое всей таблицы?
3. В каком случае используется оператор IS NULL?

4. Какие логические операции используют при записи логического выражения в раздел WHERE?

5. Какой оператор используется при сравнении атрибута со строковой константой?

Практическое занятие

Создание представлений (VIEW)

Цель занятия. Освоить работу по созданию и использованию представлений.

Задание.

База данных имеет вид, представленный на рис.

Рисунок 22 – Логическая схема базы данных «Поликлиника»

1)Создать представление, используя таблицы медкарта, пациент, врач.

1. Выбрать всех пациентов, врача – эндокринолога.
2. Выбрать врачей. которые обслуживали пациентов, возраст которых 35-45лет.
3. Выбрать пациентов и их врачей, которые обслуживались в сентябре 2017года.

2)Создать представление, используя таблицы медкарта, пациент, назначение.

1. Выбрать всех пациентов и их назначения за летние месяцы.
2. Посчитать количество назначений для пациента с заданной фамилией.
3. Посчитать количество пациентов, которые болели гриппом

3)Создать представление, используя таблицы медкарта, врач, назначение.

1. Посчитать количество посещений к заданному врачу.

4) Создать представление для выбора врача с максимальным количеством посещений.

Практическое занятие

Хранимые процедуры

Цель занятия. Освоить приемы создания и использования хранимых процедур базы данных**.**

Задание

Написать процедуру без параметров, с параметрами , с выходными параметрами для решения следующих задач.

1. Определить дни, когда было выполнено максимальное количество рейсов из из заданного города.
2. Вывести список компаний, которые перевезли более 100000 пассажиров
3. Для рейса 573 определить день с максимальной загрузкой
4. Получить список всех занятых мест рейса 582, выполненного 25.02.2017.

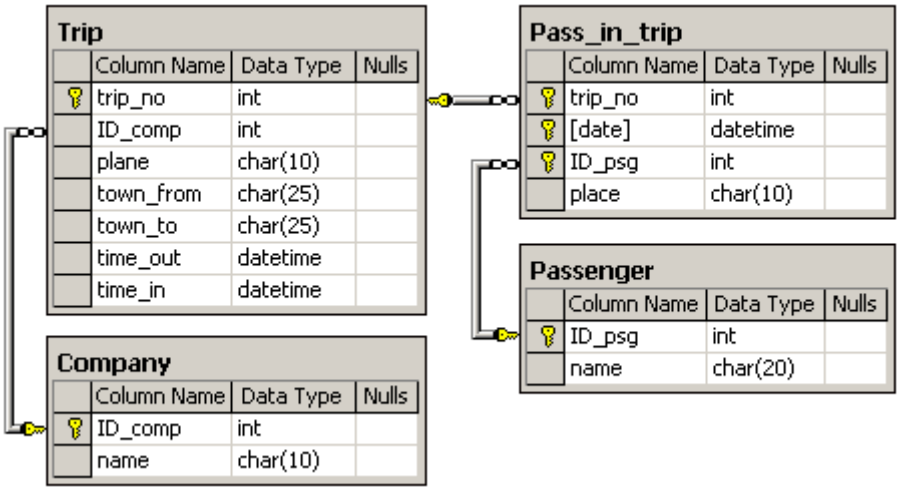


Рисунок 23 – физическая схема базы данных

Методические указания к лабораторным занятиям

Лабораторная работа №1

Создание баз данных

Цель работы: ознакомиться с СУБД Microsoft SQL Server 2014; научиться создавать базу данных и журнал транзакций.

Microsoft SQL Server 2014 – это полномасштабная реляционная система управления базами данных, включающая средства разработки и сопровождения реляционных баз данных, инструменты администрирования и анализа, которые соответствуют требованиям масштабируемости и надежности для большинства предприятий. Она может применяться в широком диапазоне разного типа решений, включая электронную коммерцию, накопление данных и другие прикладные приложения.

База данных в Microsoft SQL Server 2014 представляет собой группу объектов, которая включает по крайней мере набор объектов таблиц. После инсталляции Microsoft SQL Server 2014 в состав программного обеспечения СУБД входят следующие системные базы данных:

- master,
- model,
- msdb,
- tempdb.

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

Таблица 10 – Системные базы данных сервера, создаваемые при установке, и их файлы

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

База данных **tempdb** является глобальным ресурсом, доступным все пользователям, при этом объекты в таблице либо локальные либо глобальные. Локальные объекты существуют в пределах сеанса хранимой процедуры, триггера или пакета команд. Глобальные объекты существуют постоянно и ими могут пользоваться все приложения.

Создание новых баз данных на сервере, в том числе и системной базы **tempdb**, производиться путем копирования системной **model**, которая содержит необходимые параметры по умолчанию. Таким образом, эту базу данных можно использовать как корпоративный стандарт на формат базы данных, указывая при создании лишь имя новой базы. База данных **msbd** используется сервером (его службой **SQL Server Agent**) для планирования событий, задач и регистрации операторов. Microsoft SQL Server 2014 содержит большое количество утилит, которые позволяют решать разные задачи управления и анализа данных. Рассмотрим утилиты Microsoft SQL Server 2014: **SQL Server Management Studio** – это основная утилита разработчика базы данных, которая осуществляет:

– создание, модификацию и удаление базы данных и объектов базы данных;

– управление планируемыми задачами, такими как, резервное копирование и обеспечение пакетов аналитическое обработки данных;

– отображение данных о текущем состоянии функционирования базы данных, в частности о том, какие пользователи в ней зарегитсрированы, какие объекты заблокированы и из какой клиентской программы запущены те или другие процессы;

– управление средствами защиты, включая определение таких составляющих защиты как роли, учетные записи, удаленные и связанные серверы;

– инициализацию и управление почтовой службой базы данных;

– создание и управление каталогами полнотекстового поиска;

– управление параметрами настройки конфигурации сервера;

– создание и управление базами данных публикации и подписки, применяемыми для репликации.

Business Intelligence Development Studio – эта утилита предназначена для создания OLAP-решений.

SQL Profiler – назначение этой утилиты состоит в отслеживании и записи текущих задач SQL Server в специальный файл, который затем может быть использован для выявления проблем.

Database Tuning Advisor – эта утилита позволяет программисту или администратору создавать оптимальный набор индексов и индексированных представлений, не требуя глубокого понимая архитектуры базы данных.

SQL Server Configuration Manager – эта утилита предназначена для управления службами, поставляемыми вместе с SQL Server, а также для определения доступных протоколов для связи с SQL Server.

Visual Studio – с использование **Visual Studio. NET** можно создавать управляемые объекты и разворачивать их под Microsoft SQL 2014. Разработчик клиентских приложений имеет возможность создавать подключения к Microsoft SQL Server 2014 и управлять всеми объектами, не покидая оболочку. Создание любой базы начинается с создания файла данных. Рассмотрим этот процесс в Microsoft SQL Server 2014 на примере создания простой базы данных по учёту успеваемости студентов.

Для начала необходимо запустить среду разработки **SQL Server Management Studio**. Для этого в меню «Пуск» выбираем пункт **Программы → Microsoft SQL Server 2014 → SQL Server Management Studio**.

В появившемся диалоговом окне Соединение с сервером (рис. 1) необходимо установить соединение с сервером, где расположена необходимая база данных.

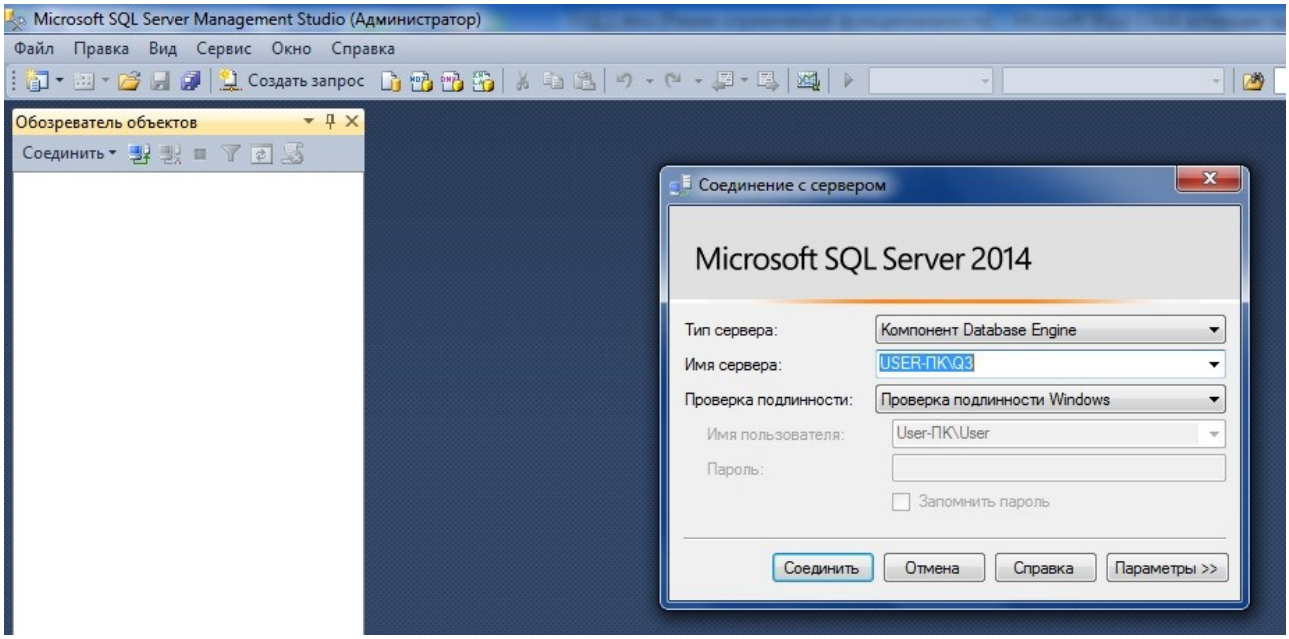


Рисунок 24 – Главная форма

Указав необходимые параметры доступа (тип сервера, имя сервера, имя входа и пароль), нажмите кнопку «Соединить». Если соединение прошло успешно, открывается стандартное окно «Среда SQL Server Management Studio». Замечание: Если при установке Microsoft SQL Server 2014 был задан логин и пароль подключения к серверу, то перед нажатием кнопки «Connect», в выпадающем списке Проверка подлинности нужно выбрать Проверка подлинности SQL Server, а затем необходимо ввести заданные при установке логин и пароль.

После нажатия кнопки «Соединить» появится окно среду разработки SQL Server Management Studio (рис.2). Данное окно имеет следующую структуру (рис.2):

1.Меню – содержит полный набор команд для управления сервером и выполнения различных операций.

2. Панель инструментов – содержит кнопки для выполнения наиболее часто производимых операций. Внешний вид данной панели зависит от выполняемой операции.

3. Панель Обозреватель объектов. Обозреватель объектов – это панель с древовидной структурой, отображающая все объекты сервера, а также позволяющая производить различные операции, как с самим сервером, так и с базой данных. Обозреватель объектов является основным инструментом для разработки базы данных.

4. Рабочая область. В рабочей области производятся все действия с базой данных, а также отображается её содержимое.

Замечание: В обозревателе объектов сами объекты находятся в папках. Чтобы открыть папку необходимо щёлкнуть по знаку «+» слева от изображения папки. Теперь перейдём непосредственно к созданию файла данных. Для этого в обозревателе объектов щёлкните правой кнопкой мыши на папке Базы данных (рис.25) и в появившемся меню выберите пункт Создать базу данных … Появится окно настроек параметров файла данных новой базы данных Создание базы данных (рис.26).

