Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
Оренбургский государственный университет

РАСПРЕДЕЛЕННЫЕ БАЗЫ ДАННЫХ

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Оренбург 2018

Содержание

1 Лабораторная работа № 1. Обследование предметной области

Цель работы: научиться выявлять предметную область информационной системы и создавать ее модель.

Используемое программное обеспечение: Ramus 2.0.

Знакомство с инструментальной средой Ramus.

Ramus имеет достаточно простой и интуитивно понятный интерфейс пользователя. При запуске Ramus по умолчанию появляется основная панель инструментов, палитра инструментов (вид которой зависит от выбранной нотации) (рисунок 1).

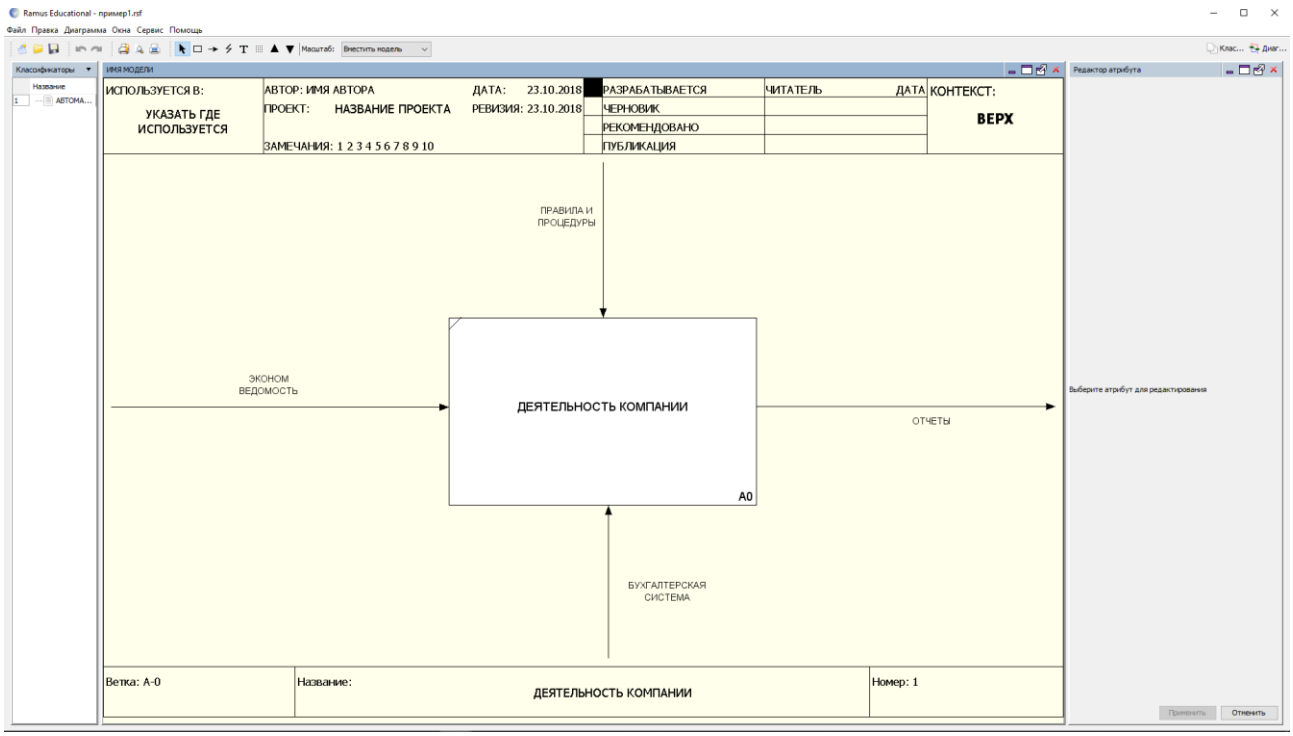


Рисунок 1 – Окно Ramus

При создании новой модели возникает диалог, из нескольких последовательных диалоговых окон, в котором следует указать, будет ли создана модель заново или она будет открыта из файла, затем внести имя модели и выбрать методологию, в которой будет построена модель (рисунок 2, 3).

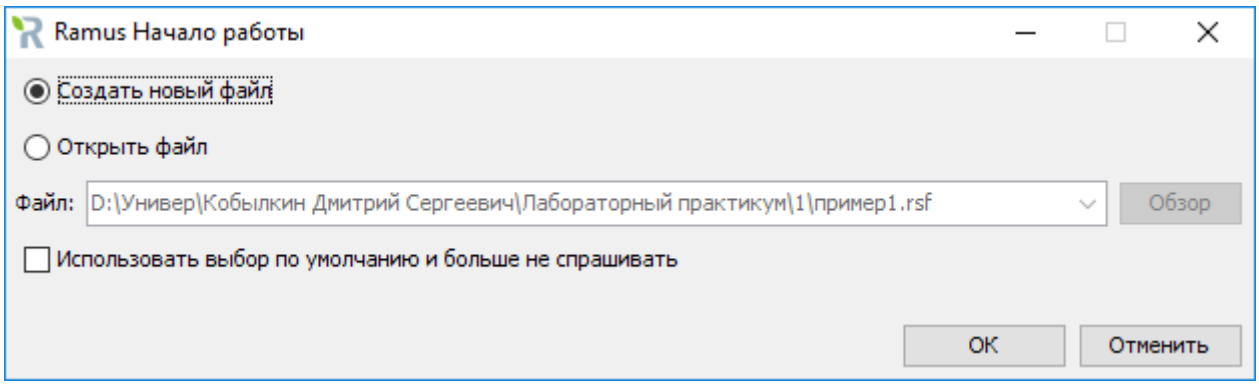


Рисунок 2 – Окно «Начало работы»

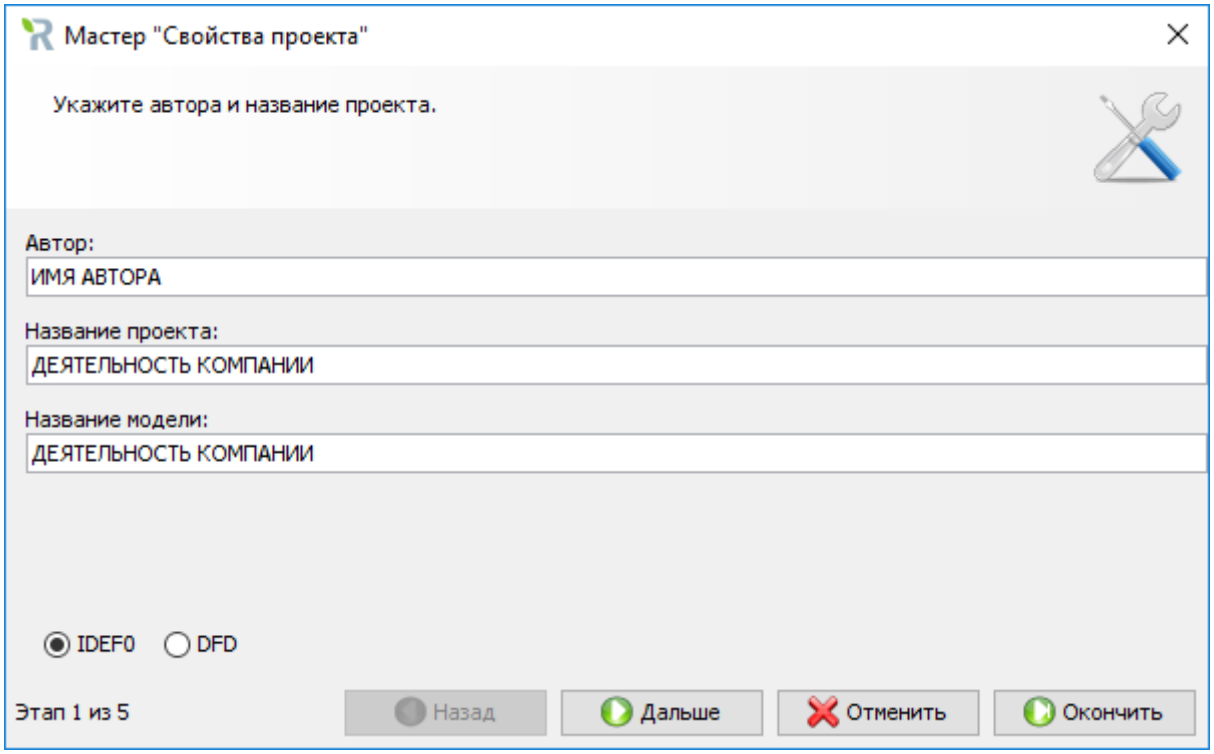


Рисунок 3 – Указание имени модели и выбор необходимой методологии построения модели

Ramus поддерживает две методологии – IDEF0, DFD, каждая из которых решает свои специфические задачи. В BP WIN возможно построение смешанных моделей, т. е. модель может содержать одновременно диаграммы как IDEF0, так и DFD. Состав палитры инструментов изменяется автоматически, когда происходит переключение с одной нотации на другую.

Модель в Ramus рассматривается как совокупность работ, каждая из которых оперирует с некоторым набором данных. Работа изображается в виде прямоугольников, данные — в виде стрелок. Если щелкнуть по любому объекту модели левой кнопкой мыши, появляется контекстное меню, каждый пункт которого соответствует редактору какого-либо свойства объекта.

1.1 Построение модели IDEF0

На начальных этапах создания информационной системы необходимо понять, как работает организация, которую собираются автоматизировать. Руководитель хорошо знает работу в целом, но не в состоянии вникнуть в детали работы каждого рядового сотрудника. Рядовой сотрудник хорошо знает, что творится на его рабочем месте, но может не знать, как работают коллеги. Поэтому для описания работы предприятия необходимо построить модель, которая будет адекватна предметной области и содержать в себе знания всех участников бизнес-процессов организации.

Наиболее удобным языком моделирования бизнес-процессов является IDEF0, где система представляется как совокупность взаимодействующих работ или функций. Такая чисто функциональная ориентация является принципиальной – функции системы анализируются независимо от объектов, которыми они оперируют. Это позволяет более четко смоделировать логику и взаимодействие процессов организации.

Процесс моделирования системы в IDEF0 начинается с создания контекстной диаграммы – диаграммы наиболее абстрактного уровня описания системы в целом, содержащей определение субъекта моделирования, цели и точки зрения на модель.

Под субъектом понимается сама система, при этом необходимо точно установить, что входит в систему, а что лежит за ее пределами, другими словами, определить, что будет в дальнейшем рассматриваться как компоненты системы,

а что как внешнее воздействие. На определение субъекта системы будут существенно влиять позиция, с которой рассматривается система, и цель моделирования – вопросы, на которые построенная модель должна дать ответ. Другими словами, вначале необходимо определить область моделирования. Описание области как системы в целом, так и ее компонентов является основой построения модели. Хотя предполагается, что в ходе моделирования область может корректироваться, она должна быть в основном сформулирована изначально, поскольку именно область определяет направление моделирования. При формулировании области необходимо учитывать два компонента – широту и глубину. Широта подразумевает определение границ модели – что будет рассматриваться внутри системы, а что снаружи. Глубина определяет, на каком уровне детализации модель является завершенной. При определении глубины системы необходимо помнить об ограничениях времени – трудоемкость построения модели растет в геометрической прогрессии с увеличением глубины декомпозиции. После определения границ модели предполагается, что новые объекты не должны вноситься в моделируемую систему.

Цель моделирования

Цель моделирования определяется из ответов на следующие вопросы:

– Почему этот процесс должен быть смоделирован?

– Что должна показывать модель?

– Что может получить клиент?

Точка зрения (Viewpoint).

Под точкой зрения понимается перспектива, с которой наблюдалась система при построении модели. Хотя при построении модели учитываются мнения различных людей, все они должны придерживаться единой точки зрения на модель. Точка зрения должна соответствовать цели и границам моделирования. Как правило, выбирается точка зрения человека, ответственного за моделируемую работу в целом.

IDEF0-модель предполагает наличие четко сформулированной цели, единственного субъекта моделирования и одной точки зрения.

Модели AS-IS и TO-BE. Обычно сначала строится модель существующей организации работы — AS-IS (как есть). Анализ функциональной модели позволяет понять, где находятся наиболее слабые места, в чем будут состоять преимущества новых бизнес-процессов и насколько глубоким изменениям подвергнется существующая структура организации бизнеса. Детализация бизнес-процессов позволяет выявить недостатки организации даже там, где функциональность на первый взгляд кажется очевидной. Найденные в модели AS-IS недостатки можно исправить при создании модели TO-BE (как будет) — модели новой организации бизнес-процессов.

Технология проектирования информационной системы подразумевает сначала создание модели AS-IS, ее анализ и улучшение бизнес-процессов, то есть создание модели TO-BE, и только на основе модели TO-BE строится модель данных, прототип и затем окончательный вариант ИС.

Иногда текущая AS-IS и будущая TO-BE модели различаются очень сильно, так что переход от начального к конечному состоянию становится неочевидным. В этом случае необходима третья модель, описывающая процесс перехода от начального к конечному состоянию системы, поскольку такой переход — это тоже бизнес-процесс.

Модели IDEF0

IDEF0 сочетает в себе небольшую по объему графическую нотацию со строгими и четко определенными рекомендациями, в совокупности предназначенными для построения качественной и понятной модели системы. Первый шаг при построении модели IDEF0 заключается в определении *назначения* модели – набора вопросов, на которые должна отвечать модель. *Границы моделирования* предназначены для обозначения ширины охвата предметной области и глубины детализации и являются логическим продолжением уже определенного назначения модели. Следующим шагом является предполагаемая *целевая аудитория*, для нужд которой создается

модель. Под *точкой зрения* понимается перспектива, с которой наблюдалась система при построении модели.

Поскольку модели IDEF0 представляют систему как множество иерархических (вложенных) функций, в первую очередь должна быть определена функция, описывающая систему в целом – контекстная функция. Функции изображаются на диаграммах как поименованные прямоугольники, или функциональные блоки. Любой блок может быть *декомпозирован* на составляющие его блоки.

Для отображения категорий информации, присутствующих на диаграммах IDEF0, существует аббревиатура ICOM, отображающая четыре возможных типа стрелок:

– I (Input) – вход – нечто, что потребляется в ходе выполнения процесса;

– C (Control) – управление – ограничения и инструкции, влияющие на вход выполнения процесса;

– O (Output) – выход – нечто, являющееся результатом выполнения процесса;

– M (Mechanism) – исполняющий механизм – нечто, что используется для выполнения процесса, но не потребляет само себя.

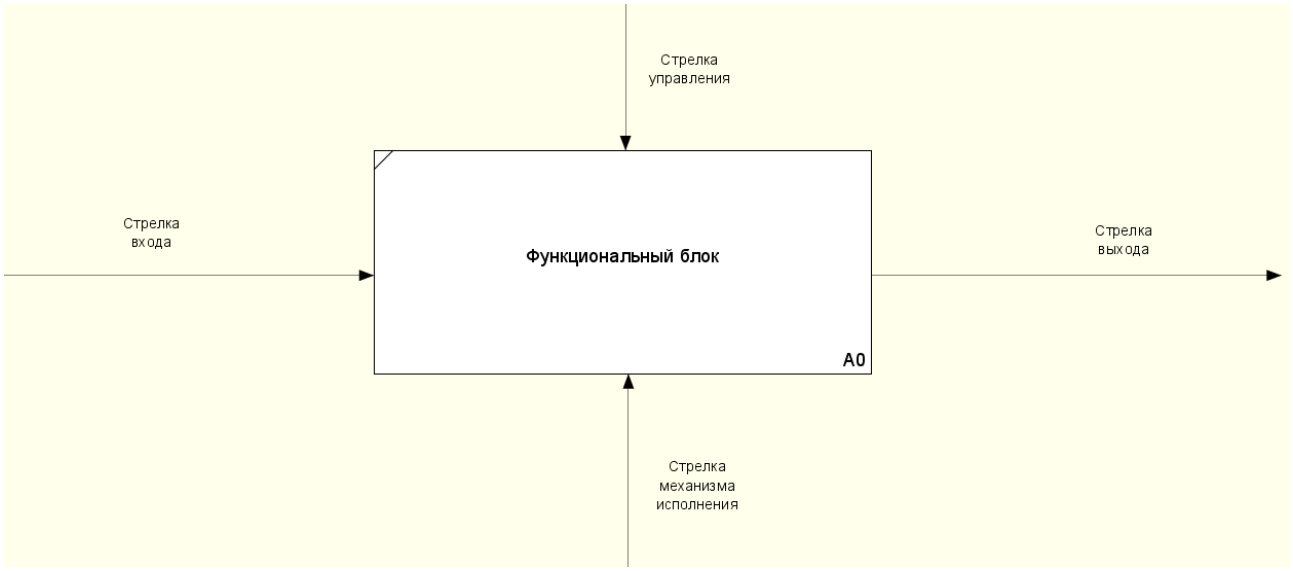


Рисунок 4 – Схема диаграммы IDEF0

Пример. В качестве примера рассмотрим деятельность вымышленной компании Quill, которая существует 5 лет и занимается в основном сборкой и продажей настольных компьютеров и ноутбуков. Годовой оборот компании составляет примерно 20 млн. долларов. Компания закупает компоненты для компьютеров от трех независимых поставщиков, а не производит компоненты самостоятельно. Она только собирает и тестирует компьютеры. Компания реализует продукцию через магазины и специализируется на покупателях, для которых главный критерий при покупке – стоимость компьютера. Предполагаемый объем рынка для компании Quill, в последующие 2 года – 50 млн. долларов.

Несмотря на некоторое увеличение продаж, прибыли уменьшаются, растет конкуренция на рынке. Чтобы не потерять позиции компания решает проанализировать текущие бизнес-процессы и реорганизовать их с целью увеличения эффективности производства и продаж. Основные процедуры в компании таковы:

- продавцы принимают заказы клиентов;
- операторы группируют заказы по типам компьютеров;
- операторы собирают и тестируют компьютеры;
- операторы упаковывают компьютеры согласно заказам;
- кладовщик отгружает клиентам заказы.

В настоящее время компания Quill использует купленную бухгалтерскую информационную систему, которая позволяет оформить заказ, счет и отследить платежи по счетам.

Улучшение деятельности компании должно касаться структуры управления компанией, эффективности производства и внутреннего контроля. В результате реорганизации может потребоваться внедрение новой корпоративной информационной системы (состоящей не только из одного бухгалтерского модуля).

Однако перед тем, как пытаться производить какие-то улучшения, необходимо разобраться в существующих бизнес-процессах.

1.2 Создание контекстной диаграммы

Для создания контекстной диаграммы выполните следующие действия:

1. Запустите Ramus. Появляется диалоговое окно начала работы, в котором следует выбрать пункт создания нового файла (рисунок 2).

2. В появившемся диалоговом окне внесите имя модели (Деятельность компании Quill), имя автора и выберите тип создаваемой диаграммы <IDEF0> (рисунок 3) и нажмите кнопку «Дальше».

3. Для создания контекстной диаграммы выберите мышью пункт «режим добавления функциональных блоков» (рисунок 5) и щелкните левой кнопкой мыши по рабочему пространству модели, куда следует поместить выбранный блок.

4. Для Создания стрелки выберите левой кнопкой мыши «режим работы со стрелками» (рисунок 6), затем поднеся указатель мыши к соответствующей части модели установите начало стрелки (рисунок 7), щелкнув мышью первый раз, затем установите конец стрелки (рисунок 8), щелкнув мышью во второй раз. Создайте стрелки на контекстной диаграмме согласно таблице 1.

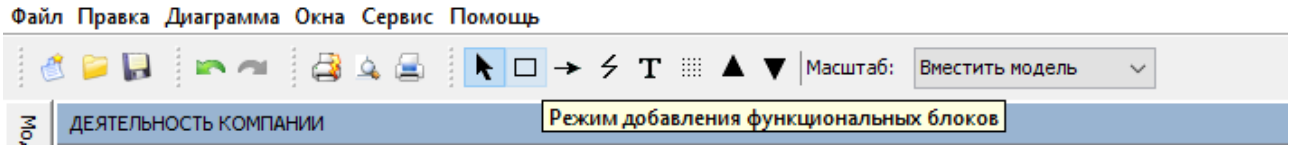


Рисунок 5 – Режим добавления функциональных блоков

Таблица 1 – Стрелки для контекстной диаграммы

Наименование стрелки	Описание	Тип
Бухгалтерская система	Оформление счетов, оплата счетов, работа с заказами	Механизм
Звонки клиентов	Запросы информации, заказы и т. д.	Вход

Правила и процедуры	Правила продажи, инструкции по сборке, процедуры тестирования, критерии производительности и т. д.	Управление
Проданные продукты	Настольные и портативные компьютеры	Выход

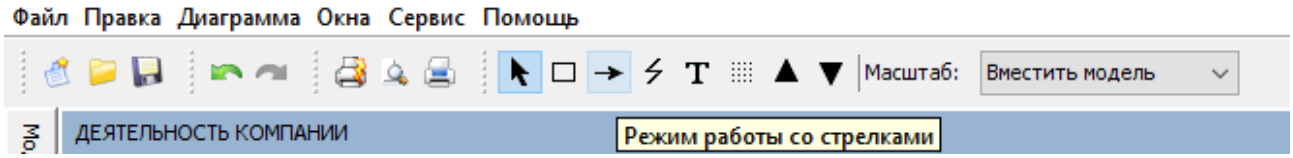


Рисунок 6 – Режим работы со стрелками

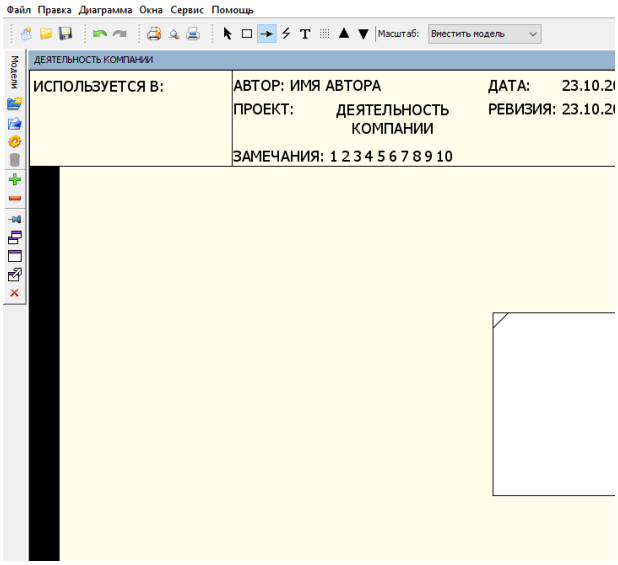


Рисунок 7 – Устанавливаем начало стрелки

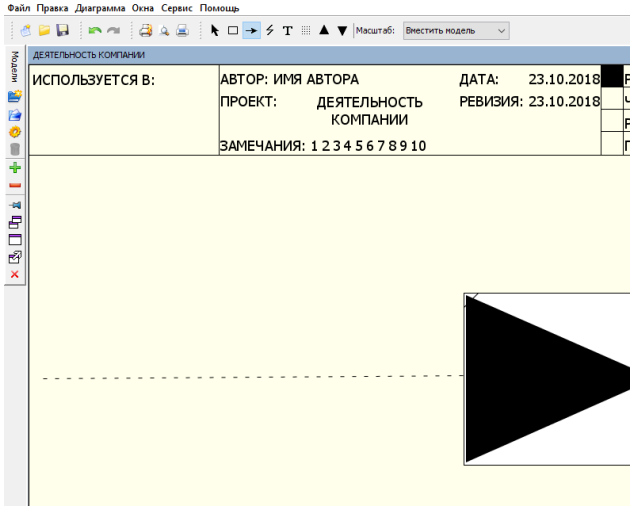


Рисунок 8 – Устанавливаем конец стрелки

Получившаяся контекстная диаграмма должна принять вид, как на рисунке 9.

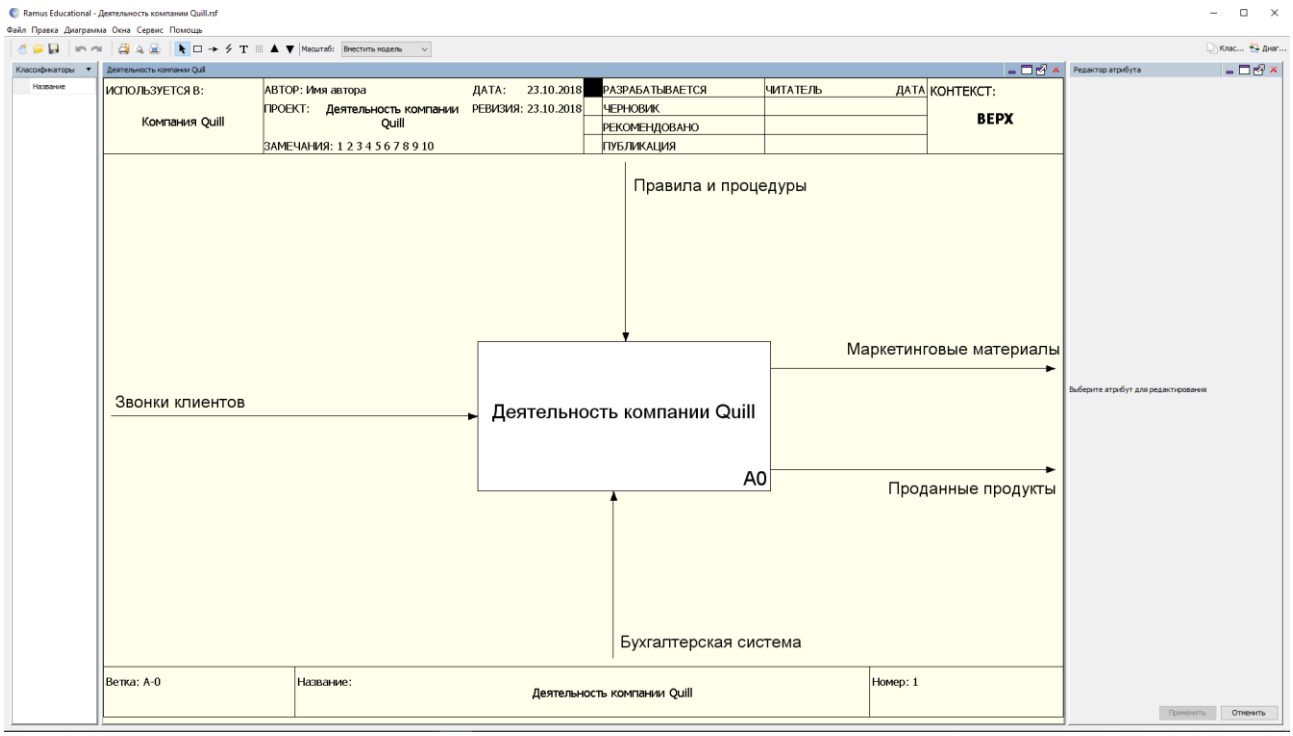


Рисунок 9 – Контекстная функциональная диаграмма IDEF0 для деятельности компании Quill

1.3 Создание диаграммы декомпозиции

Выберите кнопку перехода на нижний уровень в палитре инструментов (рисунок 10), в появившемся диалоговом окне установите число работ 3 (рисунок 11).

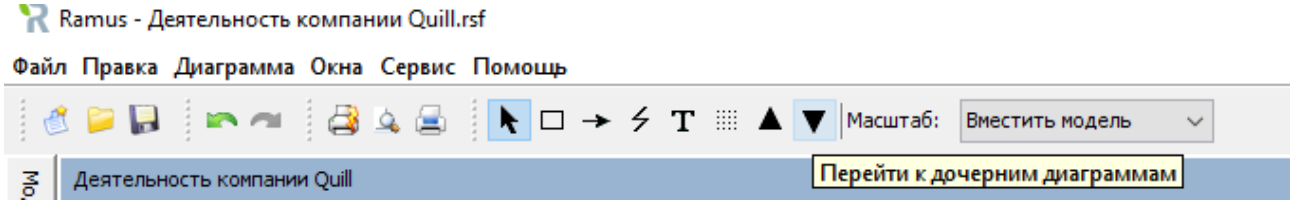


Рисунок 10 – Перейти к дочерним диаграммам

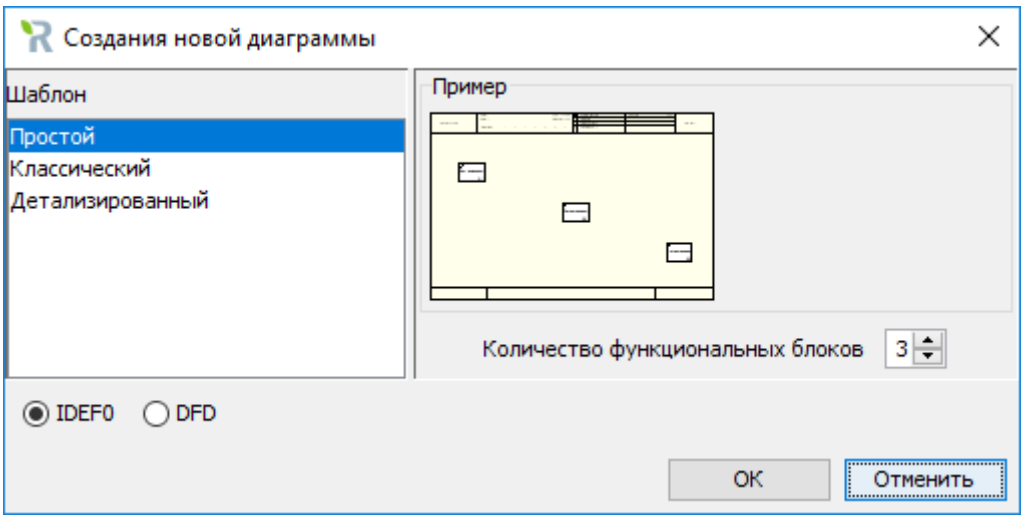


Рисунок 11 – Создание новой диаграммы

На диаграмме расположите работы, описанные в таблице 2.

Таблица 2 – Описание работ на диаграмме декомпозиции первого уровня

Функциональный блок	Описание	Статус	Источник
Продажи, маркетинг	Телемаркетинг, презентации, выставки…	WOKING	Материалы курса
Сборка, тестирование компьютеров	Сборка и тестирование настольных и портативных компьютеров	WOKING	Материалы курса
Отгрузка, получение	Отгрузка заказов клиентам и получение компонентов от поставщиков	WOKING	Материалы курса

Затем перейдите в режим рисования стрелок. Свяжите граничные стрелки так, как показано на рисунке 12.

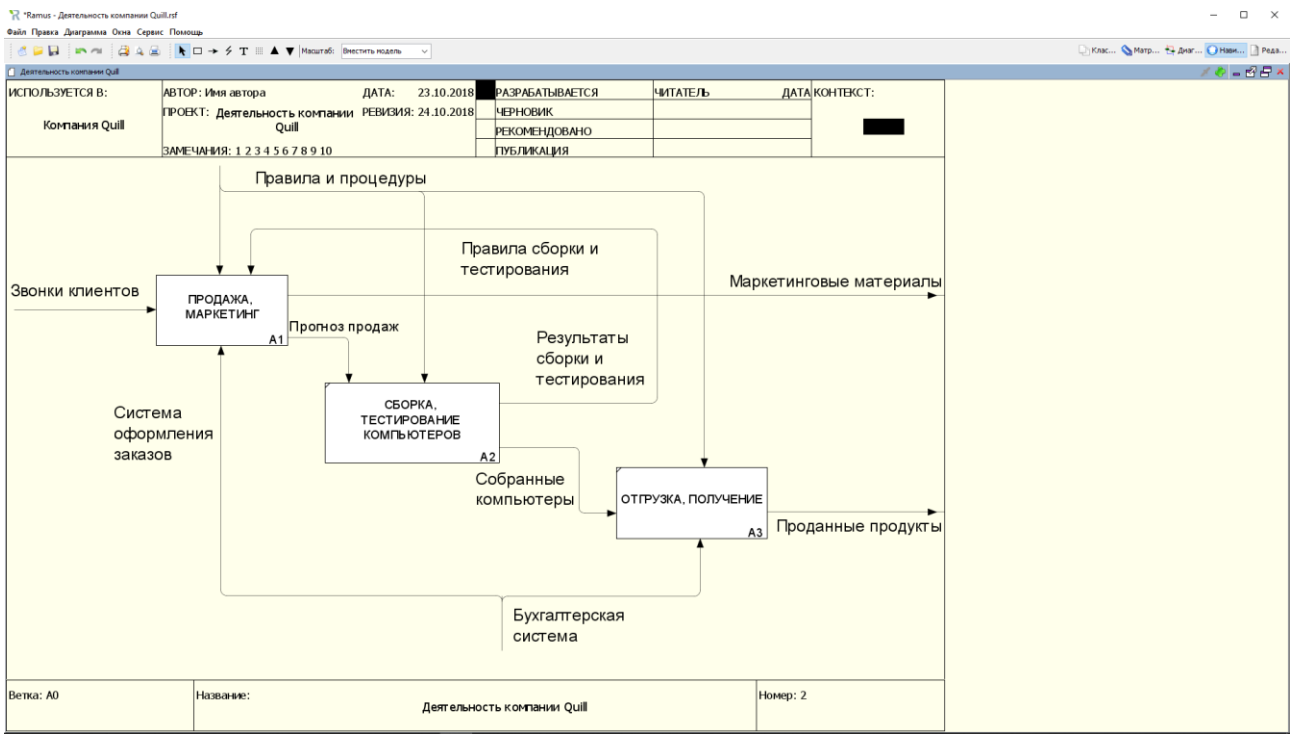


Рисунок 12 – Диаграмма декомпозиции первого уровня IDEF0 для деятельности компании Quill

Новая граничная стрелка «Маркетинговые материалы» автоматически не попадет на диаграмму верхнего уровня и имеет квадратные скобки на наконечнике (Tunnel), чтобы это исправить добавьте новую стрелку с соответствующим именем на контекстной диаграмме (рисунок 13).

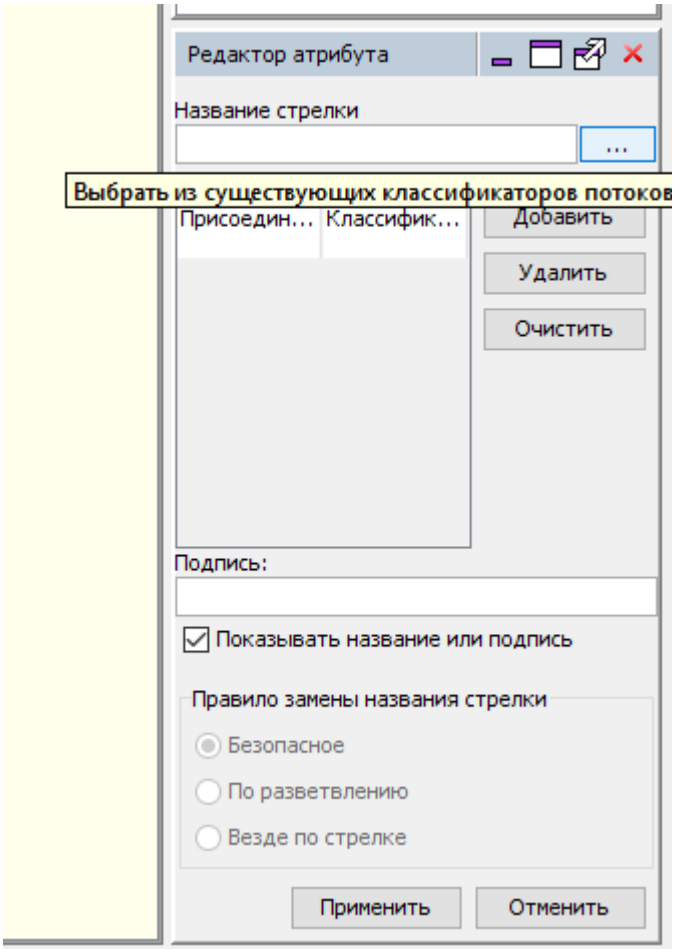


Рисунок 12 – Выбор стрелки из уже существующих классификаторов потоков

1.4 Создание диаграммы декомпозиции

Декомпозируется работа «Сборка и тестирование компьютеров». В результате проведения экспертизы получена следующая информация:

– производственный отдел получает заказы клиентов от отдела продаж по мере их поступления;

– диспетчер координирует работу сборщиков, сортирует заказы, группирует их и дает указания на отгрузку компьютеров, когда они готовы;

– каждые 2 часа диспетчер группирует заказы отдельно для настольных компьютеров и ноутбуков и направляет на участок сборки;

– сотрудники участка сборки собирают компьютеры согласно спецификациям заказа и инструкциям по сборке. Когда группа компьютеров, соответствующая группе заказов, собрана, она направляется на тестирование. Тестируется каждый компьютер, в случае необходимости заменяются неисправные компоненты;

– тестировщики направляют результаты тестирования диспетчеру, который на основании этой информации принимает решение о передаче компьютеров соответствующей группы заказов на отгрузку.

На основании этой информации создайте диаграмму декомпозиции для блока «Сборка и тестирование компьютеров», где описание бизнес-процессов приведено в таблице 3, описание стрелок в таблице 4.

Таблица 3 – Описание бизнес-процессов для работы «Сборка и тестирование компьютеров»

Функциональный блок	Описание	Статус
Отслеживание расписания и управление сборкой и тестированием	Просмотр заказов, установка расписания выполнения заказов, просмотр результатов тестирования, формирование групп заказов на сборку и отгрузку	WOKING
Сборка настольных компьютеров	Сборка настольных компьютеров в соответствии с инструкциями и указаниями диспетчера	WOKING
Сборка ноутбуков	Сборка ноутбуков в соответствии с инструкциями и указаниями диспетчера	WOKING
Тестирование компьютеров	Тестирование компьютеров и компонент. Замена неработающих компонент	WOKING

Таблица 4 – Описание стрелок для декомпозиции работы «Сборка и тестирование компьютеров»

Стрелка	Источник	Тип	Назначение	Тип назначения
Диспетчер	Персонал производственного отдела	Механизм	Отслеживание расписания и управление сборкой и тестированием	Механизм
Заказы клиентов	{Border}	Управление	Отслеживание расписания и управление сборкой и тестированием	Управление

Заказы на настольные компьютеры	Отслеживание расписания и управление сборкой и тестированием	Выход	Сборка настольных компьютеров	Управление
Заказы на ноутбуки	Отслеживание расписания и управление сборкой и тестированием	Выход	Сборка ноутбуков	Управление
Компоненты	{Tunnel}	Вход	Сборка настольных компьютеров Сборка ноутбуков Тестирование компьютеров	Вход
Настольные компьютеры	Сборка настольных компьютеров	Выход	Тестирование компьютеров	Вход
Ноутбуки	Сборка ноутбуков	Выход	Тестирование компьютеров	Вход
Персонал производственного отдела	{Tunnel}	Механизм	Сборка настольных компьютеров Сборка ноутбуков	Механизм
Правила сборки и тестирования	Правила и процедуры	Управление	Сборка настольных компьютеров Сборка ноутбуков Тестирование компьютеров	Управление
Результаты сборки и тестирования	Сборка настольных компьютеров Сборка ноутбуков Тестирование компьютеров	Выход	{Border}	Выход
Результаты тестирования	Тестирование компьютеров	Выход	Отслеживание расписания и управление сборкой и тестированием	Вход
Собранные компьютеры	Тестирование компьютеров	Выход	{Border}	Выход
Тестировщик	Персонал производственного отдела	Тестирование компьютеров		Механизм

Указание передать компьютеры на отгрузку	Отслеживание расписания и управление сборкой и тестированием	Выход	Тестирование компьютеров	Управление

Туннелируйте и свяжите на верхнем уровне граничные стрелки, если это необходимо.

1.5 Диаграммы потоков данных (DFD-модель)

Диаграммы потоков данных моделируют систему как набор действий, соединенных друг с другом стрелками. Диаграммы потоков данных содержат два новых типа объектов: объекты, собирающие и хранящие информацию – *хранилища данных*, и *внешние сущности* – объекты, которые моделируют взаимодействия с теми частями системы, которые выходят за границы моделирования.

В отличие от стрелок в IDEF0, которые иллюстрируют отношения, стрелки в DFD показывают, как объекты (включая и данные) реально перемещаются от одного действия к другому. Это представление потока в сочетании с хранилищами данных и внешними сущностями обеспечивает отражение в DFD-моделях таких физических характеристик как *движение* объектов (потоки данных), *хранение* объектов (хранилища данных), *источники и потребители* объектов (внешние сущности). Пример контекстной диаграммы DFD показан на рисунке 13.

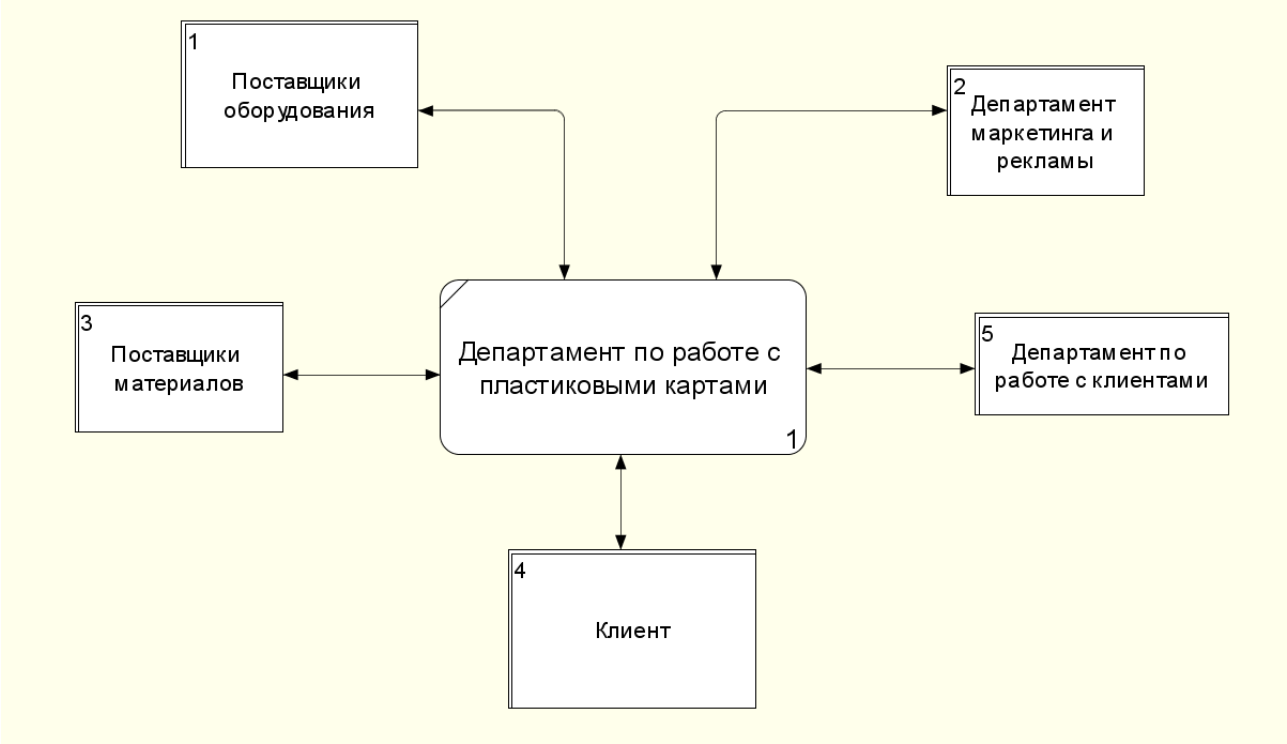


Рисунок 13 – Пример контекстной диаграммы DFD

Контекстная DFD-диаграмма часто состоит из одного функционального блока (имя которого совпадает с именем всей системы) и нескольких внешних сущностей.

Функциональный блок DFD (изображается в виде прямоугольника с закругленными углами) моделирует некоторую функцию, которая преобразует какое-либо сырье в какую-либо продукцию.

Внешние сущности обеспечивают необходимые входы для системы и / или являются приемником для ее выходов. Внешние сущности изображаются как прямоугольники и обычно размещаются у краев диаграммы. Одна и та же внешняя сущность может быть размещена на одной и той же диаграмме в нескольких экземплярах.

Стрелки (потоки данных) описывают передвижение (поток) объектов от одной части системы к другой. Поскольку все стороны обозначающего функциональный блок DFD прямоугольника равнозначны, стрелки могут начинаться и заканчиваться в любой части блока. В DFD также используются двунаправленные стрелки, обозначающие взаимный обмен информацией.

Хранилища данных – при моделировании производственных систем хранилищами данных служат места временного складирования, где хранится продукция на промежуточных стадиях обработки. В информационных системах хранилища данных представляют любой механизм, который поддерживает хранение данных для их промежуточной обработки.

1.6 Декомпозиция процесса «Продажа и маркетинг».

Работа по продажам и маркетингу заключается в ответах на телефонные звонки клиентов, предоставлении клиентам информации о ценах, оформлении заказов, внесении заказов в информационную систему и исследование рынка. На основе этой информации декомпозируйте работу «Продажа и маркетинг» (IDEF0).

Создайте следующие работы (рисунок 14):

– Предоставление информации о ценах.
– Оформление заказов.
– Исследование рынка.

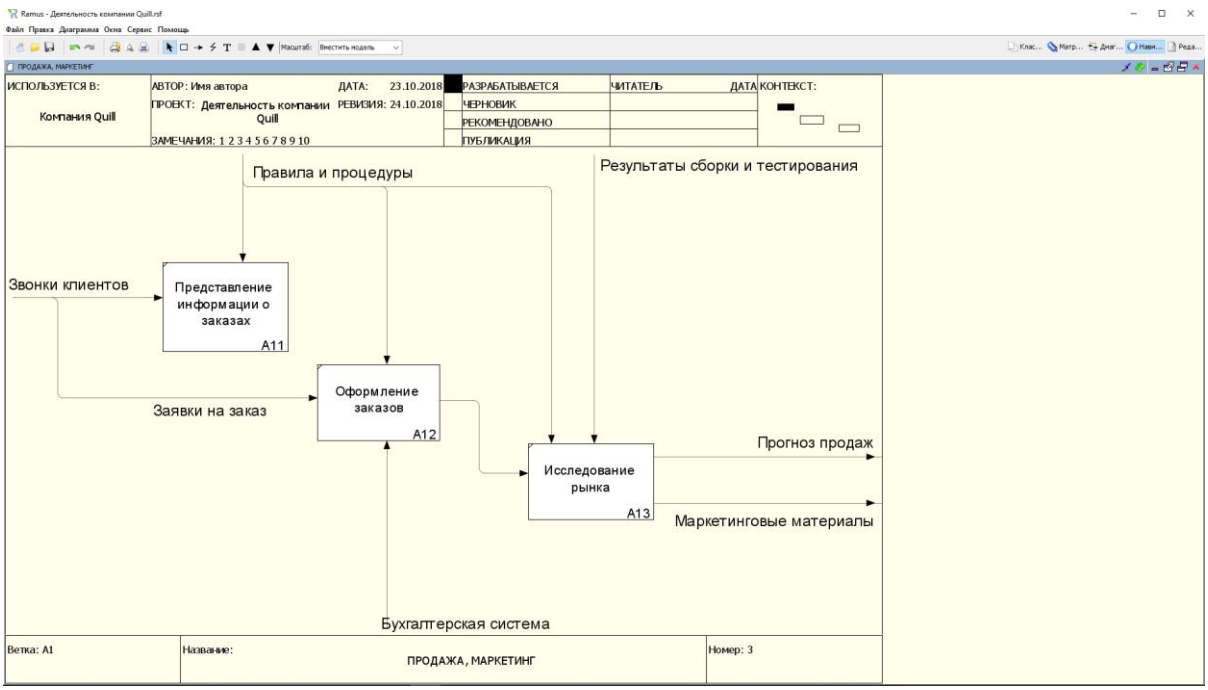


Рисунок 14 – Декомпозиция процесса «Продажа и маркетинг»

1.7 Создание диаграммы DFD

При оформлении заказа важно проверить, существует ли такой клиент в нашей базе данных, если нет – внести его в базу данных, затем оформить заказ. Оформление заказа начинается со звонка клиента. В процессе оформления заказа базы данных клиентов может редактироваться. Заказ должен включать информацию о клиенте и заказанной им продукции.

В процессе декомпозиции согласно правилам DFD мы преобразуем граничные стрелки во внутренние, начинающиеся и заканчивающиеся на внешних ссылках.

1. Декомпозируйте работу «Оформление заказов» в нотации DFD. Внесите имена работ: проверка и внесение клиента, и внесение заказа.

2. Внесите хранилища данных: список клиентов, список продуктов, список заказов. Чтобы задать их обозначение откройте свойство объекта, на вкладке «Объект» нажмите кнопку «Задать DFD объект». Далее выберите один из классификаторов (если нет ни одного, создайте в панели классификаторов) и создайте необходимые элементы, затем задайте соответствие между объектом DFD и элементом, нажав кнопку «ОК» (рисунок 15).

3. Удалите граничные стрелки с диаграммы. Внесите внешнюю ссылку – звонки клиентов.

4. Создайте внутренние ссылки. При переименовании стрелок используйте словарь.

5. Чтобы сделать стрелку двунаправленной в контекстном меню выберите пункт <Style Editor>.

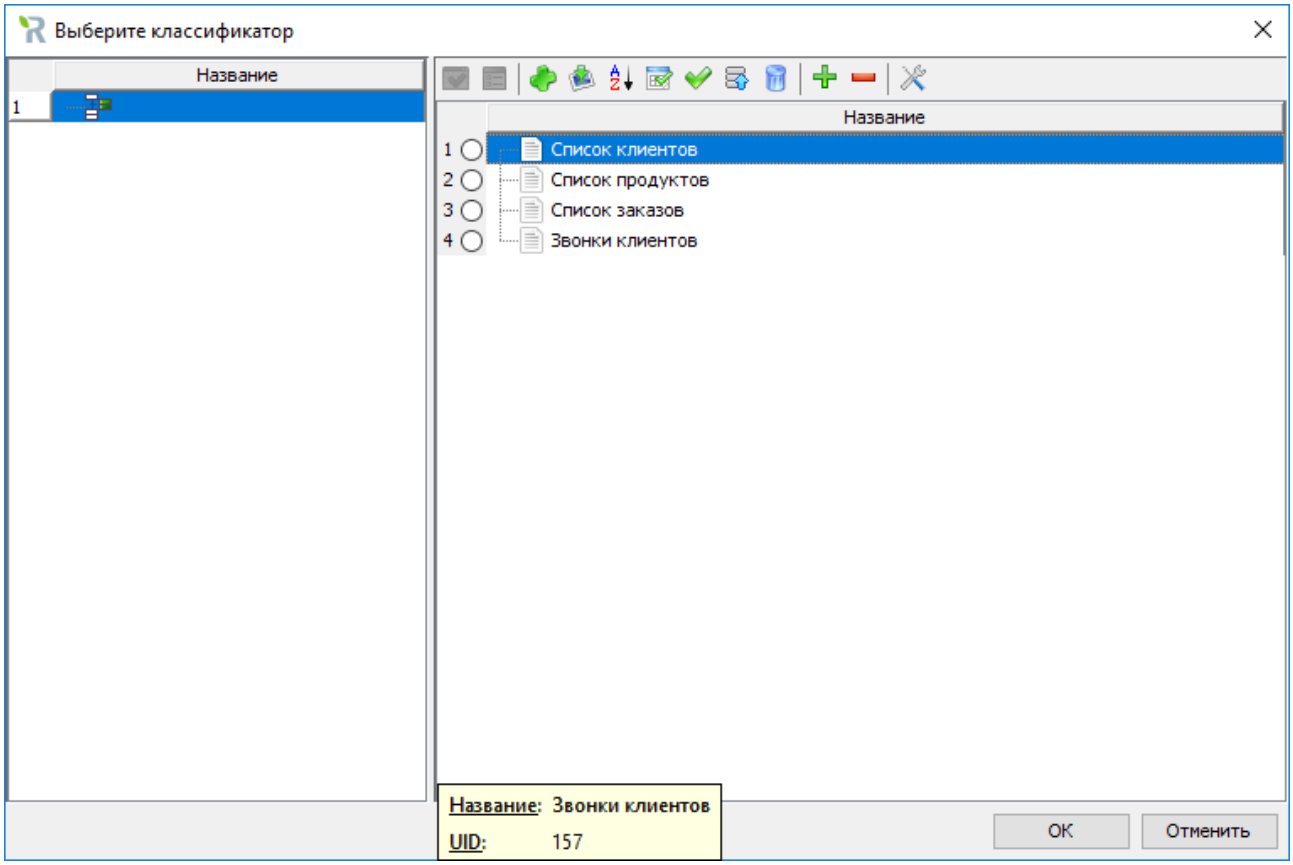


Рисунок 15 – Выбор классификатора объекта

1.8 Создание связи между стрелками

1. Декомпозируйте работу «Исследования рынка». Удалите граничные стрелки. Создайте следующие работы: разработка прогнозов продаж, разработка маркетинговых материалов, привлечение новых клиентов.

2. Внесите хранилища данных: список клиентов, список продуктов, список заказов.

3. Добавьте две внешние ссылки: маркетинговые материалы, проверка и внесение клиента.

4. Свяжите объекты диаграммы DFD.

5. В случае внесения новых клиентов в работу «Проверка и внесение клиента» на диаграмме A12 «Оформление заказов» информация должна направляться к работе «Привлечение новых клиентов» диаграммы A13 «Исследование рынка». Для этого необходимо использовать инструмент <Off-

Page Reference>. На диаграмме A12 «Оформление заказов» создайте новую граничную стрелку, исходящую от работы «Проверка и внесение клиента», и назовите ее «Информация о новом клиенте».

Результат декомпозиции представлен на рисунке 16.

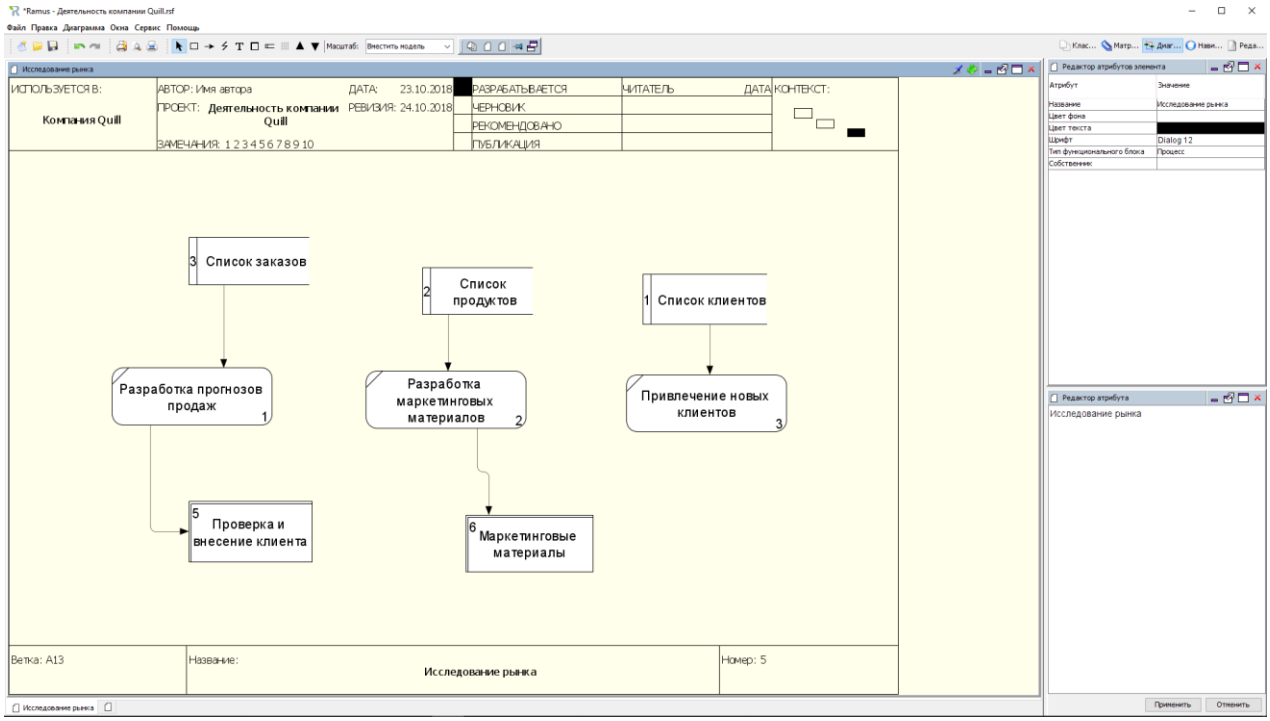


Рисунок 16 – Декомпозиция объекта «Исследование рынка»

1.9 Контрольное задание

1. Исследуйте и проанализируйте предметную область информационной системы согласно варианту, выбранному вами в прошлом семестре.

2. Создайте модель предметной области информационной системы по выбранному варианту.

1.10 Контрольные вопросы

1. Дайте понятие предметной области информационной системы. Привести примеры.

2. Сформулируйте определение модели предметной области.

3. Что собой представляет язык моделирования.

4. Что такое объект?

5. Что такое функция?

6. Дайте определение организационной единице.

7. Сформулируйте определение организационной структуры.

2 Лабораторная работа № 2. Построение технического задания

Цель работы: научиться определять цели, требования и основные исходные данные, необходимые для разработки автоматизированной информационной системы.

Используемое программное обеспечение: Microsoft Word 2016.

Техническое задание – это документ, определяющий цели, требования и основные исходные данные, необходимые для разработки автоматизированной системы управления.

Задачи, решаемые при разработке технического задания:

– установить общую цель создания информационной системы, определить состав подсистем и функциональных задач;

– разработать и обосновать требования, предъявляемые к подсистемам;

– разработать и обосновать требования, предъявляемые к информационной базе, математическому и программному обеспечению, комплексу технических средств (включая средства связи и передачи данных);

– установить общие требования к проектируемой системе;

– определить перечень задач создания системы и исполнителей;

– определить этапы создания системы и сроки их выполнения;

– провести предварительный расчет затрат на создание системы и определить уровень экономической эффективности ее внедрения.

2.1 Структура технического задания. Общие сведения

– полное наименование системы и ее условное обозначение;

– шифр темы или шифр (номер) договора;

– наименование предприятий разработчика и заказчика системы, их реквизиты;

– перечень документов, на основании которых создается информационная система;

– плановые сроки начала и окончания работ;

– сведения об источниках и порядке финансирования работ;

– порядок оформления и предъявления заказчику результатов работ по созданию системы, ее частей и отдельных средств.

2.2 Назначение и цели создания (развития) системы

– вид автоматизируемой деятельности;

– перечень объектов, на которых предполагается использование системы;

– наименования и требуемые значения технических, технологических, производственно-экономических и других показателей объекта, которые должны быть достигнуты при внедрении информационной системы.

Характеристика объектов автоматизации:

– краткие сведения об объекте автоматизации;

– анализ существующего уровня автоматизации;

– схема бизнес-процесса;

– сведения об условиях эксплуатации и характеристиках окружающей среды.

2.3 Требования к системе

– требования к структуре и функционированию системы (перечень подсистем, уровни иерархии, степень централизации, способы информационного обмена, режимы функционирования, взаимодействие со смежными системами, перспективы развития системы);

– требования к персоналу (численность пользователей, квалификация, режим работы, порядок подготовки);

– показатели назначения (степень приспособляемости системы к изменениям процессов управления и значений параметров);

– требования к надежности, безопасности, эргономике, транспортабельности, эксплуатации, техническому обслуживанию и ремонту, защите и сохранности информации, защите от внешних воздействий, к патентной чистоте, по стандартизации и унификации.