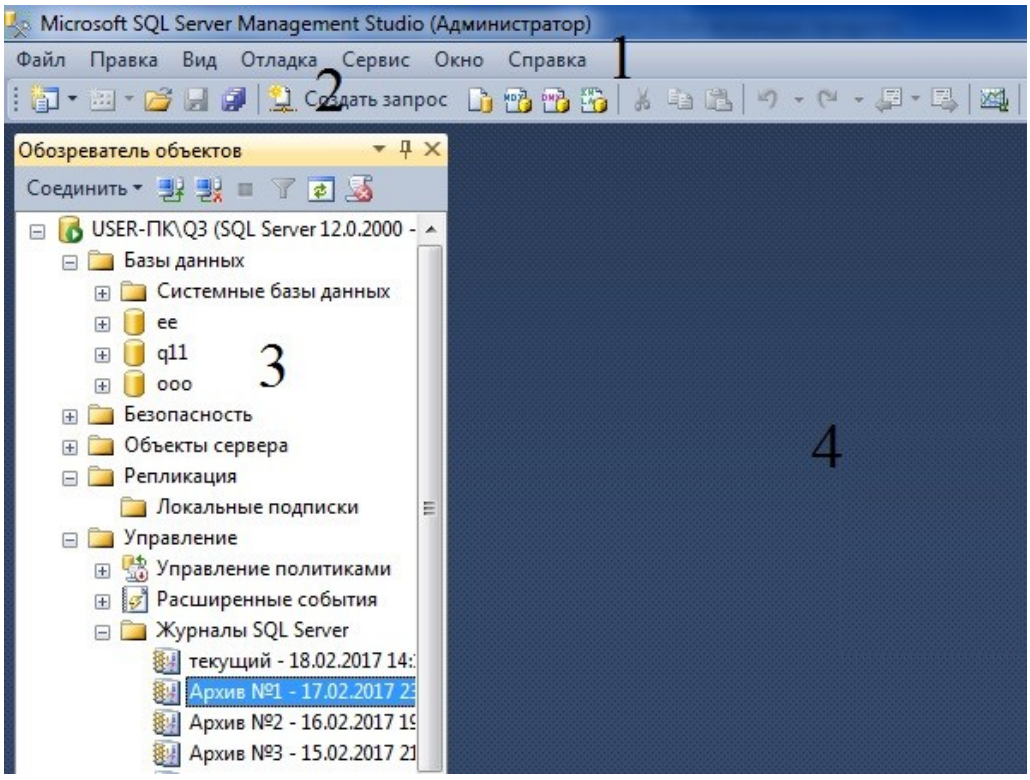


Рисунок 20 – Обозреватель объектов

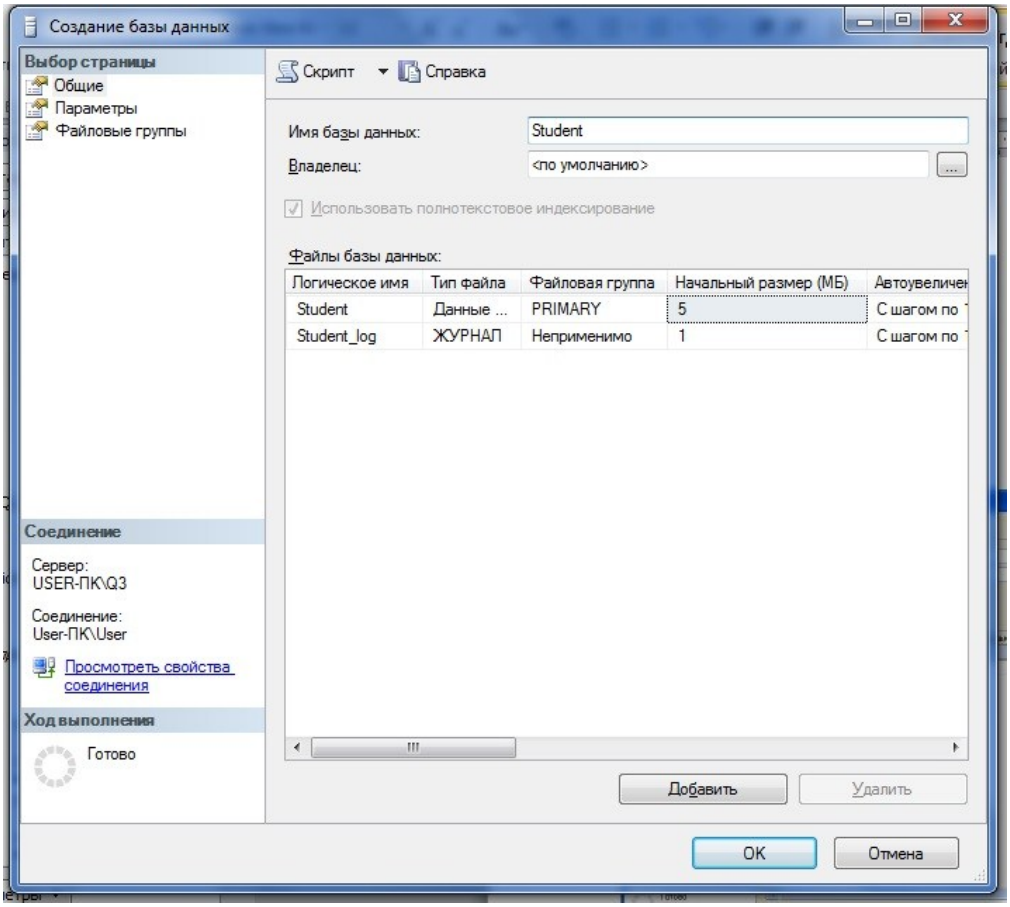


Рисунок 26 – Окно настроек параметров файла данных новой базы данных

В левой части окна настроек имеется список Выбор страницы. Этот список позволяет переключаться между группами настроек.

Для начала настроим основные настройки Общие. Для выбора основных настроек нужно просто щёлкнуть мышью по пункту Общие в списке Выбор страницы. В правой части окна Создание базы данных (рис.26) появятся основные настройки.

Рассмотрим их более подробно. Верхней части окна расположено два параметра: Имя базы данных и Владелец. Задайте параметр Имя базы данных равным «Student». Параметр Владелец оставьте без изменений. Под вышеприведёнными параметрами в виде таблицы располагаются настройки файла данных и журнала транзакций. Таблица имеет следующие столбцы:

· Логическое имя – логическое имя файла данных и журнала транзакций. По этим именам будет происходить обращение к вышеприведённым файлам в базе данных. Можно заметить, что файл данных имеет то же имя что и база данных, а имя файла журнала транзакций составлено из имени базы данных и суффикса «_log».

· Тип файла – этот параметр показывает, является ли файл файлом данных или журналом транзакций.

· Файловая группа – этот параметр показывает к какой группе файлов относится файл. Группы файлов настраиваются в группе настроек Файловые группы.

· Начальный размер (MB) – начальный размер файла данных и журнала транзакций в мегабайтах.

· Авторасширение – автоувеличение размера файла. Как только файл заполняется информацией его размер автоматически увеличивается на величину, указанную в этом параметре. Увеличение можно задавать как в мегабайтах так и в процентах. Здесь же можно задать максимальный размер файлов. Для изменения этого параметра надо нажать кнопку «…». В нашем случае (рис.3) размер файлов не ограничен. Файл данных увеличивается на 1 мегабайт, а файл журнала транзакций на 10%.

· Путь – путь к папке, где хранятся файлы. Для изменения этого параметра также надо нажать кнопку «…». · Имя файла. По умолчанию имена файлов аналогичны логическим именам. Однако файл данных имеет расширение «mdf», а файл журнала транзакций – расширение «ldf».

Замечание: Для добавления новых файлов данных или журналов транзакций используется кнопка Добавить, а для удаления кнопка Удалить.В нашем случае мы оставим все основные настройки без изменений. Теперь перейдём к другим второстепенным настройкам файла данных. Для доступа к этим настройкам необходимо щёлкнуть мышью по пункту Параметры в списке Выбор страницы. В правой части окна – следующие настройки:

· Параметры сортировки – этот параметр отвечает за обработку текстовых строк, их сравнение, текстовый поиск и т.д. Рекомендуется оставить его как «<server default>». При этом данный параметр будет равен значению, заданному на вкладке Параметры сортировки, при установке сервера.

· Модель восстановления – данный параметр отвечает за информацию, предназначенную для восстановления БД, хранящуюся в файле транзакций. Чем полнее модель восстановления, тем больше вероятность восстановления данных при сбое системы или ошибках пользователей, но и больше размер файла журнала транзакций. При наличии места на диске, рекомендуется оставить этот параметр в значении Полная.

· Уровень совместимости определяет совместимость файла данных с более ранними версиями сервера. Если планируется перенос данных на другую, более раннюю версию сервера, то её необходимо указать в этом параметре.

· Другие параметры – данные параметры являются необязательными для изменения.

Последнюю группу настроек Файловые группы. Данная группа настроек отвечает за группы файлов. Для её отображения в списке Выбор страницы необходимо щёлкнуть мышью по пункту Файловые группы.

Группы файлов представлены в таблице в правой части окна. Данная таблица имеет следующие столбцы:

· Имя – количество файлов входящих в группу.

· Файлы – файлы в группе будут только для чтения. То есть, их можно только просматривать, но нельзя изменять.· По умолчанию – группа по умолчанию. Все новые файлы данных будут входить в эту группу.

Замечание: Как и в случае с файлами данных, для добавления новых групп используется кнопка Добавить, а для удаления кнопка Удалить. В рассматриваемой базе данных нет необходимости добавлять новые группы файлов. Поэтому оставим группу настроек Файловые группы без изменений. На этом мы заканчиваем настройку свойств наших файлов. Для принятия всех настроек и создание фала данных и журнала транзакций нашей базы данных в окне Создание базы данных нажмём кнопку ОК.

Произойдёт возврат в окно среду разработки SQL Server Management Studio. На панели обозревателя объектов в папке Базы данных появиться новая база данных «Student».

Замечание: Для переименования базы данных необходимо в обозревателе объектов щёлкнуть по ней правой кнопкой мыши и в появившемся меню выбрать пункт Переименовать. Для удаления в это же меню выбираем пункт Удалить, для обновления – пункт Обновить, а для изменения свойств описанных выше – пункт Свойства.

Задания

3. Изучить утилиту SQL Server Management Studio СУБД Microsoft SQL Server 2014.
4. Создать базу данных Students.

Контрольные вопросы

1. Дайте определение системы управления базами данных (СУБД).
2. Назовите функции СУБД.
3. Назовите утилиты Microsoft SQL Server 2005. Поясните, для чего они предназначены.
4. Назначение утилиты SQL Server Management Studio.
5. Какие системные базы данных включает СУБД Microsoft SQL Server 2014?

6. Поясните назначение системной базы данных master.

7. Поясните назначение системной базы данных model.

8. Поясните назначение системной базы данных msdb.

9. Поясните назначение системной базы данных tembdb.

10.Из каких структурных элементов состоит окно SQL Server Management Studio?

11.Какая информация отображается в Обозревателе объектов.

12.Что такое файл данных базы данных?

13.Что такое журнал транзакций базы данных?

14.Поясните общие настройки файлов базы данных.

15.С какой целью используется авторасширение?

16.Какое расширение имеет файл данных?

17.Какое расширение имеет файл журнала транзакций?

Лабораторная работа №2

Создание таблиц

Цель работы. Научиться создавать таблицы и заполнять их.

Все таблицы нашей базы данных находятся в подпапке «Таблицы» папки «Student» в окне обозревателя объектов.

Создадим таблицу «Специальности». Для этого щёлкните правой кнопкой мыши по папке «Таблицы» и в появившемся меню выберите пункт «Создать таблицу …». Появиться окно создания новой таблицы.

В правой части окна расположена таблица определения полей новой таблицы. Данная таблица имеет следующие столбцы:

· **Имя столбца**. Имя столбца должно всегда начинаться с буквы и не должно содержать различных специальных символов и знаков препинания. Если имя столбца содержит пробелы, то оно автоматически заключается в квадратные скобки.

· **Тип данных** – определяет тип данных, выбираемый из раскрывающегося списка. В каждое поле допускается ввод данных только одного типа (см. табл. 11).

Таблица 11–Типы данных

Тип данных	**Класс**	**Размер в байтах**	**Описание**
bit	целое число	1	Один байт отводится для восьми элементов типа bit в таблице; если количество элементов данных такого типа меньше восьми, то остальные биты байта не используются
bigint	целое число	8	-2^{63}

			Позволяют использовать целые числа от –2 до $2^{63} - 1$
int	целое число	4	Целые числа от –214748483648 до 214748483647
smallint	целое число	2	Целые числа от –32768 до 32767
tinyint	целое число	1	Целые числа от 0 до 255
decimal, numeric	decimal / numeric	с переменной длиной	-10^{38} до $10^{38} - 1$ Заданные точность и масштаб от –10 до 10 – 1. ***Точность*** – общее количество значащих цифр в числе. Например, 10 имеет точность 2, 43.00000004 – точность 10. Точность изменяется в пределах от 0 до 38. ***Масштаб*** – возможное число значащих цифр, расположенное в дробной части. Например, 10 имеет масштаб 0, 43.00000004 – масштаб 8. **decimal(3,1)** – определят число точности 3, масштабом 1.
money	денежный	8	-2^{63} до $2^{63} - 1$ Количество денежных единиц от –2 до 2 – 1, определяемое с точностью до четырех десятичных позиций. Позволяет представлять любые денежные единицы.
smallmoney	денежный	4	Денежные единицы от –214748.3648 до 214748.3647
float (real)	приближенные численные данные	с переменной длиной	При определении данных этого типа допускается использовать параметр (**float (20)**), который определяет точность. Параметр задается в битах. Область определения – от –1.79E+308 до 1.79E+308
datetime	дата/время	8	Представляет собой данные даты и времени с 1 января 1753 года по 31 декабря 9999 года. Имеет точность в сотые доли секунды.
smalldatetime	дата/время	4	Представляет собой данные даты и времени с 1 января 1900 года по 6 июня 2079 года. Имеет точность в 1 минуту.
timestamp	Специальное числовое (двоичное) значение	8	Специальное значение, которое является уникальным в пределах данной базы данных. Это значение задается автоматически после каждой вставки или обновления записи.
unigueidentifier	Специальное числовое (двоичное) значение	16	Специальный глобально уникальный идентификатор. Уникальность такого идентификатора в пространстве и времени гарантирована.
char	символ	с переменной длиной	Символьные данные фиксированной длины. Значения данных с длиной короче заданной дополняются пробелами до указанной длины. Максимальное заданное значение длины может составлять 8000 символов.
varchar	символ	с переменной длиной	Символьные данные переменной длины. Значения данных с длиной короче заданной не дополняются пробелами. Максимальное заданное зна-