2.4 Требования к функциям (по подсистемам)

– перечень подлежащих автоматизации задач;

– временной регламент реализации каждой функции;

– требования к качеству реализации каждой функции, к форме представления выходной информации, характеристики точности, достоверности выдачи результатов;

– перечень и критерии отказов.

2.5 Требования к видам обеспечения

– математическому (состав и область применения математических моделей и методов, типовых и разрабатываемых алгоритмов);

– информационному (состав, структура и организация данных, обмен данными между компонентами системы, информационная совместимость со смежными системами, используемые классификаторы, СУБД, контроль данных и ведение информационных массивов, процедуры придания юридической силы выходным документам);

– лингвистическому (языки программирования, языки взаимодействия пользователей с системой, системы кодирования, языки ввода-вывода);

– программному (независимость программных средств от платформы, качество программных средств и способы его контроля, использование фондов алгоритмов и программ);

– техническому;

– метрологическому;

– организационному (структура и функции эксплуатирующих подразделений, защита от ошибочных действий персонала);

– методическому (состав нормативно- технической документации).

2.5 Состав и содержание работ по созданию системы

– перечень стадий и этапов работ;

– сроки исполнения;

– состав организаций — исполнителей работ;

– вид и порядок экспертизы технической документации;

– программа обеспечения надежности;

– программа метрологического обеспечения.

2.6 Порядок контроля и приемки системы

– виды, состав, объем и методы испытаний системы;

– общие требования к приемке работ по стадиям;

– статус приемной комиссии.

Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие:

– преобразование входной информации к машиночитаемому виду;

– изменения в объекте автоматизации;

– сроки и порядок комплектования и обучения персонала. Требования к документированию:

– перечень подлежащих разработке документов;

– перечень документов на машинных носителях.

Источники разработки:

– документы и информационные материалы, на основании которых разрабатывается техническое задание и система.

2.7 Пример технического задания.

Таблица 1 – Принятые сокращения

Аббревиатура	Определение
БД	База данных
БТФОМС	Белгородский территориальный фонд обязательного медицинского страхования
ЛПУ	Лечебно-профилактическое учреждение
МКБ	Международная статистическая классификация болезней и проблем, связанных со здоровьем, десятого пересмотра
МЭС	Медико-экономический стандарт
ОМС	Обязательное медицинское страхование
ПК	Программный комплекс
ПО	Программное обеспечение
СМО	Страховая медицинская организация
СУБД	Система управления базами данных
ТЗ	Техническое задание
ТФ	Территориальный фонд
ТФОМС	Территориальный фонд ОМС
ФОМС	Федеральный фонд обязательного медицинского страхования

Общие сведения.

Полное наименование системы и ее условное обозначение.

Наименование – «Программный комплекс организации и контроля взаиморасчетов между территориальными фондами ОМС».

Обозначение – ПК ВЗР.

Ниже в данном документе под «Взаиморасчеты» понимается ПК ВЗР.

Наименование предприятий

Заказчика и Подрядчика, и их реквизиты

Заказчик – Белгородский территориальный фонд обязательного медицинского страхования.

ИНН 3124014114 КПП 312301001.

308002, г. Белгород, ул. Мичурина, 62, тел. 26-03-08, факс 34-66-39.

р/с 40404810900000010002 ГРКЦ ГУ Банка России по Белгородской области, г. Белгород.

Код ОКПО 22319104.

Код ОКОНХ 96210.

ОГРН 1023101655521.

Порядок внесения уточнений и изменений

в техническое задание

Настоящее ТЗ может уточняться и дополняться в процессе разработки ПК. Согласование и утверждение дополнений к ТЗ должно проводится в установленном порядке (на основании п. 1.7 ГОСТ 34.602 – 89).

Назначение и цели создания ПК.

Цели создания ПК

Основными целями создания ПК являются:

– обеспечение территориальных фондов ОМС программными средствами для организации и контроля взаиморасчетов между ТФОМС за медицинскую помощь в объеме базовой программы обязательного медицинского страхования граждан Российской Федерации, оказанную гражданам Российской Федерации за пределами территории страхования;

– сокращение трудоемкости, времени обработки данных и подготовки документов, необходимых для обеспечения взаиморасчетов между территориальными фондами ОМС;

– рационализация использования средств системы ОМС путем повышения качества, достоверности, объективности учета и контроля финансовых средств в ходе проведения финансовых взаиморасчетов между территориальными фондами ОМС;

– статистическая обработка и формирование различных отчетов по имеющейся в ПК ВЗР информации.

Основное назначение ПК – обеспечение необходимой информационной поддержки и автоматизированной технологии решения задач ОМС, обеспечение информационного взаимодействия между субъектами ОМС в части проведения взаиморасчетов между территориальными фондами ОМС.

ПК «Взаиморасчеты» предназначен для:

– приема и обработки реестров ЛПУ по пролеченным иногородним пациентам;

– формирование реестров счетов и счетов по оплате медицинских услуг в разрезе ТФОМС, на территории которых застрахованы пролеченные пациенты;

– приема реестров счетов из других ТФОМС о застрахованных гражданах, пролеченных за пределами данной территории, их обработка;

– формирование справок по данным о взаиморасчетах между ТФОМС и о пролеченных иногородних пациентах в ЛПУ территории.

Область применения ПК «Взаиморасчеты»

ПК ВЗР предназначен для установки и использования в ТФОМС. ПК обеспечивает агрегирование информации от всех ЛПУ по пролеченным иногородним пациентам в установленном формате, проверку представленной информации на корректность, проведение медико-экономической экспертизы по определенным условиям; формирование исходящих счетов в ТФОМС за пролеченных иногородних пациентов; прием реестров счетов от других ТФОМС по застрахованным гражданам, пролеченным за пределами территории страхования, их проверку; проведение взаимозачетов.

Объекты автоматизации.

Общие сведения

Оплата медицинской помощи в пределах базовой программы обязательного медицинского страхования граждан Российской Федерации, оказанной гражданам Российской Федерации за пределами территории

страхования, производится по месту оказания медицинской помощи территориальным фондом обязательного медицинского страхования (далее – ТФ-1) по тарифам, действующим на данной территории на момент оказания медицинской помощи. Возмещение суммы, оплаченной ТФ-1, осуществляет территориальный фонд обязательного медицинского страхования по месту страхования гражданина (далее ТФ-2).

Тарифы на медицинские услуги, по которым осуществляются межтерриториальные взаиморасчеты, включают затраты, предусмотренные тарифами на данной территории, по статьям расходов: оплата труда, начисления на оплату труда, медикаменты и перевязочные средства, продукты питания, мягкий инвентарь и обмундирование согласно экономической классификации расходов бюджетов Российской Федерации.

Медицинское учреждение, входящее в систему обязательного медицинского страхования и оказавшее медицинские услуги гражданину, застрахованному по обязательному медицинскому страхованию в другом субъекте Российской Федерации, формирует и направляет реестр и счет (счета) в соответствии с порядком, принятым на территории оказания медицинской помощи, в ТФ-1.

ТФ-1 проверяет предъявленный медицинским учреждением реестр, счет (счета) и, при отсутствии претензий к лечебно-профилактическому учреждению, осуществляет оплату оказанной медицинской помощи.

ТФ-1 предъявляет ТФ-2 к оплате реестр медицинских услуг и счет на общую сумму оказанных медицинских услуг не позднее следующего календарного квартала от даты окончания оказания медицинской помощи. С целью ускорения проведения межтерриториальных взаиморасчетов и совершенствования обработки реестров счетов они направляются с помощью электронной почты.

ТФ-2 производит обработку и оплату счета, предъявленного ТФ-1 (в том числе, повторно полученного), в срок не позднее 30-ти календарных дней от даты получения реестра счета и счета на бумажном носителе.

При наличии претензий по отдельным случаям оказания медицинской помощи, ТФ-2 выставляет мотивированный отказ на бумажном носителе и в электронном виде направляет «Протокол обработки реестра счета» в срок не позднее 30-ти календарных дней от даты получения счета на бумажном носителе.

ТФ-1 дает ответ на мотивированный отказ на бумажном носителе и в электронном виде направляет исправленную часть в срок не позднее 30-ти календарных дней с момента получения мотивированного отказа на бумажном носителе, кроме случаев, требующих проведения экспертизы качества медицинской помощи.

В случаях, требующих проведения экспертизы качества медицинской помощи, она должна осуществляться в течение 3 месяцев с момента получения ТФ-1 мотивированного отказа.

Обоснованными (мотивированными) причинами отсрочки или отказа в оплате счета по каждому отдельному случаю (пациенту) являются следующие:

6. Превышение объема предоставленной медицинской помощи относительно предусмотренного базовой программой обязательного медицинского страхования граждан Российской Федерации, утверждаемой постановлением Правительства Российской Федерации.

7. Завышение тарифов на медицинскую помощь, принятых на территории оказания медицинской помощи иногороднему гражданину.

8. Факты оказания медицинской помощи в медицинском учреждении, не функционирующем в системе обязательного медицинского страхования.

9. Случаи оказания медицинской помощи, требующие экспертизы качества лечения.

10. Заполнение реестра счета с включением недостоверных и ошибочных данных, делающих невозможной идентификацию застрахованных.

ТФ-2 после оплаты счета восстанавливает финансовые средства путем предъявления реестра и счета СМО (филиалу территориального фонда,

выполняющему функцию страховщика), застраховавшей гражданина, или другим, установленным на территории порядком.

ТФ-1, ТФ-2 и страховая медицинская организация (филиал территориального фонда, выполняющий функции страховщика) используют свое право осуществления проверки достоверности тарифов на медицинские услуги, принятых на территории оказания медицинской помощи, в предъявленных им реестрах счетов; контроля объемов и качества медицинской помощи и защиты прав застрахованных.

При проведении экспертизы качества медицинской помощи оформляется акт экспертной, оценки качества медицинской помощи.

Характеристика объектов автоматизации

Объектами автоматизации Подсистемы являются рабочие процессы взаиморасчетов между БТФОМС и другими ТФОМС за медицинскую помощь в объеме базовой программы обязательного медицинского страхования граждан Российской Федерации, оказанную гражданам Российской Федерации за пределами территории страхования, а также процессы взаиморасчетов между БТФОМС и ЛПУ Белгородской области за пролеченных иногородних пациентов, включая:

1. Прием и обработка реестров из ЛПУ по пролеченным иногородним пациентам.

1.1. Прием и накопление информации о пролеченном населении других регионов (реестры и счета), поступающей от ЛПУ.

1.2. Техническая и медико-экономическая проверка реестров.

1.3. Формирование протокола обработки информации и файла, содержащего ошибочные записи для передачи в ЛПУ.

2. Формирование реестров счетов по оплате медицинских услуг, оказанных иногородним больным, включая:

2.1. Формирование сводной информации о пролеченном иногороднем населении для каждого ТФОМС.

2.2. Формирование реестров счетов по оплате медицинских услуг и счета на общую сумму медицинских услуг, оказанных иногородним больным, для предъявления в другие ТФОМС.

2.3. Учет выставленных реестров счетов по оплате медицинских услуг и счета на общую сумму медицинских услуг, оказанных иногородним больным.

2.4. Учет оплаченных счетов.

2.5. Формирование файла «Реестр счета по оплате медицинских услуг».

2.6. Формирование файла «Протокол обработки реестра счета» и передача его другому ТФОМС.

2.7. Формирование исправленных файлов «Реестр счета по оплате медицинских услуг» и передача их другим ТФОМС.

2.8. Прием и учет «Файлов обработки реестра счета», поступивших от других ТФОМС.

2.9. Формирование письма «Ответ на мотивированный отказ об оплате позиций счета по межтерриториальным расчетам».

3. Прием и обработка реестров счетов от других ТФОМС:

3.1. Прием реестров счетов по оплате медицинских услуг и счета на общую сумму оказанных медицинских услуг больным, застрахованным БТФОМС, но пролеченным на территории других ТФОМС.

3.2. Контроль на неповторяемость реестров счетов, выставляемых другими ТФОМС.

3.3. Проверка на ошибки информации о медицинской помощи, оказанной жителям Белгородской области на территории других ТФОМС (проверка правильности заполнения и наличия обязательной информации, медицинская экспертиза, сверка данных о пациенте со сводным регистром застрахованных).

3.4. Формирование файла «Протокол обработки реестра счета» и передача его другим ТФОМС.

3.5. Формирование письма «Мотивированный отказ об оплате позиций счета по межтерриториальным расчетам».

3.6. Прием файлов «Протокол обработки реестра счета» и исправленных файлов «Реестр счета по оплате медицинских услуг» из других ТФОМС.

3.7. Расчет платежа по поступившим от других ТФОМС реестрам счетов.

3.8. Подготовка данных для формирования платежных поручений на оплату счетов, выставленных другими ТФОМС и прошедшими экспертизу.

3.9. Учет оплаченных счетов.

4. Взаимозачеты между территориальными фондами ОМС:

4.1. Подготовка данных для проведения взаимозачетов с другими ТФОМС.

4.2. Формирование «Акта о согласовании проведения взаимозачета по финансовым расчетам между ТФОМС».

4.3. Учет сумм, зачтенных в порядке взаимной задолженности.

5. Анализ и контроль хода взаиморасчетов между ТФОМС, БТФОМС и ЛПУ и подготовка отчетности по взаиморасчетам.

5.1. Анализ следующих показателей за выбранный период:

5.1.1. Количество иногородних пациентов, пролеченных в ЛПУ области и объем выплат ЛПУ: суммы, представленные к оплате ЛПУ; суммы, принятые к оплате; суммы, снятые в результате экспертизы, причины снятия.

5.1.2. Объем счетов, выставленных БТФОМС в адрес других ТФОМС, на возмещение затрат по оплате медицинских услуг, оказанных ЛПУ Белгородской области иногородним больным.

5.1.3. Объем средств, поступивших от других ТФОМС в счет оплаты медицинских услуг, оказанных ЛПУ Белгородской области иногородним больным.

5.1.4. Объем счетов, отказанных в оплате другими ТФОМС и причины отказа.

5.1.5. Объем счетов, выставленных другими ТФОМС в адрес БТФОМС, за медицинские услуги, оказанные жителям Белгородской области вне территории их проживания.

5.1.6. Объем счетов, принятых БТФОМС к оплате от других ТФОМС по результатам медико-экономической экспертизы счетов.

5.1.7. Объем счетов, отказанных в оплате другими ТФОМС по результатам экспертизы и причины отказа.

5.1.8. Объем произведенной оплаты другим ТФОМС за медицинские услуги, оказанные жителям Белгородской области вне территории их проживания.

5.2. Контроль сроков оплаты предъявленных счетов (в том числе повторно полученных).

5.3. Контроль сроков оплаты выставленных счетов.

5.4. Контроль сроков представления мотивированного отказа об оплате позиций счета по межтерриториальным расчетам и «Протокола обработки реестра счета».

5.5. Контроль сроков представления ответа на мотивированный отказ об оплате позиций счета по межтерриториальным расчетам и исправленной части «Реестра счета по оплате медицинских услуг».

5.6. Формирование ежеквартальной отчетной формы № 2 – расчеты «Сведения о финансовых расчетах между территориальными фондами ОМС за медицинскую помощь, оказанную за пределами страхования гражданина Российской Федерации», в соответствии с приказом ФОМС № 26 от 20 марта 1998 г. (Информационный обмен между ТФОМС в части взаиморасчетов должен учитывать технические требования к Единой интегрированной системе управления отраслевой отчетностью и обменом данными (ЕСООД) между субъектами системы ОМС).

5.7. Формирование сводных отчетов по состоянию взаиморасчетов с другими ТФОМС.

Требования к ПК ВЗР.

Общие требования к ПК

ПК должен реализовывать функции обеспечения взаиморасчетов между ТФОМС.

Согласование внешнего вида экранных форм и формируемых выходных документов должно осуществляться в ходе технорабочего проектирования. Все

согласованные экранные формы и формируемые выходные документы должны быть включены в состав проектной документации на ПК.

Минимально необходимый состав справочников, классификаторов и словарей, обеспечивающих функционирование ПК, должен быть определен на этапе технорабочего проектирования.

ПК может дорабатываться по желанию Заказчика с расширением ее функциональности и возможностей за рамки настоящего технического задания. Доработка ПК производится по отдельному техническому заданию.

Требования к автоматизируемым функциям.

Общие требования

В ПК «Взаиморасчеты» должны формироваться и поддерживаться в актуальном состоянии следующие базы данных:

– список счетов от ЛПУ, содержащий информацию о пролеченных иногородних пациентах и оказанных им услугах;

– список счетов, выставленных в другие ТФОМС;

– список входящих счетов, выставленных другими ТФОМС БТФОМС;

– данные о взаимозачетах;

– справочные подсистемы. Справочники содержат информацию, являющуюся общей для всех субъектов системы обязательного медицинского страхования как межтерриториального, так и местного уровня, а также наиболее часто используемые в программе ВЗР данные.

– ПК должен обеспечивать выполнение функций в соответствии разделом 0 настоящего ТЗ.

– Более детальные требования к реализации функций должны быть уточнены на этапе технорабочего проектирования.

Требования к форматам и структуре входных / выходных файлов.

В ПК должен быть реализован импорт данных из следующих файлов, предоставляемых ЛПУ в ТФОМС по пролеченным иногородним пациентам:

– реестров пролеченных пациентов;

– счет-фактур пролеченных пациентов.

Формат файлов DBF.

В ПК для передачи в ЛПУ должен экспортироваться файл ошибок, содержащий ошибочные записи по результатам экспертизы.

Формат и структура входных / выходных файлов по взаимодействию между ТФОМС описан в приложении 2 к Приказу ФОМС № 70 от 23 августа 2000 г. «О порядке финансовых расчетов между территориальными фондами обязательного медицинского страхования за медицинскую помощь в объеме базовой программы обязательного медицинского страхования граждан Российской Федерации, оказанную гражданам Российской Федерации за пределами территории страхования».

Требования к работе с информацией БД, сформированной на основе данных из ЛПУ о пролеченных иногородних пациентах.

В программе ВЗР должны быть предусмотрены диалоговые окна просмотра сформированной БД в виде таблицы. В таблице с информацией о счетах из ЛПУ по пролеченным иногородним пациентам должна отображаться следующая информация (с возможностью предварительного выбора ЛПУ и периода счета):

– наименование ЛПУ.
– код ОКПО ЛПУ.
– месяц (номер п / п информации) / год.
– кол-во пролеченных пациентов.
– представленная сумма (руб. / коп.).
– снятая сумма (руб. / коп.).
– оплачено (руб. / коп.).
– признак проверки на ошибки.
– признак сформированных счетов в ТФ.
– признак наличия сформированного файла ошибок в ЛПУ.

Необходимые основные функциональные возможности:

1) печать списка отобранных счетов;

2) отбор счетов как по одному, так и по группе параметров;

3) для выбранного счета – печать «Реестра оказанных медицинских услуг гражданам, застрахованным за пределами территории»;

4) проверка на ошибки в соответствии с требованиями п. 0;

5) формирование файла ошибок;

6) формирование протокола обработки информации;

7) формирование протокола экспертизы и печать реестра ошибочных записей;

8) просмотр для выбранных счетов данных о пролеченных пациентах в виде таблицы, в которой должна отображаться следующая информация:

- регион;
- фамилия, имя, отчество;
- пол;
- дата рождения;
- ошибка;
- серия и номер полиса пациента;
- наименование СМО, выдавшей полис пациенту;
- серия и номер документа;
- место работы;
- адрес;
- особый случай;
- порядковый номер в реестре;
- тип иногороднего;
- номер и дата счета в другой ТФОМС.

В диалоговом окне с информацией о пролеченных пациентах должно быть реализовано:

- сортировка, отбор, удаление, добавление, редактирование информации;
- подсчет общего числа записей (число пациентов);

– выделение ошибочных записей;

– ручное проставление ошибок в соответствии с требованиями, описанными в пункте 0;

– просмотр услуг по выбранному пациенту;

– для выбранной записи отображение следующей информации по пролеченному пациенту:

- Фамилия, имя, отчество пациента.
- Пол пациента.
- Дата рождения пациента.
- Территория проживания, адрес пациента.
- Наименование ЛПУ.
- Особый случай.
- Наименование ошибки.

9) просмотр для выбранных счетов данных об оказанных услугах в виде таблицы, в которой должна отображаться следующая информация:

– ФИО пациента;

– серия и номер полиса;

– пол;

– дата рождения;

– ЛПУ;

– отделение;

– код услуги (код МЭС);

– основной диагноз;

– сопутствующий диагноз;

– дата начала лечения;

– дата оказания услуги;

– код услуги;

– количество;

– сумма;

– ошибка;

– код исхода лечения;

– вид медицинской помощи;

– номер амбулаторной карты;

– табельный номер врача;

– табельный номер медсестры;

– номер записи в базе ЛПУ;

– признак «особый случай».

В диалоговом окне с информацией об оказанных услугах должно быть реализовано:

– сортировка, отбор, удаление, добавление, редактирование информации;

– подсчет общего числа записей (число услуг);

– подсчет общей стоимости всех оказанных услуг;

– выделение ошибочных записей;

– ручное проставление ошибок в соответствии с требованиями, описанными в пункте 0;

– просмотр данных о пациенте по выбранной услуге;

– для выбранной записи отображение следующей информации по оказанной услуге:

- ФИО пациента.
- Пол.
- Дата рождения.
- Наименование ЛПУ.
- Наименование отделения, в котором выполнена услуга.
- Вид медицинской помощи.
- Наименование медицинской услуги (койко-дня, МЭСа).
- Количество услуг.
- Дата начала лечения.
- Дата оказания услуги.

- Стоимость лечения.
- Основной диагноз.
- Исход лечения.
- Признак «особый случай».
- Наименование ошибки.

Требования к работе с информацией БД об исходящих счетах в ТФОМС.

В форме просмотра данных об исходящих счетах должна отображаться следующая информация (с возможностью предварительного задания ТФОМС и временного интервала):

– наименование ТФОМС;

– № счет-фактуры;

– дата;

– сумма;

– отклонено (сумма);

– оплачено (сумма);

– дата;

– № документа;

– по взаимозачету (сумма);

– № документа;

– наличие исправленной части;

– признак пересылки счета;

– наличие претензий;

– отправка исправленной части.

Должны быть предусмотрены следующие возможности:

1. Сортировка, удаление, отбор записей.

2. Печать списка исходящих счетов.

3. Подсчет количества отобранных счетов.

4. Подсчет общей суммы всех отобранных счетов.

5. Печать реестра пролеченных пациентов.

6. Печать счет-фактуры по пролеченным пациентам.

7. Формирование файлов реестров счетов и их отправка по электронной почте.

8. Прием файла «Протокол обработки реестра».

9. Формирование исправленной части и их пересылка в ТФОМС.

10. Печать «мотивированного отказа».

11. Просмотр данных выбранных счетов в виде таблицы, содержащей следующую информацию:

– код территории;
– фамилия и инициалы пациента;
– пол пациента;
– дата рождения;
– сумма к оплате;
– снята сумма;
– код причины отказа;
– серия и номер полиса;
– наименование страховщика;
– код страховщика;
– серия и номер паспорта;
– адрес;
– дата начала лечения;
– дата окончания лечения;
– вид медицинской помощи;
– код основного заболевания;
– код ЛПУ;
– признак «особый случай»;
– порядковый номер в реестре;
– номер счета;
– дата счета;

– вид информации.

В диалоговом окне с информацией об оказанных услугах должно быть реализовано:

– подсчет общего числа пациентов и суммы, представленной к оплате по все пациентам;

– отбор, редактирование, сортировка, удаление и добавление данных; выделение ошибочных записей;

– при выборе записи в нижней части окна должна высвечиваться следующая информация по пролеченному пациенту:

- Фамилия, имя, отчество (полностью).
- Пол пациента.
- Территория проживания, адрес.
- Наименование ЛПУ.
- Особый случай.
- Код отказа.

Требования к работе с информацией БД о входящих счетах в БТФОМС.

В форме просмотра данных о входящих счетах должна отображаться следующая информация (с возможностью предварительного задания ТФОМС и временного интервала):

- наименование региона;
- № счета;
- дата;
- сумма;
- отклонено (сумма);
- оплачено (сумма);
- дата;
- № документа;
- по взаимозачету (сумма);
- № документа;
- проверка на ошибки;
- наличие исправленной части;
- признак пересылки протокола обработки.

Должны быть предусмотрены следующие возможности:

1. Подсчет количества и общей суммы всех отобранных счетов.

2. Сортировка, добавление, редактирование, удаление записи.

3. Печать списка отобранных счетов.

4. Просмотр данных выбранных счетов. Форма просмотра реестров по выбранным счетам аналогична форме просмотра реестров по исходящим счетам с добавлением колонки «код ошибки».

5. Проверка на ошибки в соответствии с требованиями, изложенными в п. 0.

6. Печать мотивированного отказа.

7. Формирование протокола обработки реестра счета.

Требования к справочникам.

В ПК должно быть предусмотрено ведение справочников, включающее в себя редактирование справочников, т. е. создание, изменение и удаление его элементов.

В системе должен быть предусмотрел экспорт-импорт справочников из формата DBF.

Требования к проверке на ошибки входящих реестров из ЛПУ и ТФОМС.

В программе должна быть реализована как автоматическая проверка на ошибки, так и ручное проставление ошибок. Должен быть предусмотрен выбор ошибок, на которые будет осуществляться автоматическая проверка, а также возможность добавления новых ошибок в перечень.

При автоматической проверке на ошибки должны быть соблюдены следующие правила:

1. Возможность предварительного задания перечня ошибок, на которые будет осуществляться проверка.

2. Автоматическое проставление ошибок должно быть по одному счету, группе выбранных реестров счетов, отобранных по фильтру.

3. Должна быть реализована возможность предварительного выбора проводимой экспертизы:

– техническая проверка записей (проверка правильности заполнения полей и наличия обязательной информации) (в справочнике ошибок обозначены ТЕ);

– экономическая экспертиза (соответствие стоимости лечения установленным тарифам) (в справочнике ошибок обозначены ЕК);

– медицинская экспертиза (соответствие диагноза оказанным услугам, длительности лечения, полу пациента и т. д.) (в справочнике ошибок обозначены МЕ);

– сверка с регистром застрахованных (в справочнике ошибок обозначены RE) Файл сводного регистра застрахованных, выгруженный из программы СРЗ имеет формат DBF. В ВЗР должна быть реализована как автоматическая сверка с данными СРЗ (по ФИО, дате рождения, серии, номеру полиса), так и ручной отбор данных при задании одного или нескольких параметров.

4. Если ошибочна запись в реестре пациентов, то снимается вся сумма, представленная к оплате по данному пациенту, кодом ошибки помечаются услуги, оказанные данному пациенту (в файле «Личных счетов») и заполняется поле «снятая сумма» всех услуг, оказанных пациенту. Если ошибочна запись в файле «Личных счетов», то снимается стоимость лишь данной ошибочной услуги, при этом другие услуги, оказанные пациенту, оплачиваются.

5. Если выбранный счет уже проверялся на ошибки, то должно предлагаться на выбор:

– пропустить;

– проверить заново (при этом записи, обработанные вручную, пропускаются);

– отменить проверку этого и всех последующих выделенных счетов.

6. Для всех безошибочных записей устанавливается признак «принята к оплате», для ошибочных – «отказано в оплате».

Требования к ручному проставлению ошибок:

1. Должна быть предусмотрена возможность как добавления, так и удаления ошибок.

2. При повторной обработке файла записи, обработанные вручную, не проверяются.

3. Возможность простановки и снятия ошибок по отобранной группе записей.

Требования к отчетным формам.

Должна быть предусмотрена возможность редактирования всех формируемых справок и отчетов в ПК.

ПК должен обеспечить формирование следующих выходных форм:

1. Сводная справка по пролеченным иногородним пациентам в разрезе профильных отделений. Предварительные условия для получения отчета:

– ЛПУ (одно, несколько, все);

– период;

– вид информации (по всем записям или только по безошибочным).

2. Форма ПГ за выбранный период.

3. Сводная справка по пролеченным иногородним пациентам в разрезе регионов. Предварительные условия для получения отчета:

– ЛПУ (одно, несколько, все);

– период;

– вид информации (по всем записям или только по безошибочным).

4. Расчеты с ТФОМС. Предварительные условия для получения отчета:

– ТФОМС (одно, несколько, все);

– период.

5. Расчеты с ЛПУ. Предварительные условия для получения отчета:

– ЛПУ (одно, несколько, все);

– период.

Состав выходной информации может дополняться и корректироваться в ходе выполнения проекта.

Требования к системной и информационной безопасности.

Системная и информационная безопасность предполагает функционирование системы защиты информации, которая предполагает следующее:

1. Доступ к ПК в целом должен быть ограничен только тем должностным лицам Заказчика, которые имеют отношение к работе с ВЗР.

2. Любое должностное лицо, имеющее доступ к Системе, должно иметь возможность выполнять те, и только те действия, по отношению к информации, хранящейся в БД, которые соответствуют:

– должностным обязанностям конкретного должностного лица;

– отношению конкретного должностного лица к конкретной информации.

Процедуры восстановления физической целостности данных при возникновении аварий, сбоев в работе оборудования и разрушении данных в результате несанкционированного доступа должны быть основаны на системных средствах сервера баз данных.

В процессе функционирования системы управления должно осуществляться ведение файлов, хранящих копии изменений БД за определенный промежуток времени, позволяющие восстановить актуальное состояние БД. Резервное копирование и восстановление данных осуществляется системными средствами сервера баз данных.

Частота снятия копий должна определяться частотой обновления данных в БД.

Должны быть предусмотрены процедуры реакции системы на несанкционированный доступ.

Требования к видам обеспечения.

Требования к программному обеспечению

клиентского рабочего места

На рабочих станциях пользователей должно быть установлено следующее системное программное обеспечение:

– операционная система Win2k, WinNT 4.0, WinXP;

– программное обеспечение, обеспечивающее поддержку сетевого протокола TCP/IP для связи с сервером;

– Microsoft Office 2000 или выше.

Требования к техническому обеспечению

клиентского рабочего места

Технические требования к средствам вычислительной техники определяются выбранной операционной системой.

Требования к СУБД

Программа ВЗР должна обеспечивать работу в качестве СУБД с сервером БД MS SQL Server 2000 и Microsoft Data Engine (MSDE).

Состав и содержание работ по созданию системы.

Проведение работ по созданию ПК ВЗР должно осуществляться в соответствии с ГОСТ 34.601-90.

К началу эксплуатации системы должны быть осуществлены подготовительные работы по унификации и регистрации входных и выходных документов, классификаторов и обеспечено их ведение, укомплектован и обучен состав пользователей. Все задачи выполняются работниками Заказчика в реальном масштабе времени в пределах компетенции, согласно должностным инструкциям.

Стадии и этапы разработки.

Создание ПК «Взаиморасчеты» должно производиться в соответствии со следующими этапами работ, приведенных в таблице 2.

Таблица 2 – Стадии и этапы разработки

№ п/п	Содержание работ	Начало работ	Окончание работ	Форма завершения
1	Разработка технического задания и постановки задачи на ПК ВЗР. Принятие решений по информационному обеспечению. Разработка решения по составу выходной информации по программному и организационному обеспечению			Документы «Техническое задание на создание ПК ВЗР» и «Постановка задачи для ПК ВЗР»
2	Разработка программного обеспечения и рабочей документации			ПО, пользовательская документация
3	Предварительные испытания и сдача в опытную эксплуатацию			Акт приемки в опытную эксплуатацию
4	Опытная эксплуатация			Журнал проведения опытной эксплуатации, акт завершения опытной эксплуатации

5	Сдача системы в промышленную эксплуатацию			ПО, эксплуатационная документация, программа и методика приемочных испытаний, акт и протокол проведения приемочных испытаний
6	Сертификация программных средств в отраслевом сертификационном центре			Сертификат установленного образца на ПК ВЗР
7	Авторский надзор за функционированием сданного в промышленную эксплуатацию программного обеспечения системы управления			Договор на авторское сопровождение

Порядок контроля и приемки.

Виды, состав, объем и методы испытаний

Общие положения

Организация контроля создания «Подсистемы» и ее приемка проводится в виде следующих основных испытаний:

1) предварительные испытания;

2) опытная эксплуатация;

3) приемочные испытания.

Испытания ПК проводятся в соответствии с требованиями ГОСТ 34.603.92.

Предварительные испытания

Предварительные испытания проводятся на основании распоряжения о создании комиссии для проведения предварительных испытаний.

Предварительные испытания проводятся на контрольных данных на стенде имитационного моделирования в подразделении Подрядчика в присутствии комиссии в соответствии с документом «Программа и методика предварительных испытаний». Программа стендовых испытаний должна быть согласована с Заказчиком не позднее, чем за 5 дней до их начала.

Проверяется работоспособность программного обеспечения и соответствие выполняемых функций системы требованиям документа «Техническое задание». Заканчиваются предварительные испытания оформлением акта приемки в опытную эксплуатацию.

Опытная эксплуатация

Опытная эксплуатация проводится на основании распоряжения о проведении опытной эксплуатации, которое определяет объект или несколько объектов, на которых будет проведена опытная эксплуатация, и сроки проведения опытной эксплуатации.

Опытная эксплуатация проводится с целью проверки технологичности реализованного процесса, проверки функционирования системы в целом и отдельных рабочих мест, способности к взаимодействию с другими системами, правильности выполнения поставленных задач и реализованных функций, подготовки персонала к работе в новой системе.

Опытная эксплуатация должна проводиться на полном объеме реальных данных. В процессе опытной эксплуатации должен вестись журнал, в котором будут фиксироваться результаты выполненных работ, замечания по работе программного обеспечения и предложения по изменению работы ПО.

В процессе опытной эксплуатации разработчики проводят доработку ПО по полученным замечаниям и предложениям, утверждают рабочую документацию.

Результаты опытной эксплуатации отражаются в Акте о завершении опытной эксплуатации.

Приемочные испытания

Решение о вводе системы в постоянную эксплуатацию принимает комиссия, назначаемая распоряжением о проведении приемочных испытаний. Приемочные испытания должны проводиться в соответствии с «Программой и методикой приемочных испытаний». Программа приемочных испытаний должна быть представлена Заказчику не позднее, чем за 5 дней до их начала и согласована с Заказчиком до их начала.

Комиссии предъявляется доработанное по результатам опытной эксплуатации ПО (программы и документация).

Комиссия проверяет полноту выполненных доработок, комплектность и качество документации.

На основании анализа выполненных работ и журнала проведения опытной эксплуатации, комиссия принимает решение о принятии системы в постоянную эксплуатацию.

Общие требования к приемке работ

Приемку работ по всем видам испытаний осуществляет приемочная комиссия, назначенная приказом исполнительного директора БТФОМС, в состав которой входят представители Заказчика и Подрядчика.

Место, сроки и порядок проведения испытаний определяется приемочной комиссией.

Протоколы и акты приемочной комиссии подписывают все ее члены. В случае возникновения разногласий член комиссии имеет право на выражение особого мнения или замечаний, подписываемых им. В этом случае особое мнение или замечание прилагается к протоколу или акту комиссии.

Акты приемки ПК ВЗР утверждаются исполнительным директором БТФОМС.

Требования к документации

Документация на ПК разрабатывается в соответствии с Комплексом стандартов и руководящих документов на автоматизированные системы РД 50-34. 698-90 и определяется ГОСТами, перечень которых будет приведен в пункте ниже.

Техническая документация, содержащая сведения, необходимые для установки и эксплуатации ПК ВЗР, включает в себя следующие виды документов:

- спецификация;
- инструкция по установке ПО;

– руководство пользователя;

– руководство администратора;

– программа и методика испытаний.

Источники разработки

1. Приказ ФОМС № 70 от 23 августа 2000 г. «О порядке финансовых расчетов между территориальными фондами обязательного медицинского страхования за медицинскую помощь в объеме базовой программы обязательного медицинского страхования граждан Российской Федерации, оказанную гражданам Российской Федерации за пределами территории страхования».

2. ГОСТ 23962-80. Организация работ при создании систем.

3. ГОСТ 34.602-89. Техническое задание на создание автоматизированной системы.

4. ГОСТ 34.601-90. Автоматизированные системы. Стадии создания.

5. ГОСТ 34.201-89. Виды, комплектность и обозначение документов при создании автоматизированной системы.

6. ГОСТ 34.603-92. Виды испытаний автоматизированных систем.

7. РД 50-34.698-90. Автоматизированные системы. Требования к содержанию документов.

8. ГОСТ 19.004-80. Единая система программной документации. Термины и определения.

9. ГОСТ 19.101-77. Единая система программной документации. Виды программ и программных документов.

10. ГОСТ 19.102-77. Единая система программной документации. Стадии разработки.

11. ГОСТ 19.103-77. Единая система программной документации. Обозначение программ и программных документов.

12. ГОСТ 19.104-77. Единая система программной документации. Основные надписи.

13. ГОСТ 19.105-77. Единая система программной документации. Общие требования к программным документам.

14. ГОСТ 19.106-77. Единая система программной документации. Общие требования к программным документам, выполненным печатным способом.

15. ГОСТ 19.201-78. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению.

16. ГОСТ 19.202-78. Единая система программной документации. Спецификация. Требования к содержанию и оформлению.

17. ГОСТ 19.301-79. Единая система программной документации. Программа и методика испытаний. Требования к содержанию и оформлению.

2.8 Контрольное задание

Разработать техническое задание для информационной системы согласно выбранному вами варианту.

2.9 Контрольные вопросы

1. Сформулируйте понятие технического задания для информационной системы.

2. Каковы задачи технического задания для информационной системы?

3. Опишите структуру технического задания для информационной системы.

3 Лабораторная работа № 3. Разработка технического проекта

Цель работы: научиться разрабатывать технический проект информационной системы.

Используемое программное обеспечение: Microsoft Word 2016.

Технический проект системы – это техническая документация, содержащая общесистемные проектные решения, алгоритмы решения задач, а также оценку экономической эффективности автоматизированной системы управления и перечень мероприятий по подготовке объекта к внедрению.

Технический проект информационной системы разрабатывается на основе технического задания (и эскизного проекта).

Технический проект включает в себя:

– разработку проектных решений по системе и ее частям;

– разработку документации на информационную систему и ее части;

– разработку и оформление документации на поставку комплектующих изделий;

– разработку заданий на проектирование в смежных частях проекта.

На этом этапе осуществляется комплекс научно-исследовательских и экспериментальных работ для выбора основных проектных решений и расчет экономической эффективности системы.

3.1 Структура технического проекта. Пояснительная записка

1. основания для разработки системы;

2. перечень организаций разработчиков;

3. краткая характеристика объекта с указанием основных технико-экономических показателей его функционирования и связей с другими объектами;

4. краткие сведения об основных проектных решениях по функциональной и обеспечивающим частям системы.

3.2 Функциональная и организационная структура системы

1. обоснование выделяемых подсистем, их перечень и назначение;

2. перечень задач, решаемых в каждой подсистеме, с краткой характеристикой их содержания;

3. схема информационных связей между подсистемами и между задачами в рамках каждой подсистемы.

3.3 Постановка задач и алгоритмы решения

1. организационно-экономическая сущность задачи (наименование, цель решения, краткое содержание, метод, периодичность и время решения задачи, способы сбора и передачи данных, связь задачи с другими задачами, характер использования результатов решения, в которых они используются);

2. экономико-математическая модель задачи (структурная и развернутая форма представления);

3. входная оперативная информация (характеристика показателей, диапазон изменения, формы представления);

4. нормативно-справочная информация (НСИ) (содержание и формы представления);

5. информация, хранимая для связи с другими задачами;

6. информация, накапливаемая для последующих решений данной задачи;

7. информация по внесению изменений (система внесения изменений и перечень информации, подвергающейся изменениям);

8. алгоритм решения задачи (последовательность этапов расчета, схема, расчетные формулы);

9. контрольный пример (набор заполненных данными форм входных документов, условные документы с накапливаемой и хранимой информацией, формы выходных документов, заполненные по результатам решения экономико-технической задачи и в соответствии с разработанным алгоритмом расчета).

3.4 Организация информационной базы

1. источники поступления информации и способы ее передачи;
2. совокупность показателей, используемых в системе;
3. состав документов, сроки и периодичность их поступления;
4. основные проектные решения по организации фонда НСИ;
5. состав НСИ, включая перечень реквизитов, их определение, диапазон изменения и перечень документов НСИ;
6. перечень массивов НСИ, их объем, порядок и частота корректировки информации;
7. структура фонда НСИ с описанием связи между его элементами; требования к технологии создания и ведения фонда;
8. методы хранения, поиска, внесения изменений и контроля;
9. определение объемов и потоков информации НСИ;
10. контрольный пример по внесению изменений в НСИ;
11. предложения по унификации документации.

3.5 Система математического обеспечения

1. обоснование структуры математического обеспечения;
2. обоснование выбора системы программирования;
3. перечень стандартных программ.

3.6 Принцип построения комплекса технических средств

1. описание и обоснование схемы технологического процесса обработки данных;

2. обоснование и выбор структуры комплекса технических средств и его функциональных групп;

3. обоснование требований к разработке нестандартного оборудования;

4. комплекс мероприятий по обеспечению надежности функционирования технических средств.

3.7 Расчет экономической эффективности системы

1. сводная смета затрат, связанных с эксплуатацией систем;

2. расчет годовой экономической эффективности, источниками которой являются оптимизация производственной структуры хозяйства (объединения), снижение себестоимости продукции за счет рационального использования производственных ресурсов и уменьшения потерь, улучшения принимаемых управленческих решений.

3.8 Мероприятия по подготовке объекта к внедрению системы

1. перечень организационных мероприятий по совершенствованию бизнес-процессов;

2. перечень работ по внедрению системы, которые необходимо выполнить на стадии рабочего проектирования, с указанием сроков и ответственных лиц.

3.9 Ведомость документов

1. Ведомость документов;

2. В завершение стадии технического проектирования производится разработка документации на поставку серийно выпускаемых изделий для комплектования информационной системы, а также определяются технические требования и составляются технические задания на разработку изделий, не изготовляемых серийно.

3.10 Пример технического проекта. Пояснительная записка

В связи с расширением отделов, а, следовательно, и ростом количества сотрудников, а также с обновлением ассортимента возникла необходимость в решении следующего ряда проблем: поставка нового ассортимента, реализация накопительной системы скидок и др.

Очевидно, что эффективное решение данных проблем возможно лишь на основе применения соответствующей автоматизированной информационной системы на платформе среды разработки баз данных Microsoft SQL Server 2017 и среды программирования Microsoft Visual Studio 2017 Ultimate.

3.11 Функциональная и организационная структура системы

База данных, созданная на платформе Microsoft SQL Server 2017, должна содержать 6 таблиц: поставщик, магазин, партия, транспорт, купленный товар и клиенты, на основе информации из которых возможно будет формировать заказы и контролировать их выполнение.

3.12 Постановка задач и алгоритмы решения

Задачи поставки нового ассортимента:

– осуществить выборку ассортимента по типу, наименованию, марки автомобиля;

– необходимо организовать поиск необходимого ассортимента в каталоге на наличие или отсутствие товара.

– осуществить возможность вывода на печать отчета о поставках. Реализация накопительных систем скидок:

– делая заявку на покупку товара, клиент автоматически вносится в базу данных клиентов;

– в базе данных должны храниться сведения:

- фамилия, имя, отчество клиента.
- сумма покупки.
- дата покупки.
- возможность распечатать заказ и послать поставщику.
- автоматический перенос ассортимента в новую базу данных.

Задачи по возврату бракованного ассортимента поставщику:

– ассортимент хранится в магазине, пока техник не решит, бракованный товар или исправный;

– возможность распечатать акт, и послать в сервис центр;

– автоматическое удаление списанного ассортимента из общей базы данных.

3.13 Организация информационной базы

Как уже было описано выше, база данных содержит 6 таблиц: «Postavshik», «Magazin(Sklad)», «Partiya», «Transport», «Tovar», «Client».

Таблица «Postavshik» содержит сведения о поставщике:

– *<PID> – ключевое поле таблицы – код поставщика типа <int>;

– <PType> – категория поставщика типа <char>;

– <PCost> – затраты типа <money>;

– <Pnalichie> – наличие типа <boolean>;

– <Pnaimen> – наименование типа <char>.

Таблица «Magazin(Sklad)» содержит сведения о товаре, находящемся на складе магазина:

– *<MID> – ключевое поле таблицы – код магазина типа <int>;

– <MType> – категория магазина типа <char>;

– <MRoznCost> – розничная цена типа <money>;

– <Mnaimen> – наименование типа <char>.

Таблица «Client» содержит сведения о купленном клиентом товаре, а также о самом клиенте:

– *<CID> – ключевое поле таблицы – код клиента типа <int>;

– <CFIO> – фамилия, имя, отчество клиента типа <char>;

– <CSumm> – сумма заказа типа <money>;

– <Cdate> – дата заказа типа <datatime>;

– <Ckol> – количество типа <int>.

Таблица «Partiya» содержит сведения о заказанном на данный момент товаре:

– *<PaID> – ключевое поле таблицы – код партии типа <int>;

– <PaKol> – количество типа <int>;

– <PaType> – тип партии типа <char>;

– <PaName> – наименование типа <char>.

Таблица «Transport» содержит информацию о транспорте, способе транспортировки товара, а также времени транспортировки:

– *<TID> – ключевое поле таблицы – код транспорта типа <int>;

– <TVid> – вид транспорта типа <char>;

– <TTime> – время транспортировки типа <datatime>;

– <Tcost> – затраты на транспортировку типа <money>.

Таблица «Tovar» содержит информацию о купленном товаре:

– *<TovID> – ключевое поле таблицы – код товара типа <int>;

– <TovType> – тип товара типа <char>;

– <TovCost> – стоимость товара типа <money>. Объекты базы данных связаны следующим образом:

– магазин заказывает партии товара у поставщиков.

– поставщики транспортируют товары.

– клиенты покупают товары.

3.14 Контрольное задание

Разработать технический проект для информационной системы согласно выбранному варианту на основе технического задания.

3.15 Контрольные вопросы

1. Сформулируйте определение технического проекта информационной системы.

2. Опишите структуру технического проекта информационной системы.

3. Какая взаимосвязь существует между техническим заданием и техническим проектом информационной системы?

4 Лабораторная работа № 4. Реализация баз данных и таблиц в MS SQL Server. Обеспечение целостности данных

Цель работы: научиться создавать таблицы и связи между ними с помощью компонентов Microsoft *SQL Server 2017: Query Analizer* и *Enterprise Manager*.

Используемое программное обеспечение: Microsoft SQL Server 2017.

Базы данных SQL Server бывают системные и пользовательские. Принципиальной разницы между ними нет, но системные базы данных необходимы для работы SQL Server: они хранят информацию о сервере SQL Server в целом, применяемую для управления системой. Пользовательские базы данных создают пользователи. Один сервер SQL Server может управлять несколькими такими базами данных. При установке SQL Server создается 4 системных и 2 учебных пользовательских базы данных.

4.1 Базы данных SQL Server. Системные базы данных

В нижеприведенной таблице описаны системные базы данных.

Таблица 1 – Описание системных баз данных

База данных	Описание
master model	Служит для управления пользовательскими базами данных и операциями SQL Server; хранит информацию об учетных записях, параметрах конфигурации, размещении баз данных и системных сообщениях об ошибках
tempdb	Шаблон для создания новых пользовательских баз данных. Хранилище для временных таблиц и других временных объектов
msdb	Обеспечивает поддержку службы SQL Server Agent и предоставляет хранилище для расписания и журнала заданий
distribution	Хранит протоколы и транзакции, используемые при репликации

Данные, хранящиеся в системных базах данных, можно изменять и удалять, но это не рекомендуется. Все пользовательские объекты Вы должны создавать только в пользовательских базах данных, а для чтения и изменения системных следует применять только системные хранимые процедуры.

4.2 Пользовательские базы данных

Создаваемые при установке SQL Server учебные базы данных <pubs> и <Northwind> предназначены для изучения SQL Server, но для его работы не нужны.

4.3 Объекты базы данных

База данных – это набор данных, хранящихся в таблицах, и объекты, поддерживающие хранение, извлечение, защиту и целостность этих данных. В таблице 2 перечислены основные объекты баз данных MS SQL Server.

Таблица 2 – Основные объекты баз данных MS SQL Server

Объект	Описание
Table (таблица)	Хранит данные в виде набора строк и столбцов
Data type (тип данных)	Определяет допустимый тип значений столбца или переменной. SQL Server располагает встроенными типами данных; кроме того, пользователи могут создавать свои типы данных
Constraint (ограничение)	Используется для определения правил, гарантирующих целостность данных столбца или набора столбцов таблицы; ограничения – стандартный механизм обеспечения целостности и непротиворечивости данных
Default (значение по умолчанию)	Значение, хранящееся в столбце при отсутствии фактического значения
Rule (правило)	Определяет выражение, используемое для проверки допустимости значений столбца или типа данных
Index (индекс)	Структура хранения, обеспечивающая упорядочение и быстрый доступ к данным; может также обеспечивать уникальность данны
View (представление)	Метод просмотра данных одной или нескольких таблиц или других представлений базы данных
Stored procedure (хранимая процедура)	Именованный набор выражений или пакетов Transact-SQL, выполняющихся вместе
Trigger (триггер)	Специальная форма хранимой процедуры, автоматически выполняемая при изменении данных в таблице

4.4 Типы данных MS SQL Server

Типы данных MS SQL Server приведены в таблице 3.

Таблица 3 – Типы данных MS SQL Server

Тип	Описание
Binary	Любые данные в двоичном виде. Используется для хранения файлов
Varbinary	Любые данные в двоичном виде. Используется для хранения файлов
Tinyint	Целое положительное число от 0 до 255
Char Varchar	Символьное выражение; может содержать любые символы (до 254 для одного поля)
Money	Денежное выражение для числовой величины. Выводит число с четырьмя десятичными разрядами и установленным обозначением используемой денежной единицы
Datetime	Дата и время; может содержать время, день, месяц и год
SmallDatetime	Дата и время; точность 1 минута.
Bit	Булево выражение (.T. или .F.)
Float	Числовое выражение; может содержать целые или дробные числа со знаком
Smallint	Целое число в диапазоне от -32 768 до +32 767
Int	Целое число. Можно хранить числа от -2 147 483 647 до 2 147 483 646
Image	Поле для ссылки на объект OLE
Text	Поле примечаний для ссылки на блок данных
Decimal	Точное числовое значение, которое может иметь до 38 цифр (p); число цифр после запятой (s) не может превышать p
Numeric	Точное числовое значение, которое может иметь до 38 цифр (p); число цифр после запятой (s) не может превышать p
Real	Число с плавающей точкой одинарной точности. Можно хранить числа в интервале (по модулю) от 1.401298E-45 до 3.402823E38

4.5 Пример работы с таблицами. Создание таблицы «Поставщик» с помощью <Query Analyzer>

Создать таблицу «Поставщик» в базе данных <Northwind> с помощью <Query Analyzer> (рисунок 1).

1. Запустите <Query Analizer>.

2. Введите текст следующего оператора CREATE TABLE на панели Editor <SQL Query Analyzer>:

use Northwind

```
create table Поставщик
(
Код_поставщика char(10) Primary Key,
ФИО_поставщика char(30) Null,
Тел_поставщика char(12) Null,
)
```

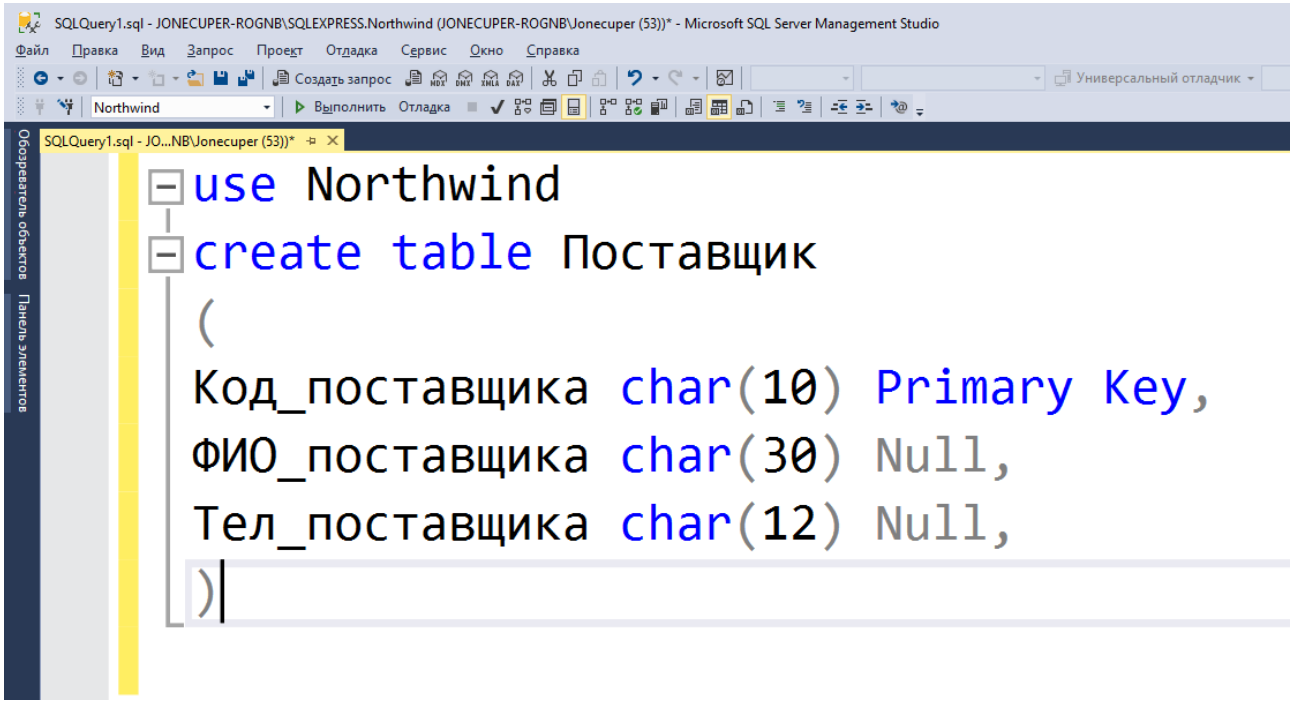


Рисунок 1 – Создание таблицы «Поставщик» с помощью <Query Analyzer>

Обратите внимание, что все ключевые слова: USE, CREATE TABLE, char, Primary Key подсвечены на экране синим цветом. Если для какого-либо из этих слов это не так, проверьте, правильно ли оно введено.

Также обратите внимание, что в раскрывающемся списке <Database> на панели инструментов выводится имя <Master>. После исполнения оператора <USE Northwind> оно изменится на <Northwind>.

3. Щелкните на панели инструментов кнопку <Execute Query> (F5).

Появляется панель <Results> с активной вкладкой <Messages>, на которой сообщается об успешном исполнении команды (рисунок 2).

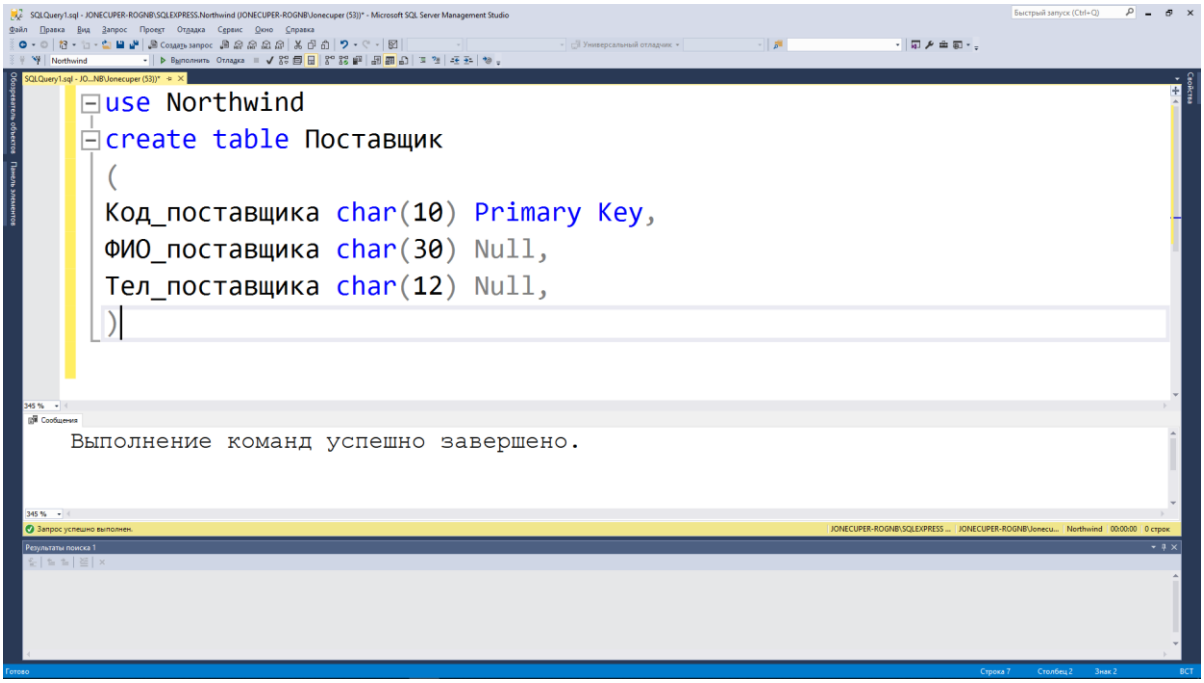


Рисунок 2 – Панель <Results> с активной вкладкой <Messages>

4. В окне <Object Browser> раскройте узел <Northwind>, а затем – узел <User Tables>.

В списке таблиц находится таблица «Поставщик». Префикс <dbo>, стоящий перед именем таблицы, указывает на владельца объекта «таблица» (рисунок 3).

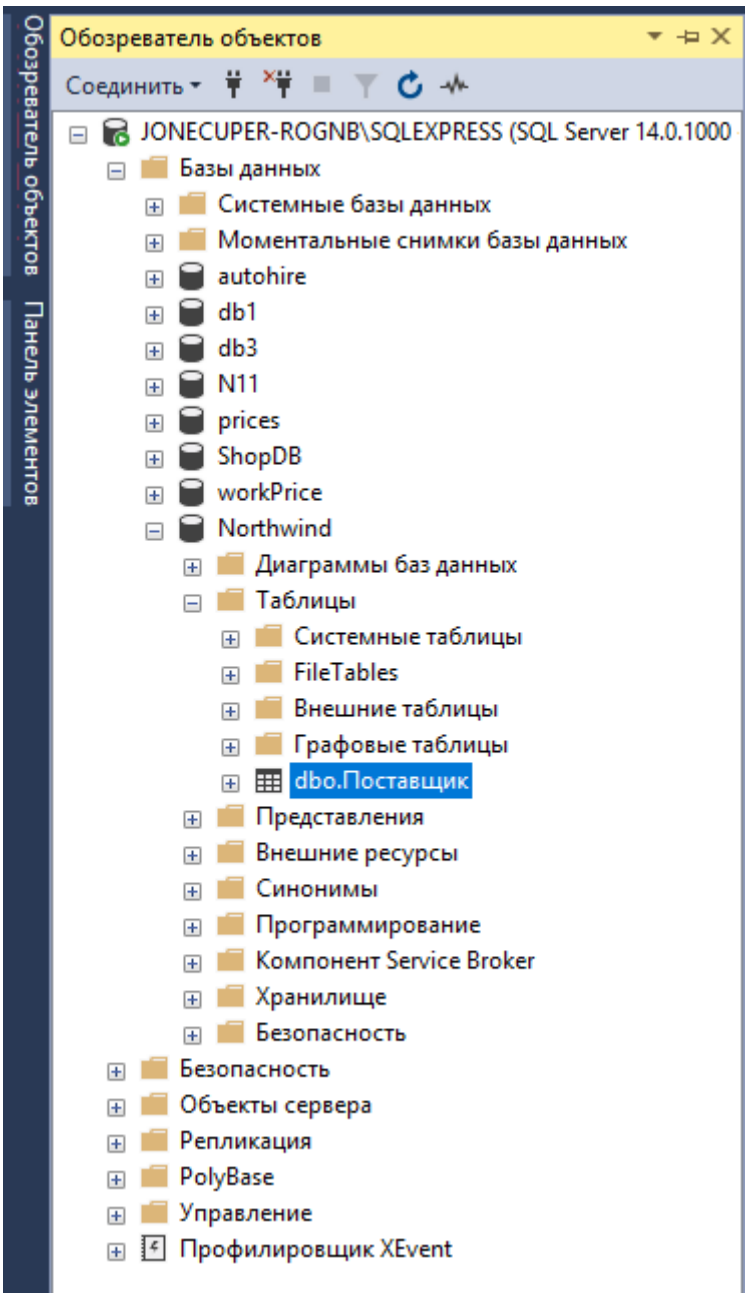


Рисунок 3 – Таблица «Поставщик» в списке таблиц

5. Щелкните правой кнопкой «dbo.Поставщик», а затем щелкните <Изменить первые 200 строк>.

Появляется окно <Open Table>, в котором выводятся имена трех созданных вами атрибутов. Ниже имен атрибутов нет никаких данных, поскольку в таблицу никакие данные вы еще не добавляли (рисунок 4).

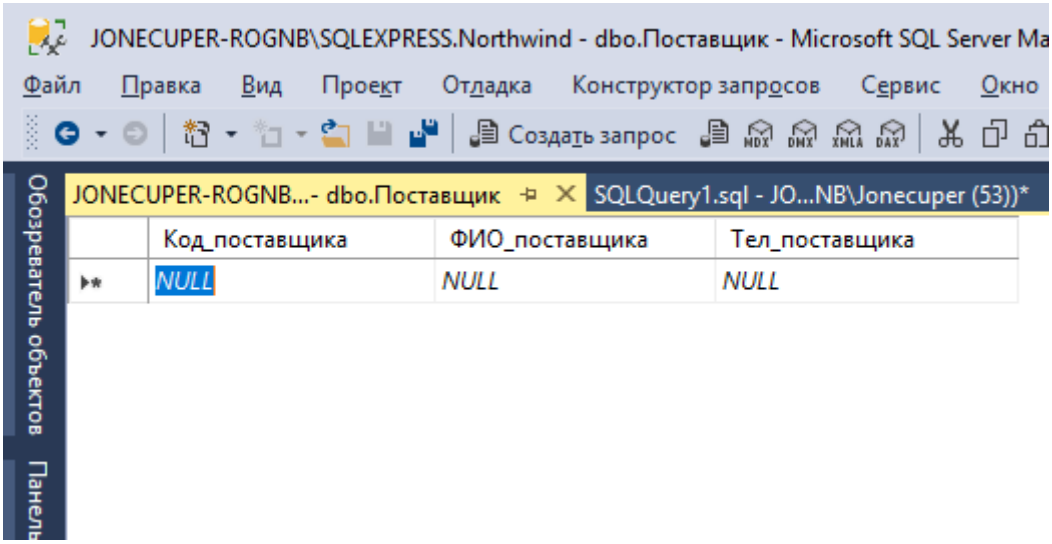


Рисунок 4 – Таблица «Поставщик» в режиме таблицы

6. Закройте окно <Open Table>.

4.6 Создание таблицы «Товар» с помощью <Enterprise Manager>

Создать таблицу «Товар» в базе данных <Northwind> с помощью <Enterprise Manager>:

1. Запустите <Enterprise Manager>.

2. В окне <Console Root> раскройте список баз данных и откройте <Northwind>, затем <Tables>, щелкните по ней правой кнопкой мыши и выберите <New Tables> (рисунок 5).

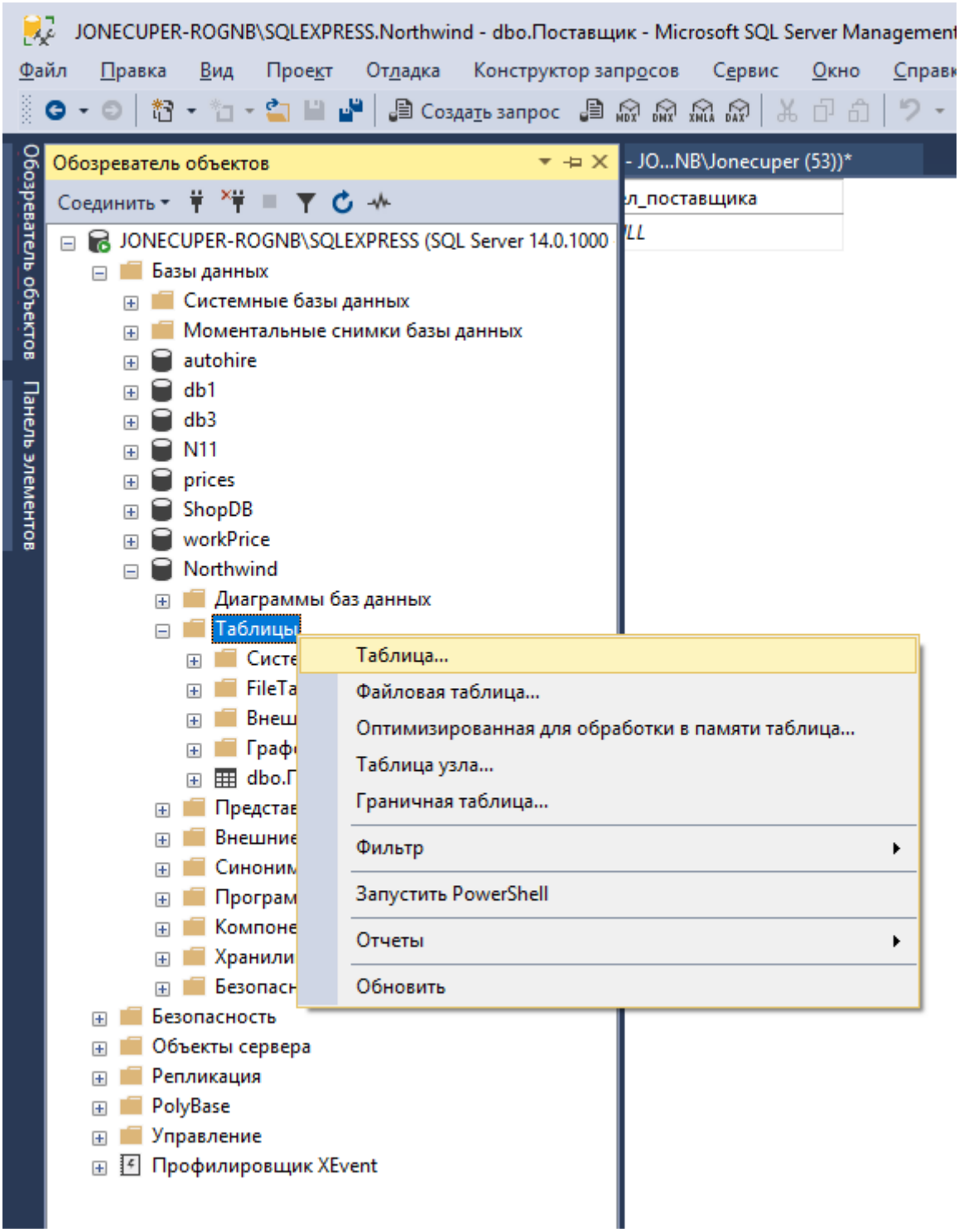


Рисунок 5 – Создание таблицы с помощью <Enterprise Manager>

Имя таблицы – «Товар», структура которой представлена ниже (рисунок 6):

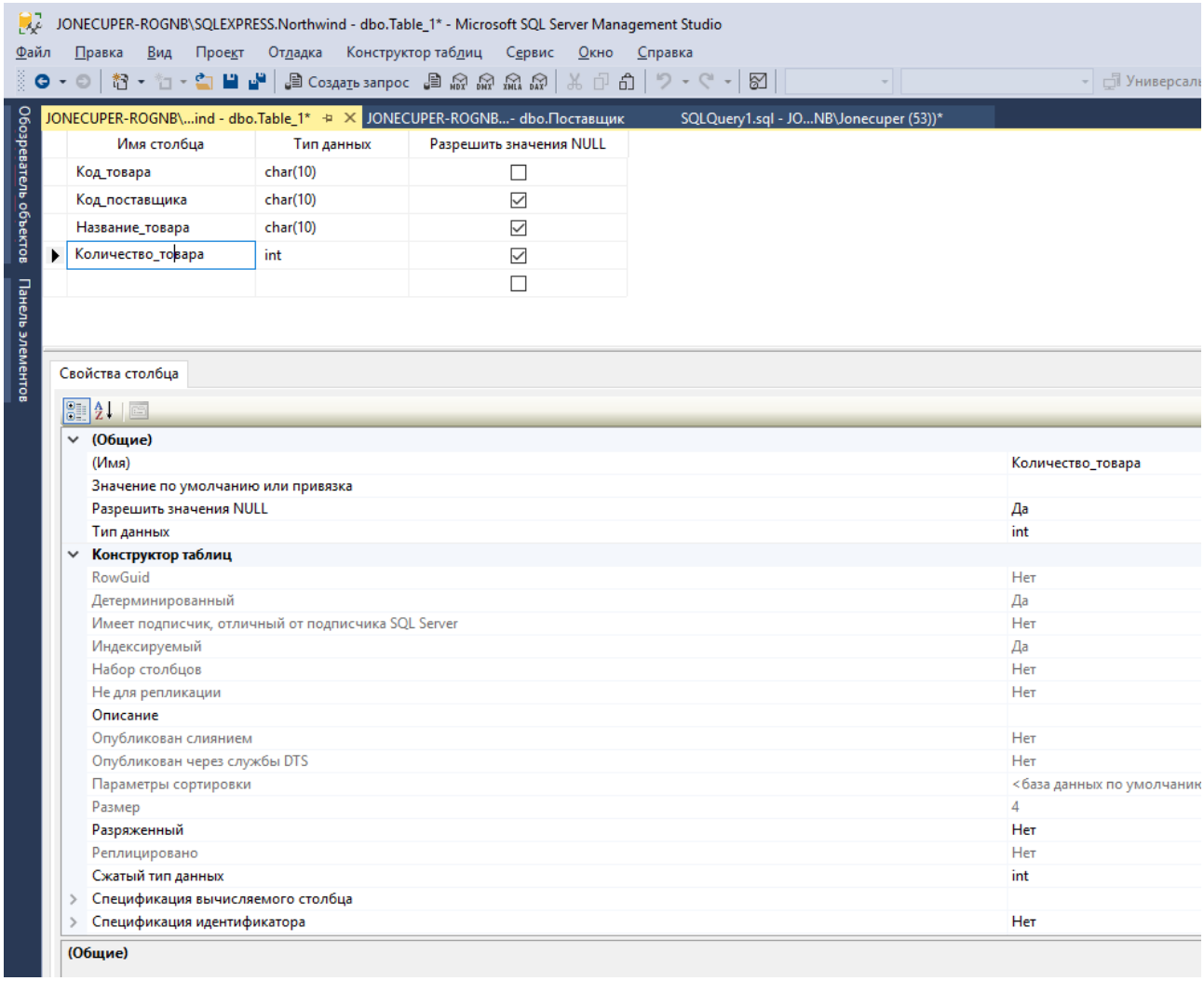


Рисунок 6 – Таблица «Товар» в режиме конструктора

3. Сделайте поле <Код_товара> ключевым. Для этого щелкните по полю правой кнопкой мыши и выберите <Set Primary Key> (рисунок 7).

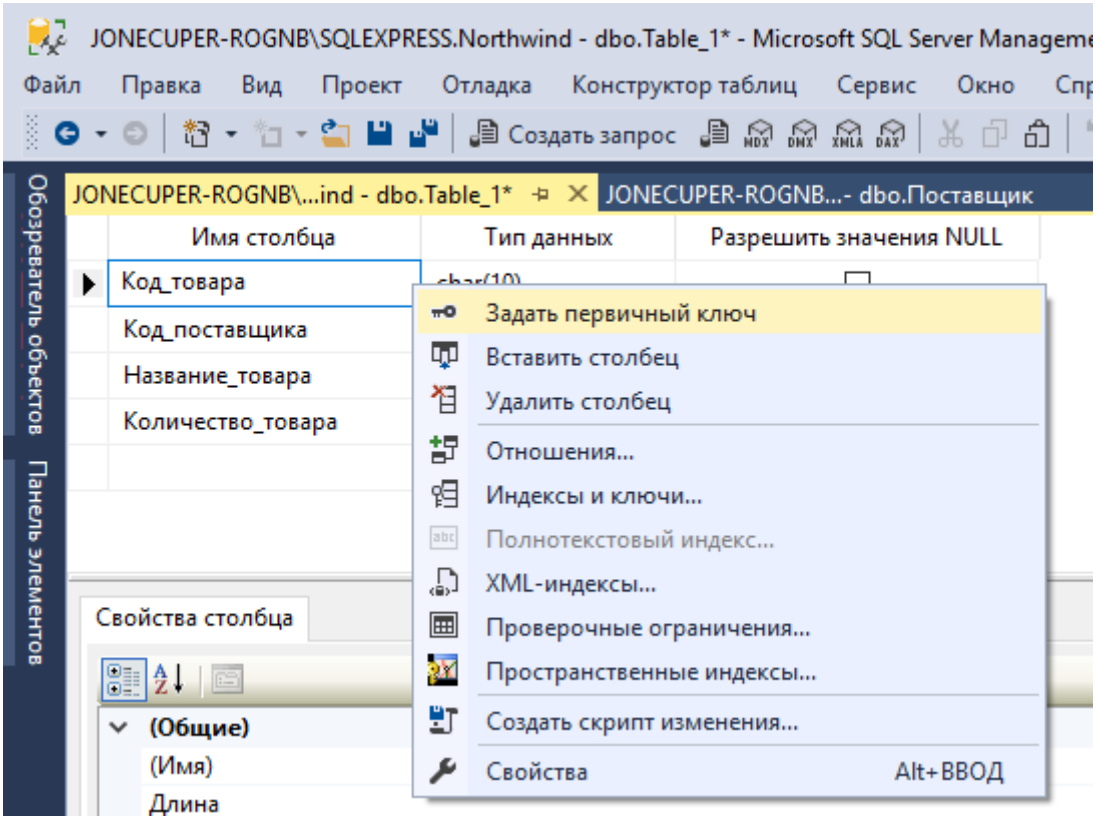


Рисунок 7 – Создание первичного ключа

4.6 Модификация таблицы «Поставщик» с помощью <Query Analyzer>

Модифицируйте таблицу «Поставщик» с помощью <Query Analyzer>:

1. На панели <Editor> поместите курсор немного ниже только что исполненного оператора.

2. Введите следующий оператор <ALTER TABLE>:

alter table Поставщик

add Адрес_поставщика char(40) Null

Здесь не обязательно использовать команду <USE Northwind>, поскольку в данный момент <Northwind> является активной базой данных. Имя активной базы данных отображается в раскрывающемся списке <Database> на панели инструментов.

3. Выделите оператор <ALTER TABLE> и щелкните кнопку <Execute Query>.

На вкладке <Messages> выводится сообщение об успешном завершении команды (рисунок 8).

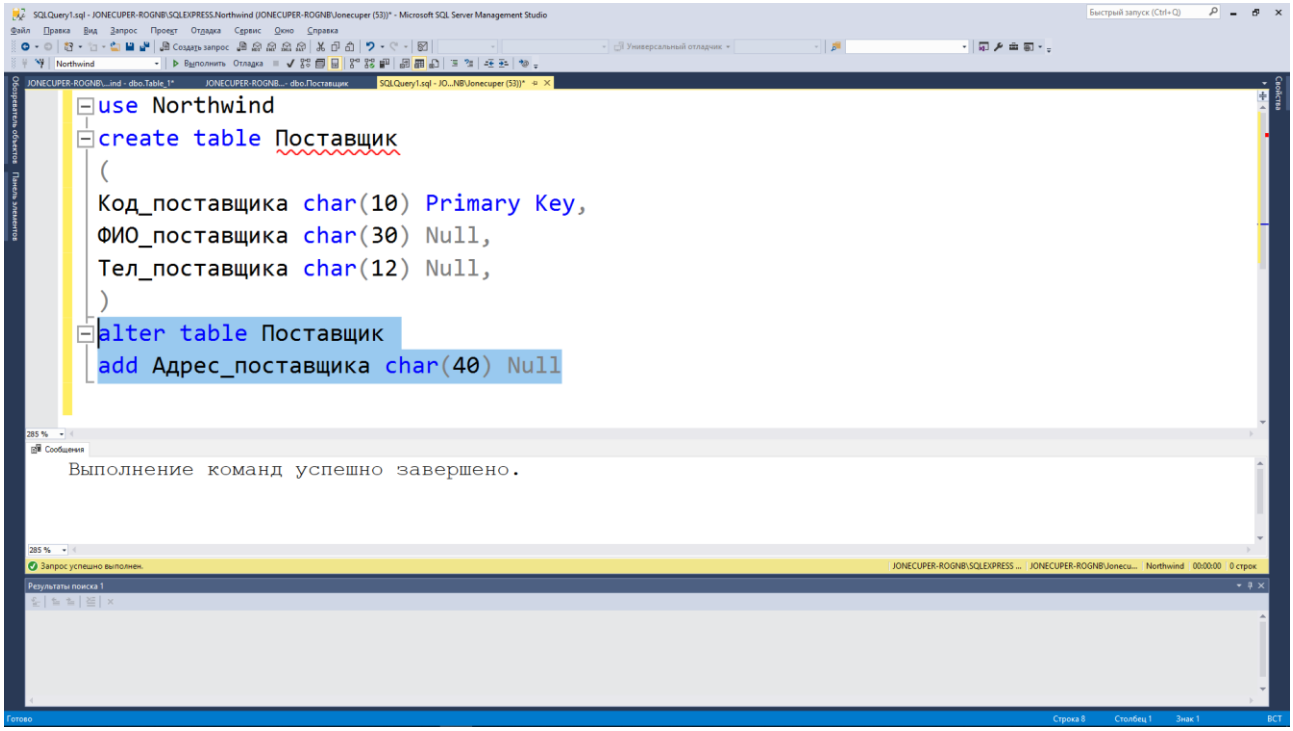


Рисунок 8 – Изменение таблицы «Поставщик»

4. В окне <Object Browser> щелкните правой кнопкой мыши «dbo.Посвтащик», а затем щелкните < Изменить первые 200 строк >.

Появляется окно <Open Table> с атрибутами таблицы «Поставщик». Теперь в таблице есть атрибут <Адрес_поставщика> (рисунок 9).

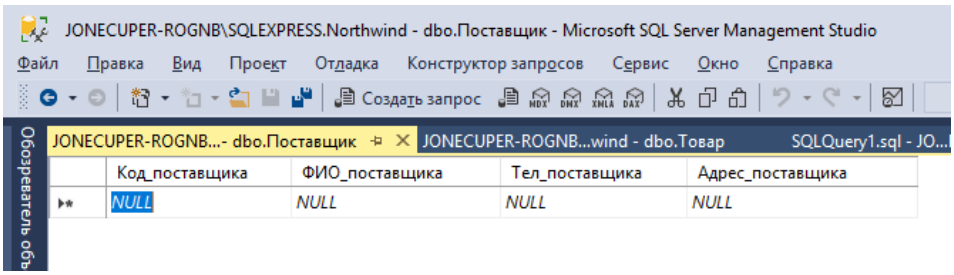


Рисунок 9 – Таблица «Поставщик» с добавленным полем <Адрес_поставщика>

5. Закройте окно <Open Table>.

4.7 Модификация таблицы «Товар» с помощью <Enterprise Manager>

Модифицируйте таблицу «Товар» с помощью <Enterprise Manager>:

В окне <Console Root> раскрыть список баз данных, открыть <Northwind>, выбрать <Tables>, в списке таблиц – «Товар», щелкнуть по ней правой кнопкой и запустить <Design Table>. В открывшемся окне добавить поле <Цена_товара> тип данных <Money> (рисунок 10).

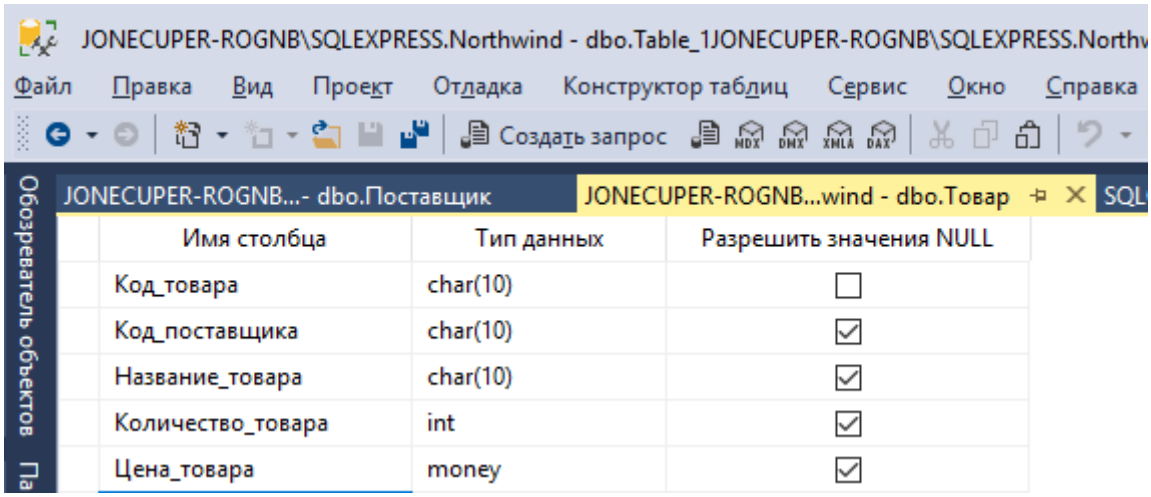


Рисунок 10 – Таблица «Товар» в режиме конструктора

4.8 Вывод сведений о таблице

С помощью <Query Analyzer>:

1. Введите на панели <Editor> следующий код несколькими строками ниже оператора <ALTER TABLE>:

EXEC sp_help Поставщик

<sp_help> — это системная хранимая процедура, которая выводит сведения об объектах базы данных.

2. Выделите и исполните оператор <EXEC>.

На вкладке <Grids> панели <Results> выводится информация о таблице «Поставщик». Обратите внимание, что эта информация напоминает сведения в

окне <Open Table>, которое было открыто с помощью окна <Object Browser> (рисунок 11).

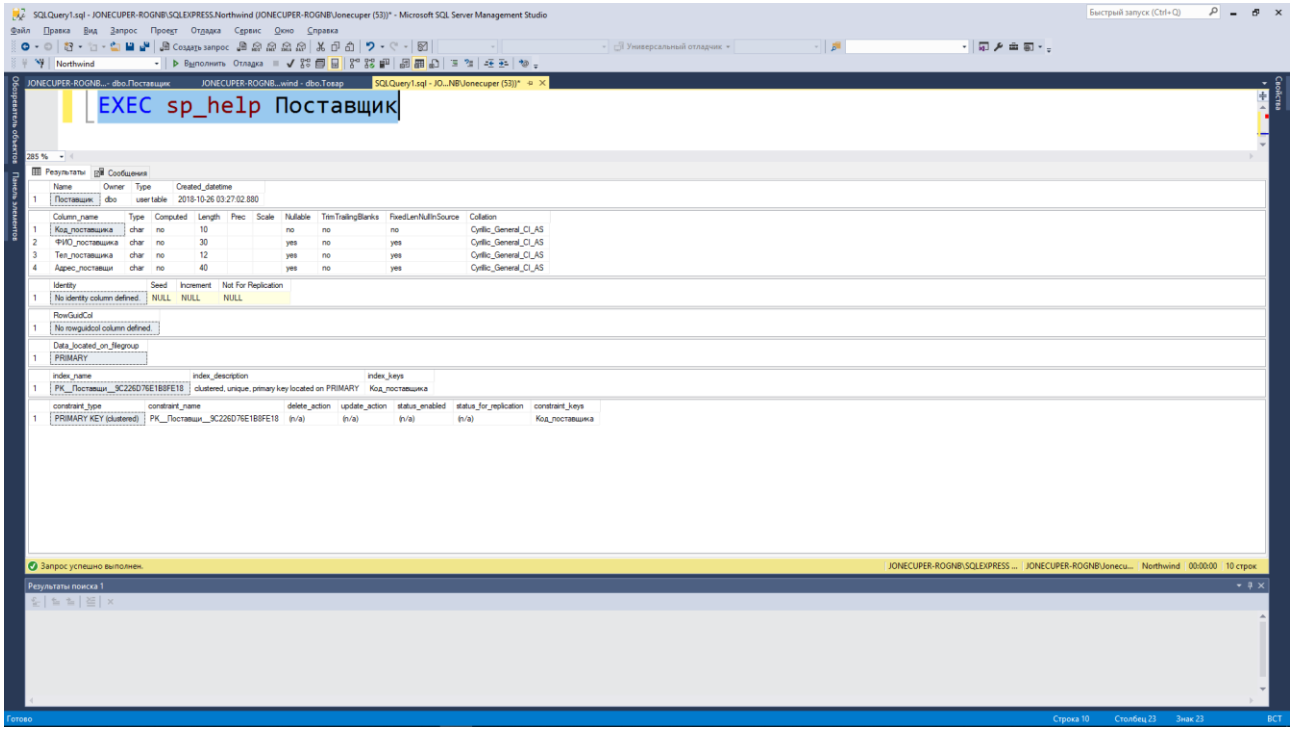


Рисунок 11 – Вывод информации о таблице «Поставщик»

3. Ознакомьтесь с информацией на вкладке <Grids>.

Обратите внимание на тип данных каждого атрибута, на то, может ли атрибут принимать значение <null>, а также на имя владельца таблицы и тип объекта.

4. Закройте окно <Object Browser Window>, но оставьте открытым окно <Query>.

С помощью <Enterprise Manager>:

В окне <Console Root> раскрыть список баз данных, открыть <Northwind>, выбрать <Tables>, в списке таблиц – «Товар», щелкнуть по ней 2 раза, либо щелкнуть правой кнопкой мыши и выбрать <Свойства> (рисунок 12).

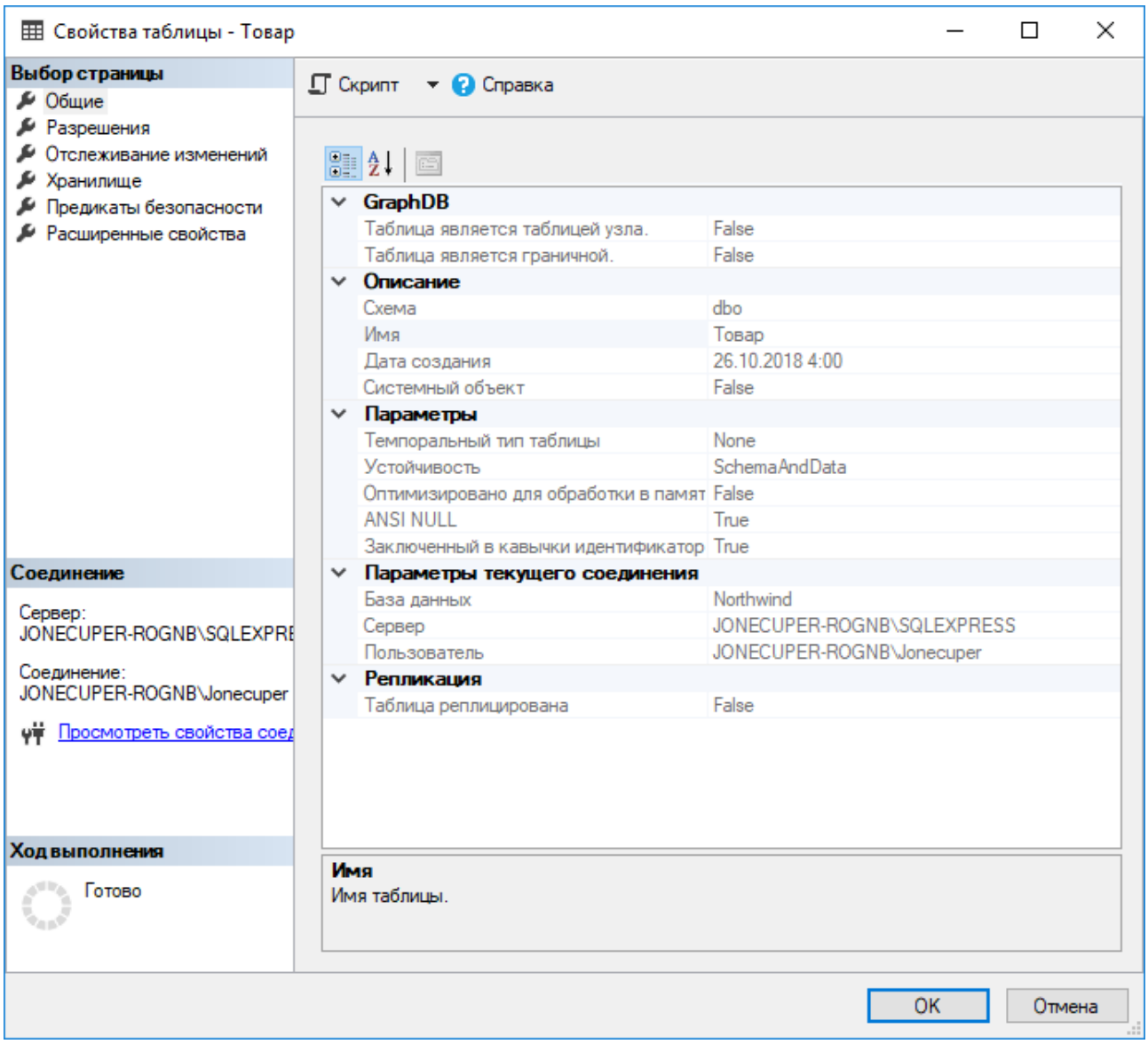


Рисунок 12 – Свойства таблицы «Товар»

4.9 Добавление и извлечение данных

С помощью <Query Analyzer>:

1. На панели <Editor> ниже последнего оператора введите следующие операторы <INSERT>:

INSERT Поставщик VALUES (01, 'Пушкин А.С.', 89142589545, 'Карбышева 7-45')

INSERT Поставщик VALUES (02, 'Иванов С.П.', 89245678952, 'Победы 34-29')

INSERT Поставщик VALUES (03, 'Захаров М.В.', 89095871535, 'Павлова 4-18')

2. Выделите операторы <INSERT> и исполните их.

На вкладке <Messages> панели <Results> отображается набор из трех сообщений. Каждое из них свидетельствует, что исполнение оператора повлияло на одну строку (рисунок 13).

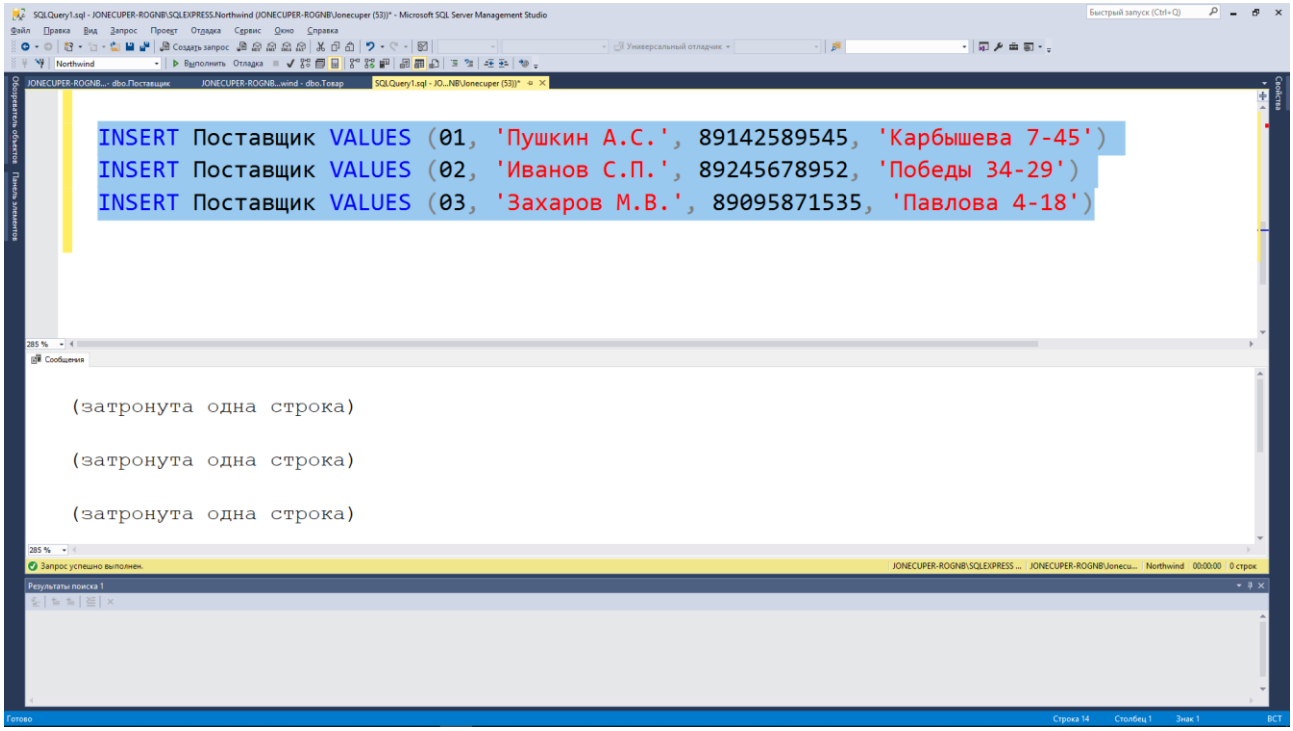
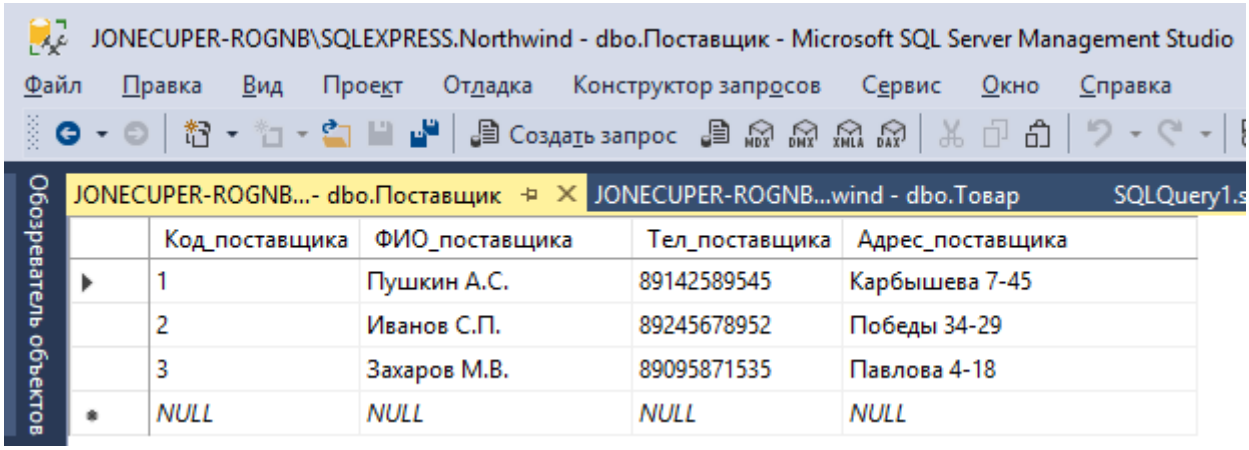


Рисунок 13 – Добавление данных в таблицу «Поставщик»

3. Проверьте внесенные данные, открыв контекстное меню «dbo.Поставщик» и выбрав < Изменить первые 200 строк > (рисунок 14).



Код_поставщика	ФИО_поставщика	Тел_поставщика	Адрес_поставщика
1	Пушкин А.С.	89142589545	Карбышева 7-45
2	Иванов С.П.	89245678952	Победы 34-29
3	Захаров М.В.	89095871535	Павлова 4-18
NULL	*NULL*	*NULL*	*NULL*

Рисунок 14 – Просмотр содержимого таблицы «Поставщик»

С помощью <Enterprise Manager>:

В окне <Console Root> раскрыть список баз данных <Northwind>, выбрать <Tables>, в списке таблиц – «Товар», щелкнуть по ней правой кнопкой мыши и запустить <Open Table>, затем <Return all rows>. В появившемся окне данных заполнить поля данными (не менее 3 записей).

4.10 Создание с помощью <Query Analizer> таблиц «Продажа» и «Клиент»

Создайте с помощью <Query Analizer> таблицы «Продажа» и «Клиент» со следующими атрибутами:

«Продажа» (<Код_продажи> (ключевое), <Код_товара>, <Код_клиента>, <Количество_проданного_товара>, <Дата>);

«Клиент» (<Код_клиента> (ключевое), <ФИО_клиента>, <Тел_клиента>, <Адрес_клиента>).

4.11 Создание связей между таблицами.

1. Откройте окно <Enterprise Manager>.

2. В окне <Console Root> раскройте список баз данных, откройте <Northwind>, выберите <Diagrams> и щѐлкните по ней правой кнопкой мыши, затем выберите <New Database Diagram> (рисунок 15).

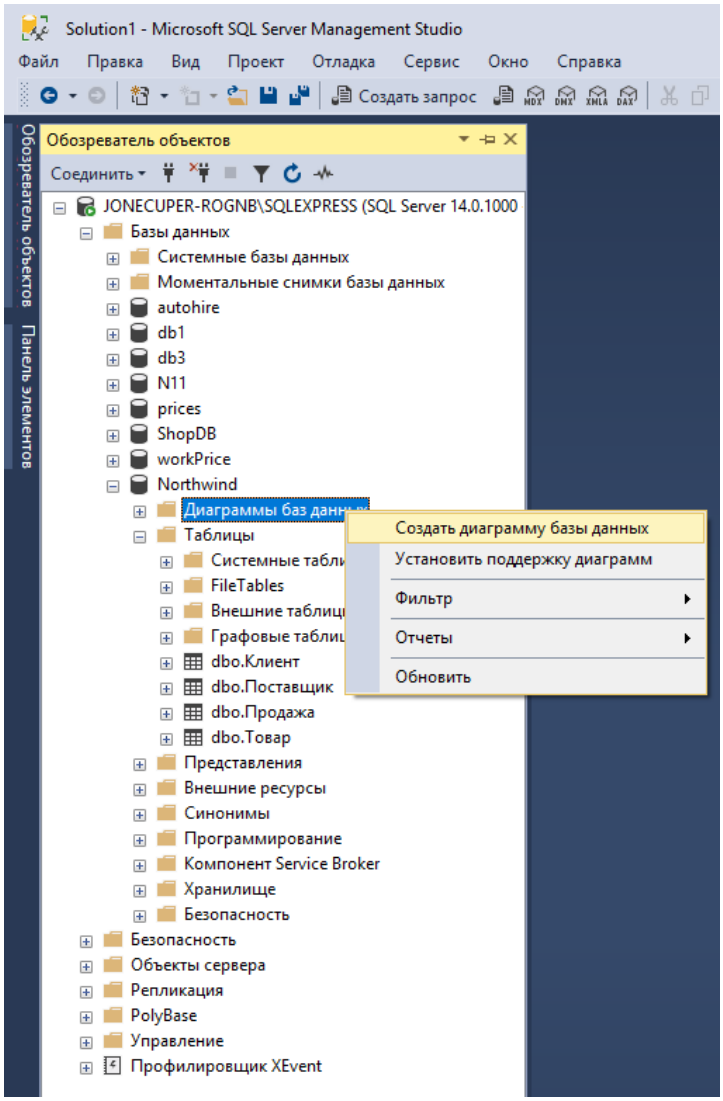


Рисунок 15 – Создание диаграммы базы данных

Появится мастер создания диаграмм, нажмите кнопку <Далее>. В окне <Available tables> выберите 4 созданных ранее таблицы и нажмите на кнопку <Add>, затем кнопку <Далее>, в следующем окне, убедившись в наличии всех таблиц, нажмите <Готово>. Появится окно редактирования диаграмм.

3. Выделите атрибут <Код_поставщика> таблицы «Поставщик» и перенесите его на тот же атрибут таблицы «Товар». Появится окно <Создание

связи>, в котором указана информация об имени связи, об атрибутах таблиц с первичным ключом и внешним ключом.

<Check existing data on creation> – означает, будут ли проверяться данные при создании связи.

<Enforce relationship for replication> – обеспечивает связь при репликации.

<Enforce relationship for INSERTs and UPDATEs> – обеспечивает связь при вставке и обновлении.

<Cascade Update related fields> – каскадное обновление связанных полей.

<Cascade Update related Records> – каскадное обновление связанных записей (рисунок 16).

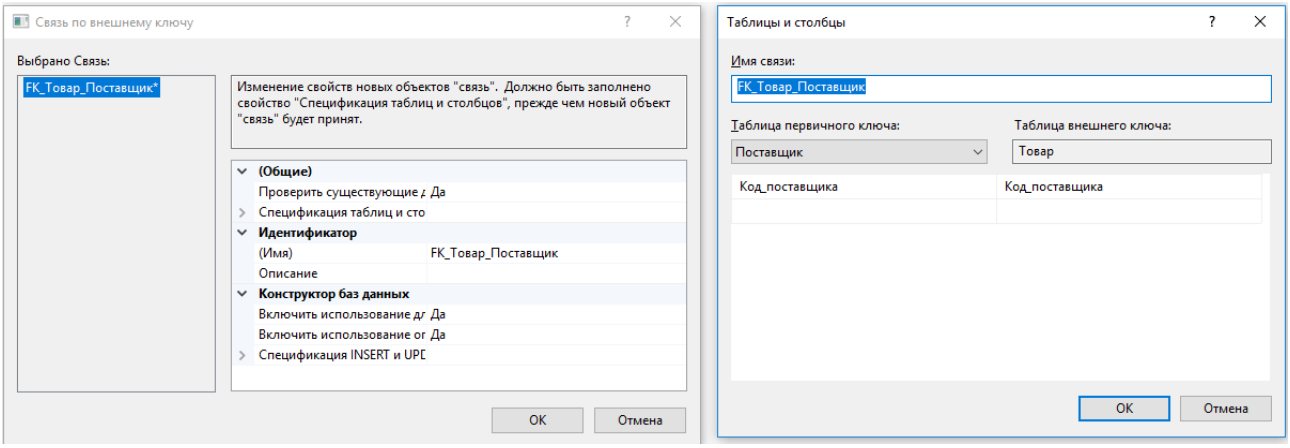


Рисунок 16 – Создание связи

4. Нажмите <OK>.

5. Создайте связи между таблицами по образцу (рисунок 17).

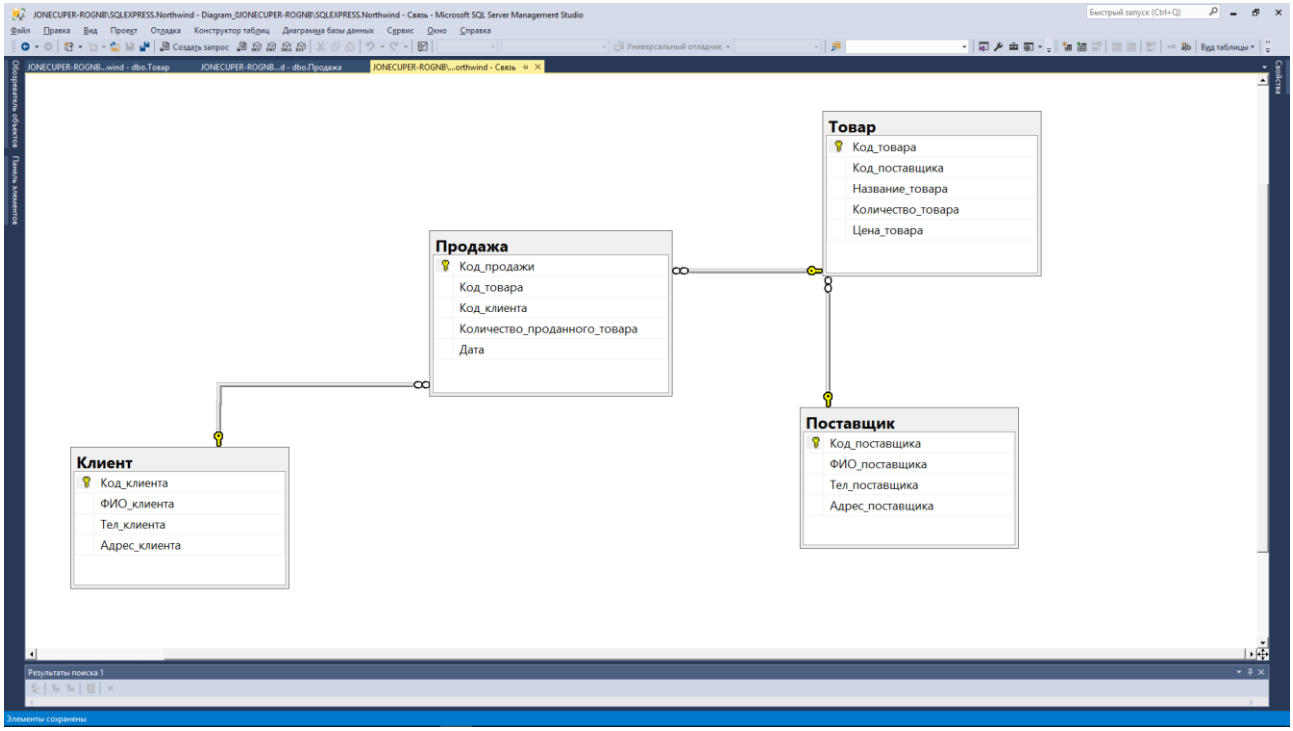


Рисунок 17 – Диаграмма связи

6. Закройте окно редактирования диаграмм и сохраните ее с именем «Связь».

7. Попробуйте удалить одну запись из таблицы «Поставщик». Обеспечение целостности данных не позволит вам это сделать.

4.12 Контрольное задание

Создать таблицы согласно выбранному варианту базы данных и обеспечить связи между ними.

4.13 Контрольные вопросы

1. Перечислите объекты базы данных MS SQL Server.
2. Какие способы создания таблиц существуют в СУБД MS SQL Server?
3. Какие типы данных допустимы при создании таблицы?
4. Как можно проверить создание таблиц?
5. Опишите способы заполнения таблиц.

6. Каким образом осуществляется обеспечение целостности данных в SQL Server?

5 Лабораторная работа № 5. Выборка и модификация данных

Цель работы: используя операторы T-SQL подготовить и реализовать серию запросов, связанных с выборкой информации и модификацией данных таблиц.

Используемое программное обеспечение: Microsoft SQL Server 2017.

SQL (англ. Structured Query Language - язык структурированных запросов) – универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных. Вопреки существующим заблуждениям, SQL является информационно-логическим языком, а не языком программирования.

SQL основывается на реляционной алгебре.

В свою очередь, язык SQL подразделяется на три подмножества:

Data Definition Language (DDL) - определяет набор команд, с помощью которых в базе данных создаются структурные объекты, т.н. метаданные, -- таблицы, домены, внешние ключи, индексы, хранимые процедуры и т.п.;

Data Manipulation Language (DML) определяет набор команд на извлечение данных из базы, а так же на вставку, изменение или удаление записей в таблице;

Data Control Language (DCL) - определяет набор операторов для разграничения доступа к данным для различных пользователей.

Transact-SQL (TRANSACT-SQL) – расширение языка SQL компаний Microsoft (для Microsoft SQL Server) и Sybase (для Adaptive Server Enterprise).

С тем чтобы сделать язык более мощным, SQL был расширен такими дополнительными возможностями как:

управляющие операторы

локальные переменные

различные дополнительные функции (для обработки строк, дат, математические и др.)

поддержка аутентификации Microsoft Windows

В настоящее время возможности программирования на стороне сервера значительно расширены за счет тесной интеграции SQL Server 2005 и технологии .NET.

Полное имя объекта (таблиц, хранимых процедур и т.д.) в TRANSACT-SQL состоит из четырех составных частей, отделенных друг от друга точками: server_name.database_name.schema_name.object_name, где

server_name – имя сервера, на котором располагается объект (в большинстве случаев имя сервера не указывается, если только объект не располагается на удаленном или связанном сервере);

database_name – имя базы данных, где располагается объект. Если объект расположен в текущей базе данных, которая определяется настройками соединения или командой use database_name, то имя базы данных в полном имени объекта обычно опускают;

schema_name – имя схемы, которой принадлежит данный объект;

object_name – собственно имя объекта.

При формировании имени объекта можно пропускать отдельные его части, если значение их очевидно по умолчанию.

Имена объектов выступают в качестве идентификаторов объектов. При помощи идентификаторов объектов можно получить доступ к самому объекту, другими словами, идентификатор является ссылкой на объект. Во время создания объекта указывается для него имя, и с этого момента имя объекта можно использовать в качестве идентификатора. Исключением являются ограничения. При создании ограничения, например первичного ключа, имя не указывается – сервер автоматически генерирует для него уникальное имя.

В SQL Server 2005 можно использовать два класса идентификаторов:

- регулярные идентификаторы;
- ограниченные (delimited) идентификаторы.

Регулярные идентификаторы образованы согласно принятому в SQL Server формату.

Ограниченные идентификаторы заключаются в квадратные скобки или двойные кавычки.

Как регулярные, так и ограниченные идентификаторы могут содержать от 1 до 128 символов. Для локальных временных таблиц длина идентификатора не должна превышать 116 символов.

первый символ идентификатора должен:

– соответствовать стандарту Unicode Standard 3.2. Согласно стандарту это может быть латинский символ от A до Z или от a до z, а также символ национального алфавита;

– быть одним из следующих символов: _– символ подчеркивания, @ – символ используется для обозначения параметров и переменных, # – символ используется для обозначения временных объектов. Имена, начинающиеся с символов ##, применяются для обозначения глобальных временных объектов. Некоторые встроенные функции SQL Server начинаются с @@. По этой причине не рекомендуется использовать подобные имена для обозначения объектов;

– последовательность символов в имени может включать:

– символы стандарта Unicode Standard 3.2;

– десятичные цифры;

– символы _, @, #, $.

идентификатор не должен быть зарезервированным словом SQL Server;

пробелы и специальные символы в имени не допускаются;

имена переменных и параметров процедур и функций должны также соответствовать перечисленным правилам.

В случае если будут использоваться идентификаторы, не соответствующий указанным правилам, то следует заключать такие идентификаторы в квадратные скобки или двойные кавычки. Квадратные скобки можно использовать в качестве ограничителей всегда, а вот двойные кавычки только при условии, что свойство QUOTED_IDENTIFIED установлено в состояние on. В Transact-SQL имеется команда “set guoted_identifier (on/off)”.

Для ограничения символьных констант можно использовать одинарные кавычки.

5.1 Использование операторов <SELECT> для выборки данных. Извлечение всех данных из таблицы «Titles»

1. Откройте <SQL Query Analyzer> и подключитесь к локальному серверу.

2. Введите следующие команды Transact-SQL на панели <Editor> окна <Query>:

```
USE Pubs
SELECT * FROM Titles
```

Этот оператор определяет базу данных, где содержится нужная для просмотра таблица. Оператор <SELECT> извлекает все данные из таблицы «Titles» в базе данных <Pubs>. Звездочка (*) в списке выбора указывает, что надо выбрать данные из всех столбцов таблицы.

3. Исполните оператор Transact-SQL.

Результирующий набор выводится на вкладке <Grids> панели <Results> (рисунок 1).

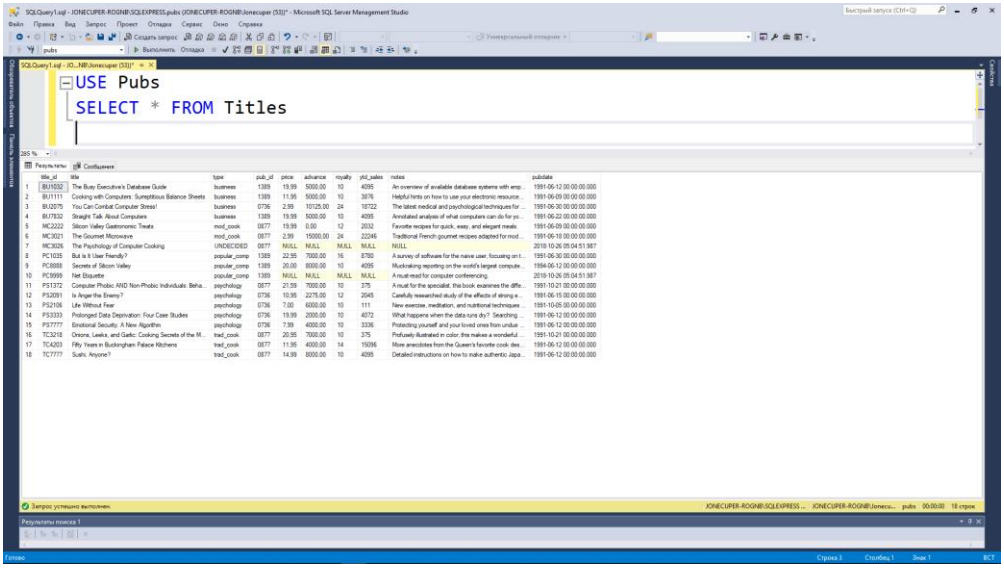


Рисунок 1 – Извлечение всех данных из таблицы «Titles»

5.2 Получение данных из определенных столбцов таблицы «Titles»

1. На панели <Editor> в окне <Query> введите следующий код Transact-SQL:

USE Pubs

SELECT Title_id, Title, Price, Ytd_sales

FROM Titles

Здесь оператор <SELECT> извлекает данные из <Title_id>, <Title>, <Price> и <Ytd_sales> базы данных <Pubs>.

2. Исполните оператор Transact-SQL.

Результирующий набор выводится на вкладке <Grids> панели <Results> (рисунок 2).

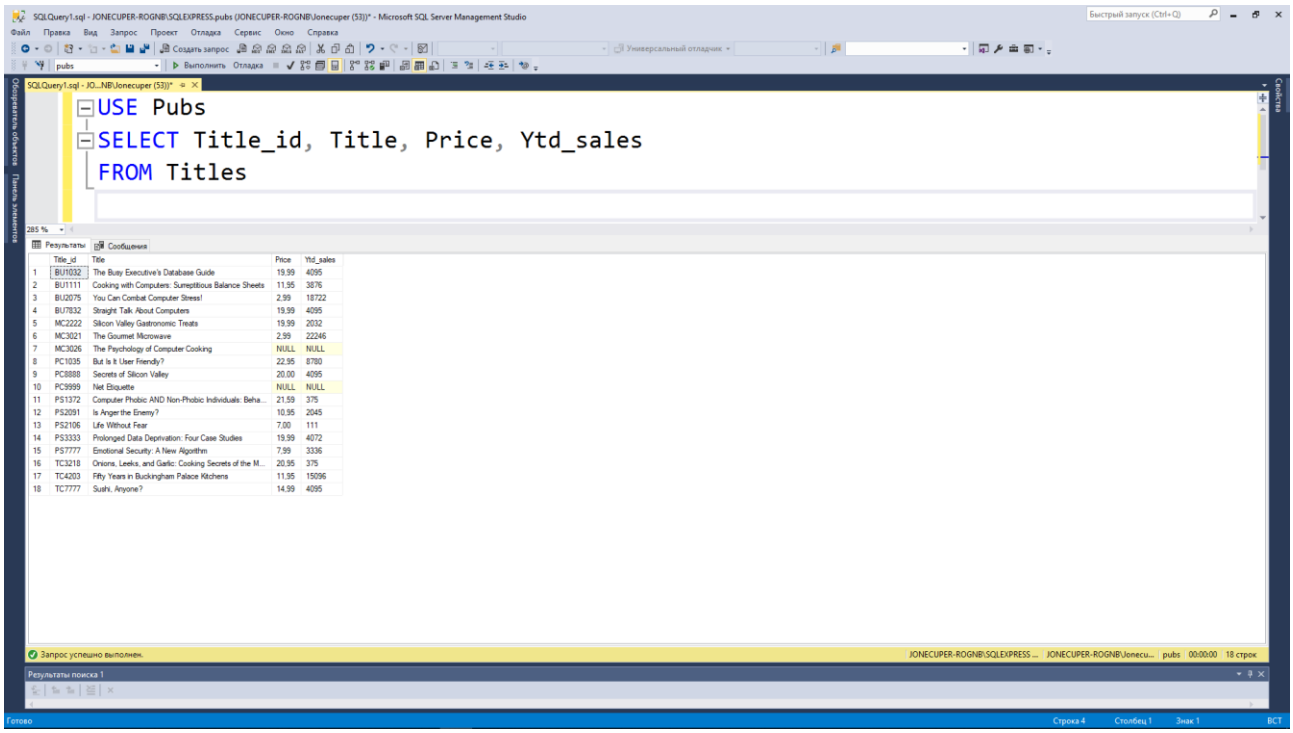


Рисунок 2 – Получение данных из определенных свойств таблицы «Titles»

5.3 Задание условия, для результирующего результирующего набора

1. На панели <Editor> в окне <Query> введите следующий код Transact-SQL:

USE Pubs

SELECT Title_id, Title, Price, Ytd_sales

FROM Titles

WHERE Price > 10

Теперь оператор <SELECT> извлечет лишь те строки, значение поля <Price> которых превышает $10.

2. Исполните оператор Transact-SQL.

Результирующий набор выводится на вкладке <Grids> панели <Results> (рисунок 3).