			чение длины может составлять 8000 символов.
varchar(max)	символ	с переменной длиной	Символьные данные, имеющие объем до 2 байт.
text	символ	с переменной длиной	Устаревший тип данных, который используется в версии Microsoft SQL Server 2014 исключительно для обеспечения совместимости с предыдущими версиями. Вместо этого типа данных следует использовать **varchar(max)**
nchar	unicode	с переменной длиной	Символьные данные в кодировке unicode фиксированной длины. Значения данных с длиной короче заданной дополняются пробелами. Максимальное заданное значение длины может составлять 4000 символов.
nvarchar	unicode	с переменной длиной	Символьные данные в кодировке unicode переменной длины. Значения данных с длиной короче заданной не дополняются пробелами. Максимальное заданное значение длины может составлять 4000 символов.
nvarchar(max)	unicode	с переменной длиной	[31] Символьные данные, имеющие объем до 2 байт.
text	unicode	с переменной длиной	Устаревший тип данных, который используется в версии Microsoft SQL Server 2014 исключительно для обеспечения совместимости с предыдущими версиями. Вместо этого типа данных следует использовать **nvarchar(max)**.
binary	binary	с переменной длиной	Двоичные данные фиксированной длины с максимальной длиной 8000 байтов.
varbinary	binary	с переменной длиной	Двоичные данные переменной длины с максимальной длиной 8000 байтов, но для обозначения длины можно использовать ключевое слово [31] max (до 2 байт).
image	binary	с переменной длиной	Устаревший тип данных, который используется в версии Microsoft SQL Server 2014 исключи тельно для обеспечения совместимости с предыдущими версиями. Вместо этого типа данных следует использовать **varbinary(max)**.-
sql_variant	другой	особый	Тип данных sql_variant может рассматриваться как приближенный аналог типа данных variant языка VB и некоторых типов данных C++. Тип данных sql_variant может использоваться, если необходимо представить в одном столбце или функции несколько разных типов данных.
xml	символ	с переменной длиной	Определяет символьное поле как содержащее данные XML. Тип данных xml обеспечивает проверку данных по схеме XML и применение специальных функций, предназначенных для обработки кода XML.

Разрешить значения null – допуск значения Null. Если эта опция столбца включена, то в случае незаполнения столбца в него будет автоматически подставлено значение Null. То есть, поле необязательно для заполнения.

Замечание: Под таблицей определения полей располагается таблица свойств выделенного столбца «Свойства столбцов». В данной таблице настраиваются свойства выделенного столбца. Некоторые из них будут рассмотрены ниже. Перейдём к созданию столбцов (полей) и настройке их свойств. В таблице определения полей задайте значения столбцов «Имя столбца», «Тип данных» и «Разрешить значения null». Таблица «Специальности» имеет три столбца (поля):

– **Код специальности** – числовое поле для связи с таблицей студенты,

– **Наименование специальности** – текстовое поле, предназначенное для хранения строк, имеющих длину не более 50 символов.

– **Описание специальности** – текстовое поле, предназначенное для хранения строк, имеющих неограниченную длину.

Замечание: Поле «Код специальности» будет являться первичным ключом.

Ключ – атрибут или совокупность атрибутов, значения которых уникально идентифицируют объект в наборе объектов. Если один и тот же набор объектов имеет несколько ключей, то один из них назначается **первичным ключом** набора объектов, а остальные ключи называются альтернативными. Каждая таблица в реляционной базе данных должна иметь первичный ключ, который может быть простым или составным, включающим несколько полей.

Выбор ключа – важный момент в проектировании моделей данных, поскольку с одной стороны, ключ должен выполнять задачу однозначной идентификации, а с другой – включать в свой состав минимально необходимое число атрибутов.

Сделаем столбец «Код специальности» счётчиком, т.к. счётчик содержит уникальные значения. Для этого выделите поле, просто щёлкнув по нему мышкой в таблице определения полей. В таблице свойств поля отобразятся свойства поля «Код специальности». Разверните группу свойств «Спецификация идентифицирующего столбца». Свойство «(Является идентифицирующим столбцом)» установите в значение «Да». Задайте свойства «Начальное значение» и «Приращение» (шаг счетчика) равными 1. Эти настройки показывают, что значение поля «Код специальности» у первой записи в таблице будет равным 1, у второй – 2, у третьей 3 и т.д.

Теперь сделаем столбец «Код специальности» ключевым полем. Выделите поле, а затем на панели инструментов нажмите кнопку с изображением ключа. В таблице определения

полей, рядом с полем «Код специальности» появиться изображение ключа, говорящее о том, что поле ключевое.

На этом настройку таблицы «Специальности» можно считать завершённой. Закройте окно создания новой таблицы, нажав кнопку закрытия в верхнем правом углу окна, над таблицей определения полей. Появиться окно с запросом о сохранении таблицы. В этом окне необходимо нажать «Да». Появиться окно «Выбор имени», предназначенное для определения имени новой таблицы.

В этом окне задайте имя новой таблицы как «Специальности» и нажмите кнопку «ОК». Таблица «Специальности» отобразиться в обозревателе объектов в папке «Таблицы» базы данных «Students».

Замечание: В обозревателе объектов таблица «Специальности» отображается как «dbo.Специальности». Префикс «dbo» обозначает, что таблица является объектом БД (Data Base Object). В дальнейшем при работе с объектами БД префикс «dbo» можно опускать.

Создайте таблицу «Дисциплины». После создания таблицы «Дисциплины» создадим таблицу «Студенты». Рассмотрим поля новой таблицы: · **Код студента** – это первичный ключ, числовой счётчик (bigint, является идентифицирующим столбцом с начальным значением 1 и приращением 1).

ФИО – текстовое поле переменной длины, с максимальной длиной 50, допускаются значения null.

Пол – текстовое поле переменной длины, с максимальной длиной 10, допускаются значения null.

Замечание. Для задания длины выделенного текстового поля необходимо в таблице свойств выделенного поля установить свойство «Длина» равное максимальному количеству знаков текста вводимого в поле .

Родители – текстовое поле переменной длины, с максимальной длиной 50, допускаются значения null.

Адрес – текстовое поле максимальной длины, допускаются значения null.

Телефон – текстовое поле переменной длины, с максимальной длиной 15, допускаются значения null.

Паспортные данные – текстовое поле максимальной длины, допускаются значения null.

Группа – текстовое поле переменной длины, с максимальной длиной 10, допускаются значения null.

Дата рождения – поле, предназначенное для хранения дат, тип данных datetime, допускаются значения null.

Дата поступления – тип данных datetime, допускаются значения null.

Очная форма обучения – является логическим полем, тип данных bit, допускаются значения null.

Номер зачётки – является целочисленным полем, тип данных bigint, допускаются значения null.

Курс – является целочисленным полем, тип данных tinyint, допускаются значения null.

Код специальности – является целочисленным полем, тип данных bigint, допускаются значения null. Данное поле является **внешним ключом**, т.е. копией первичного ключа родительской таблицы «Специальности». Внешний ключ необходим для установления связей между таблицами. Сохраните таблицу «Студенты». Создадим таблицу «Экзамены». На этом мы заканчиваем создание таблиц БД «Students».

Теперь рассмотрим операцию заполнения таблиц начальными данными. Для начала заполним таблицу «Специальности». Для заполнения этой таблицы в обозревателе объектов щёлкните правой кнопкой мыши по таблице «Специальности» и в появившемся меню выберите пункт «Открыть таблицу». В рабочей области «Microsoft SQL Server Management Studio» появится окно заполнения таблиц. Заполните таблицу «Специальности».

Замечание: Заполнение таблиц происходит полностью аналогично табличному процессору «Microsoft Excel».

Замечание: Так как поле «Код специальности» является первичным полем связи и ключевым числовым счётчиком, то оно заполняется автоматически (заполнять его не нужно). Закройте окно заполнения таблицы «Специальность» щелкнув по кнопке закрытия окна в верхнем правом углу, над таблицей. После заполнения таблицы «Специальности» заполните таблицу «Дисциплины». Откройте её для заполнения как описано выше, и заполните.

Заполните таблицу «Студенты». **Замечание:** Для заполнения дат в качестве разделителя можно использовать знак «.». Даты можно заполнять в формате «день.месяц.год».

Замечание: Поле «Код специальности» является внешним ключом (для связи с таблицей «Специальности»). Следовательно, значения этого поля необходимо заполнять значениями поля «Код специальности» таблицы «Специальности». В нашем случая это значения от 1 до 5. Если у Вас коды специальностей в таблице «Специальности» имеют другие значения, то внесите их в таблицу «Студенты». По окончании заполнения, закройте окно заполнения таблицы «Студенты».

Наконец заполним таблицу «Экзамены», как это показано на рисунке 20. Закройте окно заполнения таблицы «Экзамены». На этом заканчиваем создание и заполнение таблиц БД «Students».

Контрольные вопросы

1. Как создать таблицу в Management Studio СУБД Microsoft SQL Server 2014?
2. Какие типы данных поддерживает Microsoft SQL Server 2014?
3. Что такое ключ?
4. Что такое первичный ключ?
5. Что такое альтернативный ключ?
6. Назовите правила выбора первичного ключа.
7. Что такое внешний ключ?
8. Каким образом определяется поле счетчик?

Лабораторная работа 3

Создание связи между таблицами

Цель работы. Приобрести навыки установления связей между отношениями.

Связи между столбцами разных таблиц создаются в конструкторе диаграмм баз данных, перетаскивая столбцы между таблицами.

1. В конструкторе диаграмм баз данных выберите таблицы.
2. Щелкните селектор строк для одного или более столбцов базы данных, которые необходимо связать со столбцом в другой таблице.
3. Перетащите выбранные столбцы в связанную таблицу.
4. Отображаются два диалоговых окна: Связь по внешнему ключу и Таблицы и столбцы, второе отображается на переднем плане.
5. Имя связи устанавливается системой в формате FK_*локальная_таблица***таблица*внешнего_ключа. Можно изменить это значение.
6. Убедитесь, что Таблица первичного ключа правильно задает таблицу.
7. Сетка содержит локальные столбцы и соответствующие им внешние столбцы. Можно добавить или удалить столбцы таблицы, либо изменить сопоставления.
8. Нажмите кнопку ОК.

Открывается диалоговое окно Связь по внешнему ключу. Выбранная связь отображает созданную связь.

9. Измените свойства связи в сетке.
10. Нажмите кнопку ОК , чтобы создать связь.

Задание

Построить связи представленные на рисунке 27.

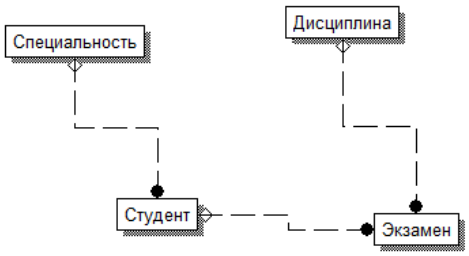


Рисунок 27 – Логическая схема данных

.

Контрольные вопросы

1. Какие связи могут быть установлены между таблицами в реляционных базах данных?

2. Как реализовывается связь «многие-ко-многим»?

3. Какова особенность связи «один-ко-одному»?

4. Какими свойствами должен обладать внешний ключ для установления связи между таблицами?

5. Какие ошибки могут возникнуть при построении диаграмм баз данных?

Лабораторная работа 4

Создание запросов

Цель работы. Приобрести навыки создания запросов.

Создание запроса в базе данных выполняется при активации правой кнопки мыши и выборе пункта меню «Создать запрос»

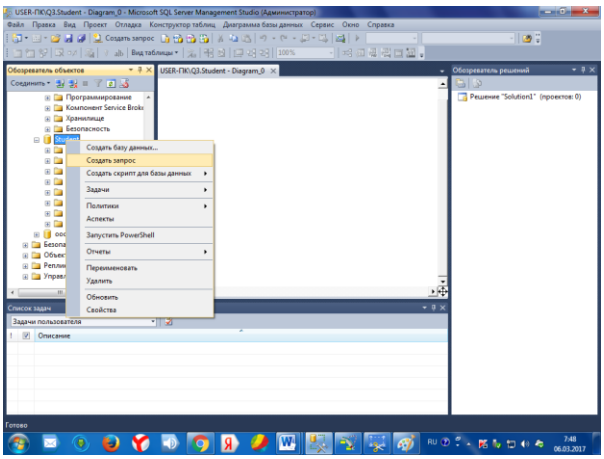


Рисунок 28 – Создание запроса

В открытом окне вводится текст запроса.

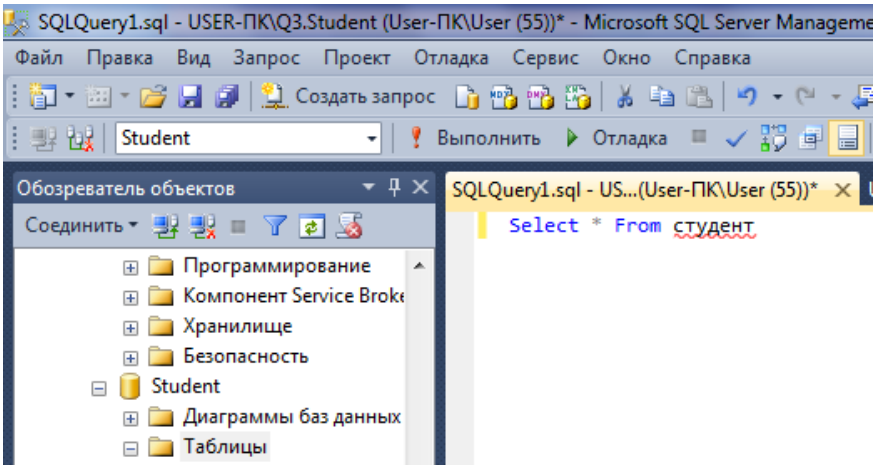


Рисунок 29 – Ввод текста запроса

Следующий шаг – выполнение запроса.

При правильной записи запроса появится результат выполнения.

Задание

Создать следующие запросы:

1. Выдать список студентов.
2. Выдать список студентов, которые родились раньше 2001года
3. Выдать список студентов, которым исполнилось 18 лет.
4. Рассчитать средний балл по всем студентам.
5. Рассчитать средний балл по дисциплине

6. Выдать список студентов женского пола
7. По студенту выдать все персональные данные.
8. Выдать список студентов в алфавитном порядке.
9. Выдать список студентов по группам
10. Выдать список студентов, которые родились в январе.

Контрольные вопросы

1. Какова структура оператора SELECT?
2. Какой оператор служит для сортировки записи?
3. Для чего предназначен оператор LIKE?
4. Каким образом выполняется переименование столбцов?
5. Привести пример оператора сравнения оператора SELECT?

Лабораторная работа 5
Создание запросов, с использованием операторов соединения

Цель работы. Приобрести навыки разработки запросов, с использованием операторов соединения таблиц.

Оператор SQL INNER JOIN формирует таблицу из записей двух или нескольких таблиц. Каждая строка из первой (левой) таблицы, сопоставляется с каждой строкой из второй (правой) таблицы, после чего происходит проверка условия. Если условие истинно, то строки попадают в результирующую таблицу. В результирующей таблице строки формируются конкатенацией строк первой и второй таблиц.

Оператор SQL INNER JOIN имеет следующий синтаксис:

```
SELECT
   column_names [,... n]
FROM
   Table_1 INNER JOIN Table_2
ON condition
```

Условие для сравнения задается в операторе ON.

Соединение может быть либо внутренним (**INNER**), либо одним из внешних (**OUTER**). Служебные слова **INNER** и **OUTER** можно опускать, поскольку внешнее соединение однозначно определяется его типом — **LEFT** (левое), **RIGHT** (правое) или **FULL** (полное), а просто **JOIN** будет означать внутреннее соединение.

Предикат определяет условие соединения строк из разных таблиц. При этом **INNER JOIN** означает, что в результирующий набор попадут только те соединения строк двух таб-

лиц, для которых значение предиката равно **TRUE**. Как правило, предикат определяет эквисоединение по внешнему и первичному ключам соединяемых таблиц, хотя это не обязательно.

Задание

Создать следующие запросы:

1. Рассчитать средний балл для каждого студента
2. Выдать оценки студентов заданной группы по заданному предмету.
3. Выдать для заданного студента все оценки с указанием дисциплины.
4. Выдать список дисциплин, изучаемых, заданным студентом.
5. Выдать список студентов, изучающих заданную дисциплину.
6. Выдать список дисциплин, изучаемых в заданной группе.
7. Выдать список групп, изучающих заданную дисциплину.
8. Выдать список дисциплин по заданной специальности.
9. Выдать список специальностей, изучающих заданную дисциплину.
10. Выдать количество студентов, обучающихся в заданной группе.

Контрольные вопросы

1. Что такое правое и левое соединение?
2. Что такое предикат в операторе соединения?
3. Что такое алиасы и как они используются?
4. Чем отличается внешнее соединение от внутреннего?
5. Сколько таблиц может использоваться в соединении?

Лабораторная работа 6
Создание представлений.

Цель работы. Приобрести навыки создания и использования представлений.

Представление — виртуальная (логическая) таблица, представляющая собой поименованный запрос (синоним к запросу), который будет подставлен как подзапрос при использовании представления.

В отличие от обычных таблиц реляционных баз данных, представление не является самостоятельной частью набора данных, хранящегося в базе. Содержимое представления динамически вычисляется на основании данных, находящихся в реальных таблицах. Изменение данных в реальной таблице базы данных немедленно отражается в содержимом всех представлений, построенных на основании этой таблицы.

На рисунках представлена последовательность шагов по созданию представления.

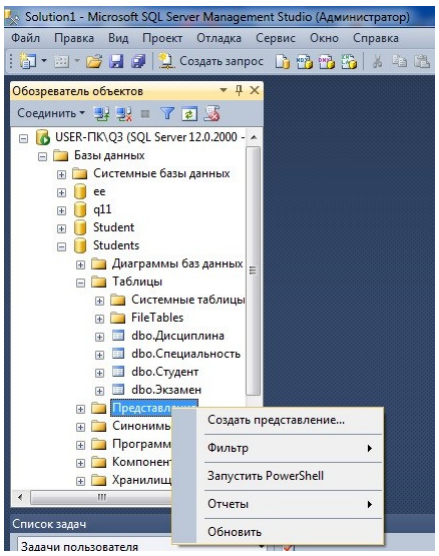


Рисунок 10– Начало создания представления

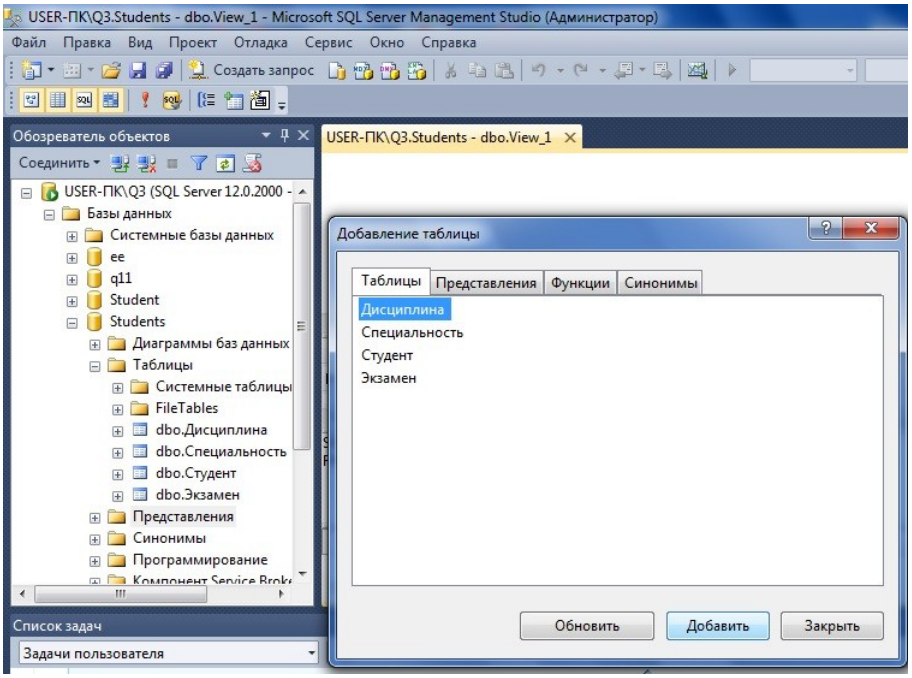


Рисунок 31– Выбор таблиц для создания представления

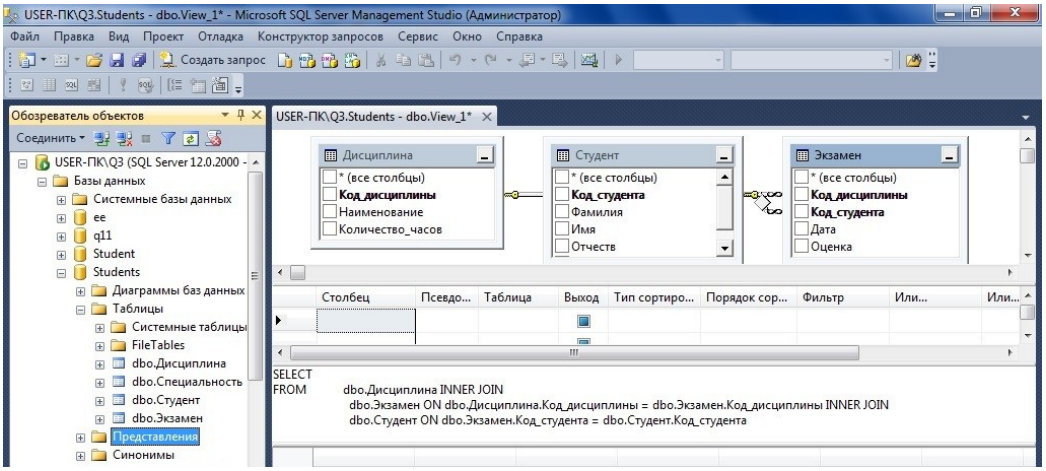


Рисунок 32– Окно задания содержимого представления

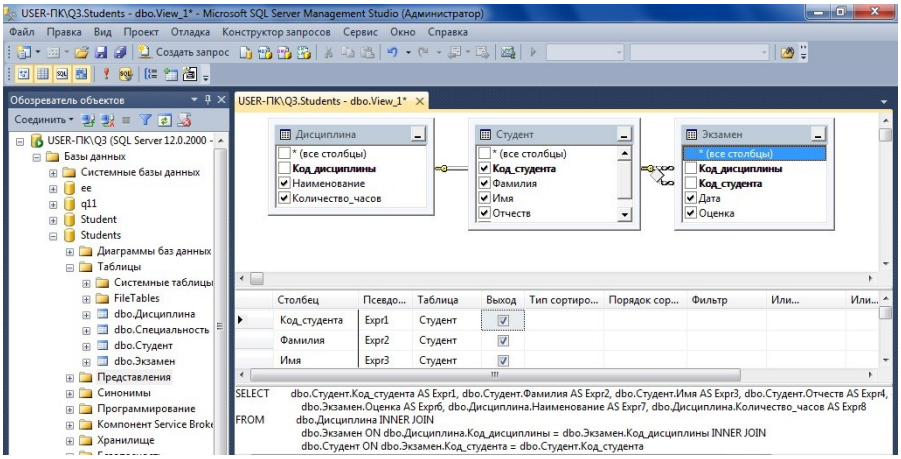


Рисунок 33 – Окно задания содержимого представления

После создания необходимо представление запустить и сохранить, задав имя (стандартно View_1).