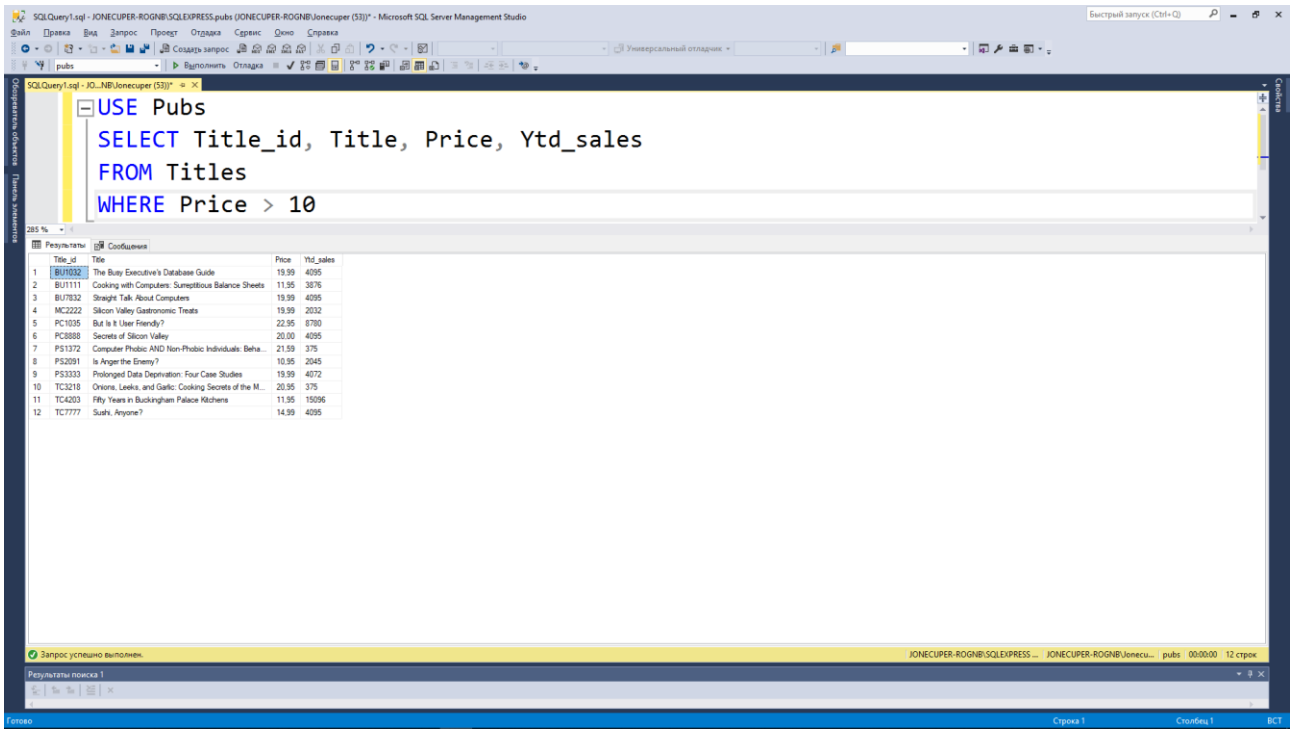


Рисунок 3 – Получение данных из определенных свойств таблицы «Titles», удовлетворяющих заданному условию

5.4 Задание порядка, в котором выводится результирующий набор

1. На панели <Editor> в окне <Query> введите следующий код Transact-SQL:

```
USE Pubs
SELECT Title_id, Title, Price, Ytd_sales
FROM Titles
WHERE Price > 10
ORDER BY Price DESC, Title
```

Результирующий набор, который вернет этот оператор <SELECT>, упорядочивается сначала по цене (по убыванию), а затем по заглавию (по возрастанию).

2. Исполните оператор Transact-SQL.

Результирующий набор выводится на вкладке <Grids> панели <Results> (рисунок 4).

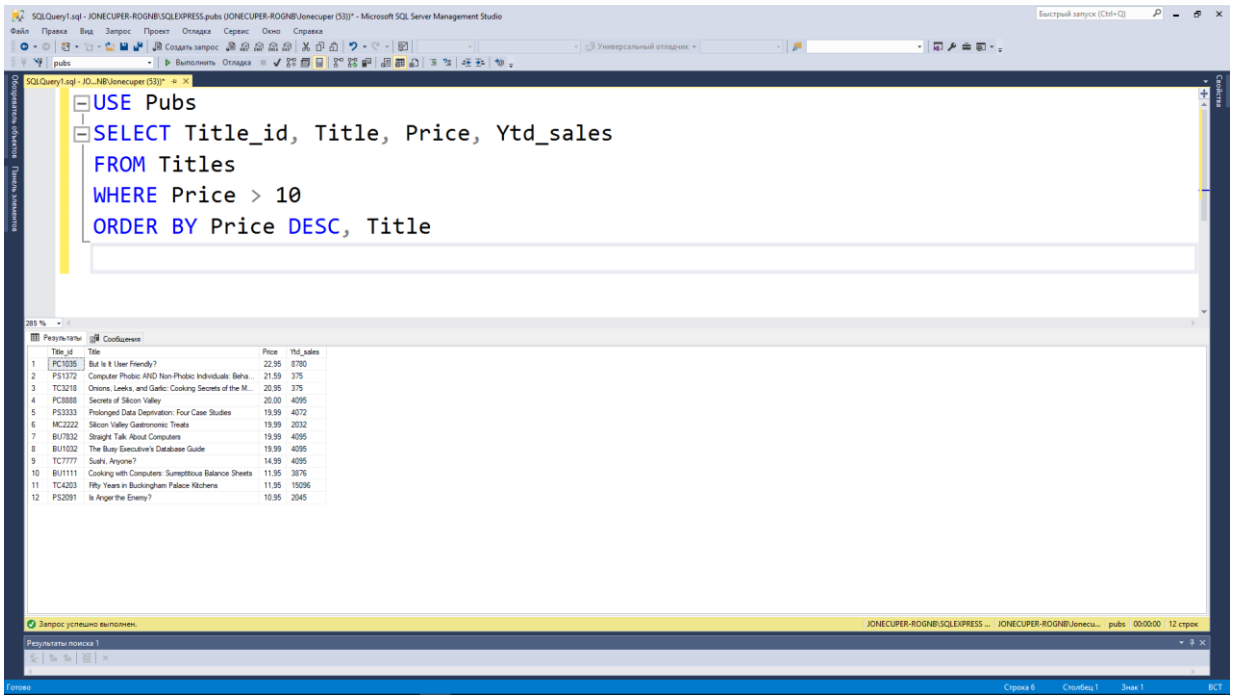


Рисунок 4 – Получение данных из определенных свойств таблицы «Titles», удовлетворяющих заданному условию и отсортированных

5.5 Группировка данных в результирующем наборе

1. На панели <Editor> в окне <Query> введите следующий код Transact-SQL:

```
USE Pubs
SELECT Type, AVG(Price) AS AvgPrice
FROM Titles
WHERE Price > 10
GROUP BY Type
ORDER BY AvgPrice DESC
```

В результирующем наборе, который вернет этот оператор <SELECT>, группируются строки с одними и теми же значениями поля <Type>. Строки, не соответствующие условиям конструкции <WHERE>, исключаются до начала любых операций по группировке. При группировке выполняется усреднение значений столбца <Price>, а полученное среднее значение вставляется в результирующий набор в виде столбца <AvgPrice>. Значения столбца <AvgPrice> упорядочиваются по убыванию.

2. Исполните оператор Transact-SQL.

Результирующий набор выводится на вкладке <Grids> панели <Results> (рисунок 5).

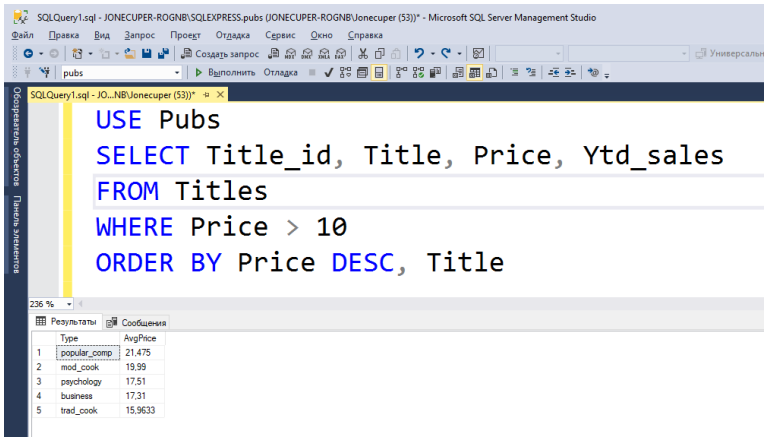


Рисунок 5 – Использование группировки данных и агрегатных функций

5.6 Создание таблицы для размещения результирующего набора

1. На панели <Editor> в окне <Query> введите следующий код Transact-SQL:

```
USE Pubs
SELECT Type, AVG(Price) AS AvgPrice
INTO TypeAvgPrice
FROM Titles
WHERE Price > 10
GROUP BY Type
ORDER BY AvgPrice DESC
```

Оператор <SELECT> создаст новую таблицу под названием <TypeAvgPrice>. В столбцах <Type> и <AvgPrice> размещаются значения результирующего набора.

2. Исполните оператор Transact-SQL.

На вкладке <Messages> панели <Results> выводится сообщение, где указано число строк, на которое повлияло исполнение оператора (рисунок 6).

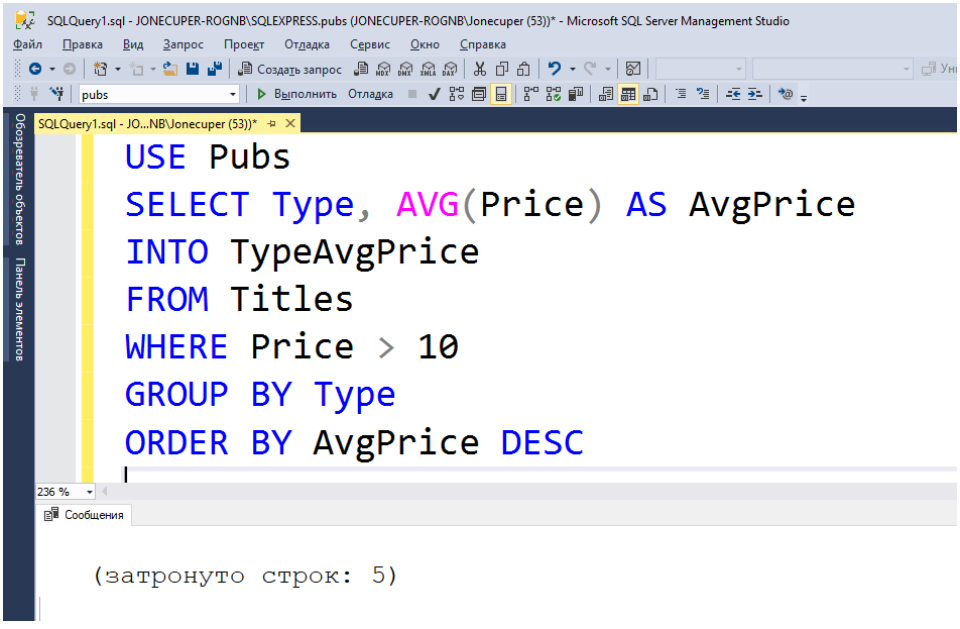


Рисунок 6 – Создание таблицы с результирующим набором данных

3. Введите и исполните следующий оператор TransactSQL:

SELECT * FROM TypeAvgPrice

Содержимое таблицы <TypeAvgPrice> выводится на вкладке <Grids> панели <Results> (рисунок 7).

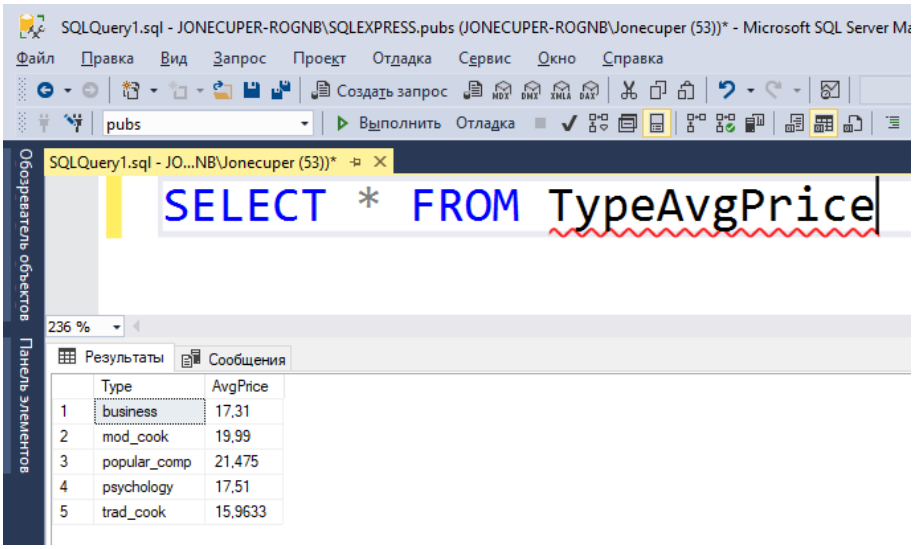


Рисунок 7 – Извлечение всех данных из таблицы «TypeAvgPrice»

4. Введите и исполните следующий оператор TransactSQL:

DROP TABLE TypeAvgPrice

На вкладке <Messages> панели <Results> выводится сообщение об успешном завершении команды (рисунок 8).

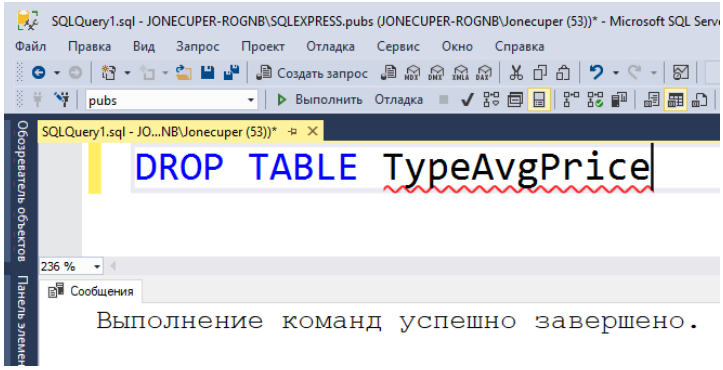


Рисунок 8 – Удаление таблицы «TypeAvgPrice»

5. Закройте <SQL Query Analyzer>.

5.7 Модификация данных в базе данных SQL Server. Создание таблицы в базе данных <Pubs>

1. Откройте <SQL Query Analyzer> и подключитесь к локальному серверу.

2. Введите на панели <Editor> в окне <Query> следующий код на Transact-SQL:

```
USE Pubs
CREATE TABLE Test1
(
RowID INT IDENTITY(1,1) NOT NULL,
Title VARCHAR(80) NOT NULL,
Type CHAR(12) NOT NULL DEFAULT ('Unknown'),
City VARCHAR(50) NULL,
Cost MONEY NULL
)
```

Этот оператор создает таблицу под названием «Test1», состоящую из пяти столбцов.

3. Исполните оператор Transact-SQL.

На вкладке <Messages> панели <Results> выводится сообщение об успешном завершении команды (рисунок 9).

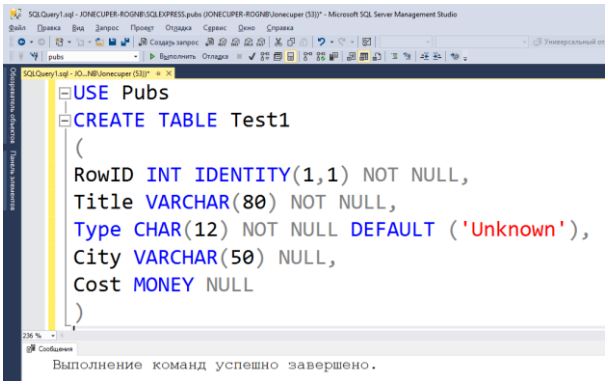


Рисунок 9 – Создание таблицы «Test1»

5.8 Добавление данных с помощью операторов <INSERT...VALUES>

1. Введите на панели <Editor> в окне <Query> следующий код на Transact-SQL:

INSERT INTO Test1 (Title, Type, Cost)

VALUES ('Test Title', 'business', 27.00)

Этот оператор вставляет строку в таблицу «Test1». В строке содержатся значения полей <Title>, <Type> и <Cost>.

2. Исполните оператор Transact-SQL.

На вкладке <Messages> панели <Results> выводится сообщение о том, что исполнение оператора повлияло на одну строку (рисунок 10).

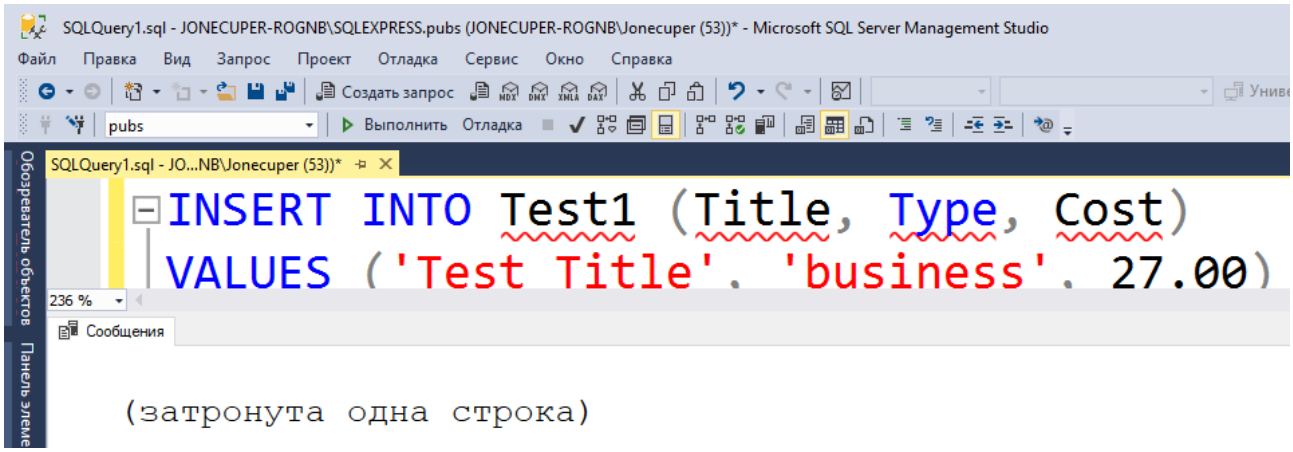


Рисунок 10 – Добавление данных в таблицу «Test1»

3. Напишите оператор <SELECT>, который позволит просмотреть все данные из таблицы «Test1».

4. Исполните оператор <SELECT>.

Содержимое таблицы «Test1» отобразится на вкладке <Grids> панели <Results>.

5. Изучите содержимое таблицы «Test1».

Обратите внимание, что в таблице имеется только одна строка – та, которую вы добавили с помощью оператора <INSERT>. SQL Server

автоматически сгенерировал значение поля <RowID>. Значение поля <City> – пустое, поскольку оно не было определено.

5.9 Добавление данных с помощью операторов <INSERT... SELECT>

1. Введите на панели <Editor> в окне <Query> следующий код на Transact-SQL:

```
INSERT INTO Test1 (Title, Type, Cost)
SELECT Title, Type, Price
FROM Pubs.dbo.Titles
```

Этот оператор берет данные из таблицы <Titles> в базе данных <Pubs> и вставляет их в таблицу «Test1».

2. Исполните оператор Transact-SQL.

На вкладке <Messages> панели <Results> выводится сообщение, где указано число строк, на которые повлияло исполнение оператора (рисунок 11).

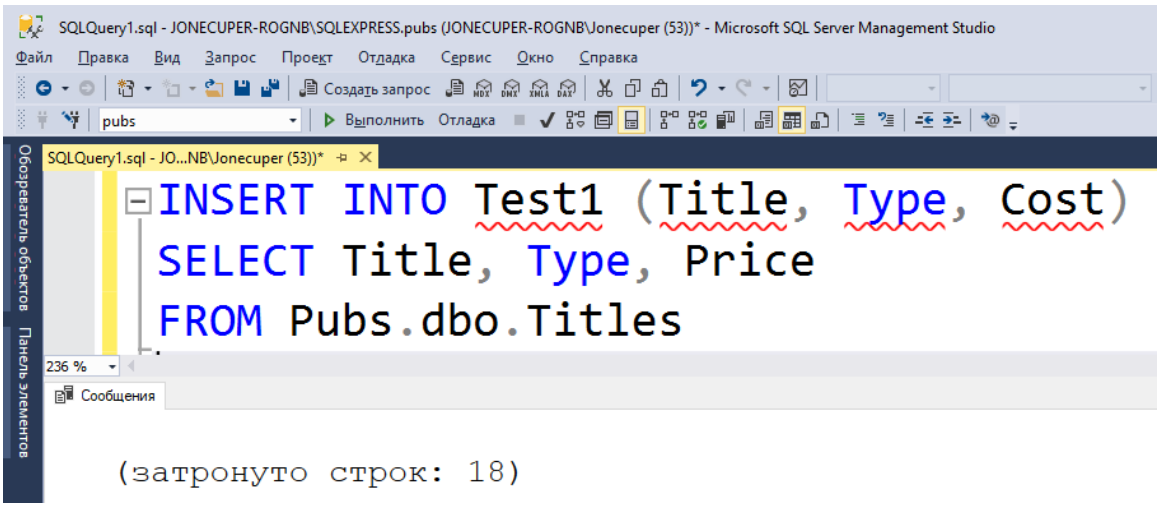


Рисунок 11 – Добавление в таблицу «Test1» данных из таблицы «Titles»

3. Воспользуйтесь оператором <SELECT> для просмотра данных таблицы «Test1». Обратите внимание, что значения поля <RowID> сгенерированы автоматически, а в поле <City> каждой строки содержится пустое значение.

5.10 Модификация данных с помощью оператора <UPDATE>

1. Просмотрите данные таблицы «Test1».

Если на панели остались результаты предыдущего запроса, то можно воспользоваться ими. В противном случае используйте для просмотра содержимого таблицы оператор <SELECT>.

2. Запишите названия нескольких книг, значение поля <Type> которых равно «business», а также их цену. Эти данные пригодятся во время модификации таблицы.

3. Введите на панели <Editor> в окне <Query> следующий код на Transact-SQL:

```
UPDATE Test1
SET Cost = Cost * 2
WHERE Type = 'business'
```

Этот оператор в два раза увеличивает значение поля <Cost> по сравнению с исходным значением для книг типа «business».

4. Исполните оператор Transact-SQL.

На вкладке <Messages> панели <Results> выводится сообщение, где указано число строк, на которые повлияло исполнение оператора (рисунок 12).

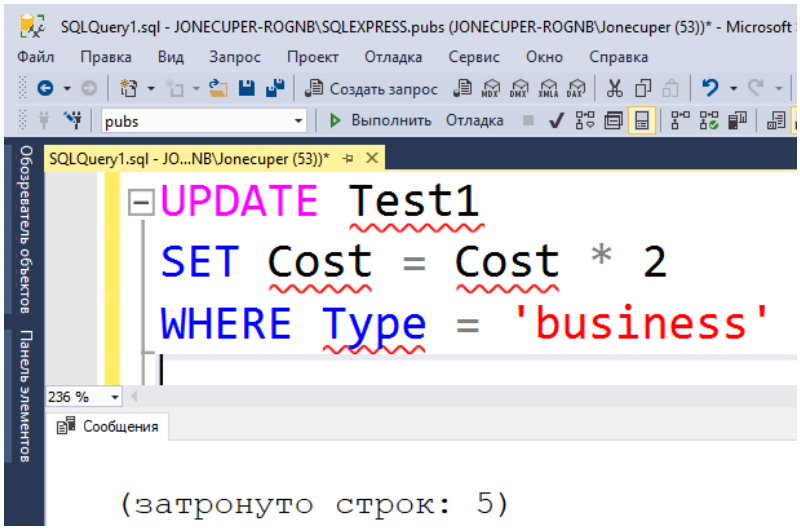


Рисунок 12 – Обновление данных в таблице «Test1»

5. Воспользуйтесь оператором <SELECT> для просмотра данных таблицы «Test1». Обратите внимание на удвоенное значение поля <Cost> для каждой книги типа «business».

5.11 Удаление данных из таблицы с помощью оператора <DELETE>

1. Введите на панели <Editor> в окне <Query> следующий код на <Transact-SQL>:

```
DELETE Test1
WHERE Title = 'Test Title'
```

Этот оператор удаляет все строки из таблицы, в столбце <Title> которой указано значение «Test Title».

2. Исполните оператор Transact-SQL.

На вкладке <Messages> панели <Results> выводится сообщение, где указано число строк, на которые повлияло исполнение оператора (рисунок 13).

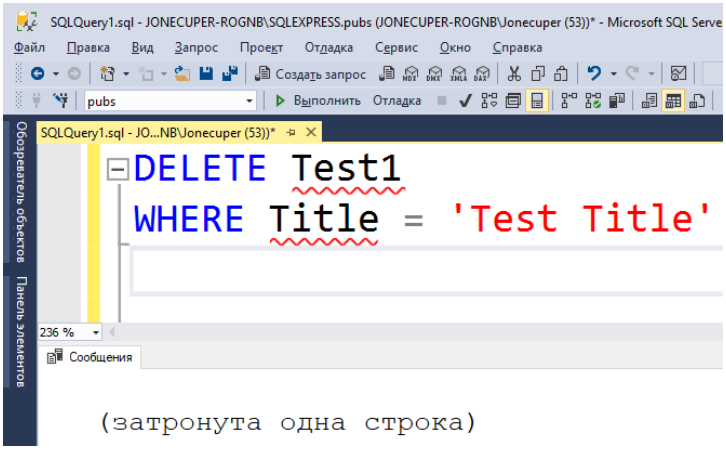


Рисунок 13 – Удаление данных из таблицы «Test1»

3. Воспользуйтесь оператором <SELECT> для просмотра данных таблицы «Test1». Обратите внимание, что строка <Test Title> удалена из таблицы.

4. Введите на панели <Editor> в окне <Query> следующий код на Transact-SQL:

```
DELETE Test1
```

Этот оператор удаляет все строки таблицы «Test1».

5. Исполните оператор Transact-SQL.

На вкладке <Messages> панели <Results> выводится сообщение, где указано число строк, на которые повлияло исполнение оператора (рисунок 14).

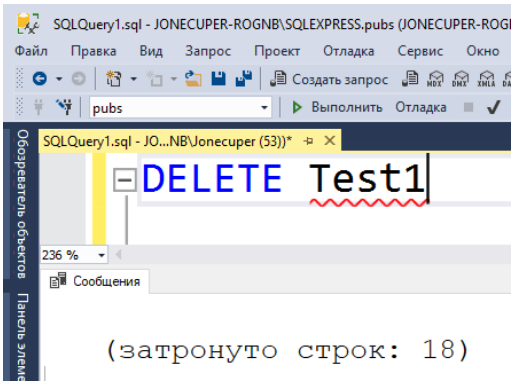


Рисунок 14 – Удаление всех строк таблицы «Test1»

6. Воспользуйтесь оператором <SELECT> для просмотра данных таблицы «Test1». Обратите внимание на отсутствие данных в таблице «Test1».

5.12 Удаление таблицы с помощью оператора <DROP TABLE>

1. Введите на панели <Editor> в окне <Query> следующий код на Transact-SQL:

DROP TABLE Test1

Этот оператора удаляет таблицу «Test1» из базы данных <Pubs>.

2. Исполните оператор Transact-SQL.

На вкладке <Messages> панели <Results> выводится сообщение об успешном завершении команды (рисунок 15).

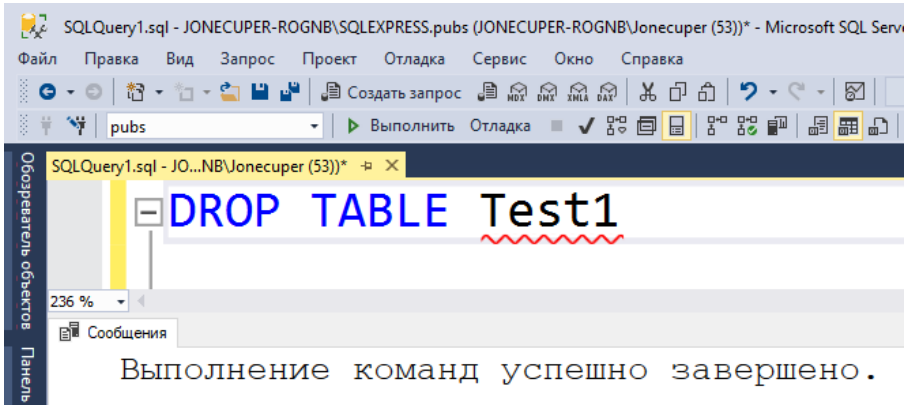


Рисунок 15 – Удаление таблицы «Test1»

3. С помощью окна <Object Browser> удостоверьтесь, что таблица «Test1» удалена из базы данных.

4. Закройте <SQL Query Analyzer>.

5.13 Контрольное задание

1. Средствами SQL Server 2000 создать четыре таблицы в базе данных <Pubs>. При создании таблиц выполнить следующие условия:

– поля номер_поставщика, номер_детали, номер_изделия во всех таблицах имеет символьный тип (char) и длину 6;

– поля рейтинг, вес и количество имеют целочисленный тип (int);

– поля фамилия, город (поставщика, детали или изделия), название (детали или изделия) имеют символьный тип nchar и длину 20;

– для всех полей допускаются значения NULL и значения-дубликаты, кроме поля номер_поставщика из таблицы «S», номер детали из таблицы «P», номер изделия из таблицы «J».

Таблица 1 – Таблица поставщиков «S»

Номер поставщика	Фамилия	Рейтинг	Город
S1	Смит	20	Лондон
S2	Джонс	10	Париж
S3	Блейк	30	Париж
S4	Кларк	20	Лондон
S5	Адамс	30	Афины

Таблица 2 – Таблица деталей «P»

Номер детали	Название	Цвет	Вес	Город
P1	Гайка	Красный	12	Лондон
P2	Болт	Зеленый	17	Париж
P3	Винт	Голубой	17	Рим
P4	Винт	Красный	14	Лондон
P5	Кулачок	Голубой	12	Париж
P6	Блюм	Красный	19	Лондон

Таблица 3 – Таблица изделий «J»

Номер изделия	Название	Город
J1	Жесткий диск	Париж
J2	Перфоратор	Рим
J3	Считыватель	Афины
J4	Принтер	Афины
J5	Флоппи-диск	Лондон
J6	Терминал	Осло
J7	Лента	Лондон

Таблица 4 – Таблица поставок «SPJ»

Номер поставщика	Номер детали	Номер изделия	Количество
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J1	400
S2	P3	J2	200
S2	P3	J3	200
S2	P3	J4	500
S2	P3	J5	600
S2	P3	J6	400
S2	P3	J7	800
S2	P5	J2	100
S3	P3	J1	200
S3	P4	J2	500
S4	P6	J3	300
S4	P6	J7	300
S5	P2	J2	200
S5	P2	J4	100
S5	P5	J5	500
S5	P5	J7	100
S5	P6	J2	200
S5	P1	J4	100
S5	P3	J4	200
S5	P4	J4	800
S5	P5	J4	400
S5	P6	J4	500

2. Убедиться в успешности выполненных действий. При необходимости исправить ошибки.

3. Используя Query Analizer, выполнить модификацию структуры таблиц «S» и «SPJ», добавив в «SPJ» поле с датой поставки. Убедиться в успешности выполненных действий. При необходимости исправить ошибки.

4. Проверить результат заполнения таблиц, написав и выполнив простейший запрос select * from имя_таблицы

5. Связать все эти таблицы и обеспечить целостность данных.

6. Согласно выданному преподавателем варианту выполнить задания, описанные ниже.

5.14 Варианты заданий на составление запросов по выборке информации из таблиц базы данных

Вариант 1

1. Выдать общее количество деталей «P1», поставляемых поставщиком «S1».

2. Получить все пары названий городов, таких, что какой-либо поставщик из первого города поставляет детали для некоторого изделия, изготовляемого во втором городе.

3. Выдать номера изделий, использующих только детали, поставляемые поставщиком «S1».

4. Получить номера деталей, поставляемых каким-либо поставщиком из Лондона, для изделия, изготавливаемого также в Лондоне.

Вариант 2

1. Выдать номера и фамилии поставщиков, поставляющих по крайней мере одну деталь, поставляемую по крайней мере одним поставщиком, который поставляет по крайней мере одну красную деталь.

2. Получить полный список деталей для всех изделий, изготавливаемых в Лондоне.

3. Выдать номера деталей, поставляемых каким-либо поставщиком из Лондона.

4. Получить номера деталей, поставляемых для всех изделий из Лондона.

Вариант 3

1. Выдать номера изделий, для которых детали поставляются по крайней мере одним поставщиком не из того же самого города.

2. Получить список всех поставок, в которых количество деталей находится в диапазоне от 300 до 750 включительно.

3. Выдать номера изделий, использующих по крайней мере одну деталь, поставляемую поставщиком «S1».

4. Получить номера и названия деталей, поставляемых для какого-либо изделия в Лондоне.

Вариант 4

1. Выдать номера и названия изделий, для которых город является первым в алфавитном списке таких городов.

2. Получить цвета деталей, поставляемых поставщиком «S1».

3. Выдать номера и фамилии поставщиков, поставляющих деталь «P1» для какого-либо изделия в количестве, большем среднего объема поставок детали «P1» для этого изделия.

4. Получить полный список деталей для всех изделий.

Вариант 5

1. Выдать названия изделий, для которых поставляются детали поставщиком «S1».

2. Получить номера деталей, поставляемых для какоголибо изделия поставщиком, находящимся в том же городе, где изготавливается это изделие.

3. Выдать номера и названия изделий, для которых поставщик «S1» поставляет несколько деталей каждого из поставляемых им типов.

4. Получить номера изделий, для которых средний объем поставки деталей «P1» больше наибольшего объема поставки любой детали для изделия «J1».

5.15 Варианты заданий на составление запросов по модификации информации из таблиц базы данных

Вариант 1

1. Увеличить на 10 рейтинг всех поставщиков, рейтинг которых в настоящее время меньше, чем рейтинг поставщика «S4».

2. Постройте таблицу, содержащую список номеров изделий, которые либо находятся в Лондоне, либо для них поставляются детали каким-нибудь поставщиком из Лондона.

Вариант 2

1. Удалить все изделия, для которых нет поставок деталей.

2. Построить таблицу с номерами поставщиков и парами номеров деталей, таких, что некоторый поставщик поставляет обе указанные детали.

Вариант 3

1. Увеличить размер поставки на 10 процентов для всех поставок тех поставщиков, которые поставляют какую-либо красную деталь.

2. Построить таблицу с комбинациями «цвет деталигород, где хранится деталь», исключая дубликаты пар (цветгород).

Вариант 4

1. Построить таблицу, содержащую список номеров деталей, которые поставляются либо каким-нибудь поставщиком из Лондона, либо для какого-либо изделия в Лондон.

2. Вставить в таблицу «S» нового поставщика с номером «S10» с фамилией Уайт из города Нью-Йорк с неизвестным рейтингом.

Вариант 5

1. Удалить все изделия из Рима и все соответствующие поставки.

2. Построить таблицу с упорядоченным списком всех городов, в которых размещаются по крайней мере один поставщик, деталь или изделие.

Вариант 6

1. Изменить цвет красных деталей с весом менее 15 фунтов на желтый.

2. Построить таблицу с номерами изделий и городов, где они изготавливаются, такие, что второй буквой названия города является «О».

Вариант 7

1. Увеличить на 10 рейтинг тех поставщиков, объем поставки которых выше среднего.

2. Построить таблицу с упорядоченным списком номеров и фамилиями поставщиков, поставляющих детали для изделия с номером «J1».

Осуществить выборку и модификацию данных согласно выбранному вами варианту базы данных.

5.16 Контрольные вопросы

1. Каким образом выполнить простейшие операции вставки строк данных в таблицу средствами T-SQL?

2. Как осуществить простейшие операции модификации строк таблицы средствами T-SQL?

3. Как выполнить просмотр таблицы?

4. Как сохранить результаты запроса в таблице?

6 Лабораторная работа № 6. Управление и манипулирование данными

Цель работы: выработка умений и навыков работы с операторами <INSERT>, <UPDATE>, <DELETE> и простейшими формами оператора <SELECT>.

Используемое программное обеспечение: Microsoft SQL Server 2017.

Язык управления данными используется для управления правами доступа к данным и выполнением процедур в многопользовательской среде. Более точно его можно назвать «язык управления доступом». Он состоит из двух основных команд:

– <GRANT> – предоставить привилегии пользователю или приложению на манипулирование объектами.

– <REVOKE> – отменить привилегии пользователя или приложения (забрать права).

Операторы манипулирования данными:

– <SELECT> – отобрать строки из таблиц.

– <INSERT> – добавить строки в таблицу.

– <UPDATE> – изменить строки в таблице.

– <DELETE> – удалить строки в таблице.

6.1 Управление данными. Предоставление права доступа к объекту базы данных с помощью <Query Analizer>

1. На панели <Editor> введите следующий оператор <GRANT>:

GRANT UPDATE, SELECT, INSERT, DELETE

ON Поставщик

TO PUBLIC

2. Выделите оператор <GRANT> и выполните его.

На вкладке <Messages> панели <Results> выводится сообщение о том, что команда успешно завершена (рисунок 1).

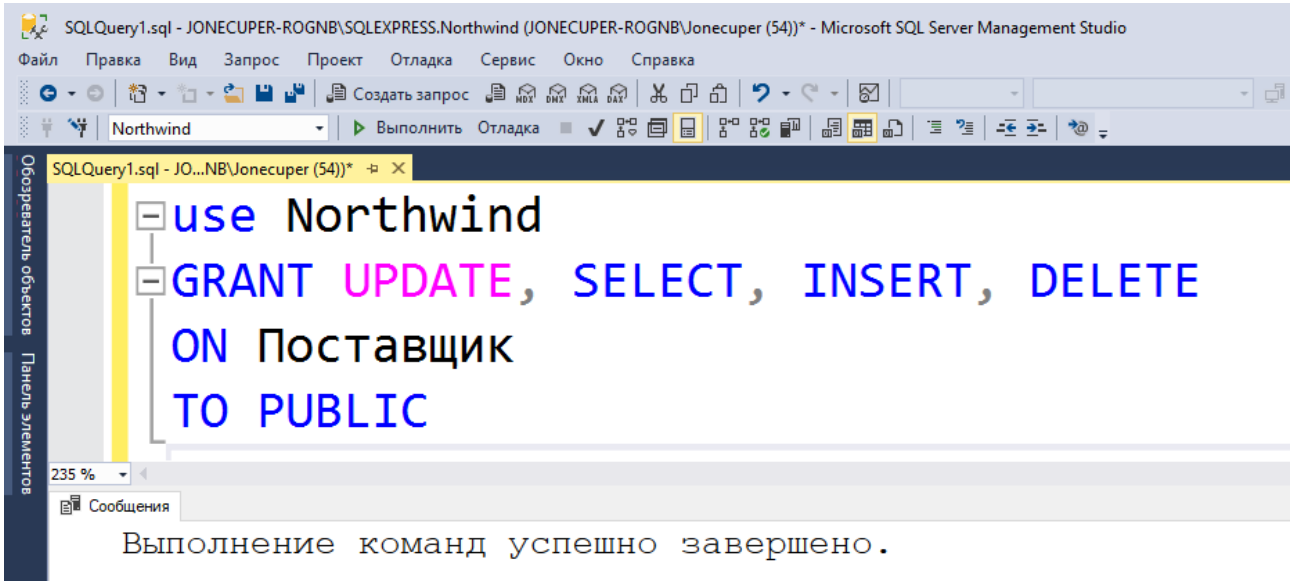


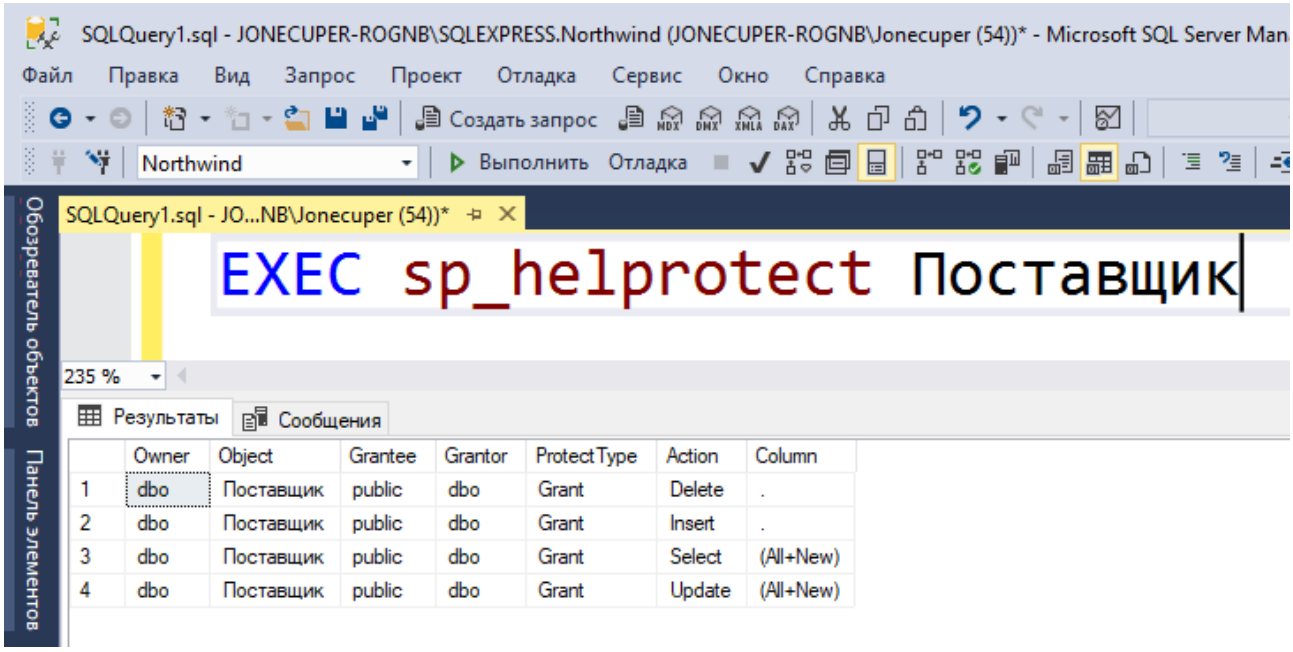
Рисунок 1 – Предоставление права доступа к таблице «Поставщик»

3. Введите следующий оператор <EXEC>:

EXEC sp_helprotect Поставщик

4. Исполните оператор <EXEC>.

На вкладке <Grids> панели <Results> отображаются данные о разрешениях пользователя для таблицы «Поставщик». Роли <Public> предоставлено право доступа <SELECT>, <UPDATE>, <INSERT>, <DELETE> для этой таблицы (рисунок 2).



	Owner	Object	Grantee	Grantor	ProtectType	Action	Column
1	dbo	Поставщик	public	dbo	Grant	Delete	.
2	dbo	Поставщик	public	dbo	Grant	Insert	.
3	dbo	Поставщик	public	dbo	Grant	Select	(All+New)
4	dbo	Поставщик	public	dbo	Grant	Update	(All+New)

Рисунок 2 – Отображение данных о разрешениях пользователя для таблицы «Поставщик» с помощью <Query Analizer>

6.2 Отзыв права доступа к объекту базы данных

1. Введите следующий оператор <REVOKE>:

REVOKE UPDATE, SELECT, INSERT, DELETE

ON Поставщик

TO PUBLIC

2. Выделите оператор <REVOKE> и выполните его.

На вкладке <Messages> панели <Results> выводится сообщение, свидетельствующее об успешном завершении команды (рисунок 3).

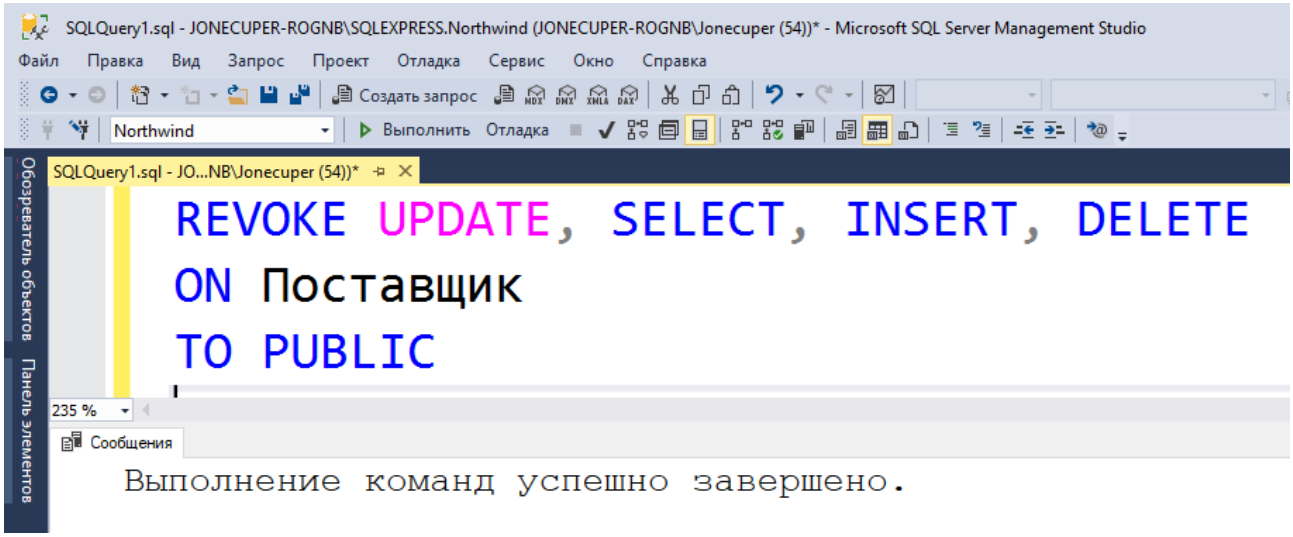


Рисунок 3 – Отзыв права доступа к таблице «Поставщик»

3. Введите следующий оператор <EXEC>:

EXEC sp_helprotect Поставщик

4. Исполните оператор <EXEC>.

На вкладке <Messages> панели <Results> выводится сообщение об отсутствии строк, совпадающих с критерием запроса. Поскольку у роли <Public> права доступа <SELECT> отозваны, теперь нет и предоставленных или отобранных прав доступа, о которых сообщает хранимая процедура (рисунок 4).

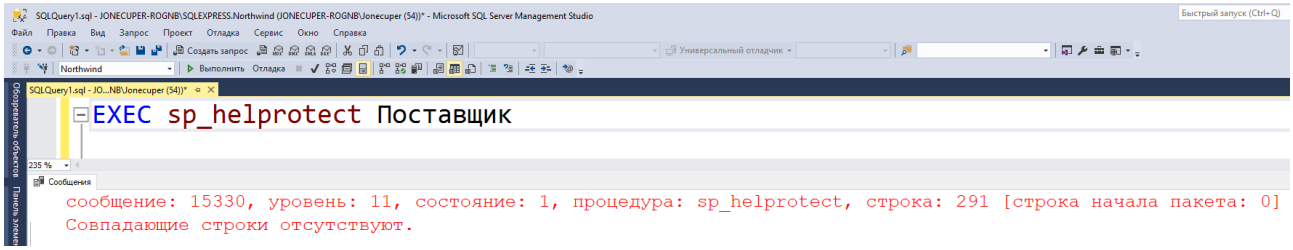


Рисунок 4 – Сообщение об отсутствии прав доступа на выборку данных из таблицы «Поставщик»

6.3 Предоставление права доступа к объекту базы данных с помощью <Enterprise Manager>

1. Запустите <Enterprise Manager>.

2. В базе данных <Northwind> выберете таблицу «Поставщик» и щелкните по ней 2 раза.

3. В появившимся окне <Свойства таблицы> нажмите на кнопку <Permissions>.

4. Для роли <public> разрешите выборку и вставку, запретите изменение и удаление (рисунок 5).

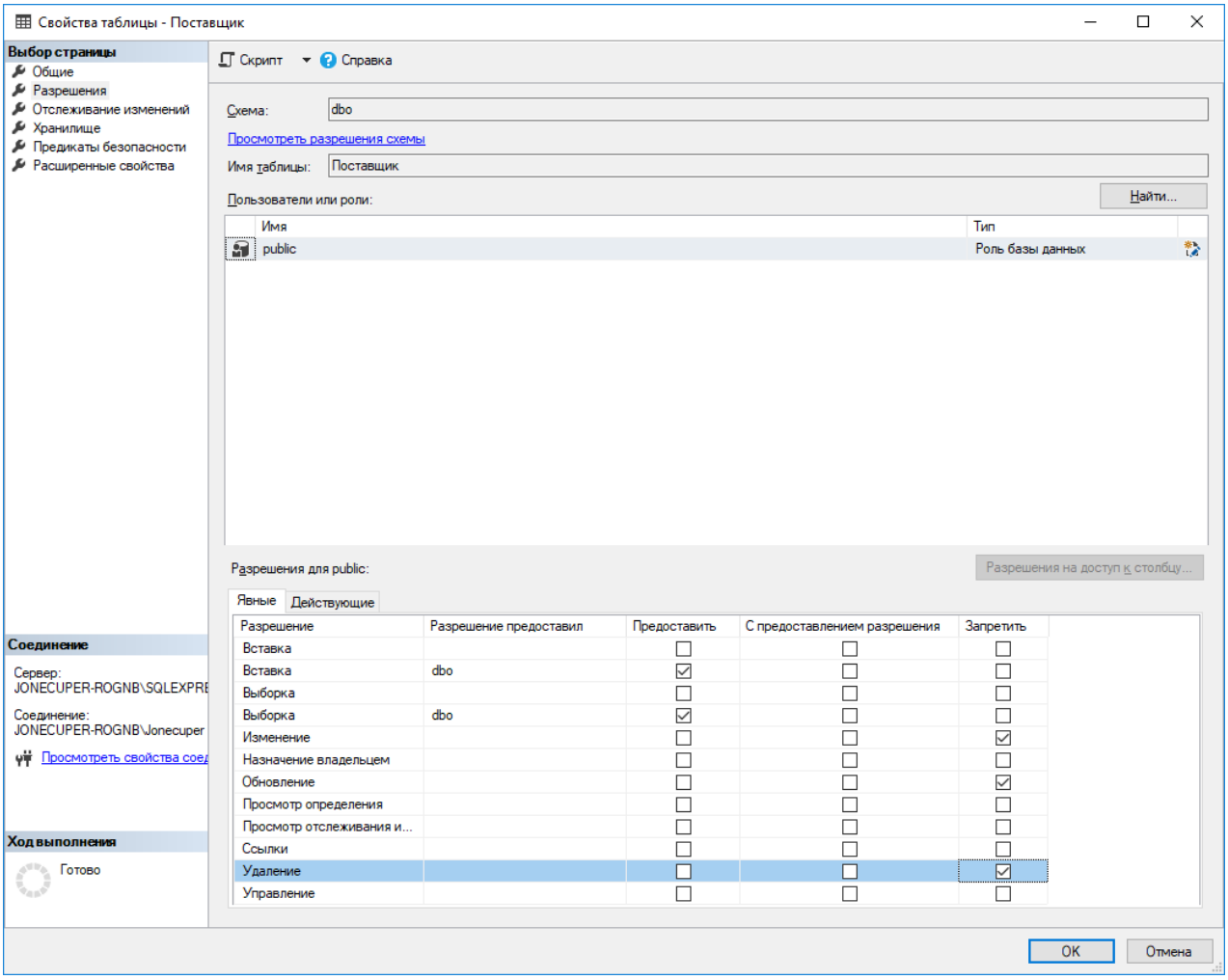


Рисунок 5 – Предоставление права доступа к таблице «Поставщик» с помощью <Enterprise Manager>

5. Выбрав роль и нажав на кнопку <Columns>, можно выбрать для каких столбцов будет разрешена или запрещена выборка и изменение.

6. Разрешите роли <public> изменение и удаление.

6.4 Манипулирование данными. Обновление данных

1. Запустите <Query Analizer>.

2. Введите следующий оператор <UPDATE> и выполните его:

```
USE Northwind
UPDATE Поставщик
SET Тел_поставщика = 89246541252 WHERE Код_поставщика = 03
```

На вкладке <Messages> появляется сообщение о том, что исполнение оператора повлияло на одну строку (рисунок 6).

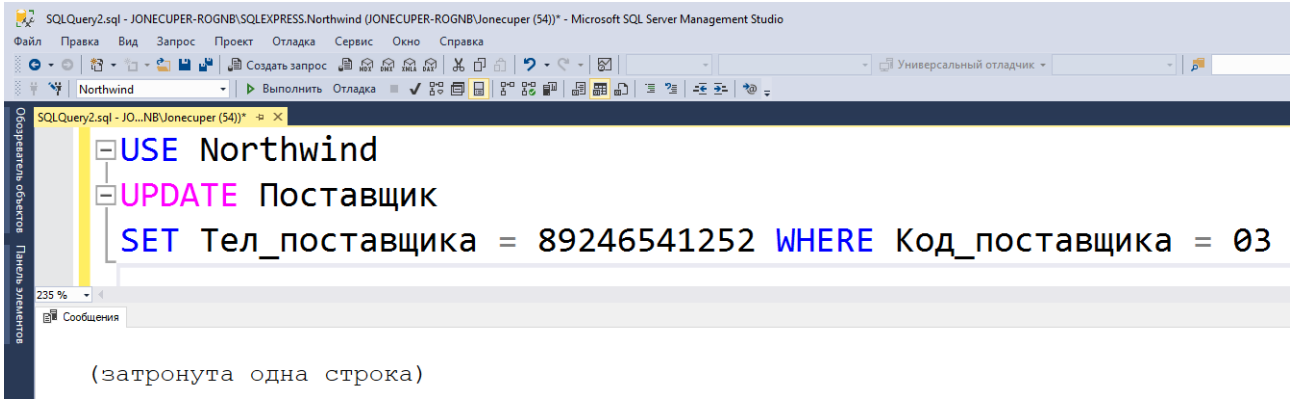


Рисунок 6 – Обновление данных в таблице «Поставщик»

6.5 Выборка данных

Исполните следующий оператор <SELECT>:

```
SELECT * FROM Поставщик
```

На панели <Grids> отображаются три строки из таблицы «Поставщик». Обратите внимание, что значение поля <Тел_поставщика> для Захарова М. В. теперь равно 89246541252 (рисунок 7).

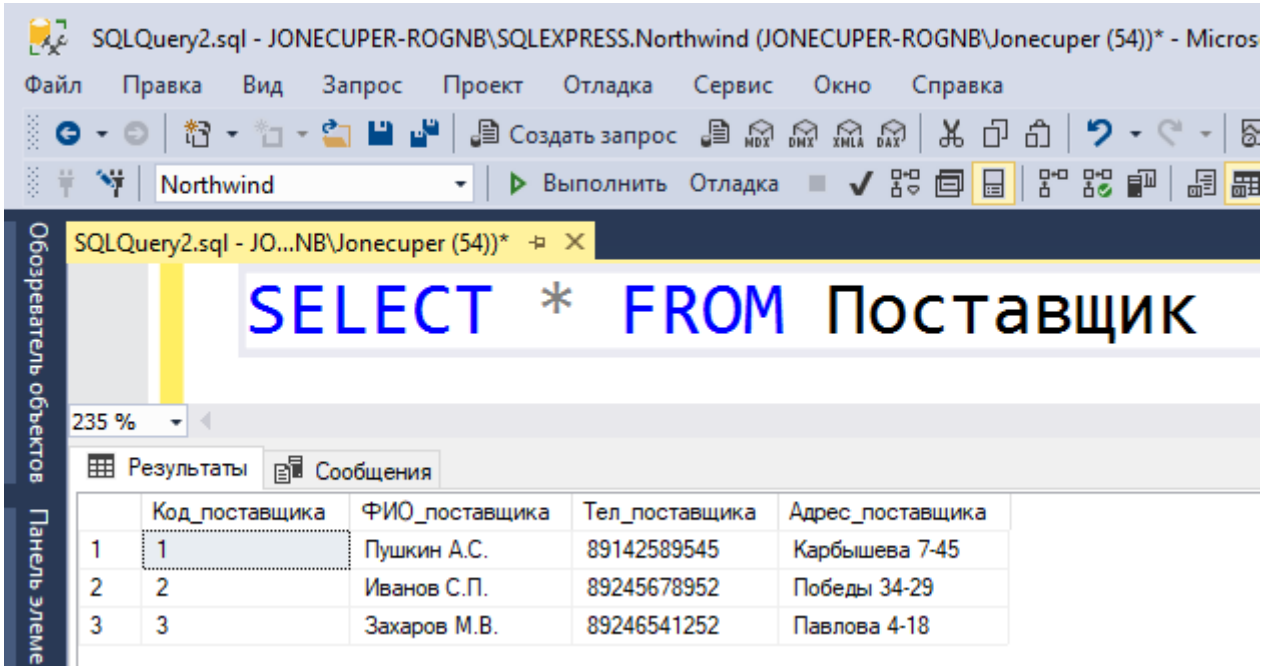


Рисунок 7 – Извлечение всех данных из таблицы «Поставщик»

6.6 Удаление данных

Наберите следующий оператор <DELETE> и исполните его:

DELETE FROM Поставщик

WHERE Код_поставщика = 02

На вкладке <Messages> выводится сообщение о том, что исполнение оператора повлиял на одну строку (рисунок 8).

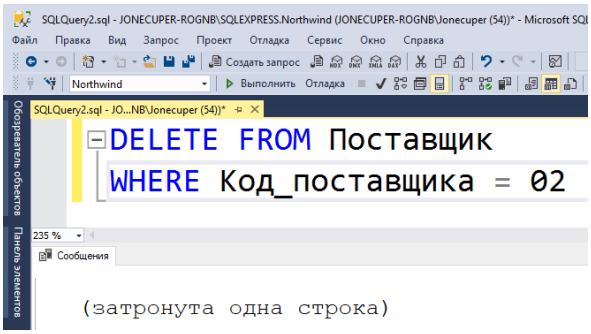


Рисунок 8 – Удаление из таблицы «Поставщик» записей, у которых <Код_поставщика> = 02

Исполните следующий оператор <SELECT>:

SELECT * FROM Поставщик

Теперь на панели <Grids> осталось только две строки из таблицы «Поставщик». Обратите внимание на отсутствие в списке имени Иванов С. П. (рисунок 9).

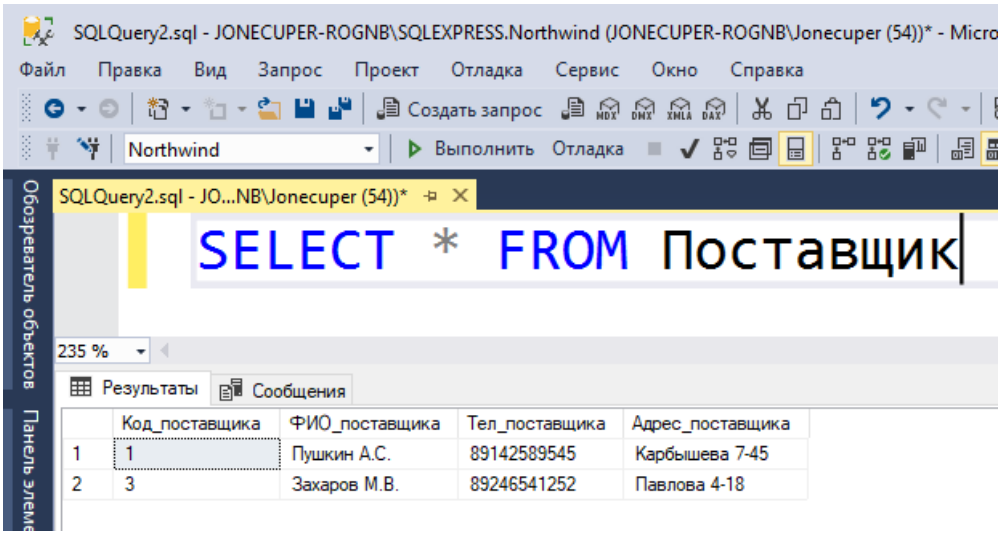


Рисунок 9 – Просмотр изменений в таблице «Поставщик»

6.7 Удаление таблицы

Введите следующий оператор <DROP> и исполните его:

DROP TABLE Поставщик

На вкладке <Messages> появляется сообщение об ошибке. Обеспечение целостности данных, созданное в лабораторной работе № 4, не позволяет удалить эту таблицу. Таблицу «Поставщик» возможно удалить, если только перед этим удалить ее связь с таблицей «Товар».