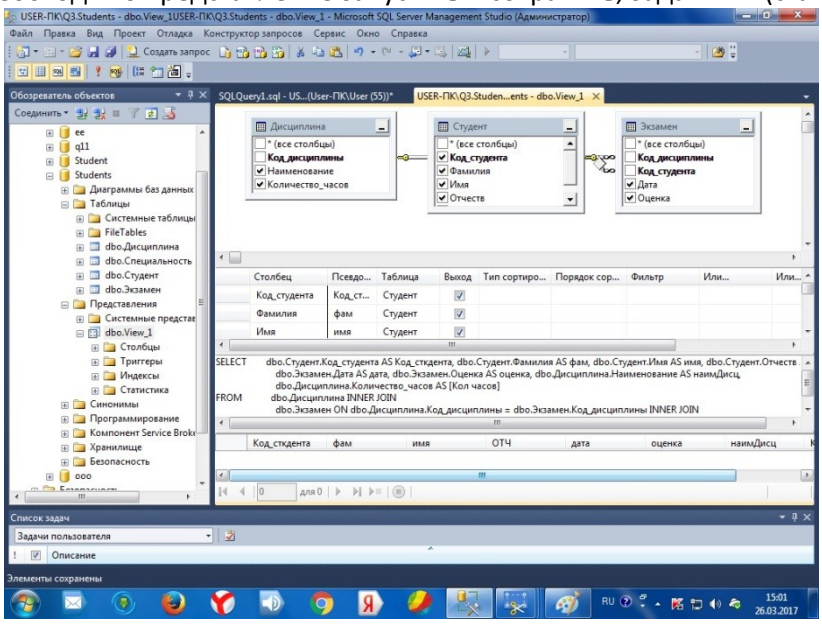


Рисунок 34 – Выполнение представления

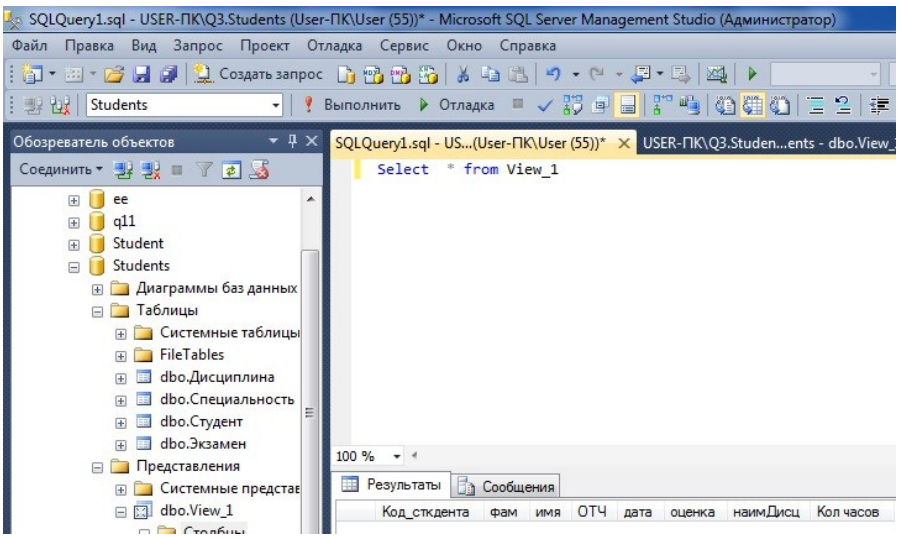


Рисунок 35 – Разработка запроса по представлению

Задание

1. Создать представление, которое содержит информацию о студентах и принадлежности их к специальности.
   1.1. Выдать список студентов по заданной специальности.
   1.2.Выдать список групп по заданной специальности.
   1.3.Рассчитать количество студентов на заданной специальности.
   1.4.Выдать все содержимое созданного представления.
2. Создать представление, которое содержит информацию о сданных студентами экзаменах.
   2.1.Выдать все содержимое созданного представления.
   2.2.Выдать список студентов, изучающих заданную дисциплину.
   2.3.Рассчитать средний балл по заданной дисциплине.
   2.4.Рассчитать количество студентов, изучающих заданную дисциплину.
   2.5.Рассчитать средний балл для заданной группы.
   2.6.Рассчитать количество студентов, средний балл которых превышает 4,5
   2.7.Выдать список студентов, средний балл которых превышает 4,5

3. Создать представление, которое содержит информацию о студентах с учетом всех таблиц.
   3.1. Рассчитать количество предметов, изучаемых студентами заданной специальности
   3.2. Выдать список предметов, изучаемых студентами заданной специальности
   3.3. Рассчитать средний балл по специальности.
   3.4. Выдать список студентов с указанием дисциплины по заданной специальности, получивших отличную оценку.
   3.5. Выдать студентов с указанием специальности, имеющих максимальный средний балл. Список отсортировать по специальности.

Контрольные вопросы

1. Что такое представление?

2. Что происходит с представлением при обновлении таблиц, участвующих в представлении?

3. Для каких целей можно использовать представление?

4. Допускается ли в представлении использования ограничений, агрегатных функций, операторов сортировки и группировки?

5. Чем представление отличается от запроса?

Лабораторная работа 7
Создание и использование хранимых процедур

Цель работы. Приобрести навыки создания и использования хранимых процедур без параметров. входными и выходными параметрами.

Хранимые процедуры похожи на процедуры из других языков программирования в том, что они могут:

– принимать входные параметры и возвращать вызывающей процедуре или пакету ряд значений в виде выходных параметров;

– содержать программные инструкции, которые выполняют операции в базе данных, в том числе вызывающие другие процедуры;

– возвращать значение состояния вызывающей процедуре или пакету, таким образом передавая сведения об успешном или неуспешном завершении (и причины последнего).

Последовательность действий при создании и использовании хранимых процедур.

1. В обозревателе объектов необходимо подключиться к экземпляру компонента Компонент Database Engine и развернуть его.

Последовательно развернуть узел Базы данных, базу данных Students и узел Программирование**.** Щелкнуть правой кнопкой мыши элемент Хранимые процедуры и выбрать пункт Создать хранимую процедуру.

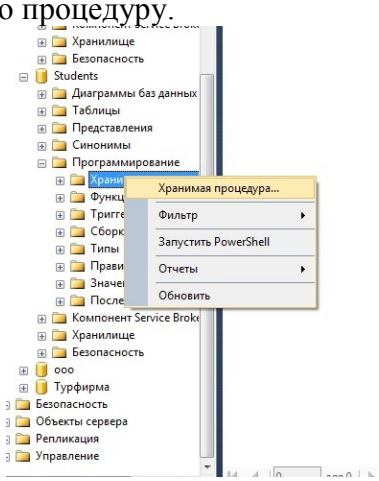


Рисунок 36– Создание хранимой процедуры.

Откроется окно с шаблоном хранимой процедуры. Необходимо настроить этот шаблон для своей процедуры.

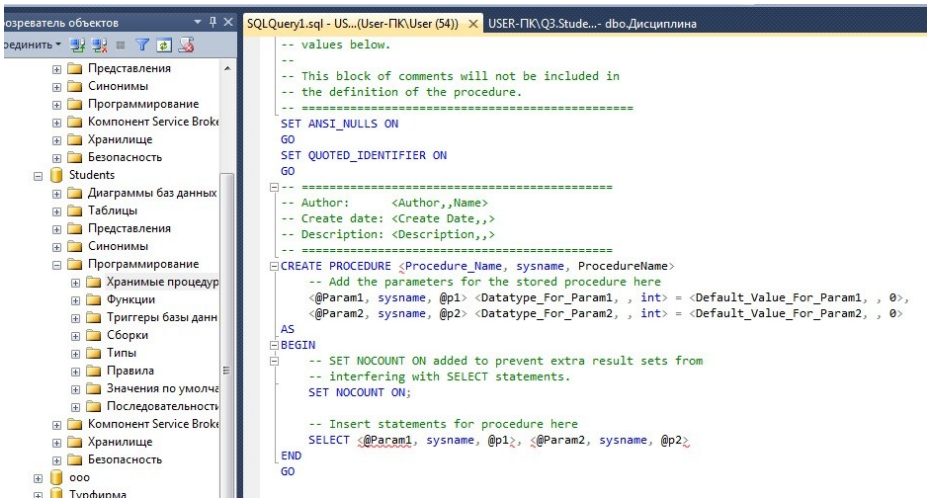


Рисунок 37– Оболочка хранимой процедуры

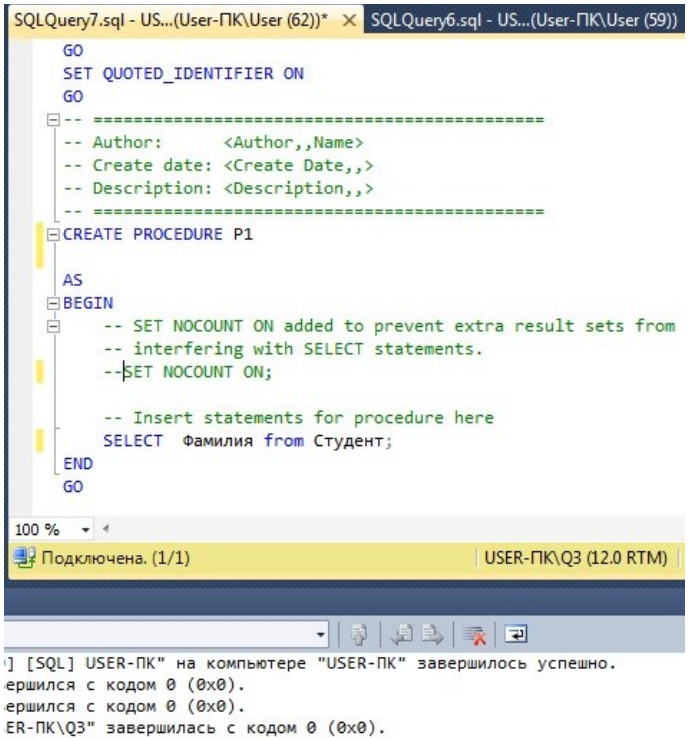


Рисунок 38– Создание хранимой процедуры

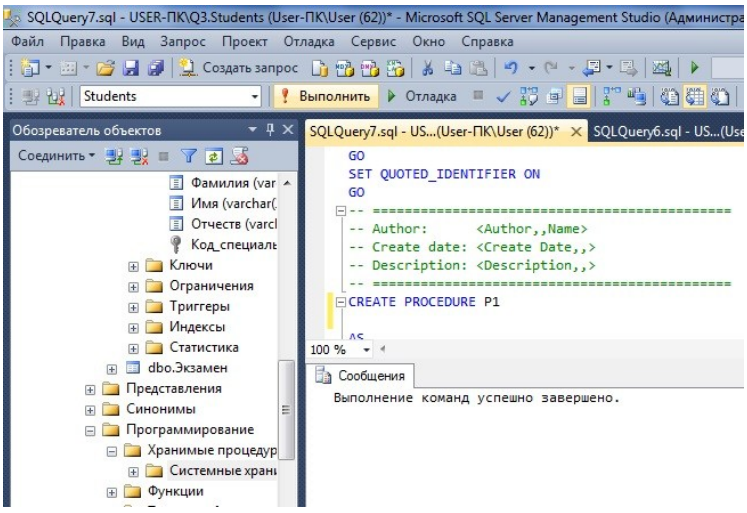


Рисунок 39– Выполнение хранимой процедуры

4 Запуск процедуры выполняется либо записью команды в окне создания запроса, либо с помощью пункта меню «Выполнить хранимую процедуру».

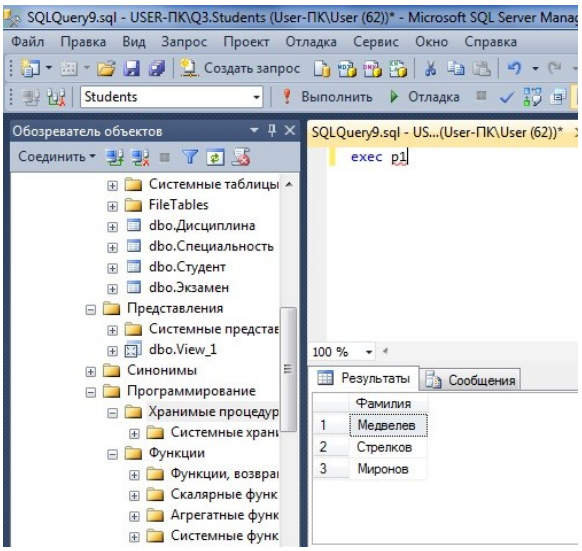


Рисунок 40– Результат хранимой процедуры

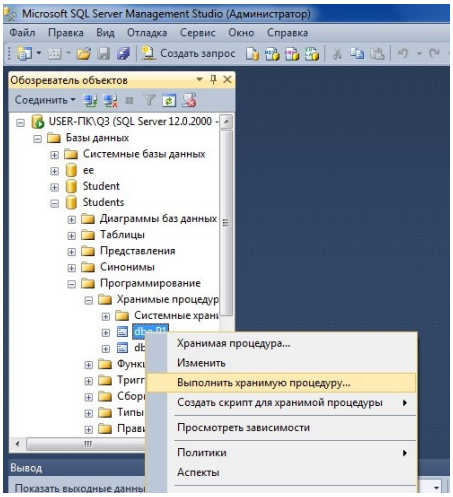


Рисунок 41– Окно для работы с хранимой процедурой

Задание

1. Создать и использовать процедуру без параметров для расчета среднего балла по всем студентам.

2. Создать и использовать процедуру с входным параметром для расчета среднего балла по заданному студенту. Входной параметр – код студента.

3. Создать и использовать процедуру с входным параметром для расчета среднего балла по дисциплине. Входной параметр – наименование дисциплины.

4. Создать и использовать процедуру с входным и выходным параметром для расчета среднего балла по всем студентам. Использовать эту процедуру для поиска студентов, средний балл которых превышает средний балл по всем студентам.

5. Создать и использовать процедуру для расчета количества отличников, хорошистов и троечников. Отличник – средний балл >4,75, хорошист – средний балл> =4 и <4.75, троечник – средний балл <4.

Контрольные вопросы

1. Что такое хранимая процедура?
2. Что такое параметр функции?
3. Какие типы параметров используются в хранимых процедурах?
4. Как используется входной и выходной параметр в случае запуска хранимой процедуры через меню «Выполнить хранимую процедуру»?
5. Чем отличаются входные и выходные параметры ?

Лабораторная работа №8

Работа с курсорами

Цель работы. Освоить технологию решения задачи по обработке базы данных с использованием курсоров.

В соответствии со стандартом SQL при работе с курсорами можно выделить следующие основные действия:

– создание или объявление курсора;
– открытие курсора , т.е. наполнение его данными, которые сохраняются в многоуровневой памяти;
– выборка из курсора и изменение с его помощью строк данных;
– закрытие курсора, после чего он становится недоступным для пользовательских программ;
– освобождение курсора, т.е. удаление курсора как объекта, поскольку его закрытие необязательно освобождает ассоциированную с ним память.

Задание

1. Разработать процедуру для вывода отчета по успеваемости в заданной группе.

Отчет о успеваемости по группам

Группа 1

Фио	средний балл
Иванов А.И.	4,8
Семенов С.К.	4,64
Петров Е.А.	4,2

_____

Средний балл по группе 4,32

К-во отличников 1

К-во хорошистов 2

К-во троечников 0

==========

Группа 2

Фио	средний балл
Голубев А.И.	4,75
СорокинС.К.	4,5
Воронов Е.А.	3,9

_____

Средний балл по группе 4,25

К-во отличников 1

К-во хорошистов 1

К-во троечников 1

2. Разработать процедуру для вывода экзаменационных ведомостей по всем предметам. Ведомость обязательно должна содержать итоги проведения экзамена. Ниже представлен вид ведомости

ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ
ПО ДИСЦИПЛИНЕ МАТЕМАТИКА

№	ФИО	№зач книжки	оценка	подпись
1	Иванов	767876	5	
2	Сидоров	78786	3	
3	Петров	76768	4	
…	…	…	…	
Отлично			1	
Хорошо			1	
удовлетв			1	
неуд			0	
итого			3	

Контрольные вопросы

7. Что такое курсор? Для решения каких задач используется курсор?
8. Какова технология использования курсора?
9. Как выполняется объявление курсора?
10. Какие существуют типы курсоров?
11. Как объявляется прокручиваемый курсор?

Лабораторная работа №9
Создание и использование триггеров

Цель работы. Освоить технологию создания и использования триггера для таблицы базы данных.

В MS SQL Server:

–триггер может быть вызван либо после выполнения операции, либо вместо выполнения операции;

–триггер вызывается один раз для всех записей таблицы, над которыми должна быть выполнена операция;

–следовательно, изменяемые записи хранятся в двух автоматически создаваемых при вызове триггера таблицах:

– таблица Inserted – содержит измененные или добавленные записи таблицы;

– таблица Deleted – содержит записи до выполнения изменений или удаленные записи таблицы;

–в теле триггера, определенного для операции Insert, доступна только таблица Inserted;

–в теле триггера, определенного для операции Delete, доступна только таблица Deleted;

–в теле триггера, определенного для операции Update, доступны обе таблицы Inserted и Deleted;

Задание

1. В таблицу Дисциплины добавить поле «Часы» и заполнить его.
2. Создать триггер обновления таблицы Дисциплины, который допускает изменение значения поля часов не более чем на 20%.

```
CREATE TRIGGER tgrDUpdate
ON Дисциплина AFTER UPDATE
AS
DECLARE @p1 int, @p1_new float, @Id int;
BEGIN
Select @p1 = Количество_часов from Deleted
Select @p1_new = Количество_часов, @Id = Код_дисциплины from Inserted
IF (@p1_new - @p1 > 0.2 * @p1)
UPDATE Дисциплина SET Количество_часов = 1.05 * @p1
WHERE Код_дисциплины = @Id
END
```

3. Создать триггер обновления таблицы Экзамены, который допускает изменение оценки только на 5 баллов.
4. Создать триггер обновления таблицы Студенты, который допускает изменение значение поля Группа путем добавления к старому символов «об».
5. Удалить последний триггер

Контрольные вопросы

1. Что такое триггер и в каком случае он используется?
2. Какие типы триггеров существуют?
3. Как создается триггер и где он хранится?

4. Какие таблицы создаются при вызове триггера?
5. Как уничтожается триггер?

Лабораторная работа № 10

Разработка приложений в visual Studio C#

Цель работы. Освоить технологию разработки приложения в среде visual Studio C#

Порядок выполнения работы представлен на рисунках

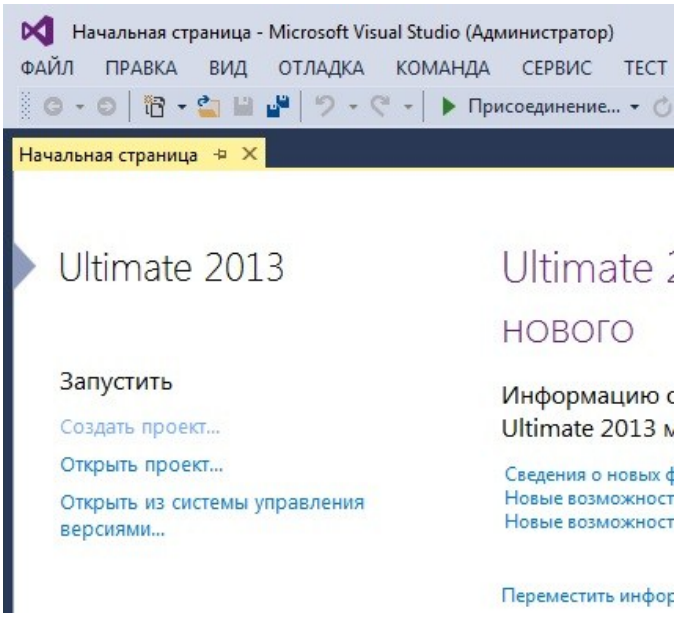


Рисунок 42– Создание проекта

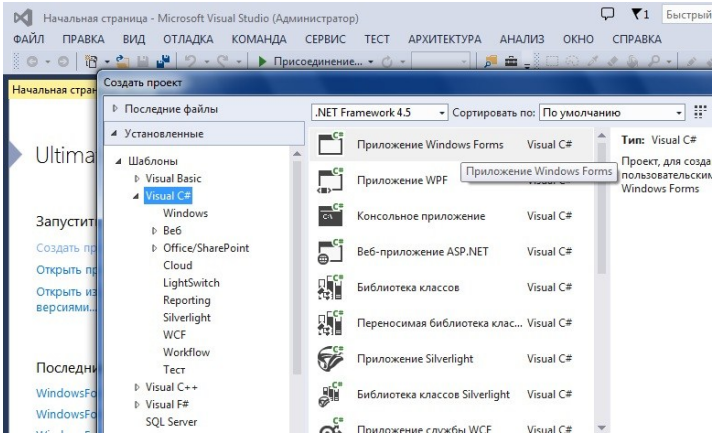


Рисунок 43– Выбор шаблона и типа приложения

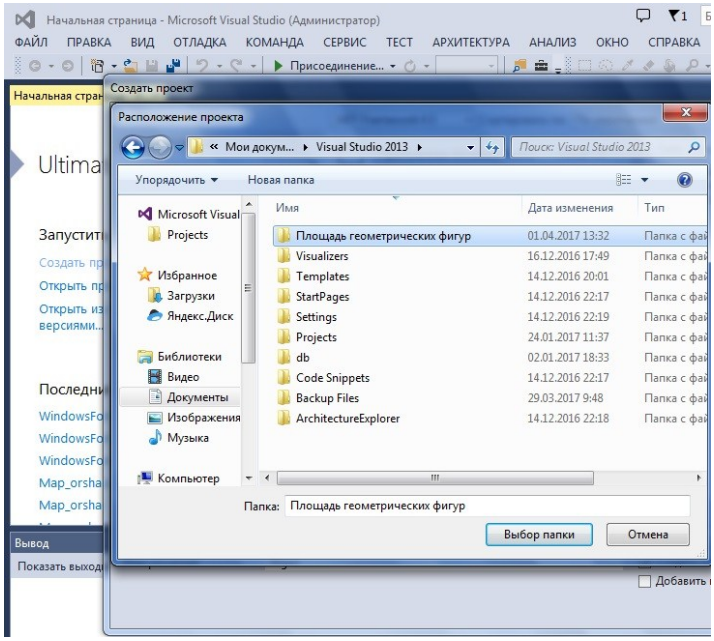


Рисунок 44 – Выбор папки для расположения файлов проекта

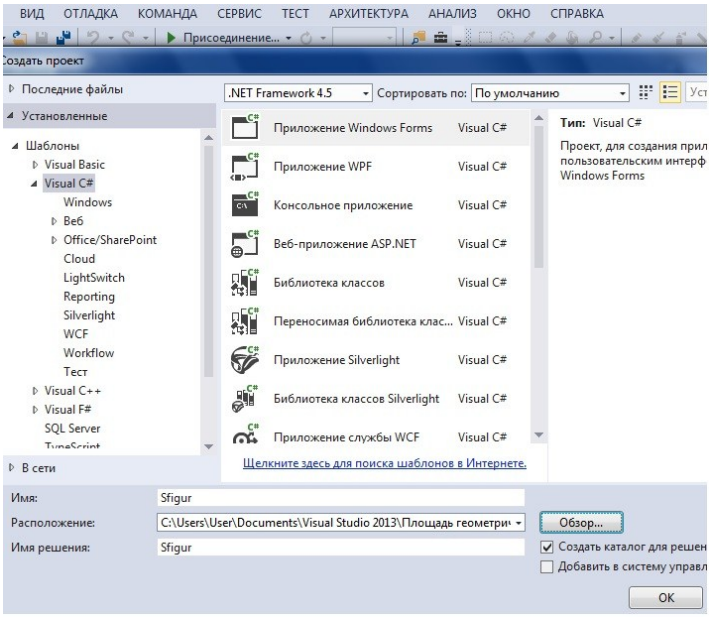


Рисунок 45 – Окно с параметрами создания проекта

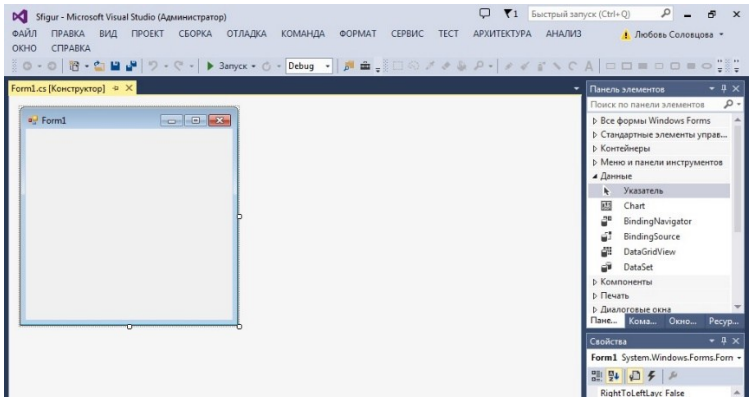


Рисунок 46– Первая форма программного приложения

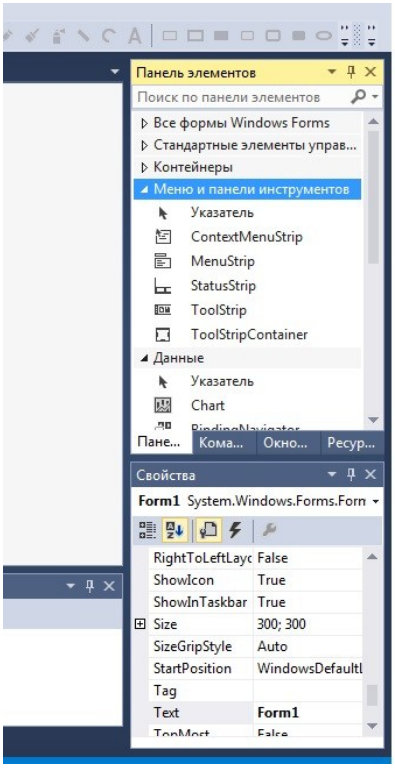


Рисунок 47– Вид панели инструментов и окно свойств Form1

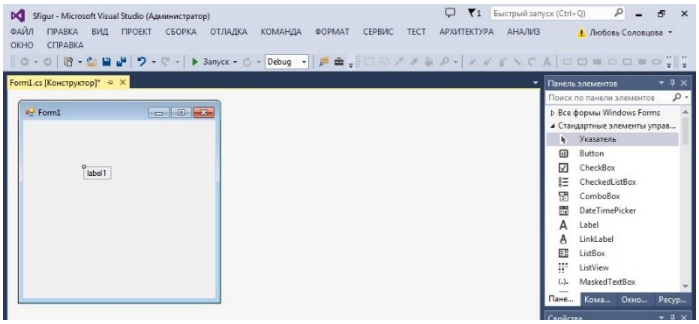


Рисунок 48 – Размещение на форме элемента Label (метка) для размещения текста

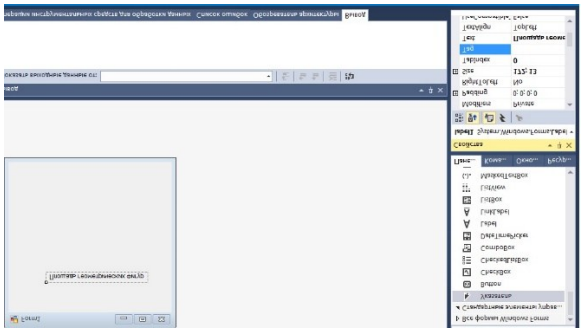


Рисунок 49 – Задание свойств элемента Label

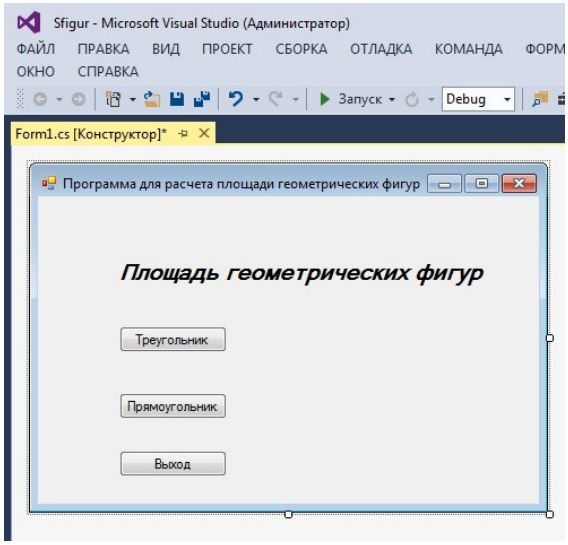


Рисунок 50 – Проектирование экранной формы. Размещение трех управляющих элементов Button(кнопка) и задание их свойств.