Закройте <SQL Query Analyzer>

6.8 Контрольное задание

Назначьте права доступа к данным в базе данных согласно выбранному вами варианту.

6.9 Контрольные вопросы

1. Какие операторы управления Вы знаете?
2. Перечислите операторы манипулирования данными.
3. Опишите синтаксис операторов <INSERT>, <UPDATE>, <DELETE>.

7 Лабораторная работа № 7. Управление и манипулирование данными

Цель работы: используя операторы T-SQL, научиться создавать хранимые процедуры и управлять ими.

Используемое программное обеспечение: Microsoft SQL Server 2017.

Хранимая процедура – компилируемый объект базы данных, хранящийся на сервере и представляющий собой набор SQL-инструкций. Хранимые процедуры могут обладать:

1) входными и выходными параметрами;

2) локальными переменными;

3) циклами и ветвлениями, то есть в них могут использоваться инструкции управления потоком;

4) в них могут производиться:

– стандартные операции с базами данных (как DDL, так и DML);

– числовые вычисления;

– операции над символьными данными.

5) результаты которых могут присваиваться переменным и параметрам.

С точки зрения приложений, работающих с базами данных, *хранимые процедуры* – это подпрограммы, которые выполнятся на сервере.

По отношению к базе данных, *хранимые процедуры* – это объекты, которые создаются и хранятся в ней. Они могут быть вызваны из клиентских приложений. При этом одна процедура может быть использована в любом количестве клиентских приложений.

7.1 Создание и вызов процедуры. Создание процедуры

CREATE PROC[EDURE] *имя_процедуры* [; *номер*]

[@параметр тип_данных [VARYING] [= *значение*] [OUTPUT]

[,…]

[WITH {RECOMPILE I ENCRYPTION|RECOMPILE|ENCRYPTION}]

[FOR REPLICATION]

AS

команды SQL

[RETURN [*код_статуса*]]

Имя_процедуры подчиняется стандартным правилам выбора имен объектов:

– в рамках одной базы данных для процедур с одинаковыми именем и владельцем номер должен быть уникальным;

– имена локальных временных процедур должны начинаться с символа #;

– имена глобальных временных процедур должны начинаться с символов ##.

Номер используется для группировки процедур как единого целого (часто в сочетании с версией приложения).

@параметр – представляет параметр, передаваемый и / или получаемый из процедуры. Процедура имеет до 1024 параметров. Параметры могут принимать значения <NULL>. Они не могут использоваться в процедуре вместо объектов баз данных, если только они не включаются в команду <EXEC>. Параметры относятся к встроенным либо пользовательским типам данных. Обратите внимание, правила, значения по умолчанию и свойства столбцов не относятся к параметрам, определяемым пользовательскими типами данных. Параметр может относиться к любому типу данных SQL Server, в том числе <text>, <image> или <cursor>. Для параметров типа <cursor> ключевые слова ***varying*** и ***output*** являются обязательными.

VARYING означает, что параметр возвращает итоговый набор. Используется только для типа данных <CURSOR>.

Значение определяет значение параметра по умолчанию. Может быть любой константой или <NULL>.

OUTPUT означает, что параметр является выходным, то есть может возвращаться стороне, вызвавшей процедуру. Параметры типа <CURSOR> обязаны быть выходными.

ENCRYPTION означает, что таблица <syscomment> должна шифроваться (благодаря чему текст процедуры скрывается от постороннего взгляда).

FOR REPLICATION означает, что процедура участвует в процессе публикации. Обратите внимание, параметры <FOR REPLICATION> и <RECOMPILE> являются взаимоисключающими.

Команды SQL – произвольное количество команд SQL, составляющих тело процедуры. Максимальный объем команд не должен превышать 128 Кбайт.

7.2 Вызов процедуры

[EXEC[UTE]] [*@код_возврата* =] *имя_процедуры* [;номер] [*@список_параметров*] [WITH RECOMPILE]

@код_возврата – переменная для сохранения кода возврата процедуры.

Номер – числовой идентификатор процедуры. При отсутствии этого аргумента выполняется процедура с максимальным номером версии.

@список параметров – список параметров процедуры. Параметры передаются как по имени (имя = значение), так и по позиции в списке, но если некоторый параметр передается по имени, все последующие параметры в списке тоже должны передаваться по имени. Чтобы пропустить параметр в середине списка, следует передавать остальные параметры по имени.

WITH RECOMPILE – процедура заново компилируется при выполнении.

Удаление процедуры:

DROP PROC имя_процедуры [, имя_процедуры[, ...]]

7.3 Системные хранимые процедуры. Просмотр системных хранимых процедур в базе данных <Master>

1. Откройте <Query Analyzer> и подключитесь к локальному серверу.

2. Откройте окно <Object Browser>, если оно еще не открыто. В окне <Object Browser> выводится иерархическое дерево объектов базы данных.

3. Раскройте узел <Master>. Появляется список типов объектов. Обратите внимание на узлы <Stored Procedures> и <Extended Procedures>.

4. Раскройте узел <Stored Procedures>. Появляется список хранимых процедур и расширенных хранимых процедур базы данных <Master>.

5. Изучите имена процедур из списка. Обратите внимание, что владельцем всех процедур является <dbo>.

6. Раскройте системную хранимую процедуру dbo.sp_who. Появляются узлы <Parameters> и <Dependencies>. 7. Раскройте узел <Parameters> (рисунок 1).

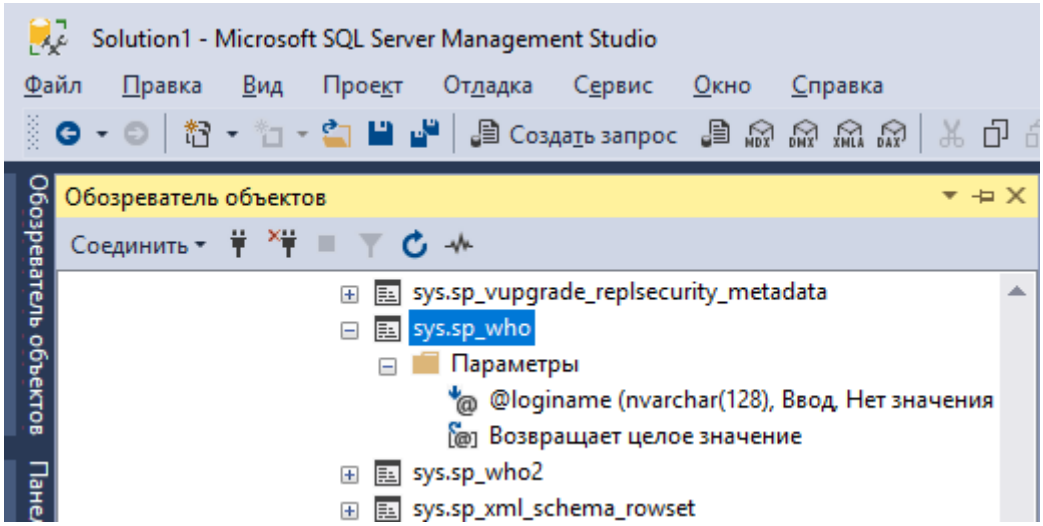


Рисунок 1 – Просмотр параметров системной хранимой процедуры «dbo.sp_who»

Обратите внимание, что для этой процедуры определены два параметра: <@RETURN_VALUE> и <@loginame. @RETURN_VALUE> – это встроенный

параметр, который используется для хранения кодов возврата, он есть у всех хранимых процедур, <@loginame> – это входной параметр.

8. Зависимость <Dependencies>.

Обратите внимание, что эта хранимая процедура зависит только от таблицы «dbo.sys-processes». Это системная таблица, которая хранится в базе данных <Master>. Системная хранимая процедура «sp_who» выполняет запрос к таблице «SysProcesses» и выводит часть ее данных в своем результирующем наборе. Эту таблицу можно найти, раскрыв узел <System Tables>.

9. Если вы уже просмотрели таблицы «dbo.sysprocesses», вернитесь к системной хранимой процедуре «sp_who» в окне <Object Browser>.

10. Прокрутите содержимое <Object Browser>, пока не увидите узел <Extended Procedures>. Обратите внимание на три хранимые процедуры с префиксом <xp_>, которые расположены непосредственно над узлом <Extended Procedures>. Они не являются расширенными.

11. Раскройте узел <Extended Procedures>.

Прокручивая список объектов этого узла, обратите внимание на наличие в нем как системных, так и расширенных хранимых процедур. Большинство процедур из этого узла являются расширенными (рисунок 2).

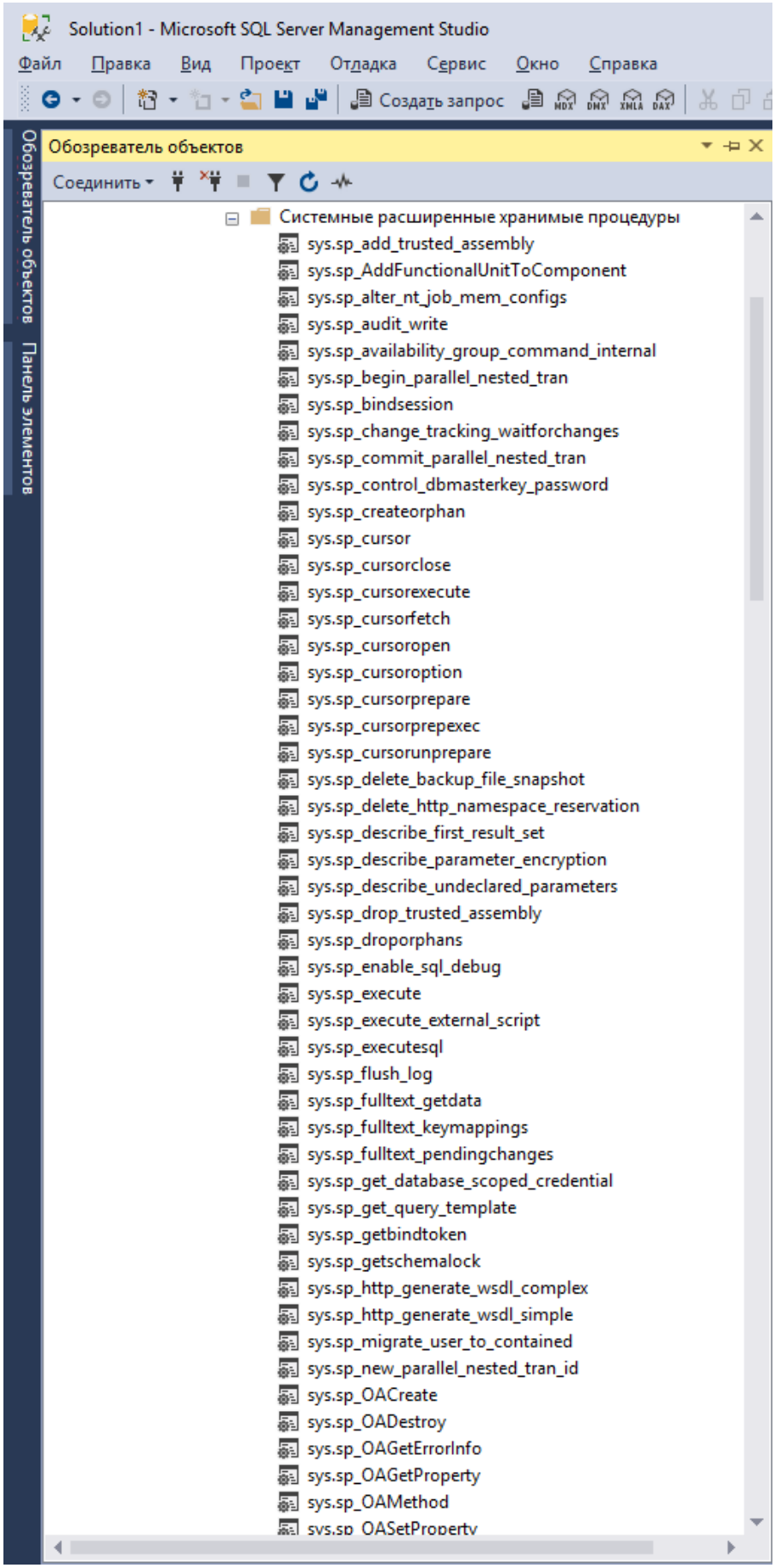


Рисунок 2 – Просмотр хранимых процедур узла Extended Procedures

12. Оставьте <Query Analyzer> открытым.

7.4 Методы просмотра содержимого хранимой процедуры

1. Щелкните правой кнопкой мыши по объекту «dbo.sp_who». Появляется контекстное меню этого объекта.

2. Выберите <Scripting Options> (рисунок 3).

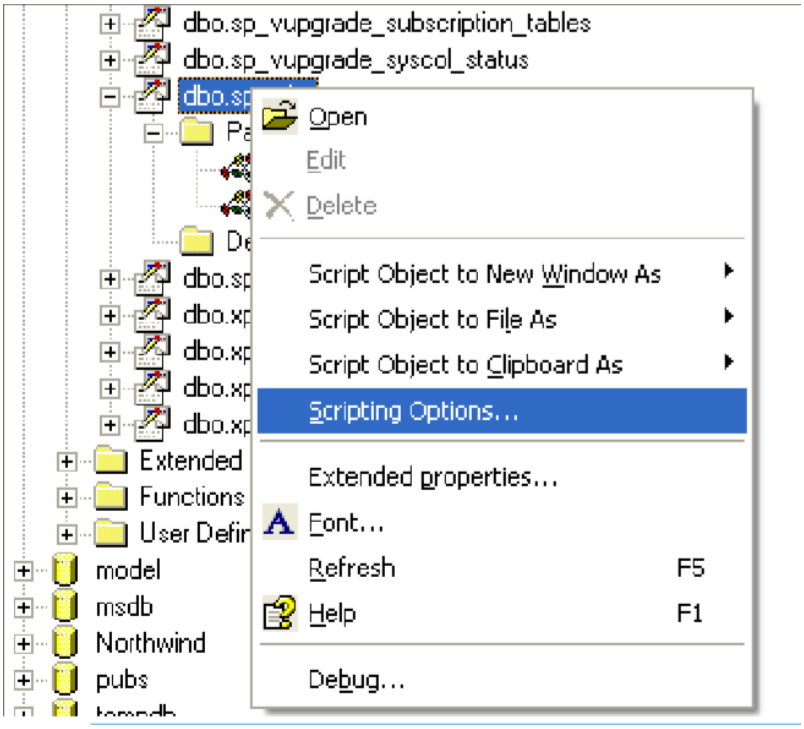


Рисунок 3 – Просмотр содержимого хранимой процедуры (1-й способ)

Выводится диалоговое окно <Options>.

3. Установите флажок <Include Descriptive Headers In The Script> и щелкните <OK> (рисунок 4).

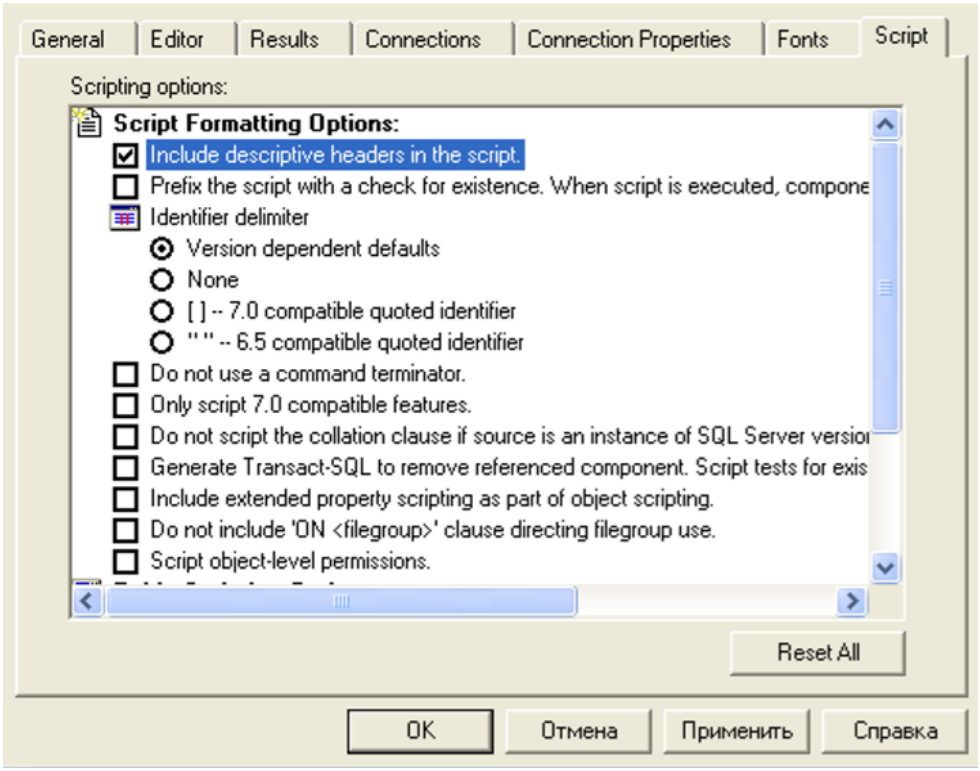


Рисунок 4 – Установление флажка <Include Descriptive Headers In The Script>

Этот параметр выводит около заголовка текста хранимой процедуры ее имя и дату создания.

4. Щелкните правой кнопкой мыши по объекту «dbo.sp_who».

Выводится контекстное меню для него.

5. Укажите <Point to Script Object To New Window As> и щелкните <Create> (рисунок 5).

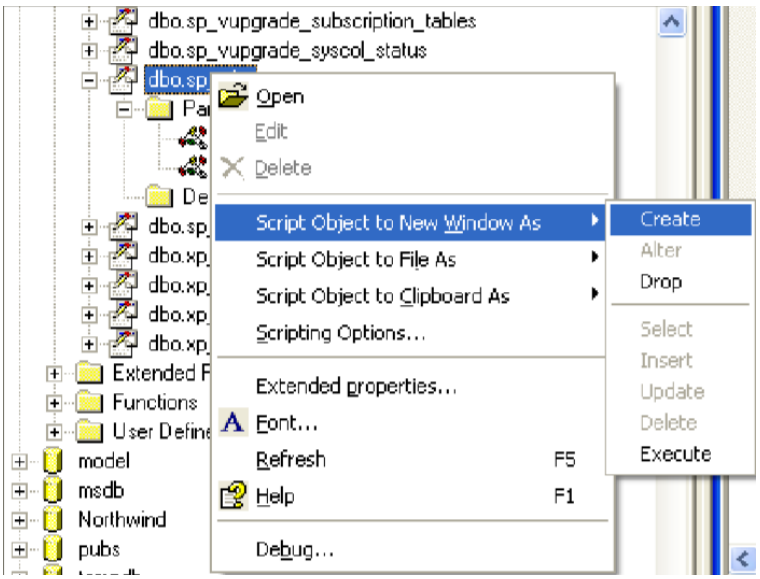


Рисунок 5 – Просмотр содержимого хранимой процедуры (2-й способ)

В окне <Query> появляется хранимая процедура «dbo.sp_who».

Обратите внимание, что рядом с заголовком файла появляются ключевые слова <CREATE PROCEDURE> (рисунок 6).

```
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS OFF
GO

/****** Object:  Stored Procedure dbo.sp_who    Script Date: 07.09.2013 18:33:01 ******/
create procedure sp_who  --- 1995/11/28 15:48
      @loginame sysname = NULL --or 'active'
as

declare  @spidlow       int,
                @spidhigh       int,
                @spid           int,
                @sid            varbinary(85)

select   @spidlow       =     0
                ,@spidhigh      = 32767


if (    @loginame is not NULL
```

Рисунок 6 – Листинг хранимой процедуры <sp_who>

6. Щелкните на панели инструментов кнопку <New Query> или нажмите <CTRL+N>. На правой панели появляется окно нового запроса.

7. На панели <Editor> в окне <Query> введите следующий код Transact-SQL:

sp_helptext [master.dbo.sp_who]

В этой команде для вывода содержимого системной хранимой процедуры «sp_who», которая содержится в базе данных <Master>, используется системная хранимая процедура «sp_helptext». Хотя указывать полное имя хранимой процедуры не обязательно, лучше это все-таки делать – в этом случае вы гарантированно откроете именно тот объект, который вам нужен.

8. Исполните команду.

Содержимое системной хранимой процедуры «sp_who» выводится на вкладке <Grids> панели <Results> (рисунок 7).

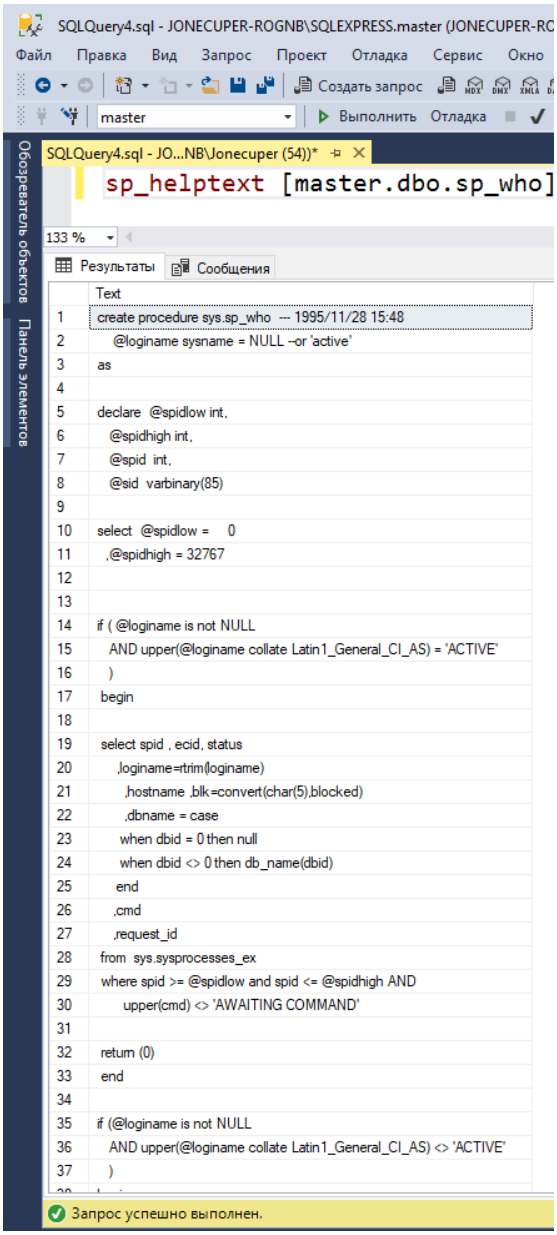


Рисунок 7 – Содержимое системной хранимой процедуры <sp_who>

7.5 Создание и исполнение хранимой процедуры в базе данных <Northwind>

1. Откройте <Query Analyzer> и подключитесь к локальному серверу.

2. Закройте окно <Object Browser>, если оно открыто.

3. Раскройте окно <Query>, чтобы оно заняло все рабочее пространство в <Query Analyzer>.

4. На панели <Editor> в окне <Query> введите следующий код:

```
USE Northwind
GO
CREATE PROCEDURE dbo.CustOrderHistRep
@CustomerID char(5)
AS
SELECT ContactName, ContactTitle
FROM Customers WHERE CustomerID = @CustomerID
SELECT ProductName, Total=SUM(Quantity)
FROM Products P, [Order Details] OD, Orders O, Customers C
WHERE
C.CustomerID = @CustomerID AND C.CustomerID = O.CustomerID
AND O.OrderID = OD.OrderID AND OD.ProductID = P.ProductID
GROUP BY ProductName
GO
```

При исполнении первого пакета база данных <Northwind> становится текущей. Далее создается процедура <CustOrderHistRep> и определяется единственный входной параметр – <@CustomerID>. Входной параметр может располагаться в одной строке с оператором <CREATE PROCEDURE>, но здесь для ясности он перенесен в отдельную строку. Аналогичный подход использован далее при разбиении по строкам кода операторов <SELECT>. Обратите внимание, что для параметра <@CustomerID> задан тип данных <char(5)>. Если выполнить запрос к таблице «[Northwind].[dbo].[Customers]», то видно, что длина

всех идентификаторов покупателей равна пяти символам. Строка, в которой находится единственное ключевое слово <AS>, является разделительной линией между созданием процедуры в таблице «SysObjects» и текстом процедуры, сохраняемым в таблице «SysComments».

5. Просмотрите операторы <SELECT>, которые расположены ниже ключевого слова <AS>.

При исполнении запроса в ответ на ввод идентификатора покупателя первый оператор <SELECT> выводит имя контактного лица и заголовок контактной информации. Второй оператор <SELECT> выводит названия и общее количество (<SUM>) каждого товара, который приобрел покупатель. Результирующий набор возвращает данные, сгруппированные по названию товара. Можно заметить, что несколько соединений реализовано в конструкции <WHERE>, а не <FROM>. Во время модификации процедуры мы переместим выражения <JOIN> в конструкцию <FROM> (рисунок 8).

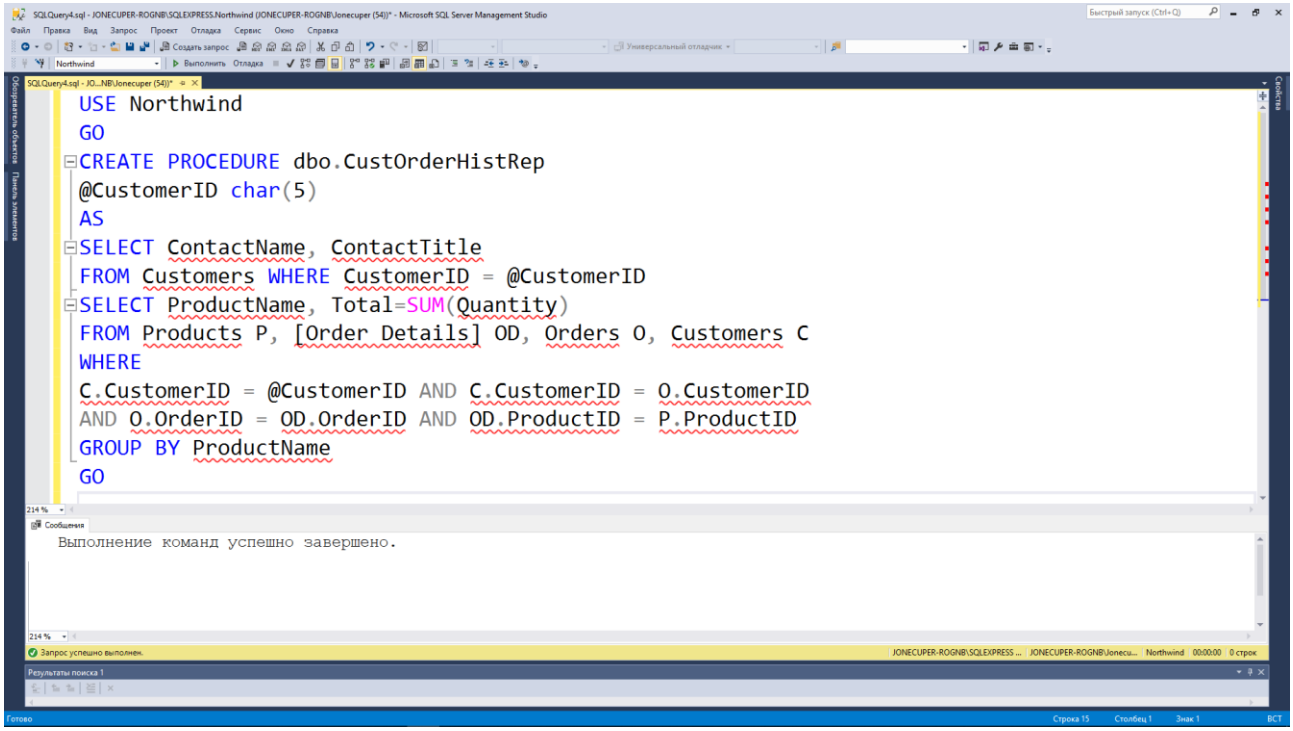


Рисунок 8 – Создание хранимой процедуры <CustOrderHistRep>

7.6 Просмотр хранимых процедур в <Query Analyzer>

1. Откройте <Object Browser> в <Query Analyzer> и выберите базу данных <Northwind>.

2. Раскройте узел <Stored Procedures>.

Появляются все хранимые процедуры, созданные в базе данных <Northwind>.

1. Раскройте хранимую процедуру «dbo.CustOrderHistRep».

Обратите внимание, что эта хранимая процедура принадлежит <dbo> и что в «dbo.CustOrderHistRep» появляются два узла: <Parameters> и <Dependencies>.

2. Раскройте узел <Parameters>.

Обратите внимание на наличие у этой хранимой процедуры двух параметров: созданного вами <@CustomerID> и встроенного параметра для хранения кода возврата <@RETURN_VALUE>.

3. Раскройте узел <Dependencies>.

Обратите внимание, что хранимая процедура зависит от четырех объектов (таблиц «Orders», «Products», «Order Details» и «Customers» из базы данных <Northwind>). От самой хранимой процедуры не зависит ни один объект (рисунок 9).

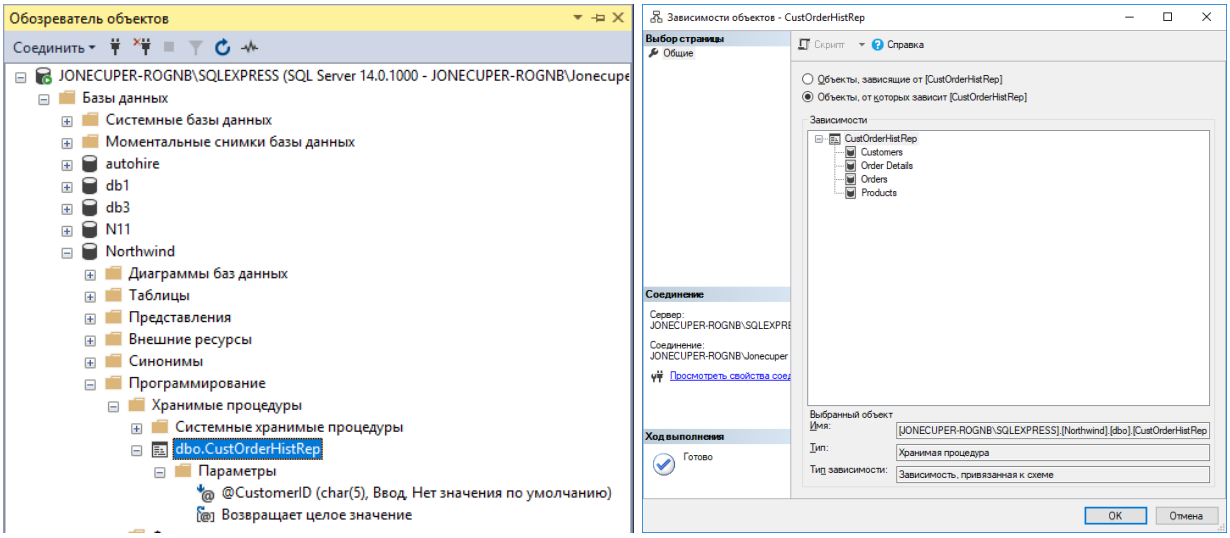


Рисунок 9 – Отображение содержимого узлов <Parameters> и <Dependencies> хранимой процедуры «CustOrderHistRep»

1. Переключитесь на панель <Editor> окна <Query>.

2. Введите в пустой строке в нижней части панели <Editor> и исполните следующую команду:

sp_depends custorderhistrep

Хранимая процедура «sp_depends» проверяет процедуру «CustOrderHistRep», чтобы определить ее зависимости.

Обратите внимание на наличие в поле <Name> повторяющихся элементов. Однако все записи с повторяющимися элементами отличаются значениями других полей (рисунок 10).

```
sp_depends custorderhistrep
```

	name	type	updated	selected	column
1	dbo.Customers	user table	no	no	ContactTitle
2	dbo.Order Details	user table	no	no	Quantity
3	dbo.Customers	user table	no	no	CustomerID
4	dbo.Customers	user table	no	no	ContactName
5	dbo.Products	user table	no	no	ProductName
6	dbo.Order Details	user table	no	no	OrderID
7	dbo.Order Details	user table	no	no	ProductID
8	dbo.Orders	user table	no	no	OrderID
9	dbo.Orders	user table	no	no	CustomerID
10	dbo.Products	user table	no	no	ProductID

Рисунок 10 – Проверка процедуры «CustOrderHistRep» для определения ее зависимости

3. Не закрывайте <Query Analyzer>, он потребуется для выполнения следующего задания.

7.7 Исполнение хранимой процедуры

1. В нижней части панели <Editor> введите в пустой строке и исполните следующую команду:

EXEC [Northwind]. [dbo]. [custorderhistrep]

@CustomerID = 'thecr'

Для запуска хранимой процедуры «CustOrderHistRep» использована сокращенная версия ключевого слова <EXECUTE> – <EXEC>. Обратите внимание на использование полного имени. Это не обязательно, но в данном случае такой прием позволяет запустить процедуру, не делая активной базу данных <Northwind>.

Возвращаются два результирующих набора. Первый из них (имя контактного лица и заголовок контактной информа ции) выводится в верхней части панели <Results>. Второй результирующий набор (название и количество товара) отображается в нижней части панели <Results> (рисунок 11).

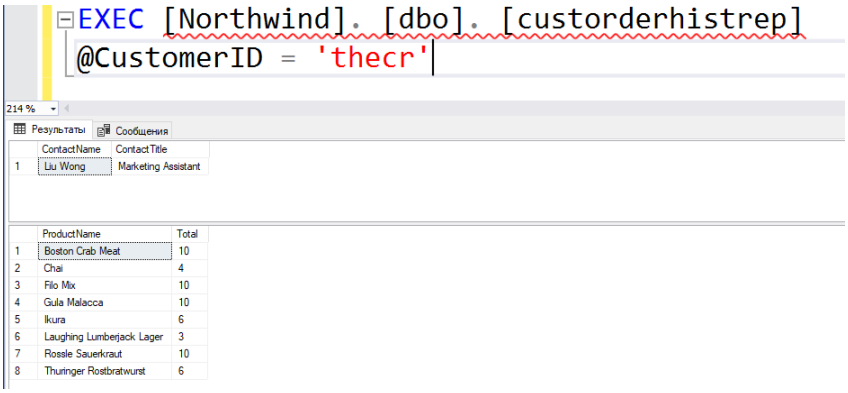


Рисунок 11 – Вызов процедуры «CustOrderHistRep» с указанием значения параметра

2. В предыдущем примере для запуска хранимой процедуры было задано ключевое слово <EXEC>. Обязательно ли его использовать?

3. Найдите в <Object Browser> хранимую процедуру «dbo.CustOrderHistRep» и щелкните по ней правой кнопкой мыши.

4. Выберите из контекстного меню <Script Object to New Windows As> и щелкните <Execute>. <Query Analyzer> загружает в редактор <Query Window> новую страницу, появляется оператор <EXECUTE> для процедуры «dbo.CustOrderHistRep». Обратите внимание на объявление двух переменных: <@RC> и <@CustomerID>. Первая переменная используется для хранения любых кодов возврата, являющихся частью процедуры, а вторая хранит значение входного параметра <@CustomerID>. Обратите внимание, что в операторе <EXEC> переменная <@RC> соответствует хранимой процедуре.

5. На панели <Editor> щелкните мышью в конце оператора <EXEC>, чтобы после слова <@CustomerID> появился мигающий курсор. Допишите к концу строки следующее: = 'thecr'. Теперь оператор <EXEC> должен принять следующий вид:

```
EXEC @RC = [Northwind].[dbo].[CustOrderHistRep] @CustomerID = 'thecr'
```

6. Исполните запрос.

7. Закройте новую страницу, созданную в окне <Query>, но не закрывайте <Query Analyzer> и исходную страницу, которая появилась на панели <Editor>.

8. Выводится сообщение <Query Analyzer> с запросом на сохранение внесенных изменений.

9. Щелкните <No>.

На панели <Editor> появляется исходная страница.

7.8 Модификация и удаление хранимой процедуры. Модификация хранимой процедуры

1. На панели <Editor> в окне <Query> введите следующий код:

```
USE Northwind
GO
CREATE PROCEDURE dbo.CustOrderHistRep
@CustomerID char(5)
AS
SELECT ContactName, ContactTitle
FROM Customers WHERE CustomerID = @CustomerID
SELECT ProductName, Total=SUM(Quantity)
FROM Products P, [Order Details] OD, Orders O, Customers C
WHERE
C.CustomerID = @CustomerID AND C.CustomerID = O.CustomerID
AND O.OrderID = OD.OrderID AND OD.ProductID = P.ProductID
GROUP BY ProductName
GO
```

2. Замените ключевое слово <CREATE> на <ALTER>.

При изменении текста процедуры ключевое слово <ALTER> позволяет изменять хранимую процедуру без потери любых ее свойств (рисунок 12).

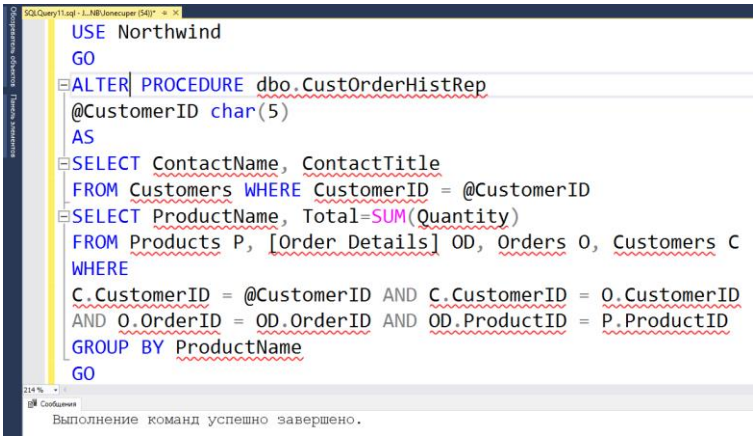


Рисунок 12 – Изменение процедуры «CustOrderHistRep»

3. Необходимо отсортировать число заказов по убыванию. Для этого ниже оператора <GROUP BY ProductName> введите следующий оператор:

ORDER BY Total DESC

4. Также решено переместить соединения таблиц из конструкции <WHERE> в конструкцию <FROM>. В окончательном виде программа выглядит примерно так:

```
USE Northwind
GO
ALTER PROCEDURE dbo.CustOrderHistRep
@CustomerID char(5)
AS
SELECT ContactName, ContactTitle
FROM Customers WHERE CustomerID = @CustomerID
SELECT ProductName, Total=SUM(Quantity)
FROM Products P INNER JOIN [Order Details] OD
ON P.ProductID = OD.ProductID
JOIN Orders O
ON OD.OrderID = O.OrderID
JOIN Customers C ON O. CustomerID = C.CustomerID
WHERE C.CustomerID = @CustomerID
GROUP BY ProductName
ORDER BY Total DESC
GO
```

5. Исполните запрос (рисунок 13).

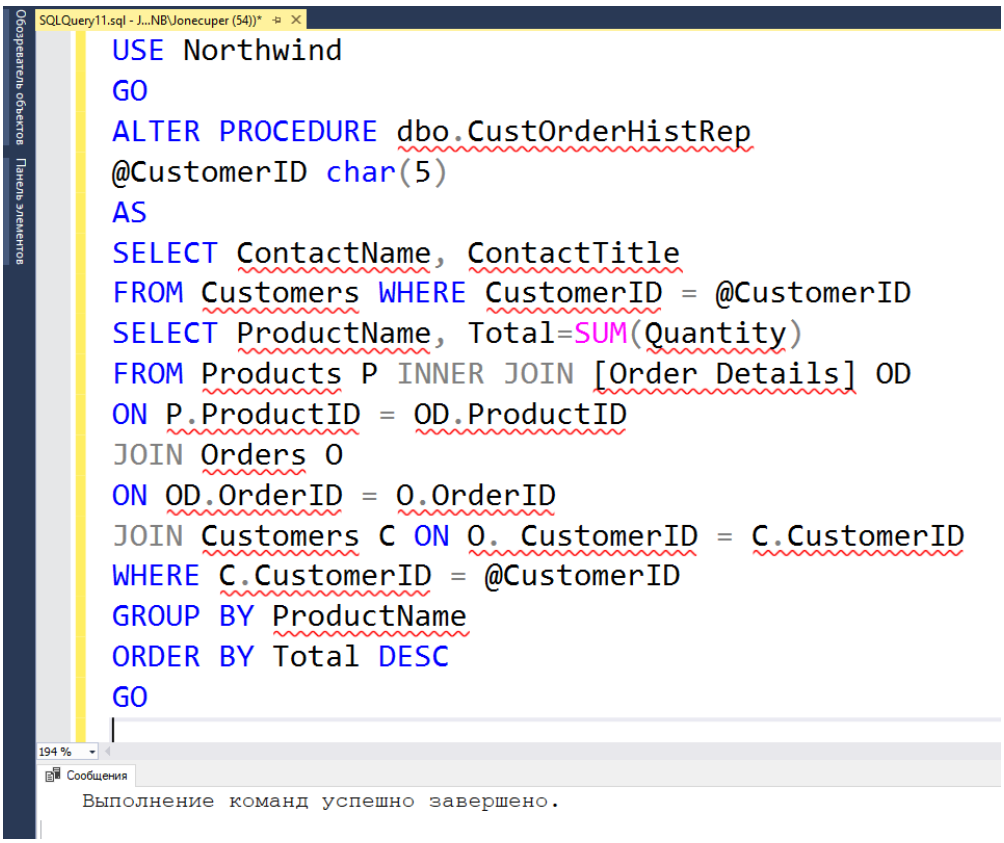


Рисунок 13 – Внесение изменений в процедуру «CustOrderHistRep»

6. Чтобы убедиться, что нужные изменения внесены, перейдите в нижнюю часть панели <Editor> нажмите <ENTER>, затем введите и исполните следующий оператор:

sp_helptext custorderhistrep

На вкладке <Grids> панели <Results> появляется текст хранимой процедуры (рисунок 14).

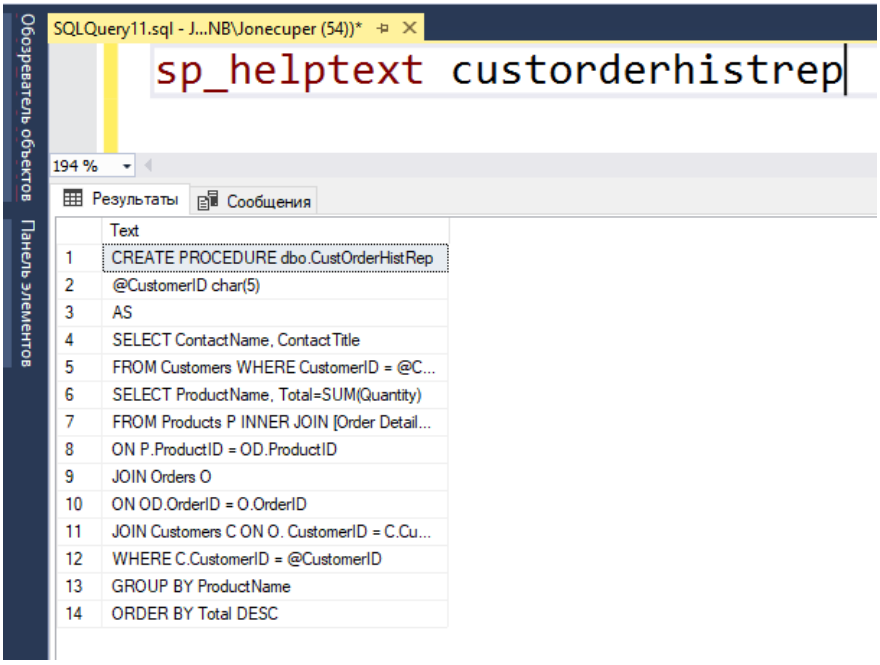


Рисунок 14 – Просмотр текста хранимой процедуры «CustOrderHistRep»

7. Оставьте <Query Analyzer> открытым.

7.9 Удаление хранимой процедуры

1. Перейдите в нижнюю часть панели <Editor> нажмите <ENTER>, затем введите и исполните следующий оператор:

DROP PROCEDURE dbo.custorderhistrep

Хранимая процедура удаляется из базы данных <Northwind>.

2. Проверьте с помощью <Object Browser> в <Query Analyzer> или <Enterprise Manager>, удалена ли хранимая процедура.

3. Закройте <Query Analyzer>.

Выводится запрос на сохранение сценария.

4. Если вы хотите сохранить сценарий, нажмите <Yes>, в противном случае – <No>.

7.10 Контрольное задание

Создайте не менее трех хранимых процедур в базе данных согласно выбранному вами варианту.

7.11 Контрольные вопросы

1. Что такое хранимая процедура?
2. Чем могут обладать хранимые процедуры?
3. Опишите методы просмотра содержимого хранимой процедуры.
4. Как создать хранимую процедуру?
5. Каким образом можно модифицировать хранимую процедуру?
6. Как осуществить просмотр хранимой процедуры?
7. Как удалить хранимую процедуру?

8 Лабораторная работа № 8. Триггеры

Цель работы: используя операторы T-SQL, научиться создавать, просматривать, модифицировать и удалять триггеры.

Используемое программное обеспечение: Microsoft SQL Server 2017.

Триггеры – это специальный вид хранимой процедуры, которую *SQL Server* вызывает при выполнении операций модификации соответствующих таблиц. Триггер автоматически активизируется при выполнении операции, с которой он связан. Триггеры связываются с одной или несколькими операциями модификации над одной таблицей.

8.1 Создание триггера

CREATE TRIGGER <имя_триггера>

ON <имя_таблицы>

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

[WITH ENCRIPTING]

AS

SQL-операторы (тело триггера)

Ограничения:

– Нельзя использовать в теле триггера операции создания объектов базы данных (новую базу, таблицу, индекс, хранимую процедуру, триггер, представление).

– Нельзя использовать в триггере команду удаления объектов <DROP>.

– Нельзя использовать в теле триггера команды изменения базовых объектов <ALTER>.

– Нельзя изменять права доступа к объектам базы данных.

– Триггер не может возвращать никаких значений.

8.2 Пример создания триггера

Создадим триггер, который срабатывает при удалении экземпляра некоторой книги, например, в случае утери этой книги читателем. При этом триггер проверяет: если остался еще хоть один экземпляр данной книги в библиотеке, и если это был последний экземпляр книги в библиотеке – удаляет описание данной книги из предметного каталога.

```
CREATE TRIGGER DEL_EXEMP ON dbo.examplar
FOR DELETE
AS
DECLARE @Ntek int
DECLARE @DEL_EX varchar(12)
Begin
Select @DEL_EX = ISBN From deleted
Exec @Ntek=COUNT_EX @DEL_EX
If @Ntek = 0 delete from BOOKS WHERE BOOKS.ISBN=@DEL_EX
END
```

8.3 Создание простых триггеров для таблицы «Товар» из базы данных <Northwind>

1. Откройте <Query Analyzer> и подключитесь к локальному серверу.
2. На панели Editor в окне <Query> введите и исполните следующий код:

```
USE Northwind
GO
CREATE TRIGGER dbo.insertindicator
ON dbo.Товар
AFTER INSERT
AS
PRINT 'The insert trigger fired.'
```

Оператор <CREATE TRIGGER> создает триггер «Insertindicator» и привязывает его к таблице «Товар» в базе данных <Northwind>. Этот триггер срабатывает при добавлении данных в таблицу «Товар» и выводит на панели <Grids> вкладки <Results> сообщение.

3. На панели <Editor> в окне <Query> введите и исполните следующий код:

```
CREATE TRIGGER dbo.updateindicator
ON dbo.Товар
AFTER UPDATE
AS
PRINT 'The update trigger fired.'
GO
CREATE TRIGGER dbo.deleteindicator
ON dbo.Товар
AFTER DELETE
AS
IF @@ROWCOUNT <> 0
PRINT 'The delete trigger fired.'
```

Оператор <CREATE TRIGGER> создает триггеры «UpdateIndicator» и «DeleteIndicator» и привязывает их к таблице «Товар» в базе данных <Northwind>. Если происходит событие <UPDATE> или <DELETE>, эти триггеры выводят на вкладке <Grids> панели <Results> сообщение. Обратите внимание, что триггер «DeleteIndicator» проверяет значение функции <@@ROWCOUNT>. Если удалена одна или несколько строк, выводится сообщение.

8.4 Проверка триггеров таблицы «Товар»

1. На панели <Editor> в окне <Query> введите и исполните следующий код:

```
INSERT INTO Товар (Код_товара, Код_поставщика, Название_товара, Количество_товара, Цена_товара)
```

VALUES (3, 2, 'Фотоаппарат Canon', 5, 2000)

В таблицу «Товар» добавляется запись, триггер «InsertIndicator» срабатывает и выводит сообщение на вкладке <Messages> панели <Results>.

2. На панели <Editor> в окне <Query> введите и исполните следующий код:

UPDATE Товар

SET Товар.Название_товара = 'Фотоаппарат_LG'

WHERE Товар.Название_товара = 'Фотоаппарат Canon'

В таблице «Товар» обновляется запись, триггер «UpdateIndicator» срабатывает и выводит сообщение на вкладке <Messages> панели <Results>.

3. На панели <Editor> в окне <Query> введите и исполните следующий код:

DELETE Товар where Название_товара = 'Фотоаппарат_LG'

Из таблицы «Товар» удаляется запись, триггер «DeleteIndicator» срабатывает и выводит сообщение на вкладке <Messages> панели <Results>.

8.4 Переименование, модификация и просмотр триггера

1. На панели <Editor> в окне <Query> введите и исполните следующую команду для запуска хранимой процедуры:

sp_rename @objname=insertindicator, @newname=insupdcontrol

Системная хранимая процедура «sp_rename» переименовывает триггер «InsertIndicator» в «InsUpdControl».

2. Исполните запрос.

На вкладке <Messages> панели <Results> выводится предупреждение о том, что переименование объекта может повлечь за собой сбои в работе сценариев и хранимых процедур. Это сообщение также свидетельствует об успешном завершении операции переименования.

3. Введите на панели <Editor> в окне <Query> и исполните следующую команду для запуска хранимой процедуры:

sp_helptrigger @tabname = Товар

Системная хранимая процедура «sp_helptrigger» выводит список триггеров, применяемых к таблице «Товар».

4. На панели <Editor> в окне <Query> введите и исполните следующий код:

```
ALTER TRIGGER dbo.insupdcontrol
ON dbo.Товар
INSTEAD OF INSERT, UPDATE
AS
PRINT 'Inserts and updates are not allowed at this time.'
```

В результате модификации переименованный триггер преобразуется в триггер «INSTEAD OF», который предотвращает все операции добавления и обновления в таблице «Товар». С помощью триггеров этого типа можно временно запретить внесение изменений в таблицу. Для включения и выключения триггеров используют оператор <ALTER TABLE>.

5. Введите на панели <Editor> в окне <Query> и исполните следующий код:

```
SET NOCOUNT ON
INSERT INTO Товар (Код_товара, Код_поставщика, Название_товара, Количество_товара, Цена_товара)
VALUES (4, 3, 'Фотоаппарат Samsung', 3, 5000)
SET NOCOUNT OFF
```

Триггер «INSTEAD OF» срабатывает и выводит сообщение, в котором говорится, что в данный момент обновление запрещено. Активирован параметр <NOCOUNT>, поэтому на вкладке <Messages> панели <Results> не выводится сообщение с числом строк, на которое повлияло исполнение запроса.

6. Чтобы убедиться, что в таблицу «Товар» не добавлено ни одной записи, исполните для этой таблицы оператор <SELECT>.

7. На панели <Editor> в окне <Query> введите и исполните следующую команду для запуска хранимой процедуры:

```
sp_helptext @objname=insupdcontrol
```

Хранимая процедура выводит содержимое триггера «InsUpdControl».

8.5 Отключение и удаление триггера

1. На панели <Editor> в окне <Query> введите и исполните следующий код:

ALTER TABLE Товар DISABLE TRIGGER insupdcontrol

Оператор <ALTER TABLE> отключает триггер «InsUpdControl» в таблице «Товар».

2. На панели <Editor> в окне <Query> введите и исполните следующий код:

INSERT INTO Товар (Код_товара, Код_поставщика, Название_товара, Количество_товара, Цена_товара)

SELECT * FROM Товар where Название_товара = 'Фотоаппарат Samsung'

Запись успешно добавляется в таблицу «Товар», добавленная запись выводится на вкладке <Grids> панели <Results>.

3. На панели <Editor> в окне <Query> введите и исполните следующий код:

DROP TRIGGER insupdcontrol, updateindicator, deleteindicator

DELETE Товар where Название_товара = 'Фотоаппарат Samsung'

Оператор <DROP TRIGGER> удаляет все три триггера, привязанных к таблице «Товар». Оператор <DELETE> удаляет запись, ранее добавленную к таблице «Товар».

8.6 Контрольное задание

Создать не менее трех триггеров в базе данных согласно выбранному вами варианту.

8.7 Контрольные вопросы

1. Что такое триггер?
2. Каково назначение триггеров?
3. В чем отличие триггера от хранимой процедуры?
4. Как создать триггер?

9 Лабораторная работа № 9. Представления

Цель работы: используя язык T-SQL, научиться создавать, модифицировать и удалять представления.

Используемое программное обеспечение: Microsoft SQL Server 2017.

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

В отличие от обычных таблиц реляционной базы 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.

9.1 Представления и подзапросы

Представления могут использовать подзапросы, включая соотнесенные подзапросы. Предположим, ваша компания предусматривает премию для тех продавцов, которые имеют заказчика с самой высокой суммой заказа для любой указанной даты. Вы можете проследить эту информацию с помощью представления:

```
CREATE VIEW Elitesalesforce
AS SELECT b.odate, a.snum, a.sname
FROM Salespeople a, Orders b
WHERE a.snum = b.snum AND b.amt = (SELECT MAX (amt)
FROM Orders c WHERE c.odate = b.odate)
```

Если, с другой стороны, премия будет назначаться только продавцу, который имел самую высокую сумму заказа за последние десять лет, вам

необходимо будет проследить их в другом представлении, основанном на первом:

```
CREATE VIEW Bonus
AS SELECT DISTINCT snum, sname
FROM Elitesalesforce a
WHERE 10 <= (SELECT COUNT (*)
FROM Elitesalesforce b
WHERE a.snum = b.snum)
```

Извлечение из этой таблицы продавца, который будет получать премию, выполняется простым запросом:

```
SELECT *
FROM Bonus
```

Если команды модификации могут выполняться в представлении, представление как сообщалось, будет *модифицируемым*; в противном случае оно предназначено только для чтения при запросе. Не противореча этой терминологии, мы будем использовать выражение «*модифицируемое представление*» (updating a view), что означает возможность выполнения в представлении любой из трех команд модификации DML (<INSERT>, <UPDATE> и <DELETE>), которые могут изменять значения.

9.2 Модифицируемое представление

Модифицируемое представление – это представление, в котором команда модификации может выполниться, чтобы изменить одну и только одну строку основной таблицы в каждый момент времени, не воздействуя на любые другие строки любой таблицы.

Критерии определения модифицируемости представления:

1. Оно должно выводиться в одну и только в одну базовую таблицу.

2. Оно должно содержать первичный ключ этой таблицы (это технически не предписывается стандартом <ANSI>, но было бы неплохо придерживаться этого).

3. Оно не должно иметь никаких полей, которые бы являлись агрегатными функциями.

4. Оно не должно содержать <DISTINCT> в своем определении.

5. Оно не должно использовать <GROUP BY> или <HAVING> в своем определении.

6. Оно не должно использовать подзапросы (это – <ANSI> ограничение, которое не предписано для некоторых реализаций).

7. Оно может быть использовано в другом представлении, но это представление должно также быть модифицируемыми.

8. Оно не должно использовать константы, строки, или выражения значений (например, «comm*100») среди выбранных полей вывода.

9. Для <INSERT>, оно должно содержать любые поля основной таблицы, которые имеют ограничение <NOT NULL>, если другое ограничение по умолчанию не определено.

9.3 Проверка значений, помещаемых в представление

Вы можете вводить значения, которые «*проглатываются*» (*swallowed*) в базовой таблице. Рассмотрим модифицируемое представление:

```
CREATE VIEW Highratings
AS SELECT cnum, rating
FROM Customers WHERE rating = 300
```

Оно просто ограничивает ваш доступ к определенным строкам и столбцам в таблице. Предположим, что вы вставляете (<INSERT>) следующую строку:

```
INSERT INTO Highratings VALUES (2018, 200)
```

Это допустимая команда <INSERT> в этом представлении. Строка будет вставлена, с помощью представления «Highratings», в таблицу заказчиков.

Однако когда она появится там, она исчезнет из представления, поскольку значение оценки не равно 300.

Значение 200 может быть просто напечатано, но теперь строка находится уже в таблице заказчиков, где вы не можете даже увидеть ее. Пользователь не сможет понять, почему введя строку, он не может ее увидеть, и будет неспособен при этом удалить ее.

Вы можете быть гарантированы от модификаций такого типа с помощью включения <WITH CHECK OPTION> (с опцией проверки) в определение представления. Мы можем использовать <WITH CHECK OPTION> в определении представления «Highratmgs».

```
CREATE VIEW Highratings
AS SELECT cnum, rating
FROM Customers
WHERE rating = 300
WITH CHECK OPTION
```

Вышеупомянутая вставка будет отклонена.

<WITH CHECK OPTION> производит действие *все_или_ничего* (*all-or-nothing*). Вы помещаете его в определение представления, а не в команду DML, так что или все команды модификации в представлении будут проверяться, или ни одна не будет проверена. Обычно вы хотите использовать опцию проверки, используя ее в определении представления, что может быть удобно.

В общем, вы должны использовать эту опцию, если у вас нет причины разрешать представлению помещать в таблицу значения, которые он сам не может содержать.

9.4 Предикаты и исключенные поля

Похожая проблема включает в себя вставку строк в представление с предикатом, базирующемся не на всех полях таблицы. Например, может показаться разумным создать представление «Londonstaff»:

CREATE VIEW Londonstaff

AS SELECT snum, sname, comm

FROM Salespeople WHERE city = 'London'

В конце концов, зачем включать значение «city», если все значения «city» будут одинаковыми.

Так как мы не можем указать значение «city» как значение по умолчанию, этим значением, вероятно, будет <NULL>, и оно будет введено в поле <city> (<NULL> используется, если другое значение по умолчанию не было определено). Так как в этом случае поле <city> не будет равняться значению «London», вставляемая строка будет исключена из представления. Это будет верным для любой строки, которую вы попробуете вставить в просмотр «Londonstaff». Все они должны быть введены с помощью представления «Londonstaff» в таблицу продавцов, и затем исключены из самого представления (если определением по умолчанию был не London, то это особый случай). Пользователь не сможет вводить строки в это представление, хотя все еще неизвестно, может ли он вводить строки в базовую таблицу. Даже если мы добавим <WITH CHECK OPTION> в определение представления

CREATE VIEW Londonstate

AS SELECT snum, sname, comm

FROM Salespeople

WHERE city = 'London'

WITH CHECK OPTION

проблема не обязательно будет решена. В результате этого мы получим представление, которое мы могли бы модифицировать или из которого мы могли бы удалять, но не вставлять в него. В некоторых случаях, это может быть хорошо; хотя, возможно, нет смысла пользователям, имеющим доступ к этому представлению, иметь возможность добавлять строки.