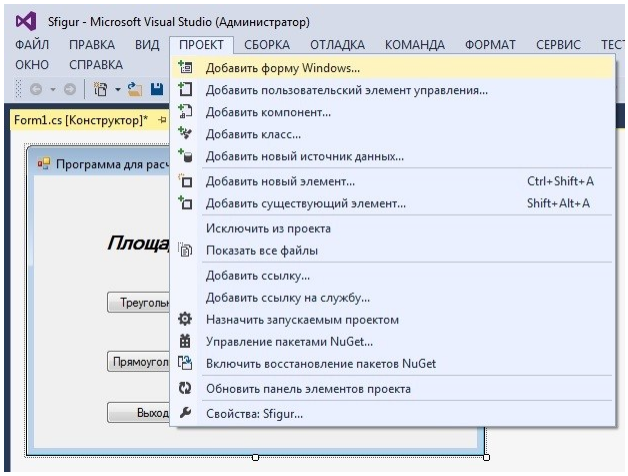


Рисунок 51 – Добавление к проекту новой формы

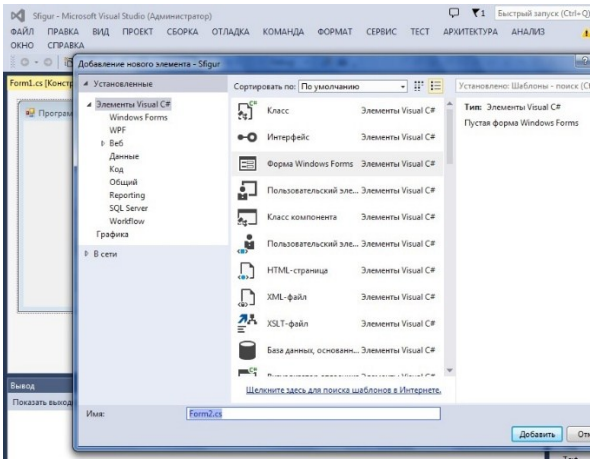


Рисунок 52 – Меню добавление к проекту новой формы

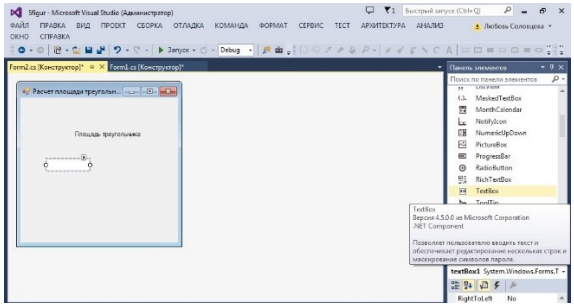


Рисунок 53 – Разработка новой формы.

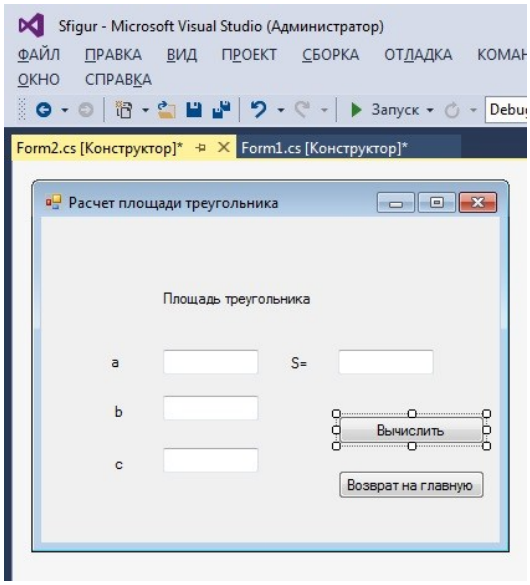


Рисунок 54 – Вид новой формы для расчета площади треугольника. Для ввода данных и представления результата используются элементы TextBox (4 шт)

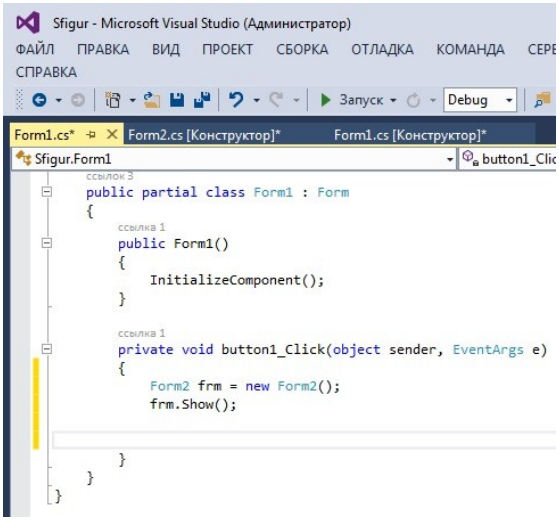


Рисунок 55 – Программный код для перехода с первой формы на вторую

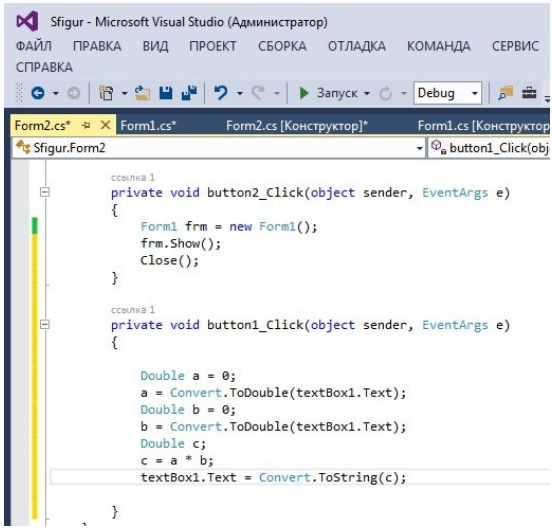


Рисунок 56 – Программный код для возврата со второй формы на первую. Фрагмент кода для расчета площади треугольника.

Лабораторная работа №11

Разработка приложения для работы с БД

Цель работы. Изучение технологии работы с данными Ado.Net.

ADO.NET предоставляет собой технологию работы с данными, которая основана на платформе .NET Framework. Эта технология представляет нам набор классов, через которые мы можем отправлять запросы к базам данных, устанавливать подключения, получать ответ от базы данных и производить ряд других операций.

Порядок выполнения работы.

1. Создать новый проект.
2. Разработать главную экранную форму с четырьмя кнопками. Первая для отображения данных таблицы Студент и ввода новых данных в таблицу. Вторая для открытия третьей формы, третья– четвертой. Четвертая - выход из программы. Для каждой кнопки прописать соответствующий код либо открытие новой формы, либо – закрытие главной формы.
3. Добавить вторую форму.
4. На форме разместить элемент dataGridView (рис.1), который на панели элементов находится в разделе ДАННЫЕ, и 2 кнопки. Первая предназначена для сохранения данных, вторая – для перехода на главную форму.

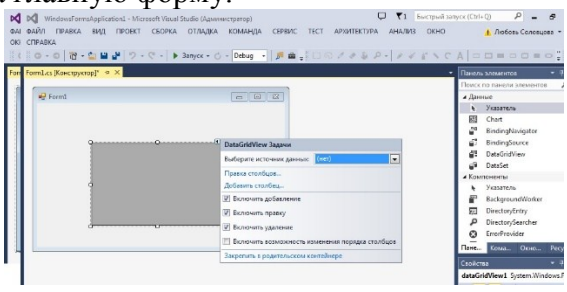


Рисунок 57 – Размещение на форме элемента dataGridView.

5. Выполним подключение БД (рис.58)

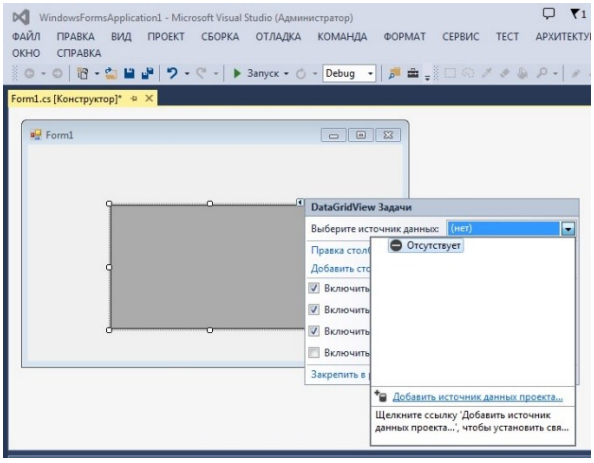


Рисунок 58– Начало добавления источника данных к приложению

6. Соединение с БД выполняем по инструкции, предлагаемой системой.(рис.59 – рис.)

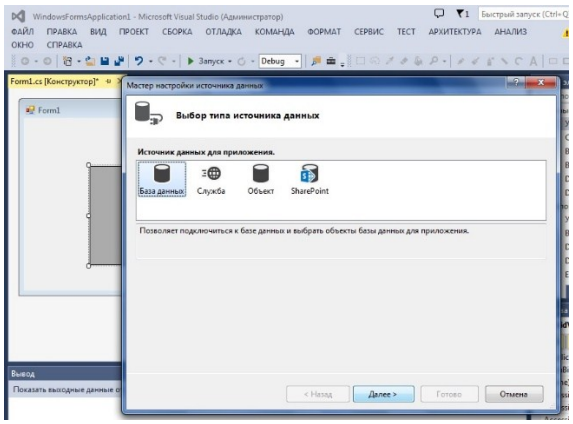


Рисунок 59 – Подключение БД

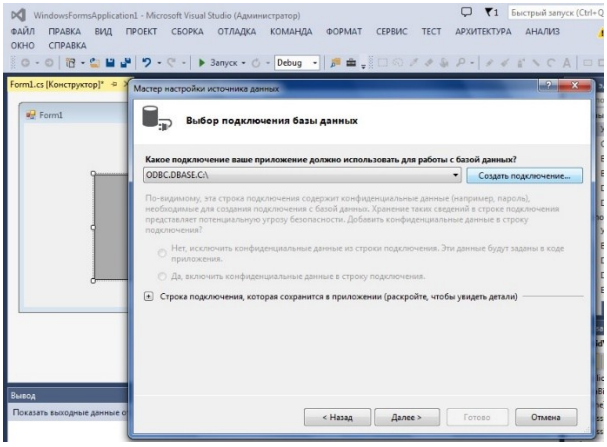


Рисунок 60 – Начальный этап подключения БД

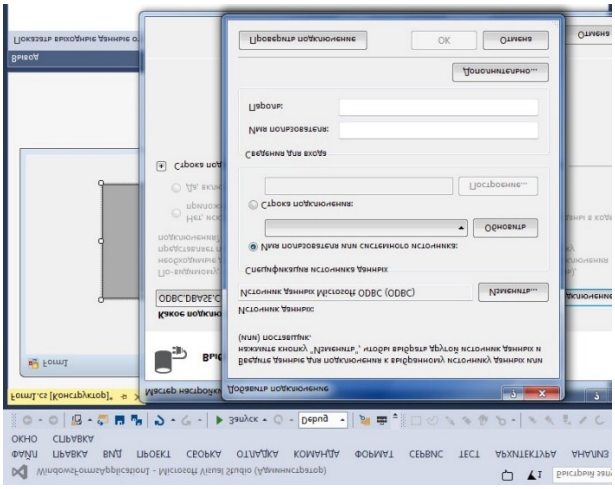


Рисунок 61 – Выбор источника данных

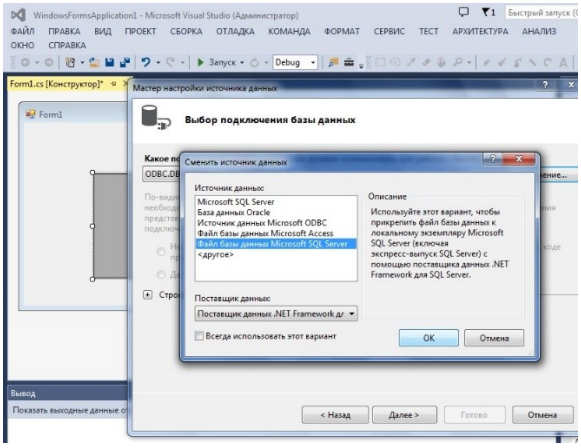


Рисунок 62 – Выбор способа подключение БД

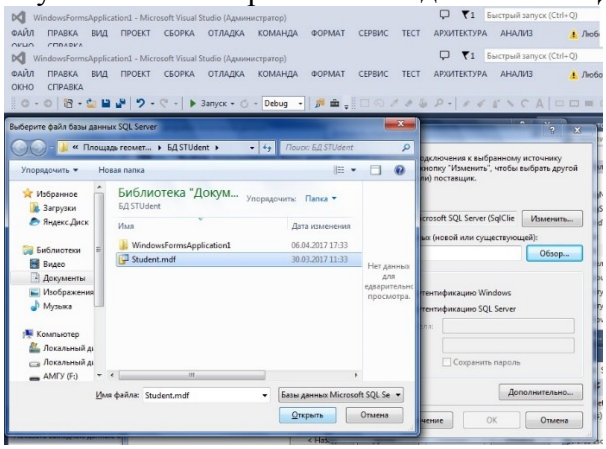


Рисунок 63 – выбор файла БД

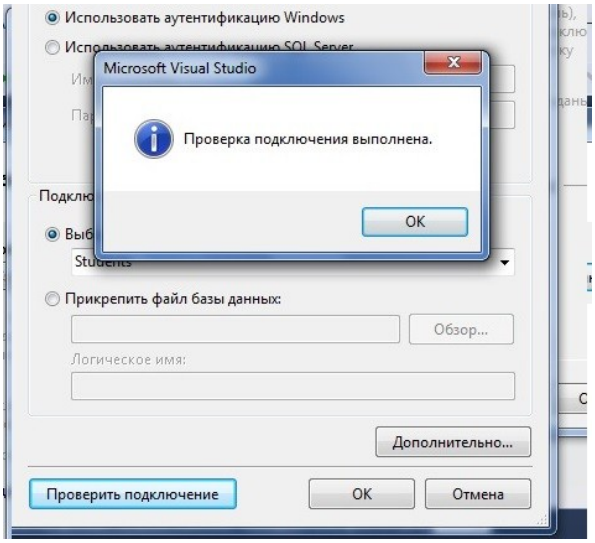


Рисунок 64 – Проверка подключение БД

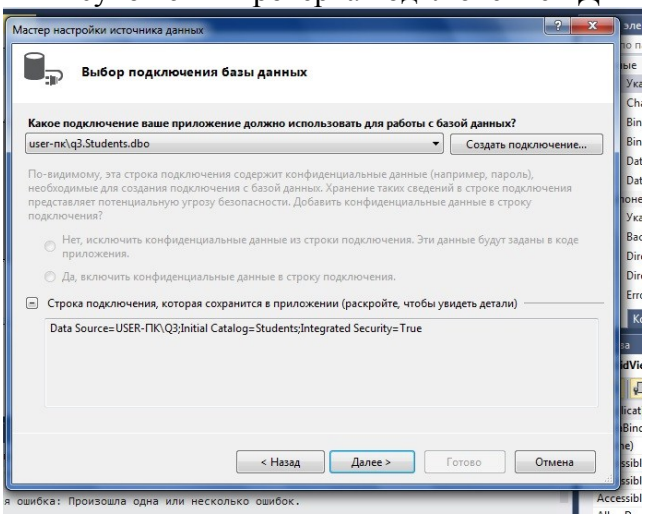


Рисунок 65– Выбор подключения БД

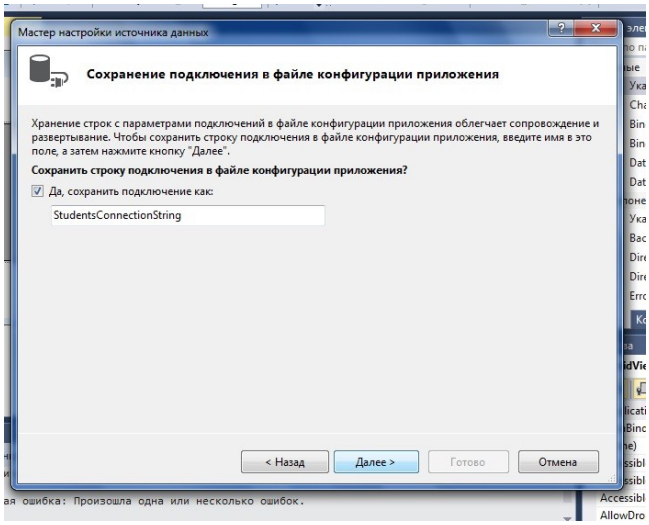


Рисунок 66 – Сохранение строки подключения

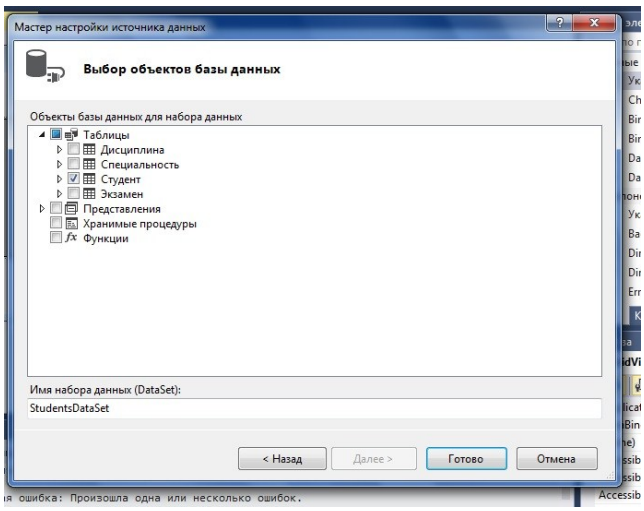


Рисунок 67 – Выбор таблицы БД

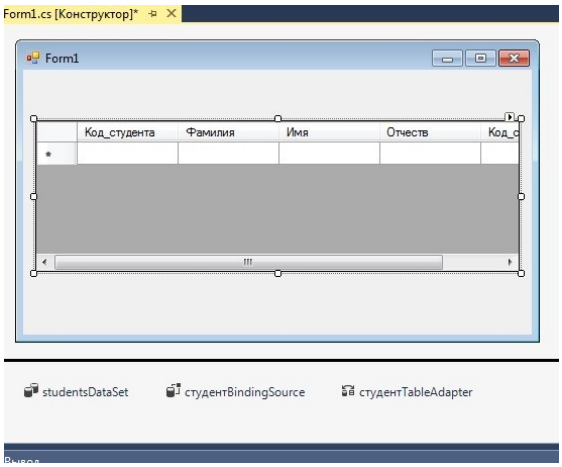


Рисунок 68– Отображение таблицы БД

7. Прописать программный код для кнопки «Сохранение». Вначале необходимо объявить пространство имен для работы с функциями и классами SQL (рисунок 13).

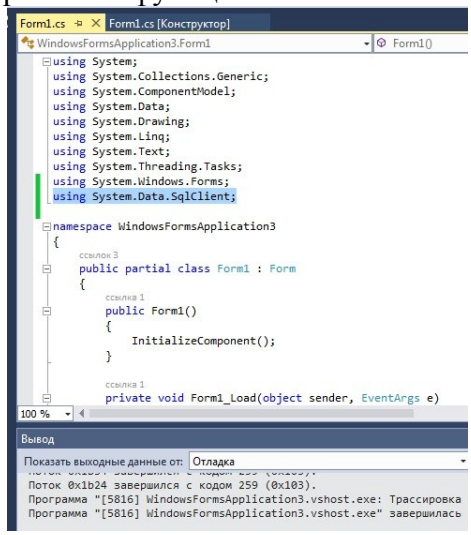
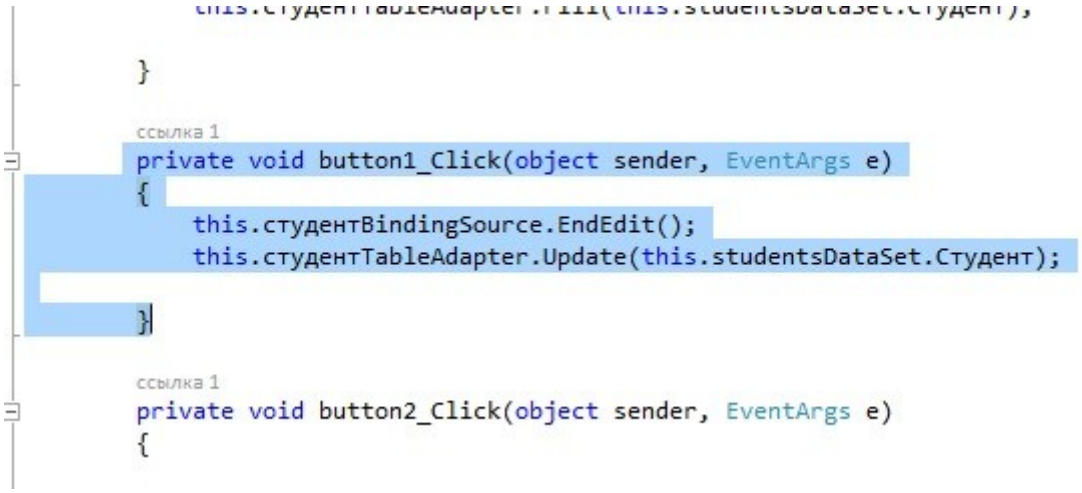


Рисунок 69 – Подключение пространства имен.

8. Прописываем программный код для обновления таблицы БД (рис.14)



```
        }

        ссылка 1
        private void button1_Click(object sender, EventArgs e)
        {
            this.студентBindingSource.EndEdit();
            this.студентTableAdapter.Update(this.studentsDataSet.Студент);

        }

        ссылка 1
        private void button2_Click(object sender, EventArgs e)
        {
```

Рисунок 70 – Программный код для обновления таблицы

9. Добавляем новую форму для обработки запросов. На форму добавить элемент dataGridView и кнопку «Выполнить запрос». Для кнопки прописать код для выполнения запроса.

```
        ссылка 1
        private void button2_Click(object sender, EventArgs e)
        {

            string commandText = "SELECT Код_студента, Фамилия FROM Студент";
            string connectionString = @"Data Source=USER-ПК\Q3;Initial Catalog=Students;Integrated Se

            SqlConnection conn = new SqlConnection(connectionString);
            conn.Open();
            SqlCommand MyCommand = new SqlCommand();
            MyCommand.Connection = conn;
            //или OleDbCommand MyCommand=conn.CreateCommand();
            MyCommand.CommandText = commandText;
            SqlDataAdapter dataAdapter = new SqlDataAdapter();
            dataAdapter.SelectCommand = MyCommand;
            DataSet ds = new DataSet();
            dataAdapter.Fill(ds, "Студент");
            dataGridView2.DataSource = ds.Tables["Студент"].DefaultView;
            conn.Close();

        }
```

Рисунок 71 – Программный код для выполнения запроса.

10. Добавить остальные формы и выполнить разработку аналогично форме 3.

```
this.студентBindingSource.EndEdit();
this.студентTableAdapter.Update(this.studentsDataSet.Студент);
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace WindowsFormsApplication3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // TODO: данная строка кода позволяет загрузить данные в таблицу
"studentsDataSet.Студент". При необходимости она может быть перемещена или удалена.
            this.студентTableAdapter.Fill(this.studentsDataSet.Студент);

        }

        private void button1_Click(object sender, EventArgs e)
        {
            this.студентBindingSource.EndEdit();
            this.студентTableAdapter.Update(this.studentsDataSet.Студент);

        }

        private void button2_Click(object sender, EventArgs e)
        {

            string commandText = "SELECT Код_студента, Фамилия FROM Студент";
            string connectionString = @"Data Source=USER-ПК\Q3;Initial Cata-
log=Students;Integrated Security=True";


            SqlConnection conn = new SqlConnection(connectionString);
            conn.Open();
            SqlCommand MyCommand = new SqlCommand();
            MyCommand.Connection = conn;
            //или OleDbCommand MyCommand=conn.CreateCommand();
            MyCommand.CommandText = commandText;
            SqlDataAdapter dataAdapter = new SqlDataAdapter();
            dataAdapter.SelectCommand = MyCommand;
            DataSet ds = new DataSet();
            dataAdapter.Fill(ds, "Студент");
            dataGridView2.DataSource = ds.Tables["Студент"].DefaultView;
            conn.Close();

        }
    }
}
```

Задание

Разработать приложение для работы с БД.

Первая форма – главная, обеспечивающая переход на остальные формы.

Вторая форма – ввод и редактирование данных таблицы Студенты.

Третья форма – обработка запроса. Выдать данные о студентах, фамилия которых начинается на букву П.

Четвертая форма – обработка запроса. Рассчитать средний балл по всем студентам по каждому студенту .