Даже если это не всегда может обеспечить Вас полезной информацией, полезно включать в ваше представление все поля, на которые имеется ссылка в предикате. Если вы не хотите видеть эти поля в вашем выводе, вы всегда сможете

исключить их из запроса в представлении, в противоположность запросу внутри представления. Другими словами, вы могли бы определить представление «Londonstaff»> подобно этому:

```
CREATE VIEW Londonstaff
AS SELECT *
FROM Salespeople
WHERE city = 'London'
WITH CHECK OPTION
```

Эта команда заполнит представление одинаковыми значениями в поле <city>, которые вы можете просто исключить из вывода с помощью запроса, в котором указаны только те поля, которые вы хотите видеть:

```
SELECT snum, sname, comm
FROM Londonstaff
```

9.5 Проверка представлений, которые базируются на других представлениях

Еще одно надо упомянуть относительно предложения <WITH CHECK OPTION> в <ANSI>: оно не делает *каскадного* изменения. Оно применяется только в представлениях, в которых определено, но не в представлениях, основанных на этом представлении. Например, в предыдущем примере:

```
CREATE VIEW Highratings
AS SELECT cnum, rating
FROM Customers
WHERE rating = 300
WITH CHECK OPTION
```

Попытка вставить или модифицировать значение оценки не равное 300 потерпит неудачу. Однако мы можем создать второе представление (с идентичным содержанием) основанное на первом:

```
CREATE VIEW Myratings
```

AS SELECT *

FROM Highratings

Теперь мы можем модифицировать оценки, не равные 300:

UPDATE Myratings

SET rating = 200

WHERE cnum = 2004

Эта команда, выполняемая так, как если бы она выполнялась как первое представление, будет допустима. Предложение <WITH CHECK OPTION> просто гарантирует, что любая модификация в представлении произведет значения, которые удовлетворяют предикату этого представления. Модификация других представлений, базирующихся на первом текущем, является все еще допустимой, если эти представления не защищены предложениями <WITH CHECK OPTION> внутри этих представлений. Даже если такие предложения установлены, они проверяют только те предикаты представлений, в которых они содержатся. Так, например, даже если представление «Myratings» создавалось следующим образом

CREATE VIEW Myratings

AS SELECT *

FROM Highratings

WITH CHECK OPTION,

проблема не будет решена. Предложение <WITH CHECK OPTION> будет исследовать только предикат представления «Myratings». Пока у «Myratings», фактически, не имеется никакого предиката, <WITH CHECK OPTION> ничего не будет делать. Если используется предикат, то он будет проверяться всякий раз, когда представление «Myratings» будет модифицироваться, но предикат «Highratings» все равно будет проигнорирован.

9.6 Создание представления <Pokupka> в базе данных <Northwind>

1. Откройте <Query Analyzer> и подключитесь к локальному серверу.

2. На панели <Editor> в окне <Query> введите и исполните следующий код Transact-SQL:

```
USE Northwind
GO
CREATE VIEW Pokupka
AS SELECT ФИО_клиента, Количество_проданного_товара, Дата
FROM Клиент, Продажа WHERE Клиент.Код_клиента = Продажа.Код_клиента
```

Этот оператор создает в базе данных <Northwind> представление «Pokupka». Оператор <CREATE VIEW> содержит запрос <SELECT>, который соединяет таблицы «Клиент» и «Продажа». В результирующий набор этого запроса войдут <ФИО_клиента>, <Количество_проданного_товара> и <Дата>.

3. Исполните оператор Transact-SQL. На вкладке <Messages> панели <Results> выводится сообщение об успешном завершении команды.

4. Если окно <Object Browser> закрыто, откройте его.

5. В дереве объектов в окне <Object Browser> найдите созданное вами новое представление. Обратите внимание, что узел «dbo.Pokupka» содержит вложенные узлы (<Columns> и <Indexes>).

6. Раскройте узел <Columns>.

Обратите внимание, что в этот узел входят три столбца, описанные в запросе <SELECT> из определения <CREATE VIEW>.

9.7 Модификация представления «Pokupka» из базы данных <Northwind>

1. На панели <Editor> в окне <Query> введите и исполните следующий код Transact-SQL:

USE Northwind

GO

ALTER VIEW Pokupka

AS SELECT ФИО_клиента, Название_товара,
Количество_проданного_товара, Дата

FROM Клиент, Продажа, Товар

WHERE Клиент.Код_клиента = Продажа.Код_клиента and
Товар.Код_товара = Продажа.Код_товара

2. Исполните оператор Transact-SQL. На вкладке <Messages> панели <Results> выводится сообщение об успешном завершении команды.

3. В дереве объектов в окне <Object Browser> найдите представление «Pokupka» и раскройте узел <Columns>. Обратите внимание на столбец <Название_товара>, добавленный к списку столбцов.

4. Закройте <Query Analyzer>.

9.8 Удаление представления «Pokupka» из базы данных <Northwind>

1. На панели <Editor> в окне <Query> введите и исполните следующий код Transact-SQL:

USE Northwind

GO

DROP VIEW Pokupka

Этот оператор удаляет представление «Pokupka» из базы данных <Northwind>.

2. Исполните оператор Transact-SQL.

На вкладке <Messages> панели <Results> выводится сообщение об успешном завершении команды.

3. В дереве объектов в окне <Object Browser> найдите узел <Views> для базы данных <Northwind>. Обратите внимание, что представление «Pokupka» больше не показано.

9.9 Контрольное задание

Создать не менее трех представлений в базе данных согласно выбранному вами варианту.

9.10 Контрольные вопросы

1. Сформулируйте определение представления.
2. Каково назначение представлений?
3. Как создать представление?
4. Что такое модифицируемое представление?
5. Каким образом модифицировать представление?

10 Лабораторная работа № 10. Индексы

Цель работы: используя язык T-SQL, научиться создавать индексы.

Используемое программное обеспечение: Microsoft SQL Server 2017.

Индекс – объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путем последовательного просмотра таблицы строка за строкой может занимать много времени. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска. Ускорение работы с использованием индексов достигается в первую очередь за счѐт того, что индекс имеет структуру, оптимизированную под поиск – например, сбалансированного дерева.

Некоторые системы управления базами данных расширяют возможности индексов введением возможности создания индексов по столбцам представлений или индексов по выражениям. Например, индекс может быть создан по выражению <upper(last_name)> и соответственно будет хранить ссылки, ключом к которым будет значение поля <last_name> в верхнем регистре. Кроме того, индексы могут быть объявлены как уникальные и как не уникальные. Уникальный индекс реализует ограничение целостности на таблице, исключая возможность вставки повторяющихся значений.

10.1 Архитектура индексов

Существует два типа индексов: *кластерные* и *некластерные*. При наличии кластерного индекса строки таблицы упорядочены по значению ключа этого индекса. Если в таблице нет кластерного индекса, таблица называется *кучей*. Некластерный индекс, созданный для такой таблицы, содержит только указатели на записи таблицы. Кластерный индекс может быть только одним для каждой

таблицы, но каждая таблица может иметь несколько различных некластерных индексов, каждый из которых определяет свой собственный порядок следования записей.

Индексы могут быть реализованы различными структурами. Наиболее частоупотребимы B*-деревья, B+-деревья, Вдеревья и хеши.

10.2 Последовательность столбцов в составном индексе

Последовательность, в которой столбцы представлены в составном индексе, достаточно важна. Дело в том, что получить набор данных по запросу, затрагивающему только первый из проиндексированных столбцов, можно. Однако в большинстве систем управления базами данных невозможно или неэффективно получение данных только по второму и далее проиндексированным столбцам (без ограничений на первый столбец).

Например, представим себе телефонный справочник, отсортированный вначале по городу, затем по фамилии, и затем по имени. Если вы знаете город, вы можете легко найти все телефоны этого города. Однако в таком справочнике будет весьма трудоемко найти все телефоны, записанные на определѐнную фамилию — для этого необходимо посмотреть в секцию каждого города и поискать там нужную фамилию. Некоторые системы управления базами данных выполняют эту работу, остальные же просто не используют такой индекс.

10.3 Производительность

Для оптимальной производительности запросов индексы обычно создаются на тех столбцах таблицы, которые часто используются в запросах. Для одной таблицы может быть создано несколько индексов. Однако увеличение числа индексов замедляет операции добавления, обновления, удаления строк таблицы, поскольку при этом приходится обновлять сами индексы. Кроме того, индексы занимают дополнительный объем памяти, поэтому перед созданием

индекса следует убедиться, что планируемый выигрыш в производительности запросов превысит дополнительную затрату ресурсов компьютера на сопровождение индекса.

10.4 Ограничения

Индексы полезны для многих приложений, однако на их использование накладываются ограничения. Возьмѐм такой запрос SQL:

SELECT first_name FROM people WHERE last_name = 'Франкенштейн'

Для выполнения такого запроса без индекса система управления базами данных должна проверить поле <last_name> в каждой строке таблицы (этот механизм известен как «полный перебор» или «полный скан таблицы», в плане может отображаться словом <NATURAL>). При использовании индекса сустема управления базами данных просто проходит по Вдереву, пока не найдѐт запись «Франкенштейн». Такой проход требует гораздо меньше ресурсов, чем полный перебор таблицы.

Теперь возьмѐм такой запрос:

SELECT email_address FROM customers WHERE email_address LIKE '%@yahoo.com'

Этот запрос должен нам найти всех клиентов, у которых email заканчивается на «@yahoo.com», однако даже если по столбцу <email_address> есть индекс, система управления базами данных всѐ равно будет использовать полный перебор таблицы. Это связано с тем, что индексы строятся в предположении, что слова / символы идут слева направо. Использование символа подстановки в начале условия поиска исключает для системы управления базами данных возможность использования поиска по B-дереву. Эта проблема может быть решена созданием дополнительного индекса по выражению <reserve(email_address)> и формированием запроса вида:

SELECT email_address FROM customers WHERE reserve(email_address) LIKE reserve('%@yahoo.com')

В данном случае символ подстановки окажется в самой правой позиции («moc.oohay@%»), что не исключает использование индекса по <reverse(email_address)>.

10.5 Редкий индекс

Редкий индекс в базах данных – это файл с последовательностью пар ключей и указателей. Каждый ключ в редком индексе, в отличие от плотного индекса, ассоциируется с определѐнным указателем на блок в сортированном файле данных. Идея использования индексов пришла, оттого что современные базы данных слишком массивны и не помещаются в основную память. Мы обычно делим данные на блоки и размещаем данные в памяти поблочно. Однако поиск записи в базе данных может занять много времени. С другой стороны, файл индексов или блок индексов намного меньше блока данных и может поместиться в буфере основной памяти, что увеличивает скорость поиска записи. Поскольку ключи отсортированы, можно воспользоваться бинарным поиском. В кластерных индексах с дублированными ключами редкий индекс указывает *на наименьший ключ* в каждом блоке.

10.6 Использование индекса и просмотр его свойств. Просмотр свойств индекса в базе данных <Northwind>

1. Откройте <Query Analyzer> и подключитесь к локальному серверу.

2. На панели <Editor> в окне <Query> введите и исполните следующий код:

```
USE Northwind
GO
sp_helpindex customers
```

На вкладке <Grids> панели <Results> появляются пять индексов (рисунок 1).

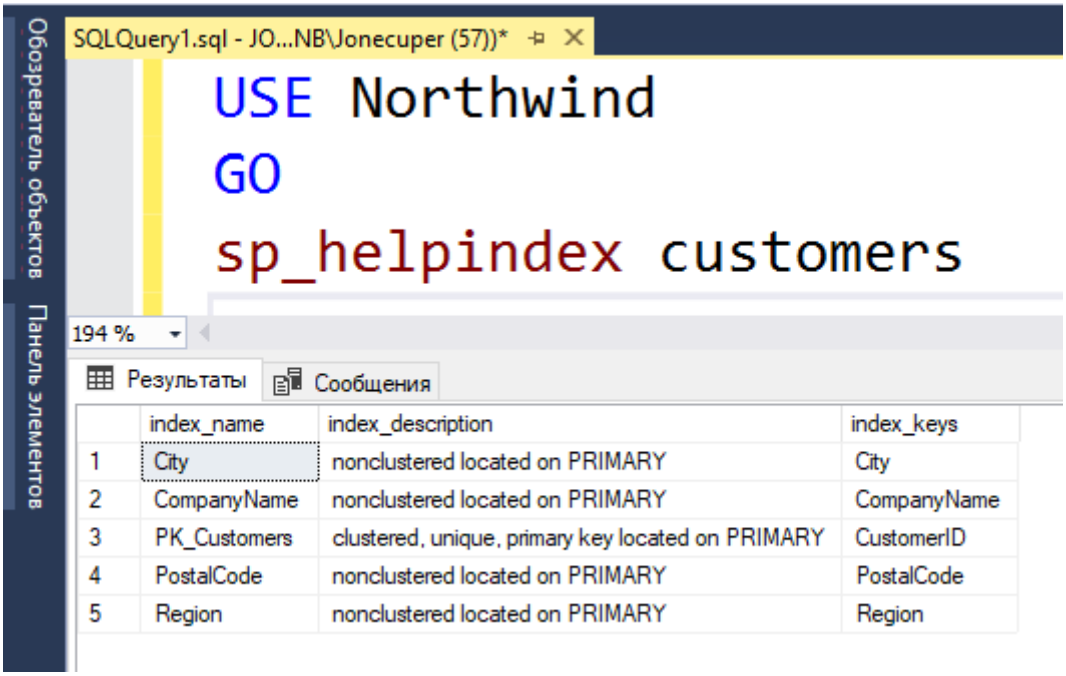


Рисунок 1 – Просмотр индексов таблицы «Customers»

3. Какой индекс отражает порядок сортировки таблицы «Customers»?

4. Есть ли в таблице «Customers» составной индекс?

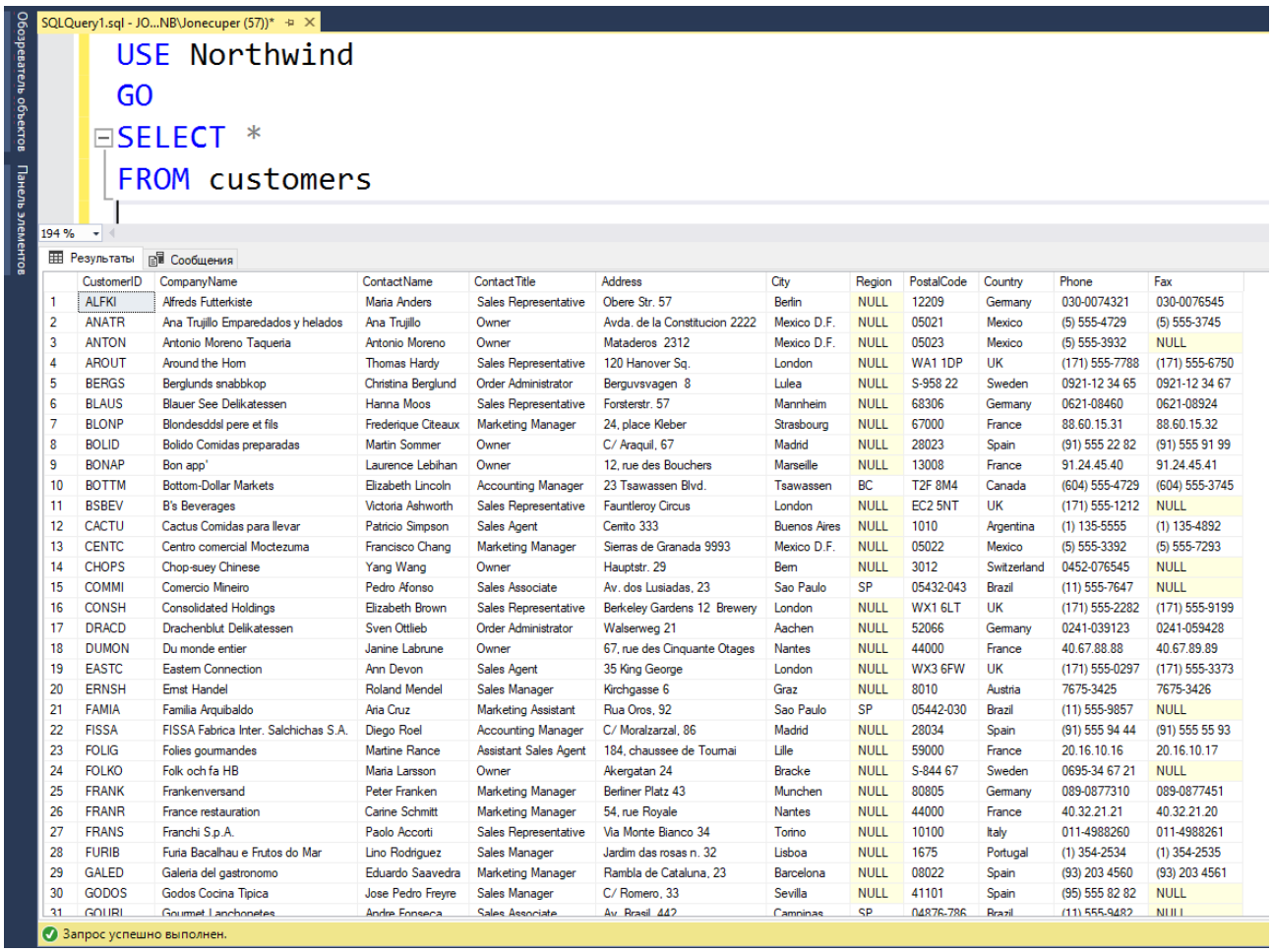
10.7 Исполнение запросов и просмотр плана исполнения

1. В <Query Analyzer> щелкните <Query>, а затем <Show Execution Plan>.

2. На панели <Editor> в окне <Query> введите и исполните следующий код:

```
USE Northwind
GO
SELECT *
FROM customers
```

На вкладке <Grids> панели <Results> появляется результирующий набор. Обратите внимание, что он упорядочен по значению <CustomerID> (рисунок 2).



	CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
1	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin	NULL	12209	Germany	030-0074321	030-0076545
2	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitucion 2222	Mexico D.F.	NULL	05021	Mexico	(5) 555-4729	(5) 555-3745
3	ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	Mexico D.F.	NULL	05023	Mexico	(5) 555-3932	NULL
4	AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London	NULL	WA1 1DP	UK	(171) 555-7788	(171) 555-6750
5	BERGS	Berglunds snabbkop	Christina Berglund	Order Administrator	Berguvsvagen 8	Lulea	NULL	S-958 22	Sweden	0921-12 34 65	0921-12 34 67
6	BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim	NULL	68306	Germany	0621-08460	0621-08924
7	BLONP	Blondesddsl pere et fils	Frederique Citeaux	Marketing Manager	24, place Kleber	Strasbourg	NULL	67000	France	88.60.15.31	88.60.15.32
8	BOLID	Bolido Comidas preparadas	Martin Sommer	Owner	C/ Araquil, 67	Madrid	NULL	28023	Spain	(91) 555 22 82	(91) 555 91 99
9	BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille	NULL	13008	France	91.24.45.40	91.24.45.41
10	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen	BC	T2F 8M4	Canada	(604) 555-4729	(604) 555-3745
11	BSBEV	B's Beverages	Victoria Ashworth	Sales Representative	Fauntleroy Circus	London	NULL	EC2 5NT	UK	(171) 555-1212	NULL
12	CACTU	Cactus Comidas para llevar	Patricio Simpson	Sales Agent	Cerrito 333	Buenos Aires	NULL	1010	Argentina	(1) 135-5555	(1) 135-4892
13	CENTC	Centro comercial Moctezuma	Francisco Chang	Marketing Manager	Sierras de Granada 9993	Mexico D.F.	NULL	05022	Mexico	(5) 555-3392	(5) 555-7293
14	CHOPS	Chop-suey Chinese	Yang Wang	Owner	Hauptstr. 29	Bern	NULL	3012	Switzerland	0452-076545	NULL
15	COMMI	Comercio Mineiro	Pedro Afonso	Sales Associate	Av. dos Lusiadas, 23	Sao Paulo	SP	05432-043	Brazil	(11) 555-7647	NULL
16	CONSH	Consolidated Holdings	Elizabeth Brown	Sales Representative	Berkeley Gardens 12 Brewery	London	NULL	WX1 6LT	UK	(171) 555-2282	(171) 555-9199
17	DRACD	Drachenblut Delikatessen	Sven Ottlieb	Order Administrator	Walserweg 21	Aachen	NULL	52066	Germany	0241-039123	0241-059428
18	DUMON	Du monde entier	Janine Labrune	Owner	67, rue des Cinquante Otages	Nantes	NULL	44000	France	40.67.88.88	40.67.89.89
19	EASTC	Eastern Connection	Ann Devon	Sales Agent	35 King George	London	NULL	WX3 6FW	UK	(171) 555-0297	(171) 555-3373
20	ERNSH	Ernst Handel	Roland Mendel	Sales Manager	Kirchgasse 6	Graz	NULL	8010	Austria	7675-3425	7675-3426
21	FAMIA	Familia Arquibaldo	Aria Cruz	Marketing Assistant	Rua Oros, 92	Sao Paulo	SP	05442-030	Brazil	(11) 555-9857	NULL
22	FISSA	FISSA Fabrica Inter. Salchichas S.A.	Diego Roel	Accounting Manager	C/ Moralzarzal, 86	Madrid	NULL	28034	Spain	(91) 555 94 44	(91) 555 55 93
23	FOLIG	Folies gourmandes	Martine Rance	Assistant Sales Agent	184, chaussee de Tournai	Lille	NULL	59000	France	20.16.10.16	20.16.10.17
24	FOLKO	Folk och fa HB	Maria Larsson	Owner	Akergatan 24	Bracke	NULL	S-844 67	Sweden	0695-34 67 21	NULL
25	FRANK	Frankenversand	Peter Franken	Marketing Manager	Berliner Platz 43	Munchen	NULL	80805	Germany	089-0877310	089-0877451
26	FRANR	France restauration	Carine Schmitt	Marketing Manager	54, rue Royale	Nantes	NULL	44000	France	40.32.21.21	40.32.21.20
27	FRANS	Franchi S.p.A.	Paolo Accorti	Sales Representative	Via Monte Bianco 34	Torino	NULL	10100	Italy	011-4988260	011-4988261
28	FURIB	Furia Bacalhau e Frutos do Mar	Lino Rodriguez	Sales Manager	Jardim das rosas n. 32	Lisboa	NULL	1675	Portugal	(1) 354-2534	(1) 354-2535
29	GALED	Galeria del gastronomo	Eduardo Saavedra	Marketing Manager	Rambla de Cataluna, 23	Barcelona	NULL	08022	Spain	(93) 203 4560	(93) 203 4561
30	GODOS	Godos Cocina Tipica	Jose Pedro Freyre	Sales Manager	C/ Romero, 33	Sevilla	NULL	41101	Spain	(95) 555 82 82	NULL
31	GOURL	Gourmet Lanchonetes	Andre Fonseca	Sales Associate	Av. Brasil, 442	Campinas	SP	04876-786	Brazil	(11) 555-9482	NULL

Рисунок 2 – Просмотр содержимого таблицы <Customers>

3. Щелкните вкладку <Execution Plan>.

План исполнения выводится на вкладке <Execution Plan> панели <Results>. Обратите внимание, что оптимизатор запросов использовал кластерный индекс «PK_Customers». На вкладке «Execution Plan» имя индекса «PK_Customers» обрезано до «PK_Cu...» (рисунок 3).

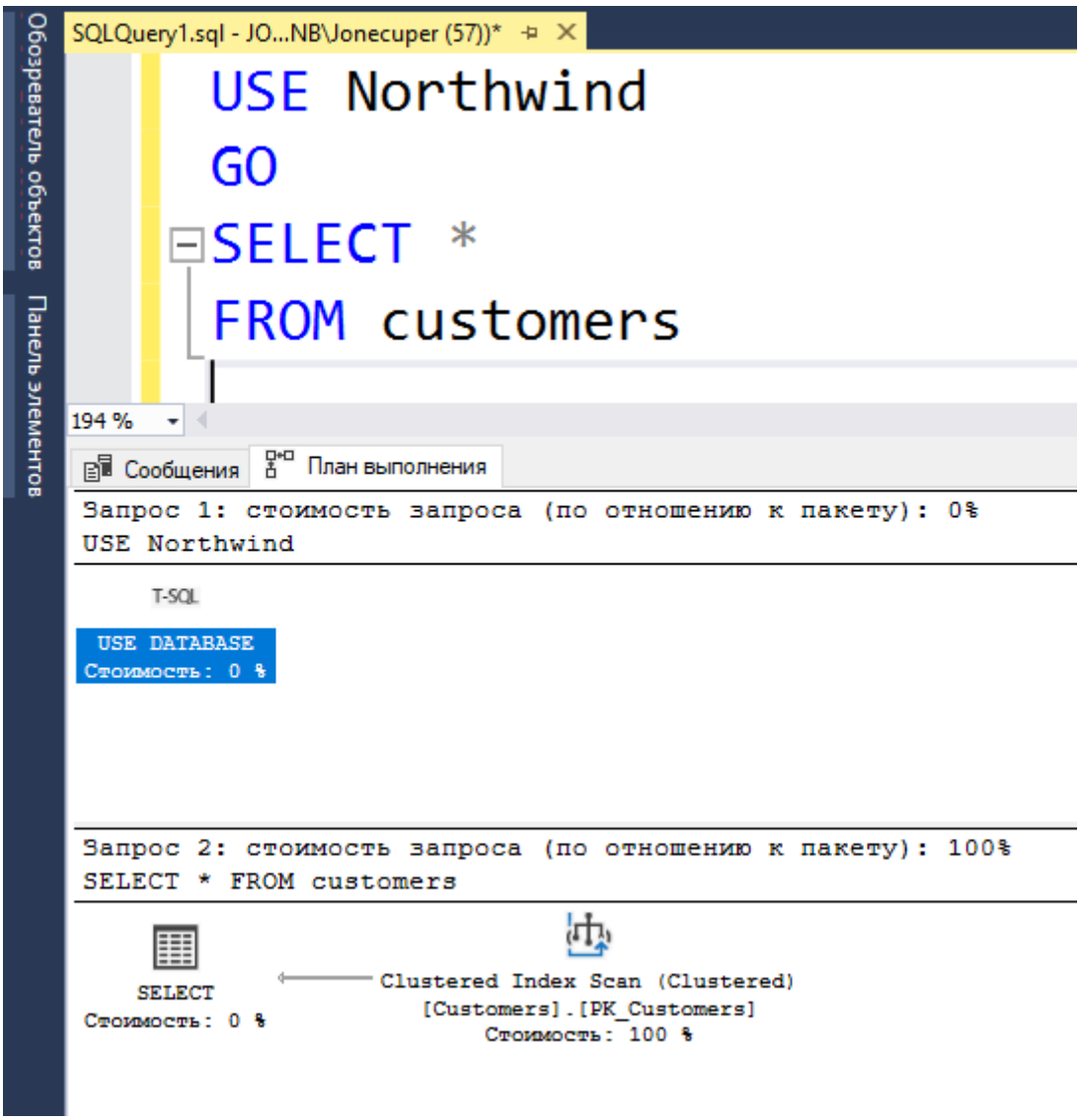


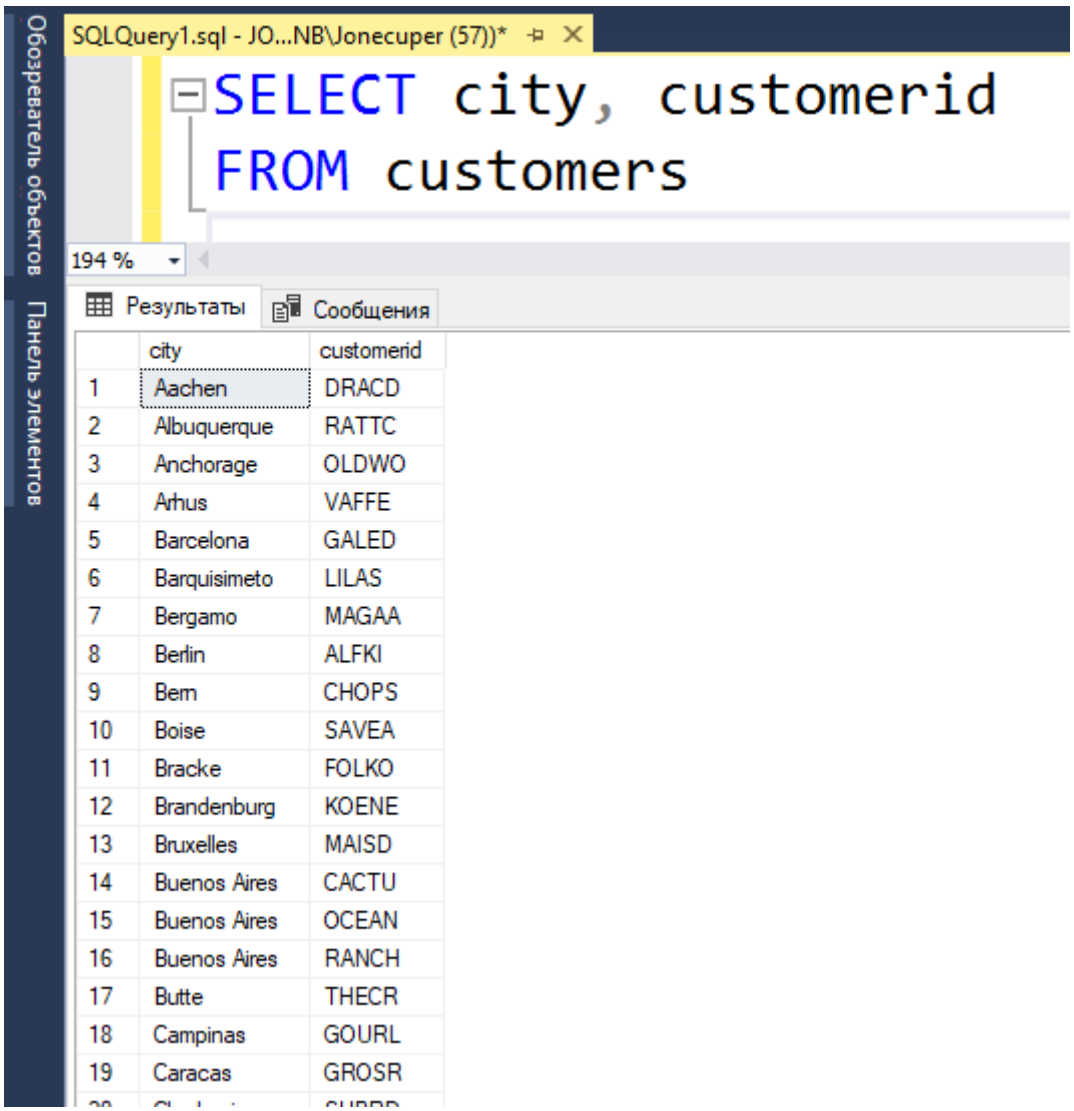
Рисунок 3 – Отображение плана исполнения просмотра содержимого таблицы «Customers»

4. На панели <Editor> в окне <Query> введите и исполните следующий код:

SELECT city, customerid

FROM customers

На вкладке <Grids> панели <Results> появляется результирующий набор. Обратите внимание, что он отсортирован по значению <City> (рисунок 4).



	city	customerid
1	Aachen	DRACD
2	Albuquerque	RATTC
3	Anchorage	OLDWO
4	Arhus	VAFFE
5	Barcelona	GALED
6	Barquisimeto	LILAS
7	Bergamo	MAGAA
8	Berlin	ALFKI
9	Bern	CHOPS
10	Boise	SAVEA
11	Bracke	FOLKO
12	Brandenburg	KOENE
13	Bruxelles	MAISD
14	Buenos Aires	CACTU
15	Buenos Aires	OCEAN
16	Buenos Aires	RANCH
17	Butte	THECR
18	Campinas	GOURL
19	Caracas	GROSR

Рисунок 4 – Просмотр содержимого полей <city> и <customerid> таблицы «Customers»

5. Щелкните вкладку <Execution Plan>.

План исполнения показывает, что оптимизатор запросов использовал некластерный индекс «City» (рисунок 5).

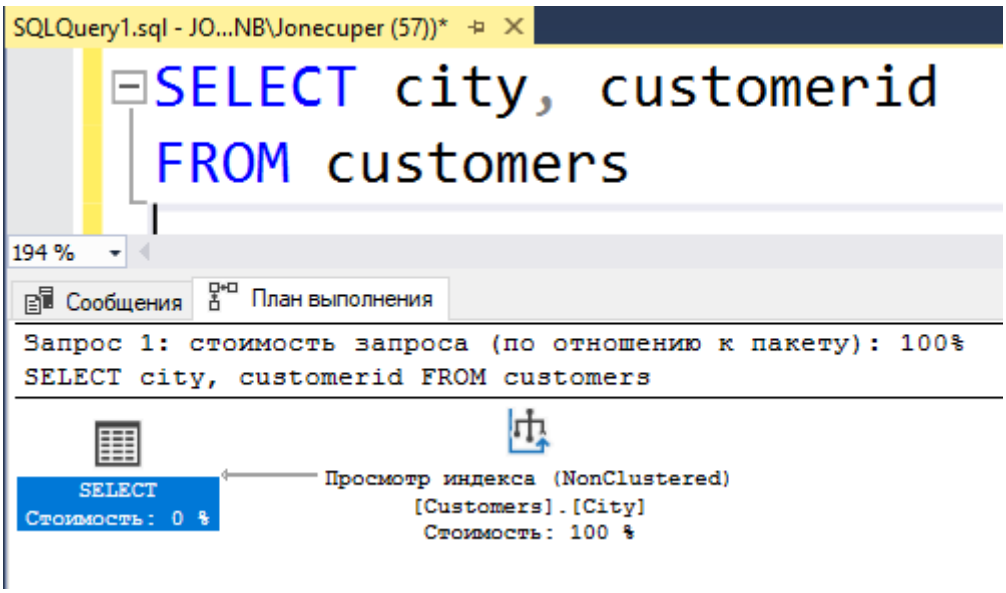


Рисунок 5 – Отображение плана исполнения просмотра содержимого полей <city> и <customerid> таблицы «Customers»

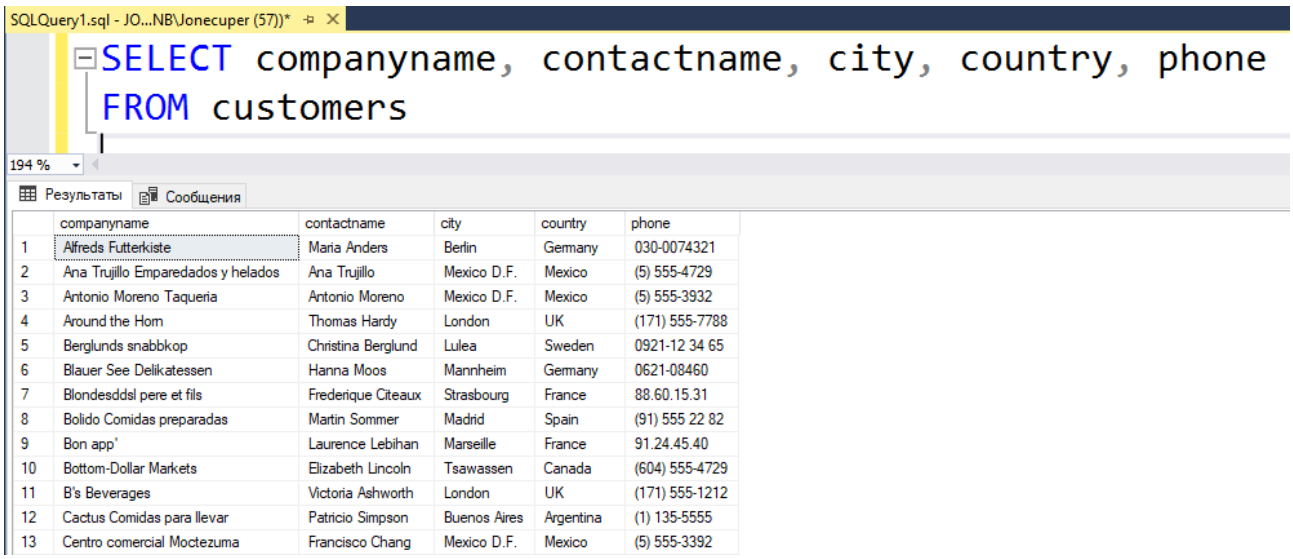
6. Почему оптимизатор запросов в этом случае выбрал индекс «City», а не «PK_Customers»?

7. На панели <Editor> в окне <Query> введите и исполните следующий код:

SELECT companyname, contactname, city, country, phone

FROM customers

На вкладке <Grids> панели <Results> появляется результирующий набор. Обратите внимание, что он упорядочен по значению столбца <CompanyName>. Этот порядок сортировки на самом деле соответствует порядку столбца <CustomerID>, значения которого включают, по крайней мере, первые три символа значений столбца <CompanyName> (рисунок 6).



	companyname	contactname	city	country	phone
1	Alfreds Futterkiste	Maria Anders	Berlin	Germany	030-0074321
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico D.F.	Mexico	(5) 555-4729
3	Antonio Moreno Taqueria	Antonio Moreno	Mexico D.F.	Mexico	(5) 555-3932
4	Around the Hom	Thomas Hardy	London	UK	(171) 555-7788
5	Berglunds snabbkop	Christina Berglund	Lulea	Sweden	0921-12 34 65
6	Blauer See Delikatessen	Hanna Moos	Mannheim	Germany	0621-08460
7	Blondesddsl pere et fils	Frederique Citeaux	Strasbourg	France	88.60.15.31
8	Bolido Comidas preparadas	Martin Sommer	Madrid	Spain	(91) 555 22 82
9	Bon app'	Laurence Lebihan	Marseille	France	91.24.45.40
10	Bottom-Dollar Markets	Elizabeth Lincoln	Tsawassen	Canada	(604) 555-4729
11	B's Beverages	Victoria Ashworth	London	UK	(171) 555-1212
12	Cactus Comidas para llevar	Patricio Simpson	Buenos Aires	Argentina	(1) 135-5555
13	Centro comercial Moctezuma	Francisco Chang	Mexico D.F.	Mexico	(5) 555-3392

Рисунок 6 – Просмотр содержимого полей <companyname>, <contactname>, <city>, <country> и <phone> таблицы «Customers»

8. Щелкните вкладку <Execution Plan>.

План исполнения показывает, что оптимизатор запросов использовал индекс «PK_ Customers». Это произошло, поскольку ни один индекс кроме него не является покрывающим для запроса. В следующем задании вы создадите покрывающий индекс для этого запроса (рисунок 7).

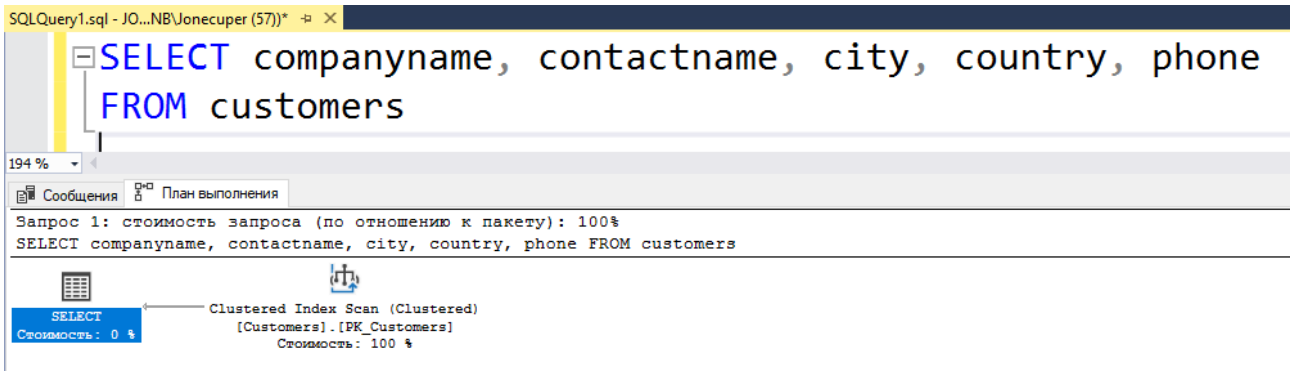


Рисунок 7 – Отображения плана исполнения просмотра содержимого полей <companyname>, <contactname>, <city>, <country> и <phone> таблицы «Customers»

Оставьте вкладку <Execution Plan> активной.

10.8 Создание составного индекса и использующего его запроса

1. На панели <Execution Plan> щелкните правой кнопкой «Customers.PK_Cu...», а затем <Manage Indexes> (рисунок 8).

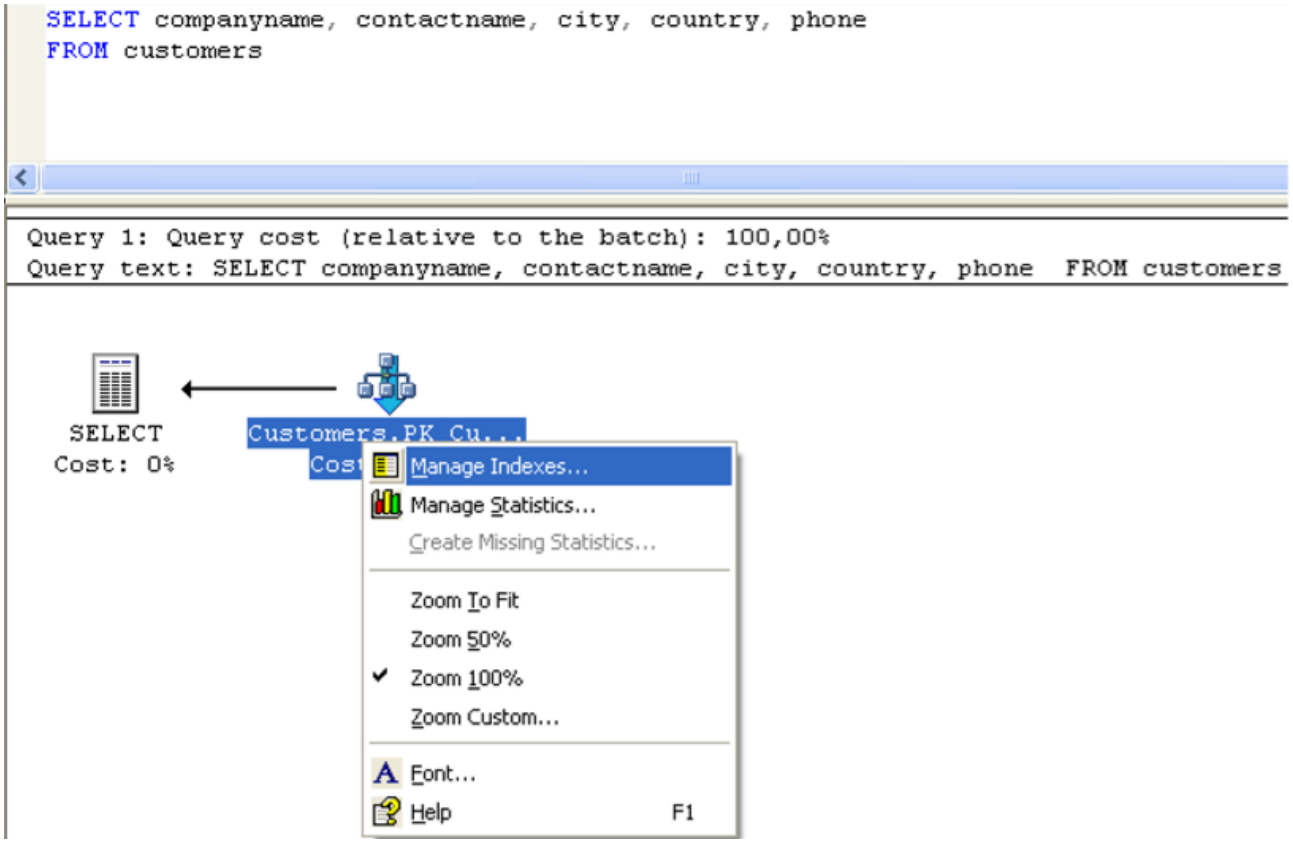


Рисунок 8 – Управление индексами

В диалоговом окне <Manage Indexes> выводится список индексов, созданных для таблицы «Customers» (рисунок 9).

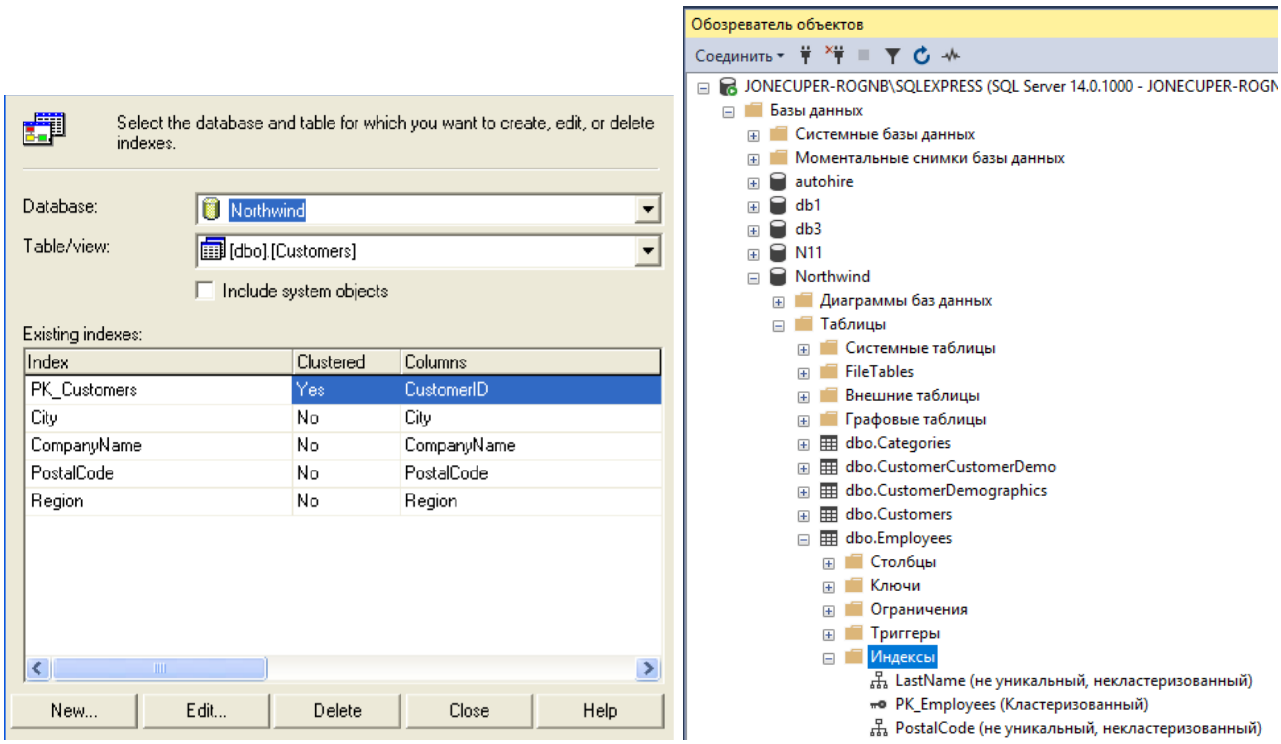


Рисунок 9 – Список индексов, созданных для таблицы «Customers»

2. Щелкните <New>.

Выводится диалоговое окно <Create New Index>.

3. В текстовом поле <Index name> наберите <Contact>.

4. В столбце ниже текстового поля <Index name> установите флажки <CompanyName>, <ContactName>, <City>, <Country> и <Phone>.

5. Выберите строку <City> и щелчками кнопки <UP> поднимите ее в начало списка.

6. Щелкните <OK> (рисунок 10).

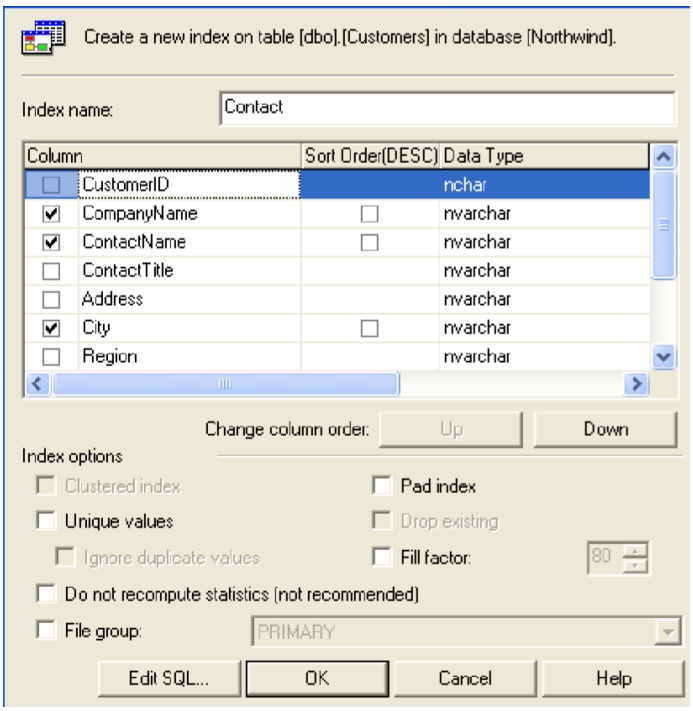


Рисунок 10 – Создание составного индекса «Contact»

В списке индексов диалогового окна <Manage Indexes> появляется индекс «Contact» (рисунок 11).

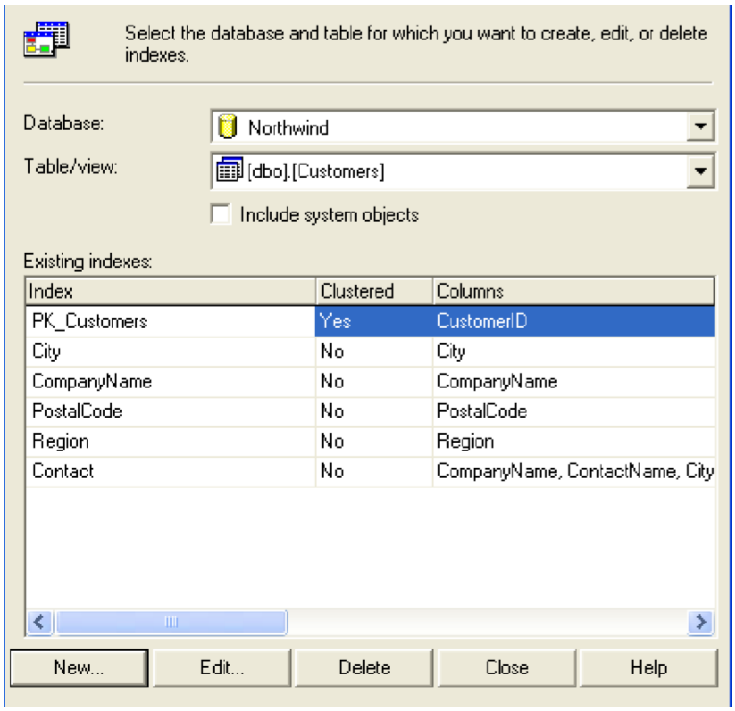


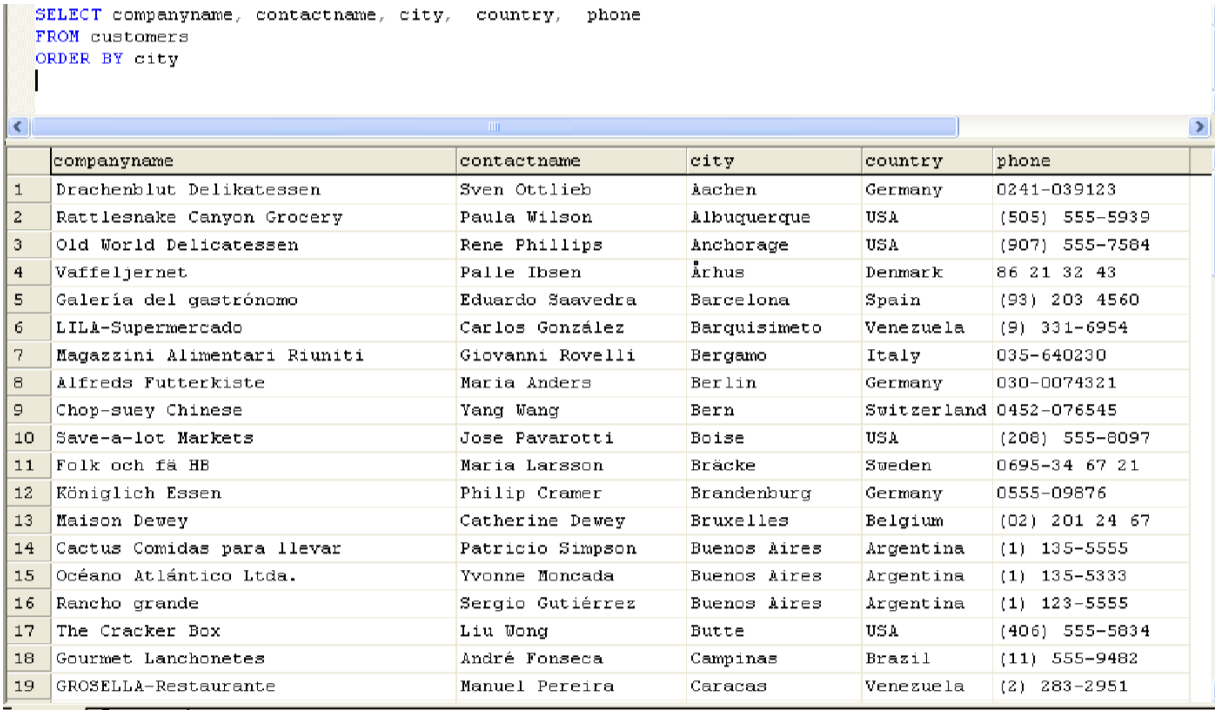
Рисунок 11 – Обновленный список индексов, созданных для таблицы «Customers»

7. Щелкните <Close>.

8. На панели <Editor> в окне <Query> введите и исполните следующий код:

```
SELECT companyname, contactname, city, country, phone
FROM customers
ORDER BY city
```

На вкладке <Grids> панели <Results> появляется результирующий набор. Обратите внимание, что он упорядочен по значению <City> (рисунок 12).



```
SELECT companyname, contactname, city,  country,  phone
FROM customers
ORDER BY city
```

	companyname	contactname	city	country	phone
1	Drachenblut Delikatessen	Sven Ottlieb	Aachen	Germany	0241-039123
2	Rattlesnake Canyon Grocery	Paula Wilson	Albuquerque	USA	(505) 555-5939
3	Old World Delicatessen	Rene Phillips	Anchorage	USA	(907) 555-7584
4	Vaffeljernet	Palle Ibsen	Århus	Denmark	86 21 32 43
5	Galería del gastrónomo	Eduardo Saavedra	Barcelona	Spain	(93) 203 4560
6	LILA-Supermercado	Carlos González	Barquisimeto	Venezuela	(9) 331-6954
7	Magazzini Alimentari Riuniti	Giovanni Rovelli	Bergamo	Italy	035-640230
8	Alfreds Futterkiste	Maria Anders	Berlin	Germany	030-0074321
9	Chop-suey Chinese	Yang Wang	Bern	Switzerland	0452-076545
10	Save-a-lot Markets	Jose Pavarotti	Boise	USA	(208) 555-8097
11	Folk och fä HB	Maria Larsson	Bräcke	Sweden	0695-34 67 21
12	Königlich Essen	Philip Cramer	Brandenburg	Germany	0555-09876
13	Maison Dewey	Catherine Dewey	Bruxelles	Belgium	(02) 201 24 67
14	Cactus Comidas para llevar	Patricio Simpson	Buenos Aires	Argentina	(1) 135-5555
15	Océano Atlántico Ltda.	Yvonne Moncada	Buenos Aires	Argentina	(1) 135-5333
16	Rancho grande	Sergio Gutiérrez	Buenos Aires	Argentina	(1) 123-5555
17	The Cracker Box	Liu Wong	Butte	USA	(406) 555-5834
18	Gourmet Lanchonetes	André Fonseca	Campinas	Brazil	(11) 555-9482
19	GROSELLA-Restaurante	Manuel Pereira	Caracas	Venezuela	(2) 283-2951

Рисунок 12 – Просмотр содержимого полей <companyname>, <contactname>, <city>, <country> и <phone> таблицы «Customers», отсортированных по полю <city>

9. Щелкните вкладку <Execution Plan> (рисунок 13).

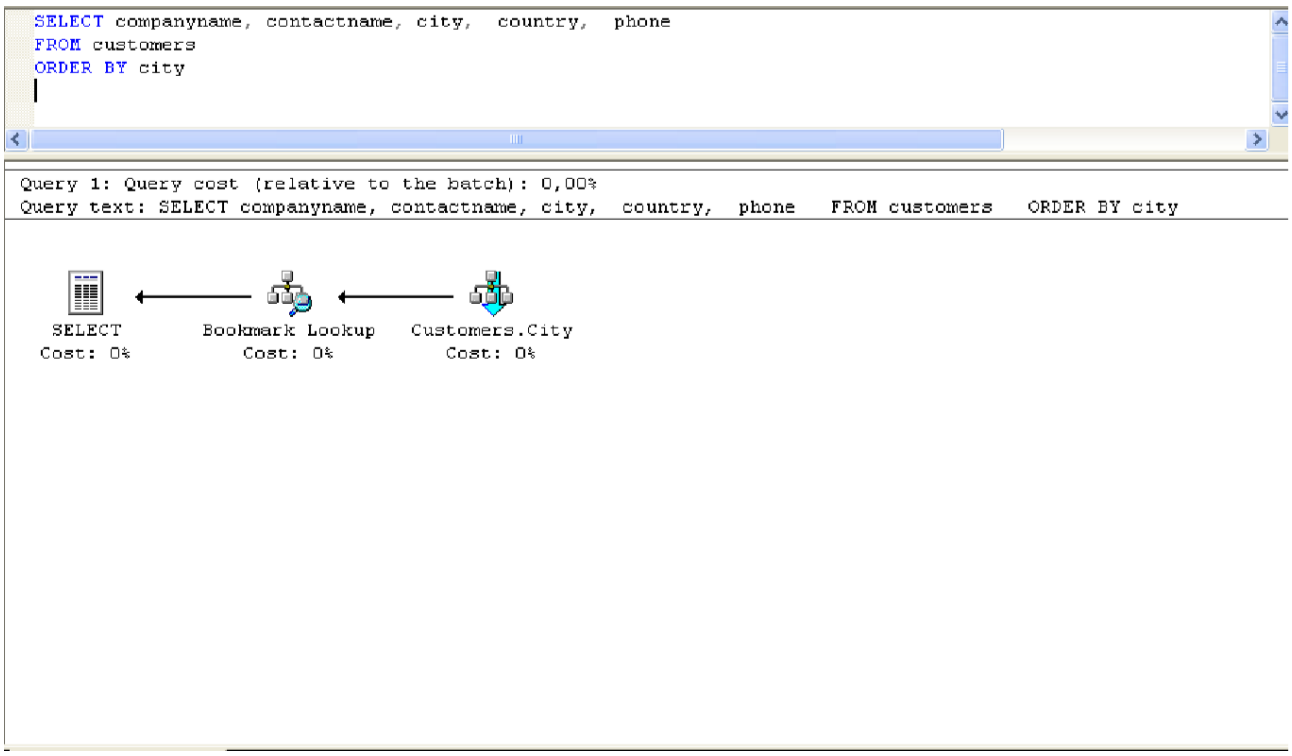


Рисунок 13 – Отображение плана исполнения просмотра содержимого полей <companyname>, <contactname>, <city>, <country> и <phone> таблицы «Customers», отсортированных по полю <city>

План исполнения показывает, что оптимизатор запросов использовал некластерный индекс «Contact». Обратите внимание, что упорядочение списка не требует вычислений, поскольку составной индекс «Contact» упорядочен вначале по значениям столбца <City>.

10. На панели <Editor> в окне <Query> введите и исполните следующий код:

SELECT companyname, contactname, city, country, phone

FROM customers

ORDER BY country

На вкладке <Grids> панели <Results> появляется результирующий набор. Обратите внимание, что он упорядочен по значению <Country> (рисунок 14).

```
SELECT companyname, contactname, city, country, phone
FROM customers
ORDER BY country
```

	companyname	contactname	city	country	phone
1	Cactus Comidas para llevar	Patricio Simpson	Buenos Aires	Argentina	(1) 135-5555
2	Océano Atlántico Ltda.	Yvonne Moncada	Buenos Aires	Argentina	(1) 135-5333
3	Rancho grande	Sergio Gutiérrez	Buenos Aires	Argentina	(1) 123-5555
4	Piccolo und mehr	Georg Pipps	Salzburg	Austria	6562-9722
5	Ernst Handel	Roland Mendel	Graz	Austria	7675-3425
6	Maison Dewey	Catherine Dewey	Bruxelles	Belgium	(02) 201 24 67
7	Suprêmes délices	Pascale Cartrain	Charleroi	Belgium	(071) 23 67 22 20
8	Tradição Hipermercados	Anabela Domingues	Sao Paulo	Brazil	(11) 555-2167
9	Wellington Importadora	Paula Parente	Resende	Brazil	(14) 555-8122
10	Que Delícia	Bernardo Batista	Rio de Janeiro	Brazil	(21) 555-4252
11	Queen Cozinha	Lúcia Carvalho	Sao Paulo	Brazil	(11) 555-1189
12	Ricardo Adocicados	Janete Limeira	Rio de Janeiro	Brazil	(21) 555-3412
13	Familia Arquibaldo	Aria Cruz	Sao Paulo	Brazil	(11) 555-9857
14	Gourmet Lanchonetes	André Fonseca	Campinas	Brazil	(11) 555-9482
15	Hanari Carnes	Mario Pontes	Rio de Janeiro	Brazil	(21) 555-0091
16	Comércio Mineiro	Pedro Afonso	Sao Paulo	Brazil	(11) 555-7647
17	Bottom-Dollar Markets	Elizabeth Lincoln	Tsawassen	Canada	(604) 555-4729
18	Mère Paillarde	Jean Fresnière	Montréal	Canada	(514) 555-8054
19	Laughing Bacchus Wine Cellars	Yoshi Tannamuri	Vancouver	Canada	(604) 555-3392

Рисунок 14 – Просмотр содержимого полей <companyname>, <contactname>, <city>, <country> и <phone> таблицы «Customers», отсортированных по полю <country>

11. Щелкните вкладку <Execution Plan> (рисунок 15).

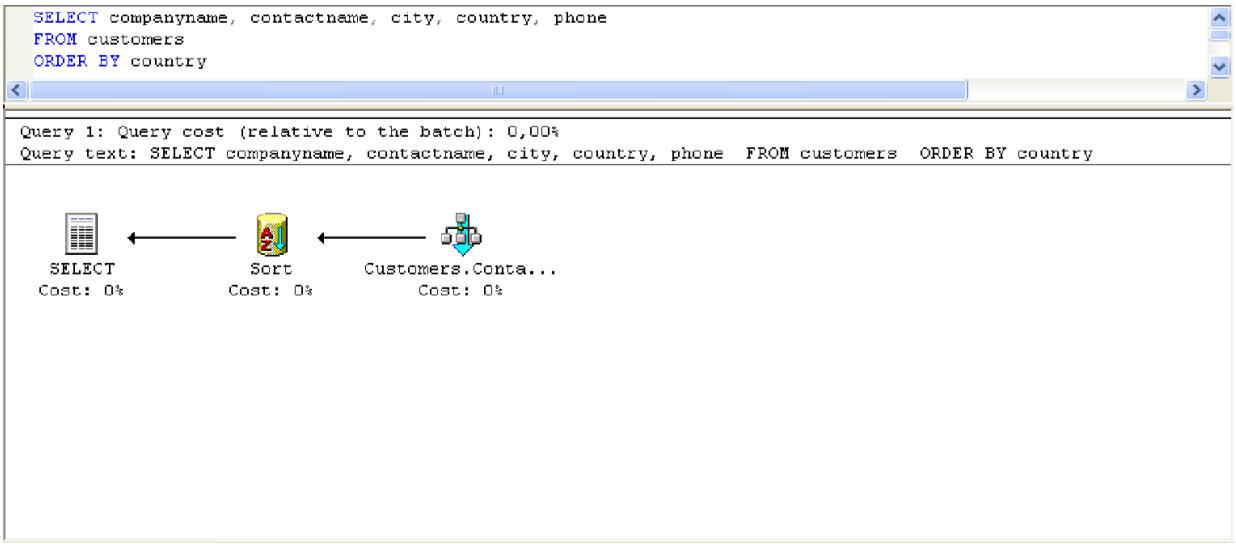


Рисунок 15 – Отображение плана исполнения просмотра содержимого полей <companyname>, <contactname>, <city>, <country> и <phone> таблицы «Customers», отсортированных по полю <country>

План исполнения показывает, что оптимизатор запросов, использовал некластерный индекс «Contact». Обратите внимание, что сортировка списка не требует обработки, поскольку составной индекс «Contact» отсортирован вначале по значениям <Country>.

10.9 Контрольное задание

Создайте составной индекс в базе данных согласно выбранному вами варианту.

10.10 Контрольные вопросы

1. Что такое индекс?
2. Какие существуют типы индексов? В чем их отличия?
3. Как создать составной индекс?

11 Лабораторная работа № 11. Управление транзакциями и блокировками в MS SQL Server 2017

Цель работы: изучить механизм формирования транзакций, уровни изоляции транзакций и взаимные блокировки.

Используемое программное обеспечение: Microsoft SQL Server 2017.

Концепция транзакций – неотъемлемая часть любой клиент-серверной базы данных.

Под *транзакцией* понимается неделимая с точки зрения воздействия на базу данных последовательность операторов манипулирования данными (чтения, удаления, вставки, модификации), приводящая к одному из двух возможных результатов: либо последовательность выполняется, если все операторы правильные, либо вся транзакция откатывается, если хотя бы один оператор не может быть успешно выполнен. Обработка транзакций гарантирует целостность информации в базе данных. Таким образом, транзакция переводит базу данных из одного целостного состояния в другое.

Поддержание механизма транзакций – показатель уровня развитости системы управления базами данных. Корректное поддержание транзакций одновременно является основой обеспечения целостности базы данных. Транзакции также составляют основу изолированности в многопользовательских системах, где с одной базой данных параллельно могут работать несколько пользователей или прикладных программ. Одна из основных задач системы управления базами данных – обеспечение изолированности, т. е. создание такого режима функционирования, при котором каждому пользователю казалось бы, что база данных доступна только ему. Такую задачу системы управления базами данных принято называть *параллелизмом транзакций*.

Большинство выполняемых действий производится в теле транзакций. По умолчанию каждая команда выполняется как самостоятельная транзакция. При

необходимости пользователь может явно указать ее начало и конец, чтобы иметь возможность включить в нее несколько команд.

При выполнении транзакции система управления базами данных должна придерживаться определенных правил обработки набора команд, входящих в транзакцию.

Характеристики транзакций:

– неделимость;

– согласованность;

– изолированность;

– устойчивость.

Транзакция неделима в том смысле, что представляет собой единое целое. Все ее компоненты либо имеют место, либо нет. Не бывает частичной транзакции. Если может быть выполнена лишь часть транзакции, она отклоняется.

Транзакция является *согласованной*, потому что не нарушает бизнес-логику и отношения между элементами данных. Это свойство очень важно при разработке клиент-серверных систем, поскольку в хранилище данных поступает большое количество транзакций от разных систем и объектов. Если хотя бы одна из них нарушит целостность данных, то все остальные могут выдать неверные результаты.

Транзакция всегда *изолирована*, поскольку ее результаты самодостаточны. Они не зависят от предыдущих или последующих транзакций – это свойство называется сериализуемостью и означает, что транзакции в последовательности независимы.

Транзакция устойчива. После своего завершения она сохраняется в системе, которую ничто не может вернуть в исходное (до начала транзакции) состояние, т. е. происходит фиксация транзакции, означающая, что ее действие постоянно даже при сбое системы. При этом подразумевается некая форма хранения информации в постоянной памяти как часть транзакции.

Указанные выше правила выполняет сервер. Программист лишь выбирает нужный уровень изоляции, заботится о соблюдении логической целостности данных и бизнес-правил. На него возлагаются обязанности по созданию эффективных и логически верных алгоритмов обработки данных. Он решает, какие команды должны выполняться как одна транзакция, а какие могут быть разбиты на несколько последовательно выполняемых транзакций. Следует по возможности использовать небольшие транзакции, т. е. включающие как можно меньше команд и изменяющие минимум данных. Соблюдение этого требования позволит наиболее эффективным образом обеспечить одновременную работу с данными множества пользователей.

11.1 Блокировки

Повышение эффективности работы при использовании небольших транзакций связано с тем, что при выполнении транзакции сервер накладывает на данные блокировки.

Блокировкой называется временное ограничение на выполнение некоторых операций обработки данных. Блокировка может быть наложена как на отдельную строку таблицы, так и на всю базу данных. Управлением блокировками на сервере занимается менеджер блокировок, контролирующий их применение и разрешение конфликтов. Транзакции и блокировки тесно связаны друг с другом. Транзакции накладывают блокировки на данные, чтобы обеспечить выполнение характеристик транзакций. Без использования блокировок несколько транзакций могли бы изменять одни и те же данные.

Блокировка представляет собой метод управления параллельными процессами, при котором объект базы данных не может быть модифицирован без ведома транзакции, т. е. происходит блокирование доступа к объекту со стороны других транзакций, чем исключается непредсказуемое изменение объекта.

Различают два вида блокировки:

– *блокировка записи* – транзакция блокирует строки в таблицах таким образом, что запрос другой транзакции к этим строкам будет отменен;

– *блокировка чтения* – транзакция блокирует строки так, что запрос со стороны другой транзакции на блокировку записи этих строк будет отвергнут, а на блокировку чтения – принят.

В системах управления базами данных используют протокол доступа к данным, позволяющий избежать проблемы параллелизма. Его суть заключается в следующем:

– транзакция, результатом действия которой на строку данных в таблице является ее извлечение, обязана наложить блокировку чтения на эту строку;

– транзакция, предназначенная для модификации строки данных, накладывает на нее блокировку записи;

– если запрашиваемая блокировка на строку отвергается из-за уже имеющейся блокировки, то транзакция переводится в режим ожидания до тех пор, пока блокировка не будет снята;

– блокировка записи сохраняется вплоть до конца выполнения транзакции.

Решение проблемы параллельной обработки баз данных заключается в том, что строки таблиц блокируются, а последующие транзакции, модифицирующие эти строки, отвергаются и переводятся в режим ожидания. В связи со свойством сохранения целостности базы данных транзакции являются подходящими единицами изолированности пользователей. Действительно, если каждый сеанс взаимодействия с базой данных реализуется транзакцией, то пользователь начинает с того, что обращается к согласованному состоянию базы данных – состоянию, в котором она могла бы находиться, даже если бы пользователь работал с ней в одиночку.

Если в системе управления базами данных не реализованы механизмы блокирования, то при одновременном чтении и изменении одних и тех же

данных несколькими пользователями могут возникнуть следующие *проблемы одновременного доступа*:

– проблема последнего изменения возникает, когда несколько пользователей изменяют одну и ту же строку, основываясь на ее начальном значении; тогда часть данных будет потеряна, т. к. каждая последующая транзакция перезапишет изменения, сделанные предыдущей. Выход из этой ситуации заключается в последовательном внесении изменений;

– проблема «грязного» чтения возможна в том случае, если пользователь выполняет сложные операции обработки данных, требующие множественного изменения данных перед тем, как они обретут логически верное состояние. Если во время изменения данных другой пользователь будет считывать их, то может оказаться, что он получит логически неверную информацию. Для исключения подобных проблем необходимо производить считывание данных после окончания всех изменений;

– проблема неповторяемого чтения является следствием неоднократного считывания транзакцией одних и тех же данных. Во время выполнения первой транзакции другая может внести в данные изменения, поэтому при повторном чтении первая транзакция получит уже иной набор данных, что приводит к нарушению их целостности или логической несогласованности;

– проблема чтения фантомов появляется после того, как одна транзакция выбирает данные из таблицы, а другая – вставляет или удаляет строки до завершения первой. Выбранные из таблицы значения будут некорректны.

Для решения перечисленных проблем в специально разработанном стандарте определены четыре уровня блокирования. Уровень изоляции транзакции определяет, могут ли другие (конкурирующие) транзакции вносить изменения в данные, измененные текущей транзакцией, а также может ли текущая транзакция видеть изменения, произведенные конкурирующими транзакциями, и наоборот. Каждый последующий уровень поддерживает требования предыдущего и налагает дополнительные ограничения:

– уровень 0 – запрещение «загрязнения» данных. Этот уровень требует, чтобы изменять данные могла только одна транзакция; если другой транзакции необходимо изменить те же данные, она должна ожидать завершения первой транзакции;

– уровень 1 – запрещение «грязного» чтения. Если транзакция начала изменение данных, то никакая другая транзакция не сможет прочитать их до завершения первой;

– уровень 2 – запрещение неповторяемого чтения. Если транзакция считывает данные, то никакая другая транзакция не сможет их изменить. Таким образом, при повторном чтении они будут находиться в первоначальном состоянии;

– уровень 3 – запрещение фантомов. Если транзакция обращается к данным, то никакая другая транзакция не сможет добавить новые или удалить имеющие строки, которые могут быть считаны при выполнении транзакции. Реализация этого уровня блокирования выполняется путем использования блокировок диапазона ключей. Подобная блокировка накладывается не на конкретные строки таблицы, а на строки, удовлетворяющие определенному логическому условию.

11.2 Управление транзакциями

Под *управлением транзакциями* понимается способность управлять различными операциями над данными, которые выполняются внутри реляционной системы управления базами данных. Прежде всего, имеется в виду выполнение операторов <INSERT>, <UPDATE> и <DELETE>. Например, после создания таблицы (выполнения оператора <CREATE TABLE>) не нужно фиксировать результат: создание таблицы фиксируется в базе данных автоматически. Точно так же с помощью отмены транзакции не удастся восстановить только что удаленную оператором <DROP TABLE> таблицу.

После успешного выполнения команд, заключенных в тело одной транзакции, немедленного изменения данных не происходит. Для окончательного завершения транзакции существуют так называемые *команды управления транзакциями*, с помощью которых можно либо сохранить в базе данных все изменения, произошедшие в ходе ее выполнения, либо полностью их отменить.

Существуют три *команды*, которые используются для *управления транзакциями*:

– COMMIT – для сохранения изменений;

– ROLLBACK – для отмены изменений;

– SAVEPOINT – для установки особых точек возврата.

После завершения транзакции вся информация о произведенных изменениях хранится либо в специально выделенной оперативной памяти, либо во временной области отката в самой базе данных до тех пор, пока не будет выполнена одна из команд управления транзакциями. Затем все изменения или фиксируются в базе данных, или отбрасываются, а временная область отката освобождается.

Команда <COMMIT> предназначена для сохранения в базе данных всех изменений, произошедших в ходе выполнения транзакции. Она сохраняет результаты всех операций, которые имели место после выполнения последней команды <COMMIT> или <ROLLBACK>.

Команда <ROLLBACK> предназначена для отмены транзакций, еще не сохраненных в базе данных. Она отменяет только те транзакции, которые были выполнены с момента выдачи последней команды <COMMIT> или <ROLLBACK>.

Команда <SAVEPOINT> (точка сохранения) предназначена для установки в транзакции особых точек, куда в дальнейшем может быть произведен откат (при этом отката всей транзакции не происходит). Команда имеет следующий вид:

SAVEPOINT имя_точки_сохранения

Она служит исключительно для создания точек сохранения среди операторов, предназначенных для изменения данных. Имя точки сохранения в связанной с ней группе транзакций должно быть уникальным.

Для отмены действия группы транзакций, ограниченных точками сохранения, используется команда <ROLLBACK> со следующим синтаксисом:

ROLLBACK TO имя_точки_сохранения

Поскольку с помощью команды <SAVEPOINT> крупное число транзакций может быть разбито на меньшие и поэтому более управляемые группы, ее применение является одним из способов управления транзакциями.

11.3 Управление транзакциями в среде MS SQL Server. Определение транзакций

MS SQL Server предлагает множество средств управления поведением транзакций. Пользователи в основном должны указывать только начало и конец транзакции, используя команды SQL или API (прикладного интерфейса программирования). Транзакция определяется на уровне соединения с базой данных и при закрытии соединения автоматически закрывается. Если пользователь попытается установить соединение снова и продолжить выполнение транзакции, то это ему не удастся. Когда транзакция начинается, все команды, выполненные в соединении, считаются телом одной транзакции, пока не будет достигнут ее конец.

MS SQL Server поддерживает *три вида определения транзакций*:

- явное;
- автоматическое;
- подразумеваемое.

По умолчанию MS SQL Server работает в режиме автоматического начала транзакций, когда каждая команда рассматривается как отдельная транзакция.

Если команда выполнена успешно, то ее изменения фиксируются. Если при выполнении команды произошла ошибка, то сделанные изменения отменяются и система возвращается в первоначальное состояние.

Когда пользователю понадобится создать транзакцию, включающую несколько команд, он должен явно указать транзакцию.

Сервер работает только в одном из двух режимов определения транзакций: автоматическом или подразумевающемся. Он не может находиться в режиме исключительно явного определения транзакций. Этот режим работает поверх двух других.

Для установки режима автоматического определения транзакций используется команда:

SET IMPLICIT_TRANSACTIONS OFF

При работе в режиме неявного (подразумевающегося) начала транзакций *MS SQL Server* автоматически начинает новую транзакцию, как только завершена предыдущая. Установка режима подразумевающегося определения транзакций выполняется посредством другой команды:

SET IMPLICIT_TRANSACTIONS ON

Явные транзакции требуют, чтобы пользователь указал начало и конец транзакции, используя следующие команды:

– начало транзакции: в журнале транзакций фиксируются первоначальные значения изменяемых данных и момент начала транзакции;

BEGIN TRAN[SACTION]

[имя_транзакции |

@имя_переменной_транзакции

[WITH MARK ['описание_транзакции']]]

– конец транзакции: если в теле транзакции не было ошибок, то эта команда предписывает серверу зафиксировать все изменения, сделанные в транзакции, после чего в журнале транзакций помечается, что изменения зафиксированы и транзакция завершена;

COMMIT [TRAN[SACTION]

[имя_транзакции |

@имя_переменной_транзакции]]

– создание внутри транзакции точки сохранения: система управления базами данных сохраняет состояние базы данных в текущей точке и присваивает сохраненному состоянию имя точки сохранения;

SAVE TRAN[SACTION]

{имя_точки_сохранения |

@имя_переменной_точки_сохранения}

– прерывание транзакции; когда сервер встречает эту команду, происходит откат транзакции, восстанавливается первоначальное состояние системы и в журнале транзакций отмечается, что транзакция была отменена. Приведенная ниже команда отменяет все изменения, сделанные в базе данных после оператора <BEGIN TRANSACTION> или отменяет изменения, сделанные в базе данных после точки сохранения, возвращая транзакцию к месту, где был выполнен оператор <SAVE TRANSACTION>.

ROLLBACK [TRAN[SACTION]

[имя_транзакции |

@имя_переменной_транзакции

| имя_точки_сохранения

|@имя_переменной_точки_сохранения]]

Функция <@@TRANCOUNT> возвращает количество активных транзакций.

Функция <@@NESTLEVEL> возвращает уровень вложенности транзакций.

BEGIN TRAN

SAVE TRANSACTION point1

В точке <point1> сохраняется первоначальное состояние таблицы «Товар»

DELETE FROM Товар WHERE Код_товара=2

SAVE TRANSACTION point2

В точке <point2> сохраняется состояние таблицы «Товар» без товаров с кодом 2.

DELETE FROM Товар WHERE Код_товара=3

SAVE TRANSACTION point3

В точке <point3> сохраняется состояние таблицы «Товар» без товаров с кодом 2 и с кодом 3.

DELETE FROM Товар WHERE Код_товара<>1

ROLLBACK TRANSACTION point3

Происходит возврат в состояние таблицы без товаров с кодами 2 и 3, отменяется последнее удаление.

SELECT * FROM Товар

Оператор <SELECT> покажет таблицу «Товар» без товаров с кодами 2 и 3.

ROLLBACK TRANSACTION point1

Происходит возврат в первоначальное состояние таблицы.

SELECT * FROM Товар

COMMIT

Первоначальное состояние сохраняется.

11.4 Вложенные транзакции

Вложенными называются *транзакции*, выполнение которых инициируется из тела уже активной транзакции.

Для создания вложенной транзакции пользователю не нужны какие-либо дополнительные команды. Он просто начинает новую транзакцию, не закрыв предыдущую. Завершение транзакции верхнего уровня откладывается до завершения вложенных транзакций. Если транзакция самого нижнего (вложенного) уровня завершена неудачно и отменена, то все транзакции верхнего уровня, включая транзакцию первого уровня, будут отменены. Кроме того, если несколько транзакций нижнего уровня были завершены успешно (но

не зафиксированы), однако на среднем уровне (не самая верхняя транзакция) неудачно завершилась другая транзакция, то в соответствии с их характеристиками произойдет откат всех транзакций всех уровней, включая успешно завершенные. Только когда все транзакции на всех уровнях завершены успешно, происходит фиксация всех сделанных изменений в результате успешного завершения транзакции верхнего уровня.

Каждая команда <COMMIT TRANSACTION> работает только с последней начатой транзакцией. При завершении вложенной транзакции команда <COMMIT> применяется к наиболее «глубокой» вложенной транзакции. Даже если в команде <COMMIT TRANSACTION> указано имя транзакции более высокого уровня, будет завершена транзакция, начатая последней.

Если команда <ROLLBACK TRANSACTION> используется на любом уровне вложенности без указания имени транзакции, то откатываются все вложенные транзакции, включая транзакцию самого высокого (верхнего) уровня. В команде <ROLLBACK TRANSACTION> разрешается указывать только имя самой верхней транзакции. Имена любых вложенных транзакций игнорируются, и попытка их указания приведет к ошибке. Таким образом, при откате транзакции любого уровня вложенности всегда происходит откат всех транзакций. Если же требуется откатить лишь часть транзакций, можно использовать команду <SAVE TRANSACTION>, с помощью которой создается точка сохранения.

11.5 Пример вложенных транзакций

```
BEGIN TRAN
INSERT Товар (Название, остаток)
VALUES ('v',40)
BEGIN TRAN
INSERT Товар (Название, остаток)
VALUES ('n',50)
```

```
BEGIN TRAN
INSERT Товар (Название, остаток)
VALUES ('m',60)
ROLLBACK TRAN
```

Здесь происходит возврат на начальное состояние таблицы, поскольку выполнение команды <ROLLBACK TRAN> без указания имени транзакции откатывает все транзакции.

11.6 Блокировки в среде MS SQL Server. Управление блокировками

Пользователю чаще всего не нужно предпринимать никаких действий по управлению блокировками. Всю работу по установке, снятию и разрешению конфликтов выполняет специальный компонент сервера, называемый *менеджером блокировок. MS SQL Server* поддерживает различные уровни блокирования объектов (или детализацию блокировок), начиная с отдельной строки таблицы и заканчивая базой данных в целом. Менеджер блокировок автоматически оценивает, какое количество данных необходимо блокировать, и устанавливает соответствующий тип блокировки. Это позволяет поддерживать равновесие между производительностью работы системы блокирования и возможностью пользователей получать доступ к данным. Блокирование на уровне строки позволяет наиболее точно управлять таким доступом, поскольку блокируются только действительно изменяемые строки. Множество пользователей могут одновременно работать с данными с минимальными задержками. Платой за это является увеличение числа операций установки и снятия блокировок, а также большое количество служебной информации, которое приходится хранить для отслеживания установленных блокировок. При блокировке на уровне таблицы производительность системы блокирования резко увеличивается, так как необходимо установить лишь одну блокировку и снять ее только после завершения транзакции. Пользователь при этом имеет максимальную скорость доступа к данным. В то же время они не доступны никому другому, потому что вся таблица заблокирована. Приходится ожидать, пока текущий пользователь завершит работу.

Действия, выполняемые пользователями при работе с данными, сводятся к операциям двух типов: их чтению и изменению. В операции по изменению включаются действия по добавлению, удалению и собственно изменению данных. В зависимости от выполняемых действий сервер накладывает определенный *тип блокировки* из следующего перечня:

Коллективные блокировки. Они накладываются при выполнении операций чтения данных (например, <SELECT>). Если сервер установил на ресурс коллективную блокировку, то пользователь может быть уверен, что уже никто не сможет изменить эти данные.

Блокировка обновления. Если на ресурс установлена коллективная блокировка и для этого ресурса устанавливается блокировка обновления, то никакая транзакция не сможет наложить коллективную блокировку или блокировку обновления.

Монопольная блокировка. Этот тип блокировок используется, если транзакция изменяет данные. Когда сервер устанавливает монопольную блокировку на ресурс, то никакая другая транзакция не может прочитать или изменить заблокированные данные. Монопольная блокировка не совместима ни с какими другими блокировками, и ни одна блокировка, включая монопольную, не может быть наложена на ресурс.

Блокировка массивного обновления. Накладывается сервером при выполнении операций массивного копирования в таблицу и запрещает обращение к таблице любым другим процессам. В то же время несколько процессов, выполняющих массивное копирование, могут одновременно вставлять строки в таблицу.

Помимо перечисленных основных типов блокировок MS SQL Server поддерживает ряд специальных блокировок, предназначенных для повышения производительности и функциональности обработки данных. Они называются блокировками намерений и используются сервером в том случае, если транзакция намеревается получить доступ к данным вниз по иерархии и для других транзакций необходимо установить запрет на наложение блокировок, которые будут конфликтовать с блокировкой, накладываемой первой транзакцией.

Ранее рассмотренные блокировки относятся к данным. Помимо перечисленных в среде MS SQL Server существует два других типа блокировок:

блокировка диапазона ключей и *блокировка схемы* (метаданных, описывающих структуру объекта).

Блокировка диапазона ключей решает проблему возникновения фантомов и обеспечивает требования сериализуемости транзакции. Блокировки этого типа устанавливаются на диапазон строк, соответствующих определенному логическому условию, с помощью которого осуществляется выборка данных из таблицы.

Блокировка схемы используется при выполнении команд модификации структуры таблиц для обеспечения целостности данных.

«*Мертвые*», *или тупиковые, блокировки* характерны для многопользовательских систем. «Мертвая» блокировка возникает, когда две транзакции блокируют два блока данных и для завершения любой из них нужен доступ к данным, заблокированным ранее другой транзакцией. Для завершения каждой транзакции необходимо дождаться, пока блокированная другой транзакцией часть данных будет разблокирована. Но это невозможно, так как вторая транзакция ожидает разблокирования ресурсов, используемых первой.

Без применения специальных механизмов обнаружения и снятия «мертвых» блокировок нормальная работа транзакций будет нарушена. Если в системе установлен бесконечный период ожидания завершения транзакции (а это задано по умолчанию), то при возникновении «мертвой» блокировки для двух транзакций вполне возможно, что, ожидая освобождения заблокированных ресурсов, в тупике окажутся и новые транзакции. Чтобы избежать подобных проблем, в среде MS SQL Server реализован специальный механизм разрешения конфликтов тупикового блокирования.

Для этих целей сервер снимает одну из блокировок, вызвавших конфликт, и откатывает инициализировавшую ее транзакцию. При выборе блокировки, которой необходимо пожертвовать, сервер исходит из соображений минимальной стоимости.

Полностью избежать возникновения «мертвых» блокировок нельзя. Хотя сервер и имеет эффективные механизмы снятия таких блокировок, все же при

написании приложений следует учитывать вероятность их возникновения и предпринимать все возможные действия для предупреждения этого. «Мертвые» блокировки могут существенно снизить производительность, поскольку системе требуется достаточно много времени для их обнаружения, отката транзакции и повторного ее выполнения.

Для минимизации возможности образования «мертвых» блокировок при разработке кода транзакции следует придерживаться следующих правил:

– выполнять действия по обработке данных в постоянном порядке, чтобы не создавать условия для захвата одних и тех же данных;

– избегать взаимодействия с пользователем в теле транзакции;

– минимизировать длительность транзакции и выполнять ее по возможности в одном пакете;

– применять как можно более низкий уровень изоляции.

11.7 Уровни изоляции MS SQL Server

Уровень изоляции определяет степень независимости транзакций друг от друга. Наивысшим уровнем изоляции является сериализуемость, обеспечивающая полную независимость транзакций друг от друга. Каждый последующий уровень соответствует требованиям всех предыдущих и обеспечивает дополнительную защиту транзакций.

MS SQL Server поддерживает все четыре уровня изоляции, определенные стандартом ANSI. Уровень изоляции устанавливается командой:

```
SET TRANSACTION ISOLATION LEVEL
{ READ COMMITTED
| READ UNCOMMITTED
| REPEATABLE READ
| SERIALIZABLE }
```

READ UNCOMMITED – незавершенное чтение, или допустимо черновое чтение. Низший уровень изоляции, соответствующий уровню 0. Он гарантирует только физическую целостность данных: если несколько пользователей одновременно изменяют одну и ту же строку, то в окончательном варианте строка будет иметь значение, определенное пользователем, последним изменившим запись. По сути, для транзакции не устанавливается никакой блокировки, которая гарантировала бы целостность данных. Для установки этого уровня используется команда:

SET TRANSACTION ISOLATION

LEVEL READ UNCOMMITTED

READ COMMITTED – завершенное чтение, при котором отсутствует черновое, «грязное» чтение. Тем не менее, в процессе работы одной транзакции другая может быть успешно завершена и сделанные ею изменения зафиксированы. В итоге первая транзакция будет работать с другим набором данных. Это проблема неповторяемого чтения. Данный уровень изоляции установлен в *MS SQL Server* по умолчанию и устанавливается посредством команды:

SET TRANSACTION ISOLATION

LEVEL READ COMMITTED

REPEATABLE READ – повторяющееся чтение. Повторное чтение строки возвратит первоначально считанные данные, несмотря на любые обновления, произведенные другими пользователями до завершения транзакции. Тем не менее, на этом уровне изоляции возможно возникновение фантомов. Его установка реализуется командой:

SET TRANSACTION ISOLATION

LEVEL REPEATABLE READ

SERIALIZABLE – сериализуемость. Чтение запрещено до завершения транзакции. Это максимальный уровень изоляции, который обеспечивает полную изоляцию транзакций друг от друга. Он устанавливается командой:

SET TRANSACTION ISOLATION

LEVEL SERIALIZABLE

В каждый момент времени возможен только один уровень изоляции.

В общем случае результат выполнения транзакции не должен зависеть от других выполняющихся одновременно транзакций. Но полная изоляция транзакций делает невозможной параллельную обработку данных. В большинстве случаев такая строгость не нужна, и поэтому были введены так называемые «*уровни изоляции*» (*Isolation Level*), которые определяют степень параллелизма выполнения транзакций. Чем ниже уровень изоляции, тем выше степень параллелизма и тем больше риск «неправильного» выполнения транзакции.

В стандарте ANSI SQL вводятся четыре уровня изоляции. И по названиям, и по поведению уровни изоляции в *Microsoft SQL Server 2017* полностью соответствуют описанным в стандарте. В стандарте уровни изоляции описываются при помощи «*феноменов*» – побочных эффектов низкой изолированности транзакций.

Всего их четыре:

1) грязная запись (*Dirty Write*);

2) грязное чтение (*Dirty Read*);

3) неповторяющееся чтение (*Repeatable Read*);

4) фантомы (*Phantoms*).

Используются *четыре уровня* изоляции, которые устраняют вышеперечисленные феномены:

1. *Read Uncommitted* – самый низкий уровень изоляции. При этом уровне изоляции ликвидируется феномен «грязная запись», но транзакция может читать «грязные» данные незафиксированных транзакций. «Грязные» они потому, что если незафиксированная транзакция будет откачена, то получится, что были прочитаны никогда не существовавшие данные.

2. *Read Committed* – этот уровень изоляции решает проблему грязного чтения, т. е. в транзакции с таким уровнем изоляции невозможно прочитать данные незафиксированных транзакций, однако остается феномен

неповторяющегося чтения. Суть этого феномена в том, что если первая транзакция один раз прочитала данные, а потом вторая их изменила и зафиксировалась, то повторное чтение тех же данных первой транзакцией вернет уже измененные данные. *Microsoft SQL Server* использует этот уровень изоляции по умолчанию.

3. *Repeatable Read* – этот уровень решает предыдущую проблему, но при этом возможно появление фантомов. Изменение однажды прочитанных первой транзакцией данных другими транзакциями (до фиксации первой) невозможно, однако если первая транзакция сделала выборку по какому-то условию, а потом вторая транзакция добавила новые данные, этому условию удовлетворяющие, и зафиксировалась, то повторная выборка первой транзакцией по тому же условию вернет в том числе и добавленные данные – фантомы.

4. *Serializable* – при этом уровне изоляции никакие фантомы невозможны в принципе, равно как и другие феномены. Этот уровень изоляции ни на какие феномены не опирается, просто требуется, чтобы результат параллельного выполнения транзакций был таким же, как если бы они выполнялись последовательно.

11.8 Контрольное задание

1. Создайте таблицу в базе данных.

```
create table SomeTable(x int, y int)
go
```

2. Вставьте одну запись в таблицу.

```
insert into SomeTable (x,y) values (1,1)
```

3. В одном окне <MS SQL Query Analyzer> напишите скрипт обновления записи с явно объявленной транзакцией.

```
begin tran
update SomeTable
set x = 2
```

where y = 1

commit

Выделить строки и запустить скрипт на выполнение (F5). <MS SQL Query Analyzer> выполнит выделенные строки, т. е. мы получим незафиксированную транзакцию.

4. В другом окне <MS SQL Query Analyzer> попробуйте выбрать данные из таблицы при <read committed> и <read uncommitted> уровнях изоляции.

set transaction isolation level read uncommitted

begin tran

select * from SomeTable

commit

set transaction isolation level read committed

begin tran

select * from SomeTable

commit

5. В первом окне выделите текст как на рисунке 1 и запустите скрипт на выполнение.

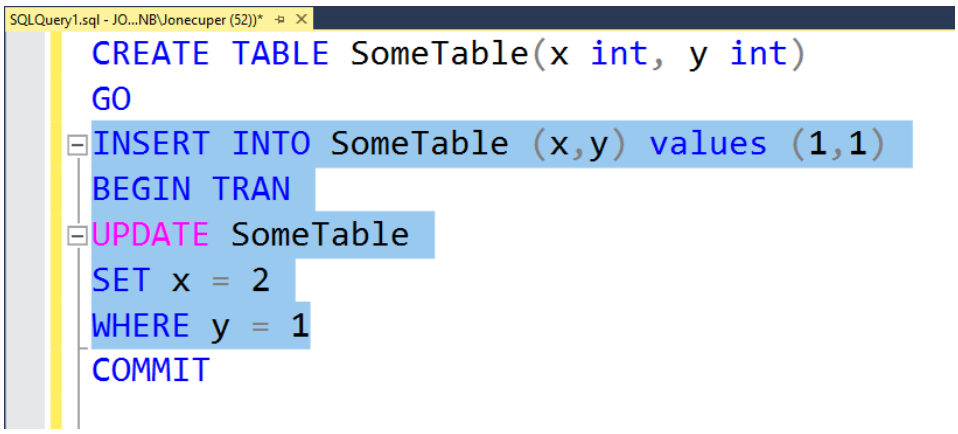


Рисунок 1 – Запуск скрипта на выполнение

Повторите запросы из пункта 4. Объясните полученные результаты.

6. Аналогично пунктам 3–5 продемонстрируйте проблемы при уровнях изоляции <Repeatable Read> и <Serializable>. Для переключения уровней изоляции используйте следующие скрипты:

```
set transaction isolation level repeatable read
begin tran
select * from SomeTable
commit

set transaction isolation level serializable
begin tran
select * from SomeTable
commit
```

7. Создайте вторую таблицу в базе данных.

```
create table AnotherTable(x int, y int)
go
```

8. Вставьте в обе таблицы по две записи.

```
delete from SomeTable
insert into SomeTable (x,y) values (1,1)
insert into SomeTable (x,y) values (2,2)

insert into AnotherTable (x,y) values (1,1)
insert into AnotherTable (x,y) values (2,1)
```

9. В одном окне <MS SQL Query Analyzer> напишите скрипт обновления записей в двух таблицах в контексте одной транзакции.

```
begin tran
update Sometable
set x = 1
where y = 1
update Anothertable
set x = 2
```

where y = 2

commit

10. Во втором окне <MS SQL Query Analyzer> напишите скрипт обновления записей в двух таблицах, но в обратном порядке.

begin tran

update Anothertable

set x = 1

where y = 1

update Sometable

set x = 2

where y = 2

commit

11. В первом окне выделите строки 1–4 и запустить скрипт на выполнение, переключитесь на второе окно, выделите строки 1–4, запустите скрипт на выполнение. Переключитесь на первое окно, выделите оставшийся скрипт, выполните его. Переключитесь на второе окно, выделите оставшийся скрипт и выполните его. Объясните полученный результат.

11.9 Контрольные вопросы

1. Что понимается под управлением транзакциями?

2. Какие существуют команды, которые используются для управления транзакциями?

3. Какие существуют виды определения транзакций?

4. Для чего предназначены блокировки?

5. Какие существуют типы блокировок?

12 Лабораторная работа № 12. Разработка базы данных. В ms sql server

Цель работы: научиться разрабатывать базу данных в *MS SQL Server*: создавать таблицы, схему данных, хранимые процедуры, триггеры, представления, индексы, назначать права доступа к данным.

Используемое программное обеспечение: Microsoft SQL Server 2017.

12.1 Контрольное задание

1. Выполните контрольные задания в лабораторных работах № 4–10.

2. Создайте базу данных согласно выбранному вами варианту, содержащую:

– не менее трех хранимых процедур;

– не менее трех триггеров;

– не менее трех представлений;

– созданный составной индекс.

3. Продемонстрировать:

– запросы, где осуществляется выборка и модификация данных из базы данных;

– назначенные права доступа к данным.

12.2 Контрольные вопросы

1. Какие способы создания таблиц существуют в системе управления базами данных MS SQL Server?

2. Какие типы данных допустимы при создании таблицы?

3. Каким образом возможно проверить создание таблиц?

4. Каким образом осуществляется обеспечение целостности данных в MS SQL Server?

5. Как выполнить простейшие операции вставки строк данных в таблицу средствами T-SQL?

6. Как выполнить простейшие операции модификации строк таблицы средствами T-SQL?

7. Как выполнить просмотр таблицы?

8. Как сохранить результаты запроса в таблице?

9. Перечислите операторы управления.

10. Перечислите операторы манипулирования данными.

11. Что такое хранимая процедура?

12. Чем могут обладать хранимые процедуры?

13. Что такое триггер?

14. Каково назначение триггеров?

15. В чем отличие триггеров от хранимых процедур?

16. Сформулируйте определение представления.

17. Каково назначение представлений?

18. Что такое индекс?

19. Какие существуют типы индексов, в чем их отличия?

13 Лабораторная работа № 13. Разработка бизнес-правил

Цель работы: научиться разрабатывать бизнес-правила.

Используемое программное обеспечение: Microsoft SQL Server 2017.

Определяя бизнес-правила, нужно выявить ограничения, на основе которых строится работа с системой и данными, а также меры их защиты. В систему бизнес-правил входят все ограничения, налагаемые на систему, в том числе целостность данных и безопасность системы.

Организовать безопасность системы и целостность данных возможно несколькими способами:

1) настройка аутентификации и авторизации пользователя;

2) создание связей между таблицами и обеспечение целостности;

3) настройка управления правами доступа к данным;

4) создание триггеров на обеспечение целостности данных и управление транзакциями.

Аутентификацию можно настроить, нажав правой кнопкой на локальный сервер в <Enterprise Manager>, далее выбрать свойства и перейти на вкладку <Security>. Права доступа к базам данных определятся также в свойствах баз данных.

13.1 Пример создания триггеров, запрещающего добавление, обновление и удаление строк в таблице. Триггер для запрета добавления строк в таблицу

Рассмотрим создание триггера, который запрещает добавление строк в таблицу «Товар», выводя сообщение <Вставка строк запрещена>:

```
create trigger товар_insert
on Товар
for Insert
as
```

print 'Вставка строк запрещена'

rollback tran

При выполнении запроса в <*MS Query Analizer*> на вставку строк в таблицу «Товар», выводится сообщение <Вставка строк запрещена>. Если использовать для заполнения строк <*MS Enterprise Manager*>, сообщение выводиться не будет, но вставленные поля сохраняться не будут, так как в триггере прописана команда <rollback tran>, которая отменяет транзакцию.

13.2 Триггер для запрета обновления строк в таблице

Рассмотрим создание триггера, который реагирует на модификацию существующих данных. При изменении поля <код_поставщика> в таблице «Поставщик» будет изменяться <код_поставщика> в таблице «Товар»:

```
create trigger поставщик_updated
on поставщик
for update
as
IF UPDATE код_поставщика
UPDATE товар
SET код_поставщика = код_поставщика + inserted.код_поставщика – deleted.код_поставщика
FROM товар, deleted, inserted
WHERE товар.код_поставщика = inserted.код_поставщика
AND товар.код_поставщика = deleted.код_поставщика
```

13.3 Триггера для запрета удаления строк из таблицы

Рассмотрим создание триггера, который выполняет удаление полей из таблицы товар, если удалить соответствующие поля из таблицы поставщик:

```
create trigger товар_delete
```

```
on товар
for delete
as
select count(*)
from товар, поставщик
where товар.код_поставщика=поставщик.код_поставщика
```

13.4 Контрольное задание

Разработать бизнес-правила к базе данных согласно выбранному вами варианту.

13.5 Контрольные вопросы

1. Что входит в систему бизнес-правил?
2. Какими способами можно организовать безопасность системы и целостность данных?

14 Лабораторная работа № 14. Разработка концептуальной модели приложения-клиента

Цель работы: научиться разрабатывать концептуальную модель приложения-клиента.

Используемое программное обеспечение: Ramus 2.0.

Концептуальная модель приложения – это модель, которую проектировщик хочет довести до понимания пользователя. Используя приложение и читая документацию к нему, пользователь выстраивает в голове модель функционирования системы. Хорошо, если модель, возникшая в голове пользователя, и модель, задуманная проектировщиком, совпадают. Шансы на это выше, если проектировщик предварительно создаст четкую концептуальную модель.

Концептуальная модель – это еще не пользовательский интерфейс. Она абстрактно – в терминах задач, нажатий на клавиши, манипуляций мышью или экранной графики – описывает, что именно пользователь должен делать с системой, и какие концепты ему необходимо знать. Основная идея заключается в том, что тщательная разработка подробной концептуальной модели, на основе которой потом проектируется пользовательский интерфейс, делает приложение более простым и понятным для понимания. При этом необходимо, во-первых, сделать концептуальную модель максимально простой с использованием минимального количества концептов для обеспечения необходимой функциональности; и, во-вторых, максимально ориентировать концептуальную модель на конкретные задачи, т. е. исключить или ограничить работу пользователя с концептами, не фигурирующими в данной области задач.

Важным компонентом концептуальной модели является анализ объектов и действий – список всех видимых пользователю объектов приложения и действий, которые пользователь может совершать над каждым объектом. В реализации системы могут присутствовать и другие объекты, но предполагается,

что они будут невидимыми для пользователя. В частности, в состав концептуальной модели не могут входить чисто имплементационные объекты.

Объекты концептуальной модели приложения могут образовывать структурную иерархию, в которой дочерние блоки будут перенимать действия родительских. В зависимости от приложения объекты могут также образовывать иерархию включения, в которой некоторые объекты включают в себя другие. Использование двух этих типов иерархии в концептуальной модели значительно облегчает проектирование и разработку связного и понятного пользовательского интерфейса.

Подобный анализ объектов и действий помогает управлять реализацией системы, поскольку он указывает наиболее удобный вид иерархии объектов, а также методы работы, которые предусматривает каждый вид. Он также облегчает структуру команд приложения, т. к. позволяет разработчику увидеть, какие действия применимы к разным объектам и могут быть спроектированы как обобщенные. В свою очередь это делает структуру команд более легкой для изучения пользователем: вместо того, чтобы осваивать большое количество объектно-ориентированных команд достаточно изучить несколько обобщенных, применяемых к разным объектам.

14.1 Пример разрабатываемого приложения

К примеру, если бы разрабатываемое приложение было программой, помогающей человеку управлять своим банковским счетом, концептуальная модель включала бы такие объекты, как проверки, счета, суммы денег и такие действия как размещение, отзыв, аннулирование, запрос баланса. Концептуальная модель должна исключать все объекты, не входящие в данную область задач (например, буферы, диалоговые окна, состояния, базы данных, строки), а также действия: нажатие клавиш, резервное копирование баз данных, очистка буфера обмена.

Поскольку компьютерная бухгалтерия иногда имеет свойства, не присущие бумажной, некоторые дополнительные функции могут попасть в концептуальную модель (например, такой объект как «шаблон операции» и такое действие как «определение шаблона»). Однако следует понимать, что каждый дополнительный концепт, добавленный в область задач, может создать две проблемы:

1) пользователи, знающие область задач, не распознают новый концепт, следовательно, его придется изучать дополнительно;

2) в геометрической прогрессии возрастет сложность приложения, поскольку каждый добавленный концепт начинает взаимодействовать с многочисленными другими концептами приложения.

Вот почему следует строго ограничивать количество дополнительных концептов. Отсюда девиз разработчика пользовательского интерфейса: «Чем меньше, тем лучше».

После разработки концептуальной модели необходимо написать сценарий, описывающий работу пользователя с приложением. При этом нужно использовать лишь терминологию из области задач. К примеру, в случае с банковским приложением следовало бы написать следующий сценарий: «Джон использует программу для проверки баланса своего текущего счета. Он кладет деньги на текущий счет, а потом переводит их с текущего счета на сберегательный». Следует отметить, что этот сценарий ссылается только на объекты и действия области задач, а не на свойства пользовательского интерфейса.

В процессе разработки пользовательского интерфейса абстрактные концепты концептуальной модели переводятся в конкретные представления и действия пользователя. Для достижения оптимальных результатов пользовательский интерфейс проектируется после разработки концептуальной модели.

Сценарии при этом могут быть переписаны на уровне пользовательского интерфейса.

Разработка концептуальной модели как первая стадия проектирования пользовательского интерфейса имеет ряд *преимуществ*.

Разделение области задач на объекты и действия позволяет определить действия, одинаковые для нескольких объектов. Впоследствии один и тот же пользовательский интерфейс может быть использован для работы с разными объектами. Например, если в хорошо продуманном приложении пользователь может использовать объекты А и Б, и он хочет создать объект А, но умеет создавать только Б, он сумеет создать и А, поскольку это действие происходит единообразно для двух данных объектов. Как, впрочем, и копирование, перемещение, удаление, редактирование, печать и т. д. Это в свою очередь делает пользовательский интерфейс более простым и последовательным, а значит и более удобным в изучении и использовании.

Даже если не брать в расчет упрощение, вызванное существованием обобщенных действий, при проектировании пользовательской модели разработчики должны признать относительную важность концептов, значимость концептов для области задач (в противоположность технической области), типовую иерархию и иерархию включения объектов. Все эти явления в значительной степени облегчают проектирование пользовательского интерфейса.

Концептуальная модель представляет собой начальный этап разработки терминологии программного продукта, то есть словаря терминов, которые будут использоваться для идентификации каждого объекта и действия, реализованного в программе. Это способствует сохранению устойчивости терминологии не только в самой программе, но и в сопутствующей документации. Недостатками программы, разработанной без такой терминологии, являются:

1) использование многочисленных терминов для определения одного концепта;

2) использование одного термина для определения разных концептов.

Наконец, разработка концептуальной модели представляет собой первоначальный вид объектной модели (по крайней мере, для объектов,

необходимых пользователю), которую разработчики могут в дальнейшем использовать при реализации программного обеспечения. Это особенно актуально если разработчик пишет программу на объектно-ориентированных языках, таких как *Java, C#* или *C++*.

14.2 Контрольное задание

Разработать концептуальную модель приложения-клиента к базе данных согласно выбранному вами варианту.

14.3 Контрольные вопросы

1. Что такое концептуальная модель приложения клиента?
2. Привести пример концептуальной модели приложения клиента.
3. Преимущества разработки концептуальной модели.

15 Лабораторная работа № 15. Разработка приложения-клиента в среде Visual Studio 2017 на языке C#

Цель работы: научиться разрабатывать клиентское приложение в среде Microsoft Visual Studio, используя платформу .NET: создавать меню; реализовывать отображение данных из таблиц базы данных, созданной в MS SQL Server; осуществлять поиск, сортировку, фильтрацию данных; реализовывать выполнение запросов и хранимых процедур.

Используемое программное обеспечение: Microsoft SQL Server 2017, Microsoft Visual Studio 2017 Enterprise.

15.1 Настройка подключения к базе данных

Для разработки приложения-клиента в среде программирования Microsoft Visual Studio используется технология ADO.NET.

Установка соединения с хранилищем данных выполняется двумя путями: с помощью свойства <ConnectionString> или с помощью компонента <ADOConnection>.

Для подключения к базе данных необходимо запустить среду Visual Studio и выбрать пункт меню <Средства-Подключиться к базе данных> (рисунок 1). Откроется диалоговое окно (рисунок 2).

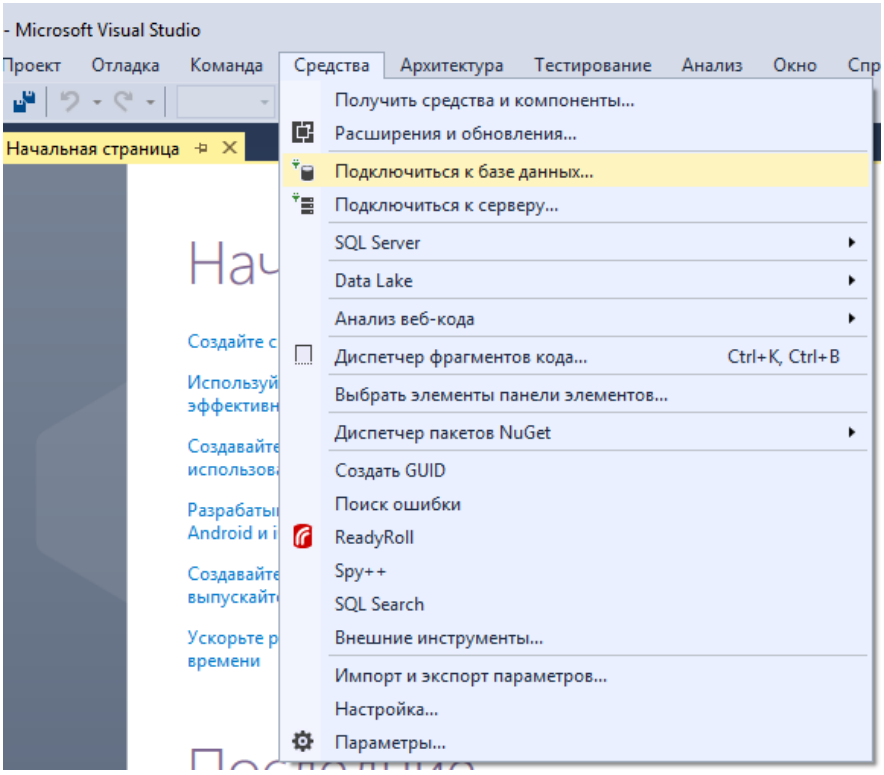


Рисунок 1 – Команда для открытия диалогового окна добавления подключения к существующей базе данных

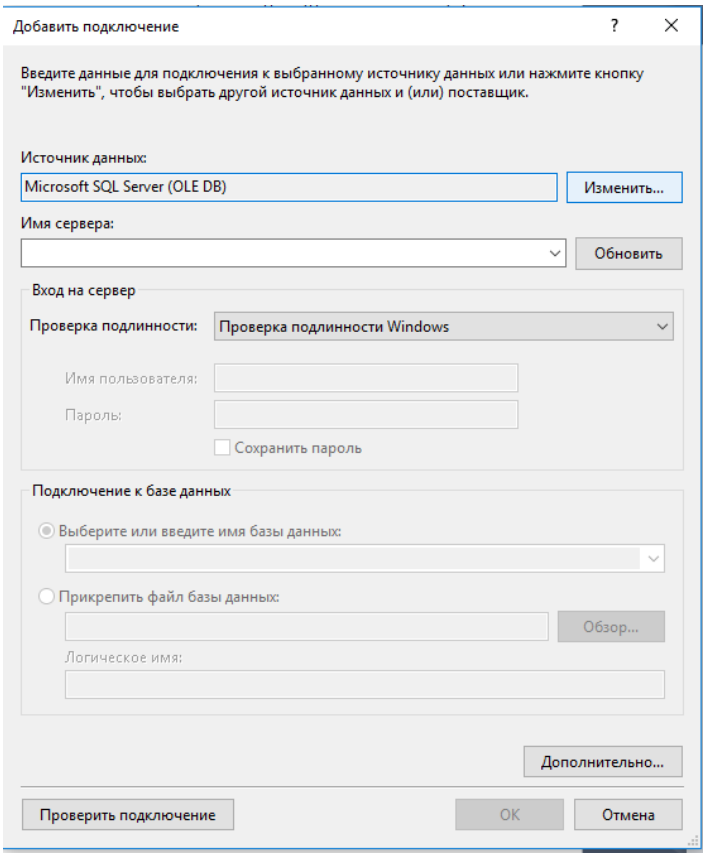


Рисунок 2 – Окно «Добавить подключение»

Для соединения с данными СУБД Microsoft SQL Server необходимо выбрать правильный драйвер для работы с базой данных, также называемый поставщиком данных: <Microsoft OLE DB Provider for SQL Server> для этого нажмите кнопку «Изменить», находящуюся напротив надписи «Источник данных», после этого откроется диалоговое окно для выбора источника данных (рисунок 3).

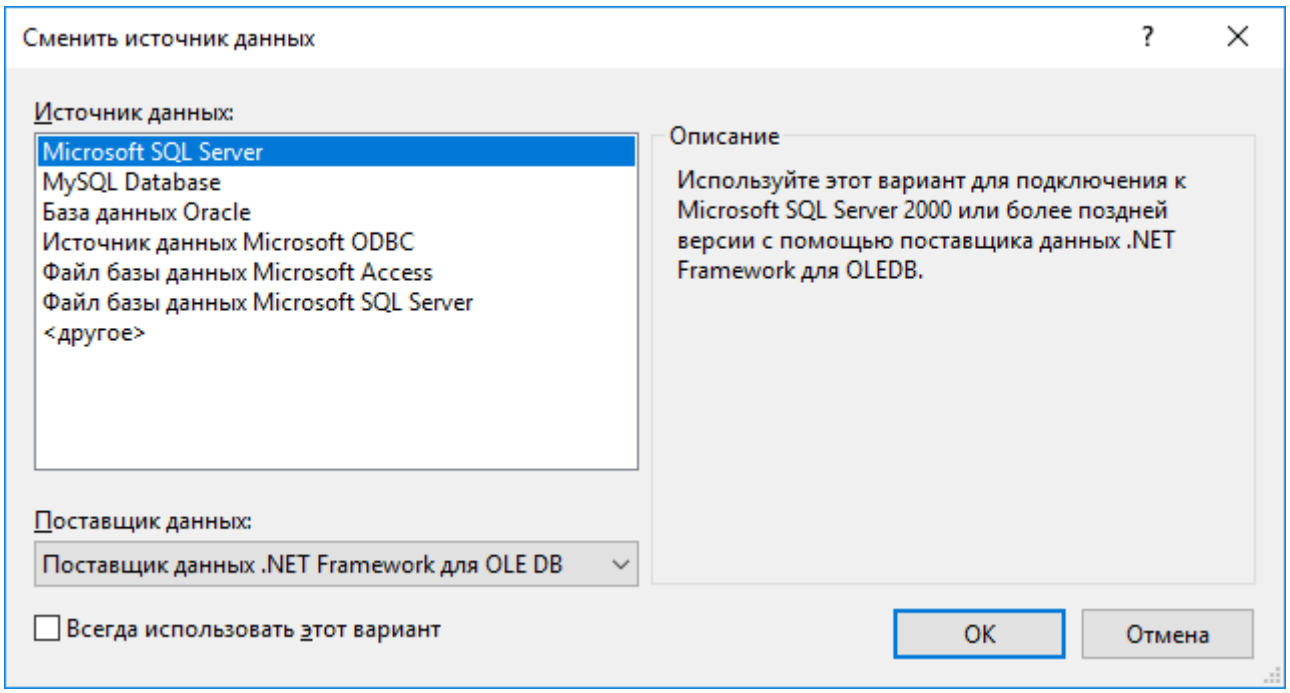


Рисунок 3 –Выбор поставщика данных

При нажатии на кнопку <ОК> происходит возврат к окну добавления подключения <Connection>, содержимое которого несколько изменяется в зависимости от выбора провайдера.

Теперь необходимо указать имя сервера, имя пользователя и способ авторизации для защищенных баз данных, таких способа всего два: проверка подлинности Windows или проверка подлинности самого сервера, в первом случае, авторизацию на себя берет операционная система Windows и пароль указывать нужно в соответствии с настройками пользователя системы, во втором

случае, необходимо будет указать и отдельный пароль для подключения к серверу MS Server, последним выберите имя базы данных, введя его вручную или выбрав из всплывающего списка. Нажав кнопку <Test Connection>, можно проверить правильность функционирования соединения, в случае успеха (рисунок 4) или неудачи возникнет новое модельное окно с соответствующим сообщением.

Основные параметры строки подключения к MS SQL Server БД:

– **Data Source** – указывает имя экземпляра SQL Server, к которому нужно подключиться

– **Initial Catalog** – параметр, указывающий на имя базы данных на сервере, к которой нужно подключиться

– **Integrated Security** – позволяет использовать для подключения к серверу данные учетной записи Windows или имя входа SQL Server.

– **User Id** – позволяет указать имя входа SQL Server для подключения к серверу

– **Password** – пароль имени входа SQL Server

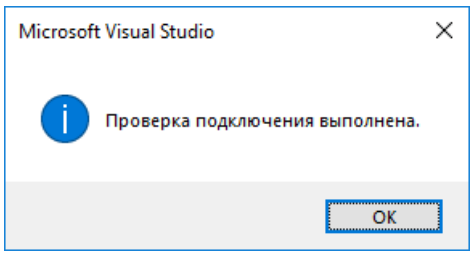


Рисунок 4 – Всплывающее модальное окно с сообщением о результате успешной проверки

Нажав кнопку <Advanced> (Дополнительно) откроется окно расширенных свойств подключения, в котором задается уровень защиты при сетевом доступе к базе данных; задается предельное время ожидания соединения в секундах; задается перечень допустимых операций: <Read> – только чтение; <ReadWrite> – чтение и запись; <Share Deny None> – нет запрета на чтение и запись; <Share

Deny Read> – запрещено открытие для чтения; <Share Deny Write> – запрещено открытие для записи; <Share Exclusive> – эксклюзивное (монопольное) использование; <Write> – только запись. Т.е. устанавливаются уровни изоляции «грязного чтения».

В случае использования компонента <ADOConnection> для активации соединения после настройки достаточно установить свойству Connected этого компонента значение True или при выполнении приложения выбрать метод Open.

В случае использования любого из компонентов доступа к данным (<ADODataSet>, <ADOTable>, <ADOQuery> и <ADOStoredProc>) для активизации соединения после настройки используют свойство <Active>.

Для обеспечения доступа к таблицам хранилищ данных по технологии ADO служит компонент <ADOTable>. Для установки соединения с хранилищем данных этого компонента через провайдеров ADO служит свойство <ConnectionString> или <Connection>. Для управления набором данных таблицы в приложение включает компонент источника данных <DataSource>. При этом свойству <DataSet> этого компонента в качестве значения задается имя компонента <ADOTable>. Для отображения данных таблицы к источнику данных подключаются различные компоненты отображения, к примеру, <DataGridView>.

После создания строки подключения можно создавать экземпляр SqlConnection:

```
string conStr = @"Data Source = .\SQLEXPRESS; Initial Catalog =
BookShopDB; Integrated Security = True";
SqlConnection connection = new SqlConnection(conStr);
```

Предварительно необходимо подключить ссылку на библиотеку с данными SQL клиента:

```
using System.Data.SqlClient;
```

Для открытия соединения с сервером у объекта **SqlConnection** есть метод Open(), а для закрытия – метод Close().

Строку подключения удобно хранить в конфигурационном файле, который легко отредактировать в случае необходимости, для этого необходимо воспользоваться объектом:

```
var setting = new ConnectionStringSettings{
Name = "MyConnectionString1",     //имя строки подключения в конфигурационном файле
ConnectionString = @"Data Source=.\SQLEXPRESS; Initial Catalog=BookShopDB; Integrated Security=True;"};
Configuration config;  // Объект Config представляет конфигурационный файл
config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);  // Объект ConfigurationManager предоставляет доступ к файлам конфигурации
config.ConnectionStrings.ConnectionStrings.Add(setting);
config.Save();
```

После установления связи компонента <ADOTable> с хранилищем данных с помощью свойства <TableName> типа <WideString> задается имя таблицы. Не все провайдеры ADO допускают непосредственный доступ к таблицам, поэтому может потребоваться доступ с помощью SQL-запроса. Вариант доступа к данным таблицы определяет свойство <TableDirect> типа <Boolean>. Если оно имеет значение «False» (по умолчанию), то компонент <ADOTable> автоматически генерирует SQL-запрос для доступа к таблице, в противном случае выполняется непосредственный доступ к данным таблицы.

Для осуществления транзакций над данными (добавления, удаления, изменения и др.) используется компонент <BindingNavigator>, в свойстве <DataSource> которого выбираем <DataSource1> (т.е. выбранное имя источника данных).

Управление записями, выведенными в поле просмотра, осуществляется с помощью навигатора < BindingNavigator >, который состоит из следующих кнопок (соответственно слева направо) (рисунок 5):

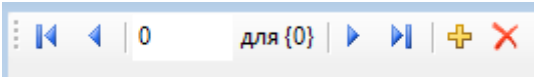


Рисунок 5 – Компонент BindingNavigator

1. Переход к первой записи.
2. Переход к предыдущей записи.
3. Переход к следующей записи.
4. Переход к последней записи.
5. Добавить запись.
6. Удалить запись.

А также еще 4-х кнопок, требующих дополнительных настроек:

7. Изменить запись.
8. Сохранить изменения.
9. Не сохранять изменения.
10. Обновить.

15.2 Создание базы данных

Рассмотрим создание приложения-клиента в среде программирования Visual Studio к базе данных интернет-магазина по продаже книг «BookShopDB».

1. Создайте базу данных интернет-магазина по продаже книг <BookShopDB>, которая состоит из восьми таблиц («Books», «Authors», «Deliveries», «PublishHouse», «Orders», «OrderIremList», «AuthorList», «Clients»):

Таблица 1 – Books

Название поля	Тип поля	Описание поля
BookID	int	Код закупаемой книги
TitleBookName	nvarchar	Название книги
Pages	int	Количество страниц
AuthorListID	int	Код списка авторов
PublishID	int	Код издательства
Price	money	Цена за книгу

screen	image	Изображение книги
screen_format	varchar	Поле для хранения типа формата изображения

Таблица 2 – Authors

Название поля	Тип поля	Описание поля
AuthorID	int	Код автора
FirstName	nvarchar	Имя автора
MiddleName	nvarchar	Отчество автора
LastName	nvarchar	Фамилия автора
BirthDate	date	Дата рождения автора

Таблица 3 – Deliveries

Название поля	Тип поля	Описание поля
DeliveriesID	int	Код поставщика
DeliveriesFirstName	nvarchar	Имя ответственного лица
DeliveriesMiddleName	nvarchar	Отчество ответственного лица
DeliveriesLastName	nvarchar	Фамилия ответственного лица
DeliveriesCompanyName	nvarchar	Название компании-поставщика
AddressToDeliveries	nvarchar	Юридический адрес
PhoneNumber	char	Контактный телефон
INN	char	ИНН

Таблица 4 – PublishHouse

Название поля	Тип поля	Описание поля
PublishID	int	Код издательства
TitlePublishName	nvarchar	Название издательства
TitleCityName	nvarchar	Город издательства

Таблица 5 – Orders

Название поля	Тип поля	Описание поля
OrderID	int	Код заказа
ClientID	int	Код клиента
OrderItemListID	int	Код списка заказанных книг (из одного заказа)
OrderDate	datetime	Дата и время заказа
TotalPrice	money	Итоговая сумма заказа
Amount	int	Количество книг в заказе
DeliveriesID	int	Код поставщика

Таблица 6 – OrderIremList

Название поля	Тип поля	Описание поля
ItemListID	int	Код списка заказанных книг (из одного заказа)
AmountOfItems	int	Количество книг в заказе
OrderID	int	Код заказа

Таблица 7 – AuthorList

Название поля	Тип поля	Описание поля
AuthorListID	int	Код списка авторов
AuthorID	int	Код автора

Таблица 8 – Clients

Название поля	Тип поля	Описание поля
ClientID	int	Код клиента
ClientFirstName	nvarchar	Имя клиента
ClientMiddleName	nvarchar	Отчество клиента
ClientLastName	nvarchar	Фамилия клиента
ClientBirthDate	date	Дата рождения клиента
ClientPhoneNumber	char	Номер телефона клиента
ClientEmail	nvarchar	Электронная почта клиента
ClientAddress	nvarchar	Почтовый адрес клиента

2. Свяжите их согласно представленной диаграмме (рисунок 6):

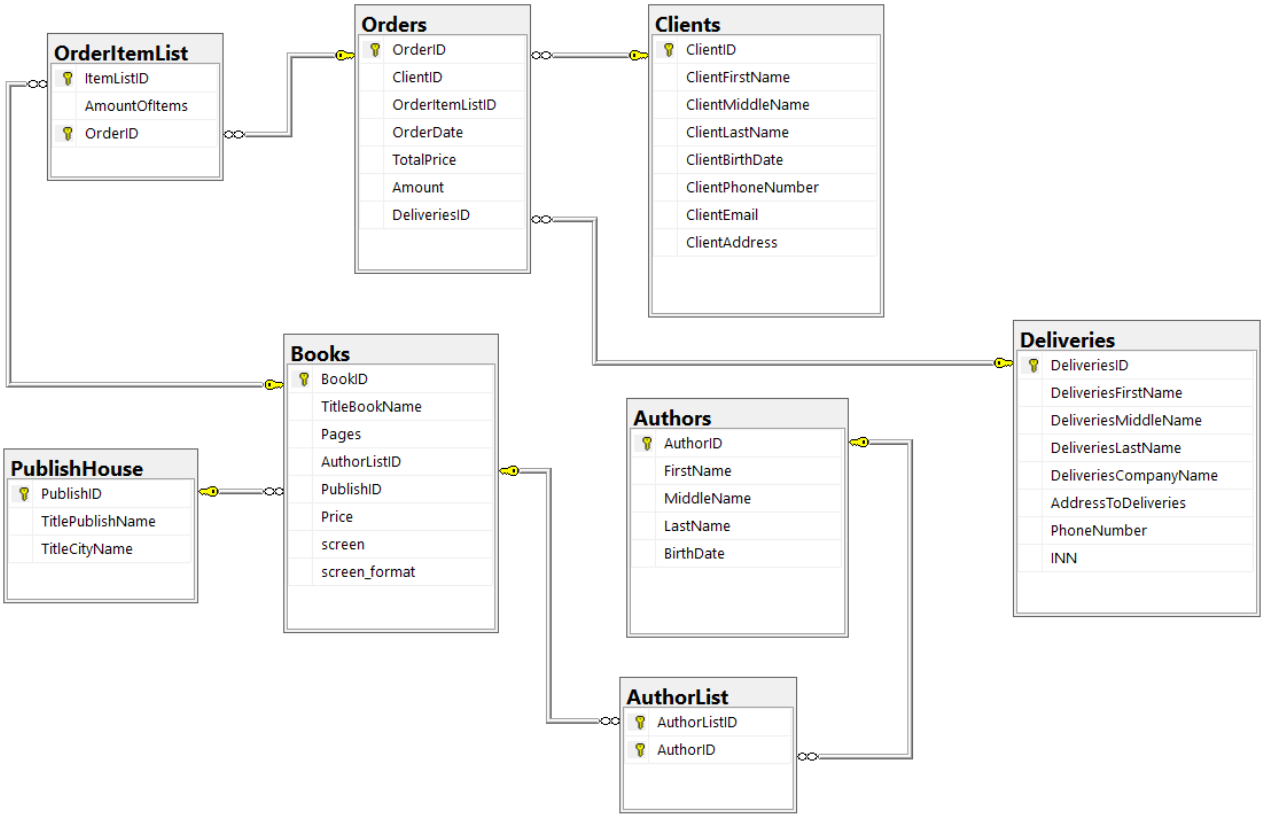


Рисунок 6 – Диаграмма базы данных книжного магазин

3. Заполните поля в таблицах данными.

4. Создайте хранимую процедуру для определения количества заказов, совершенных за указанный период:

CREATE PROC spCountOfOrders @d1 DATETIME, @d2 DATETIME, @c INT OUTPUT AS SELECT @c=count(OrderID) from Orders WHERE OrderDate BETWEEN @d1 AND @d2 SET @c=ISNULL(@c,0)

5. Создайте хранимую процедуру для определения списка заказанных книг для определенного заказа:

CREATE PROC ListOrdersSelectCommand (@OrderID int) AS SET NOCOUNT ON; select bk.BookID, bk.TitleBookName, bk.Price from Orders as ord join OrderItemList as orl on ord.OrderItemListID = orl.OrderID and ord.OrderID = @OrderID join Books as bk on bk.BookID = orl.ItemListID

6. Создайте хранимую процедуру для определения списка всех авторов одной книги:

CREATE PROC [dbo].[spBookAuthorsList] @ListID int OUTPUT AS begin select distinct al.AuthorID, TitleBookName as bookname, left(ars.FirstName,1) + '. ' + left(ars.MiddleName,1) + '. '+ars.LastName as Authors from books bk join AuthorList al on bk.AuthorListID = al.AuthorListID and al.AuthorListID = @ListID join Authors ars on ars.AuthorID = al.AuthorID end

7. Создайте хранимую процедуру для объединения ФИО автора книги книги:

CREATE PROC spListAllAuthors AS select Authors.AuthorID, left(Authors.FirstName,1) + '. ' + left(Authors.MiddleName,1) + '. ' + Authors.LastName as AuthorName from Authors

8. Создайте хранимую процедуру для определения списка книг в конкретном заказе:

PROC spListOfOrders @ParamOrderID int OUTPUT AS begin select bk.BookID, bk.TitleBookName, bk.Price, orl.AmountOfItems from Orders as ord join OrderItemList as orl on ord.OrderItemListID = orl.OrderID and ord.OrderID = @ParamOrderID join Books as bk on bk.BookID = orl.ItemListID end

15.3 Создание клиентского приложения

При создании клиентского приложения будем использовать встроенные инструменты для работы с данными.

Создадим приложение типа Windows Forms (.NET Framework) и назовем его BookShopDB, для этого выберем в главном меню Visual Studio пункт создания нового проекта (рисунок 7).

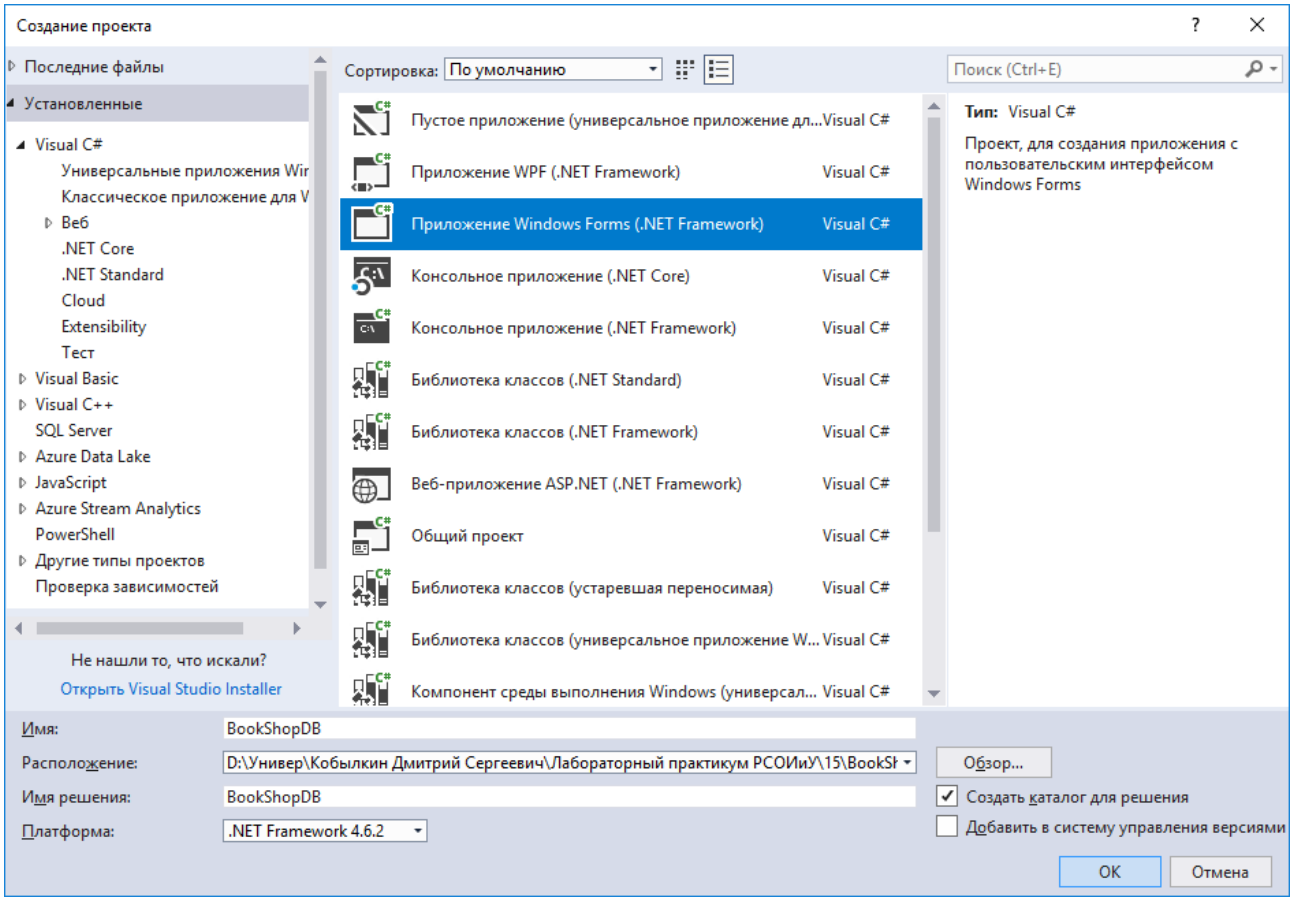


Рисунок 7 – Создание проекта

15.4 Добавление компонентов на форму

Используя окно *панели элементов*, добавьте на форму (<Form1>) следующие компоненты, расположенные на указанных вкладках:

– <DataSet> (вкладка <Данные>);

– <ADOConnection> (добавится автоматически при добавлении подключения к источнику данных через <DataSet>);

– <MenuStrip> (вкладка <Меню и панели инструментов>);

– <Panel> (вкладка контейнеры) на который установить <DataGridView> (вкладка <Данные>) и <BindingNavigator> (вкладка <Данные>);

– 3 компонента <GroupBox> (вкладка <Контейнеры>);

– 7 компонентов <Lable> (вкладка <Стандартные элементы управления>);

– 5 компонентов <TextBox> (вкладка <Стандартные элементы управления>);

– <ComboBox> (вкладка <Стандартные элементы управления>);

– 4 компонента <Button> (вкладка <Стандартные элементы управления>);

– <Panel> (вкладка контейнеры) на который установить PictureBox (вкладка <Стандартные элементы управления>).

Расположите компоненты на форме следующим образом (рисунок 8):

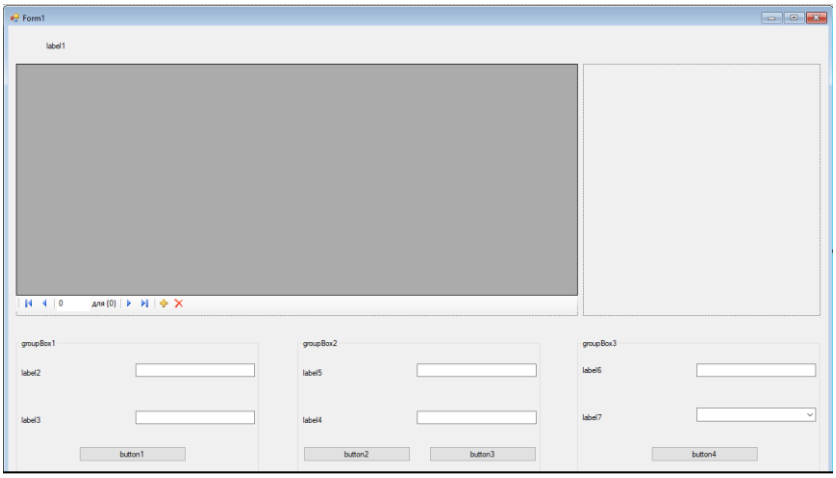


Рисунок 8 - Расположение необходимых компонентов на главной форме клиентского приложения

У компонента <DataGridView1> настройте следующее свойство: <Dock> - нажать кнопку «Вверх», у компонента <BindingNavigator> выберите это же свойство и установите значение «Вниз».

15.5 Выбор источника данных

У компонента <DataGridView1> найдите группу свойств «Данные», выберите свойство <DataSource> и выберите «Добавить источник данных проекта» как на рисунке 9:

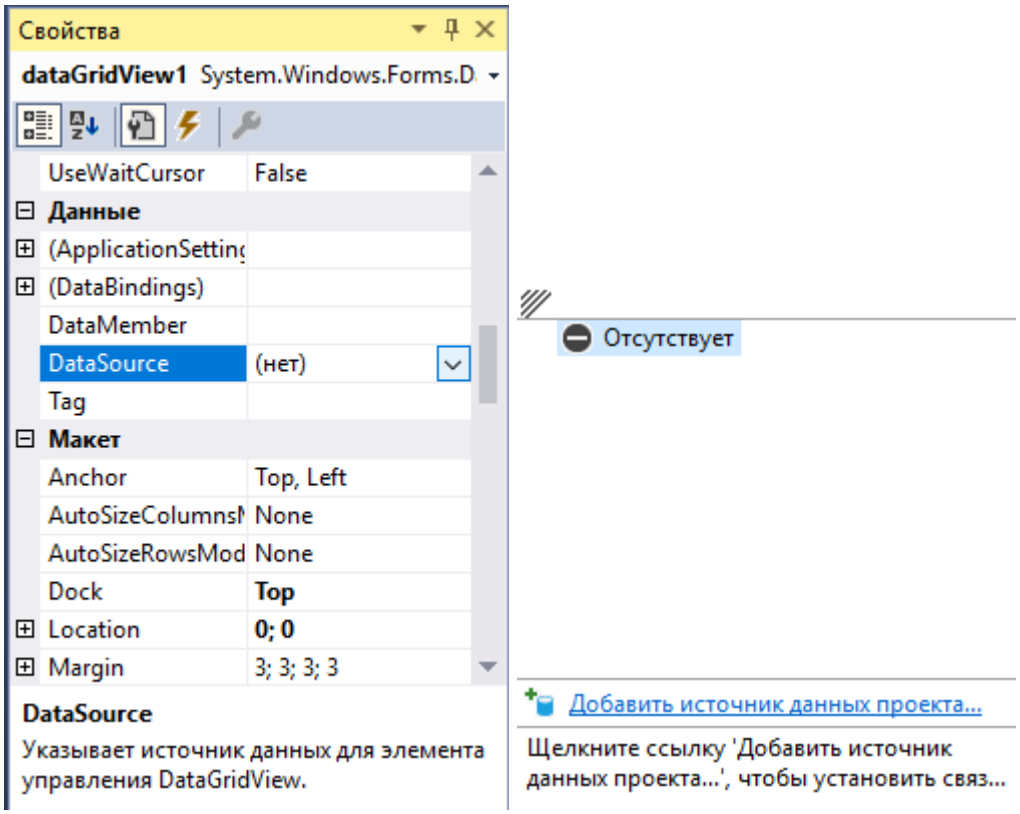


Рисунок 9 – Выбор источника данных

В появившемся диалоговом окне выберите пункт «База данных» и нажмите кнопку «Далее» (рисунок 10), в следующем окне выберите «Набор данных», снова нажмите кнопку «Далее» (рисунок 11). Затем выберите подключение из созданного ранее или создайте новое (рисунок 12), нажмите кнопку «Далее». В следующем окне выберите имя строки подключения, затем нажмите кнопку

«Далее» (рисунок 13). В появившемся окне, выберите объекты из базы данных, которые нужно подключить в качестве источника данных (рисунок 14) и задайте имя источника данных, затем нажмите кнопку «Готово».

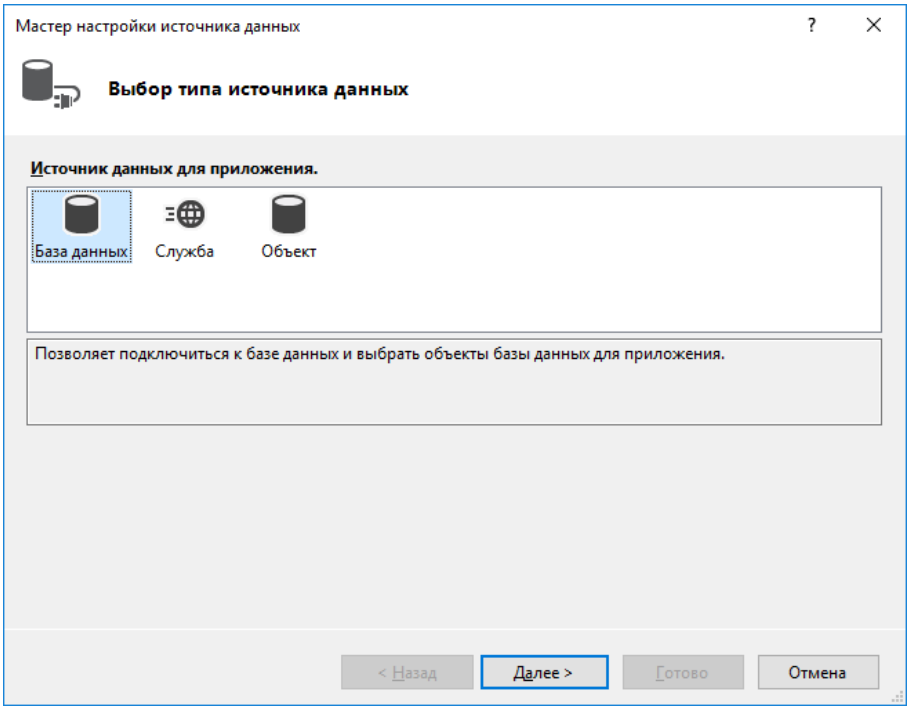


Рисунок 10 – Выбор типа источника данных

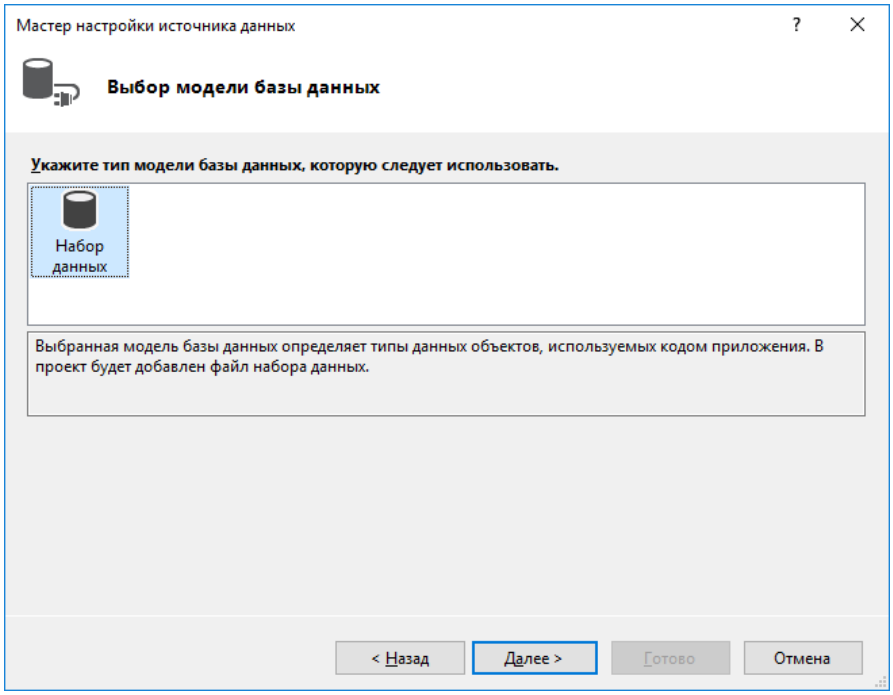


Рисунок 11 – Выбор модели базы данных

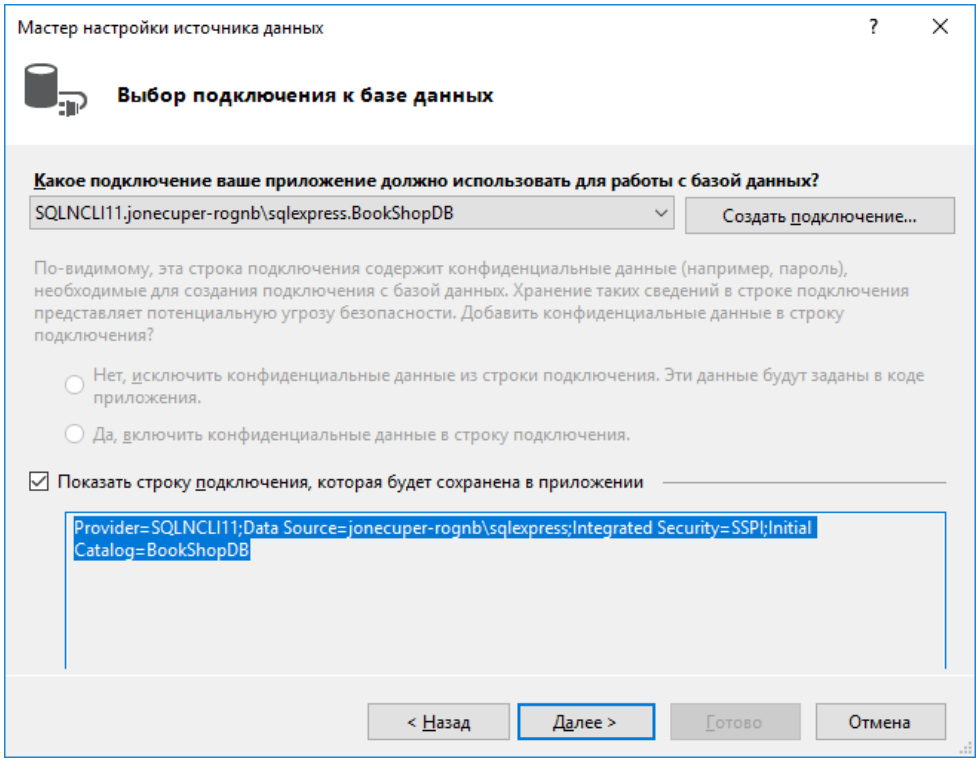


Рисунок 12 – Выбор подключения к базе данных

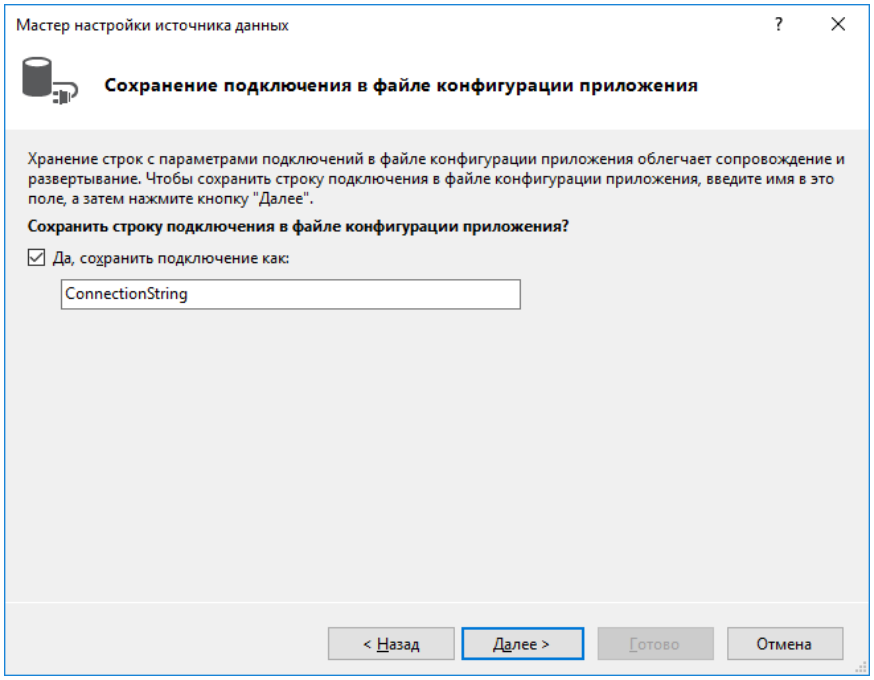


Рисунок 13 – Сохранение подключения в конфигурационном файле

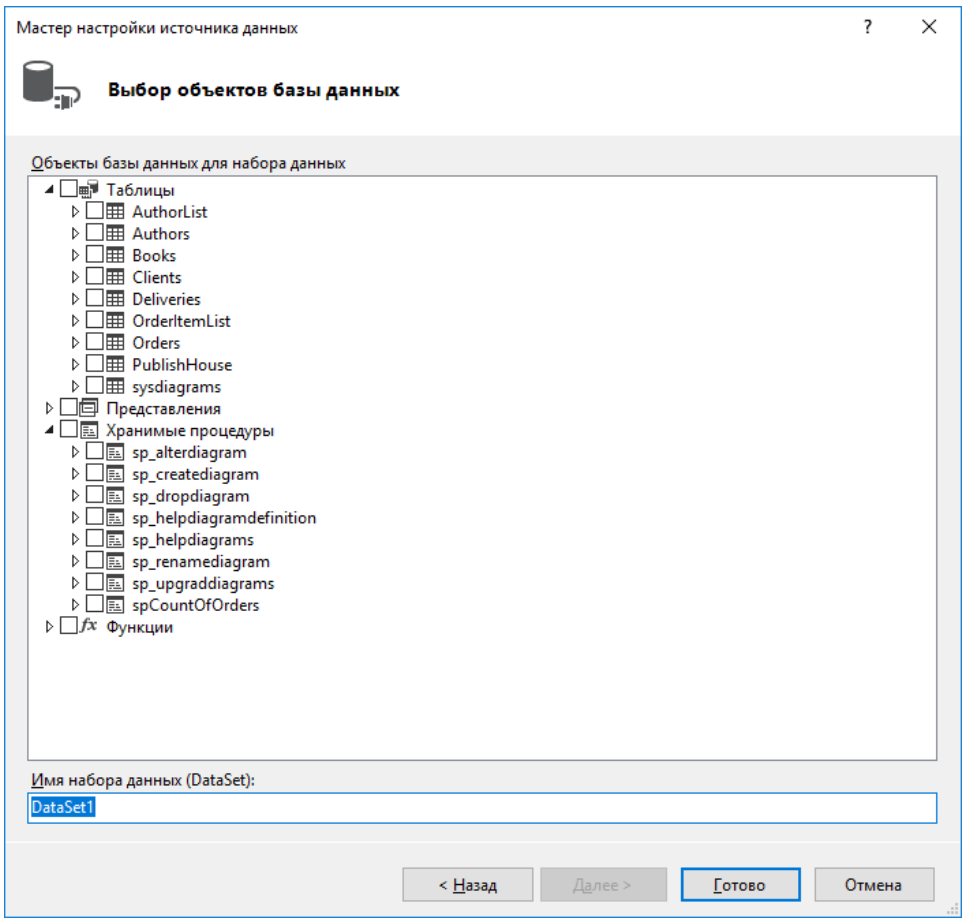


Рисунок 14 – Выбор объектов базы данных

15.5 Добавление меню и первичная настройка элементов главной формы

Добавьте меню для приложения как на рисунке 15:

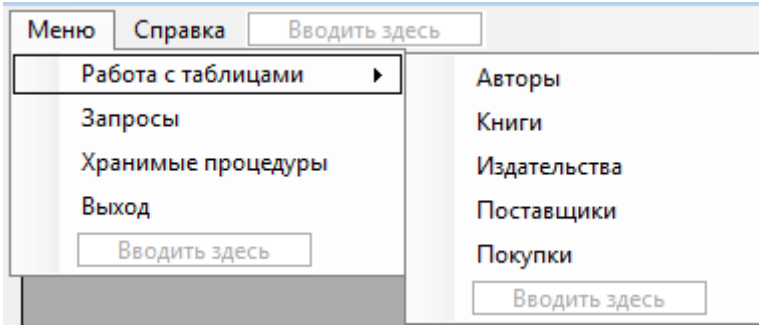


Рисунок 15 – Создание меню

– В меню <Справка> создайте подменю <Помощь> и <О программе>.

– У компонента <Lable1> измените свойство <Text> на «Нет информации для отображения», поменяйте размер шрифта в свойстве <Font>.

– У компонента <GroupBox1> измените свойство <Text> на «Поиск», у <GroupBox2> – на «Фильтр», у <GroupBox3> – на «Сортировка».

– У компонентов <Lable2>, <Lable5>, <Lable6> измените свойство <Text> на «Название поля», у <Lable3> и <Lable4> – на «Значение», у <Lable7> – на «Тип сортировки».

– У компонентов <TextBox1>, <TextBox2>, <TextBox3>, <TextBox4>, <TextBox5> и <ComboBox1> измените свойство <Text> на пустое значение.

– У компонента <Button1> измените свойство <Text> на «Поиск», у <Button2> – на «Снять фильтр», у <Button3> – на «Ok», у <Button4> – на «Сортировка».

– У компонента <ComboBox1> настройте свойство <Items>: добавьте 2 линии – <По возрастанию> и <По убыванию>.

Выровняйте компоненты. Добавьте еще одну кнопку, измените ее свойство <Text> как на рисунке, запустите компиляцию (нажмите на кнопку <Пуск> или <F5>). Должна получиться ниже представленная форма (рисунок 16):

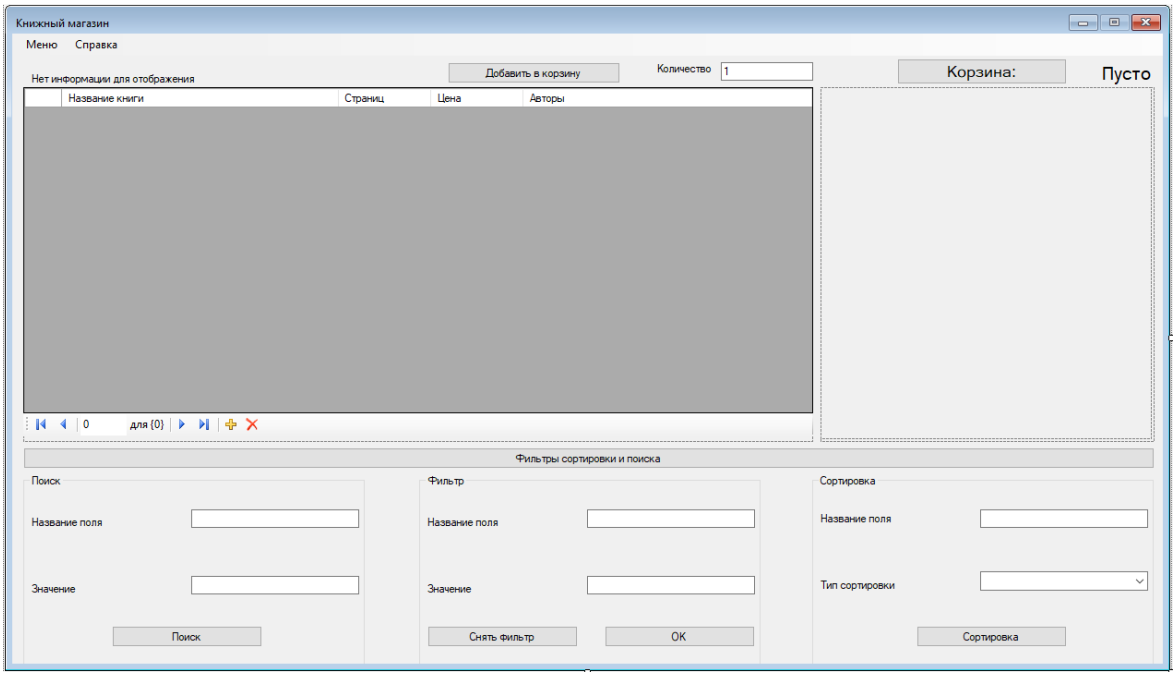


Рисунок 16 – Главная форма клиентского приложения

15.6 Создание остальных форм приложения

Далее создайте новые формы для представлений авторов, клиентов, добавления даты, поставщиков, книг, издательства, оформления и представления заказов. Для создания новой формы перейдите в окно обозревателя решений, выделите свой проект правой клавишей мыши и выберите пункт добавления новой формы (рисунок 17).

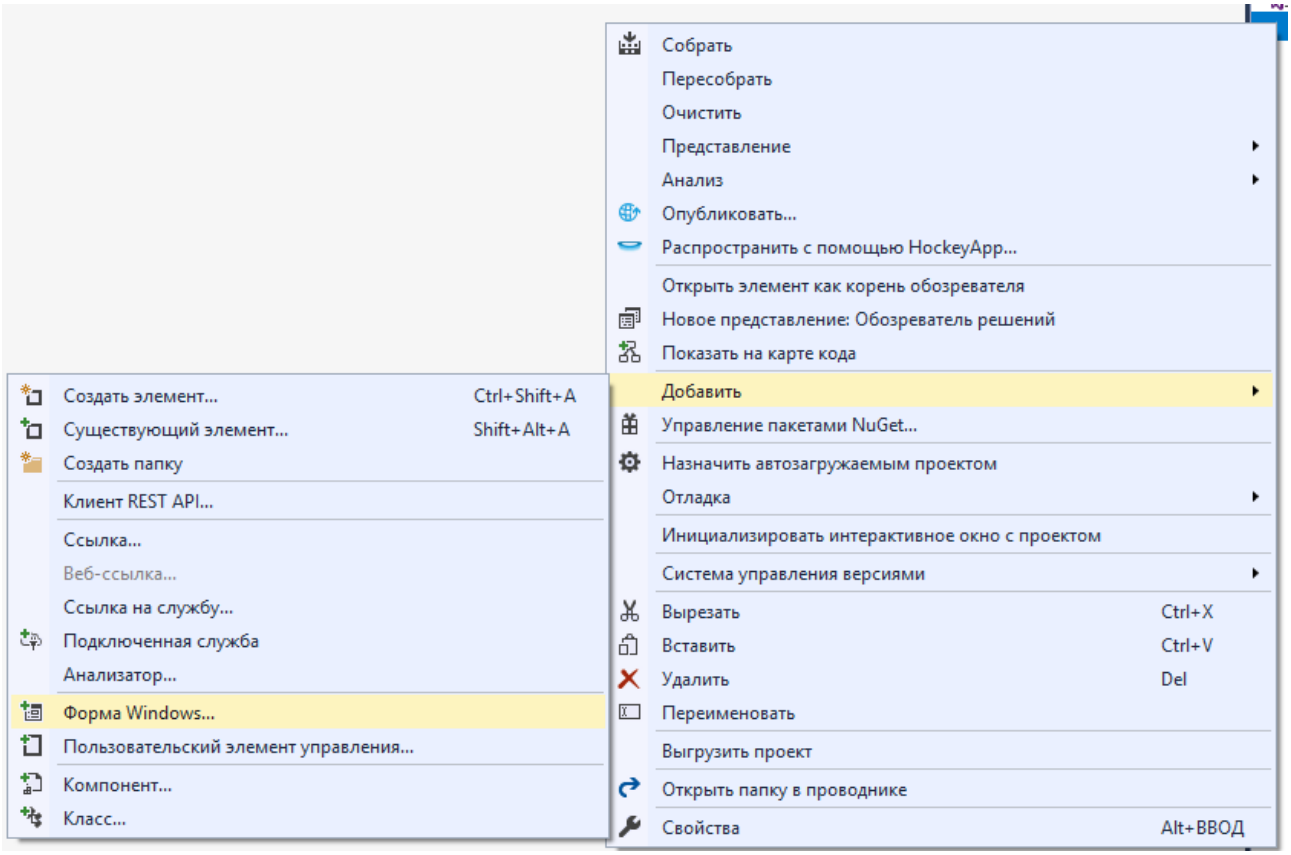


Рисунок 17 – Добавление новой формы

Добавьте новые формы FrmAuthors, FrmClients, FrmDate, FrmDeliveries, FrmEditBooks, FrmPublishHouse, FrmPurchases, FrmViewOrdersInfo (рисунки 18 - 26).

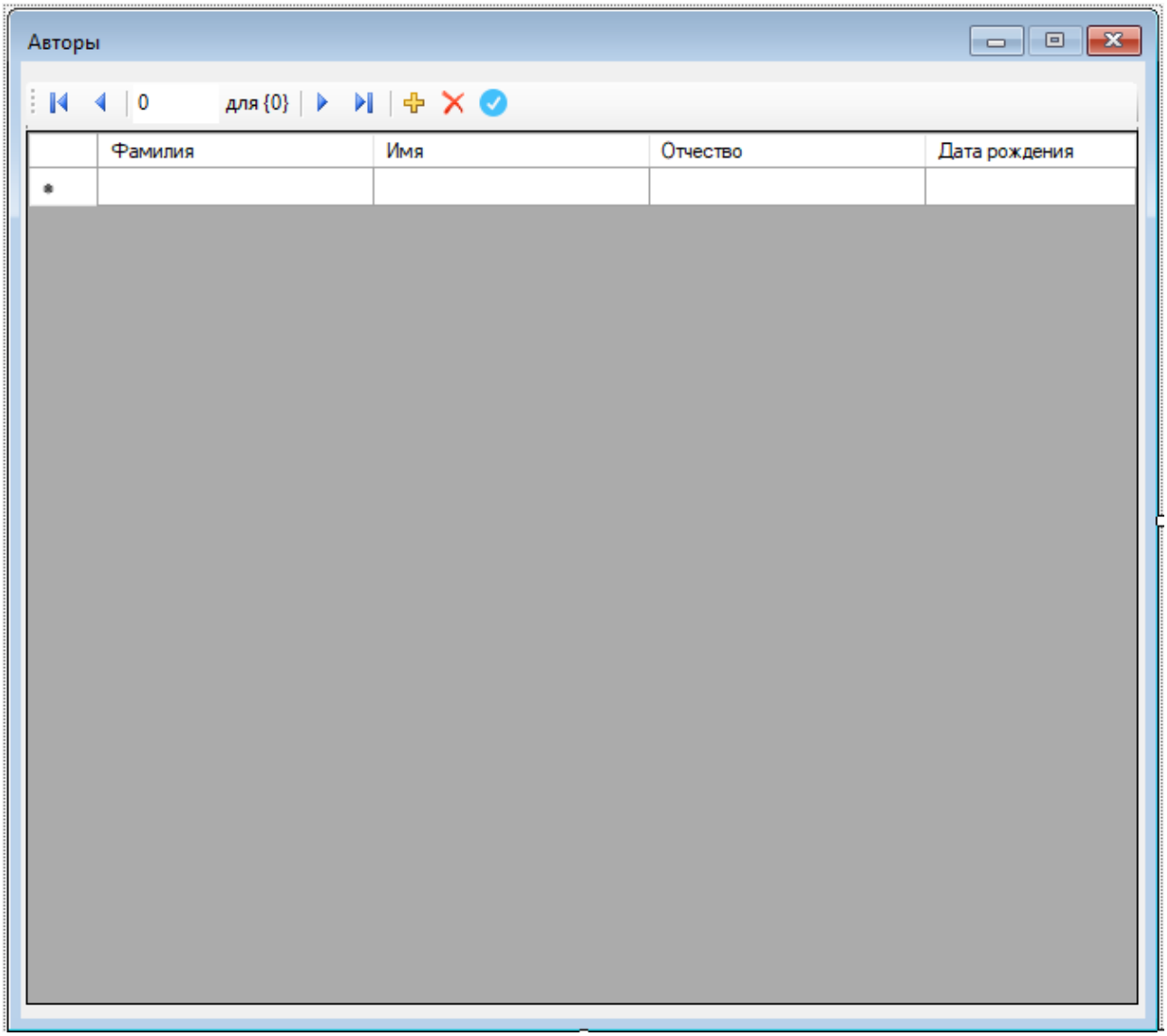


Рисунок 18 – Форма «Авторы» <FrmAuthors>

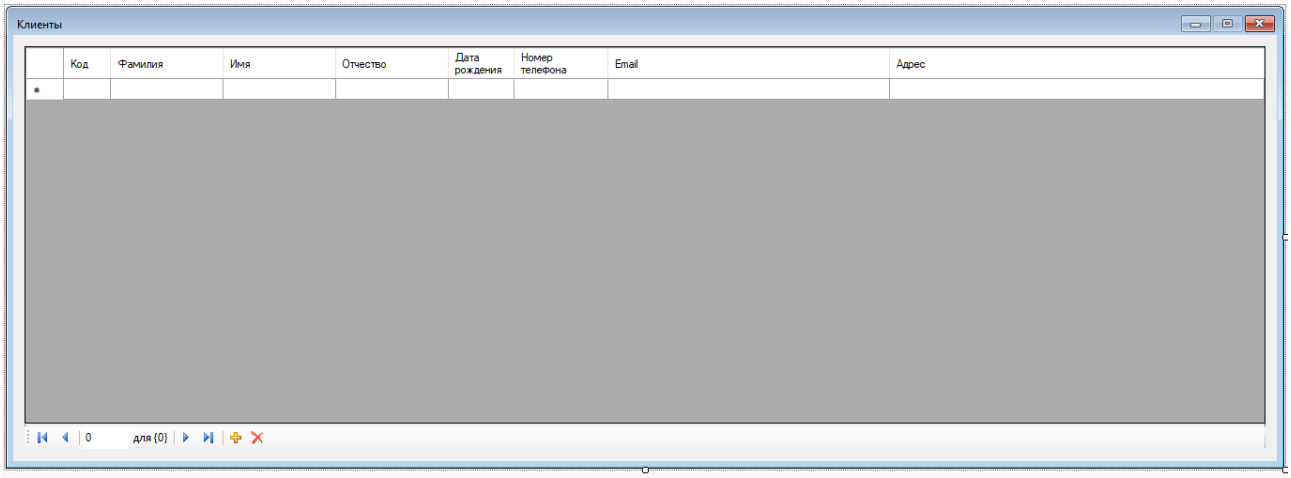


Рисунок 19 – Форма «Клиенты» <FrmClients>

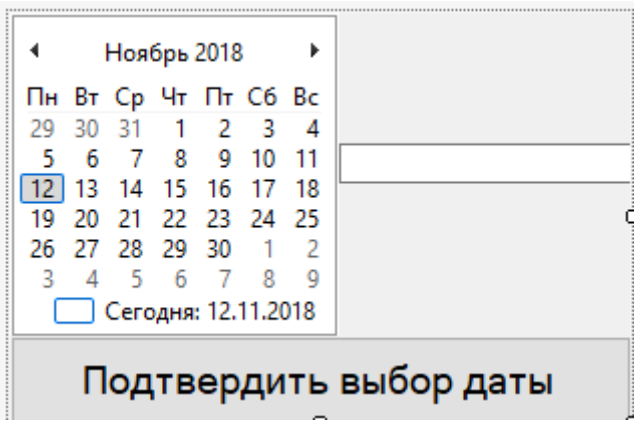


Рисунок 20 – Форма «Добавление даты» <FrmDate>

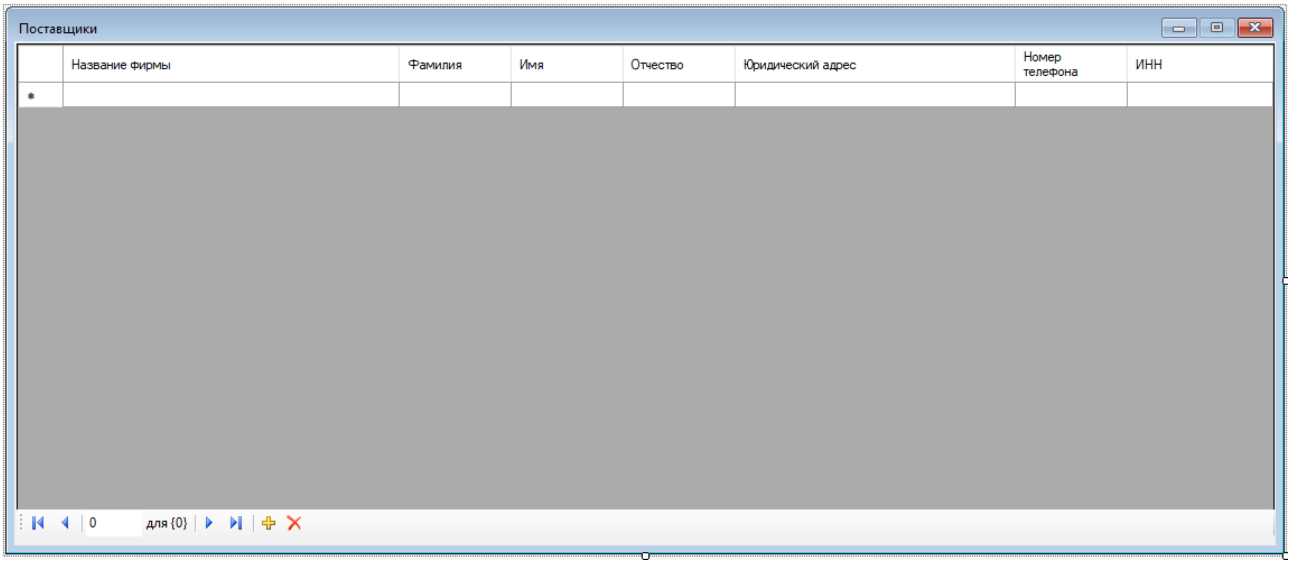


Рисунок 21 – Форма «Поставщики» <FrmDeliveries>

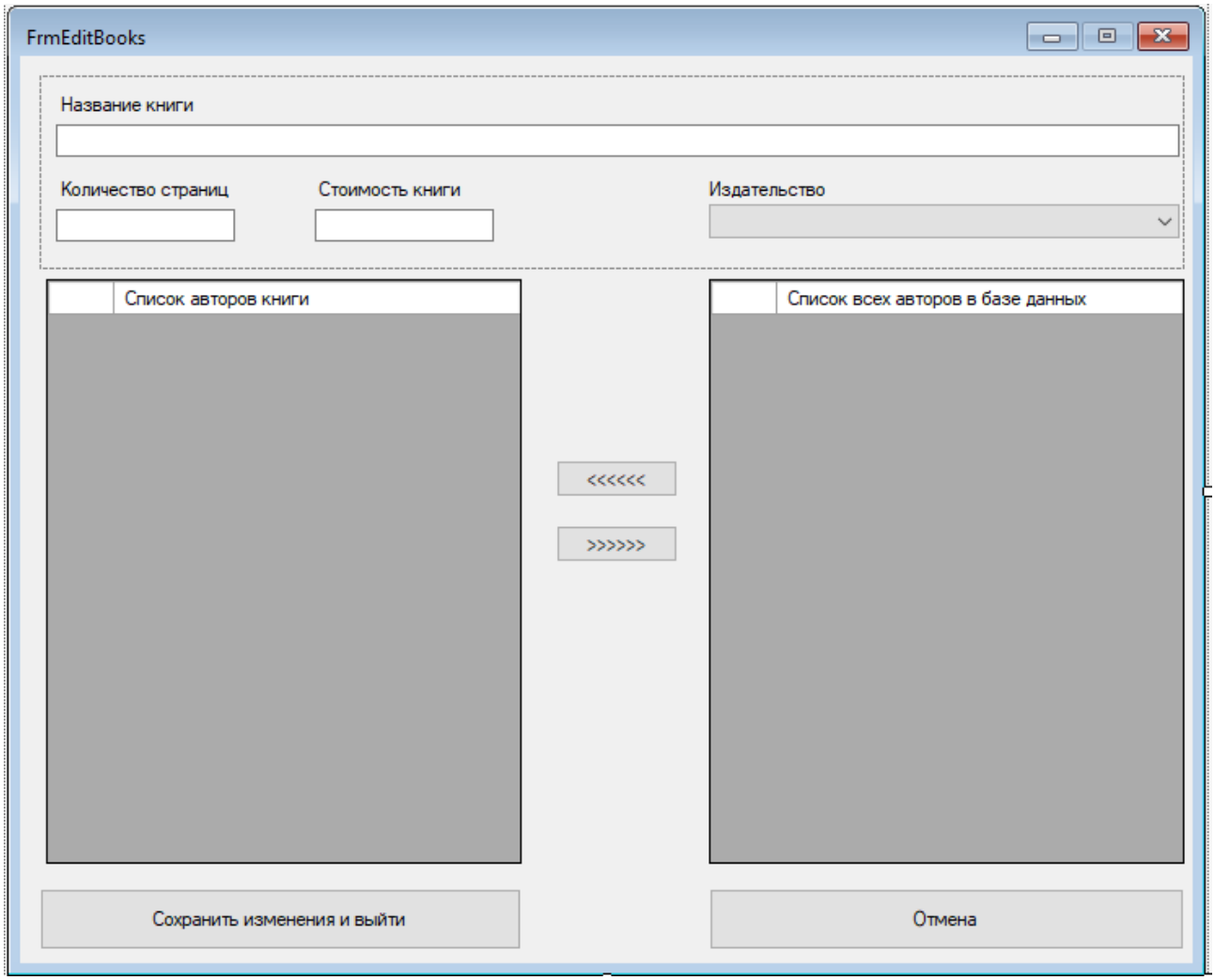


Рисунок 22 – Форма «Книги» (добавление или редактирование) <FrmEditBooks>

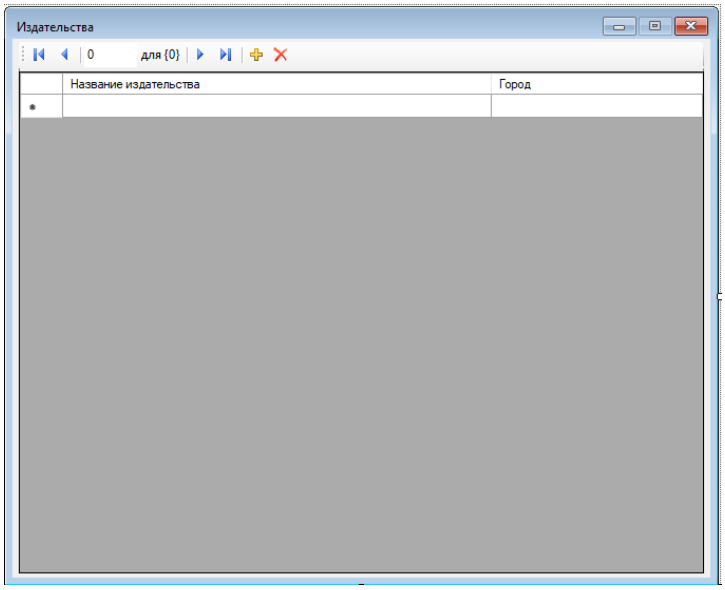


Рисунок 23 – Форма «Издательства» <FrmPublishHouse>

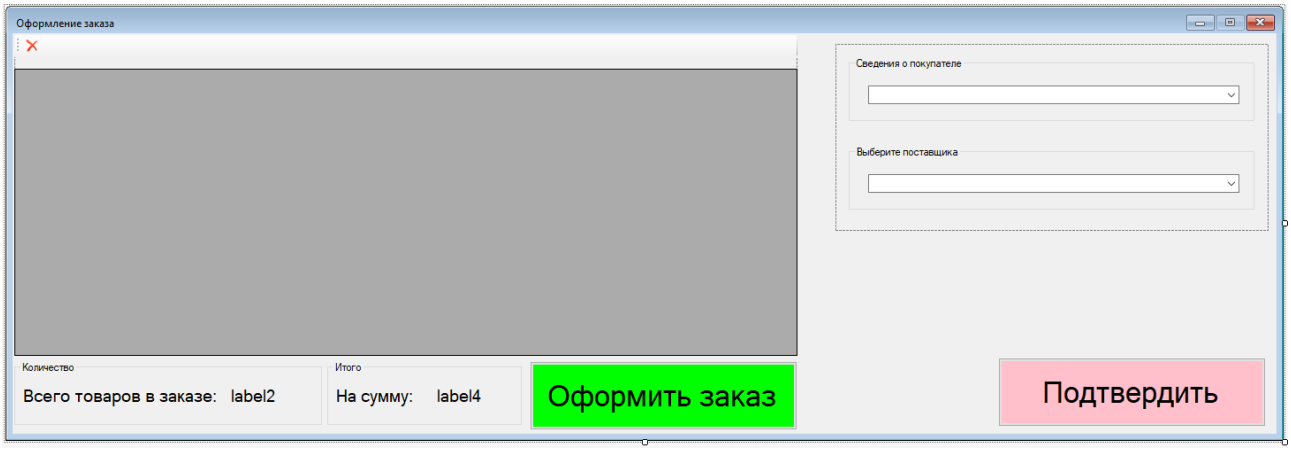


Рисунок 24 – Форма «Оформление заказа» <FrmPurchases>

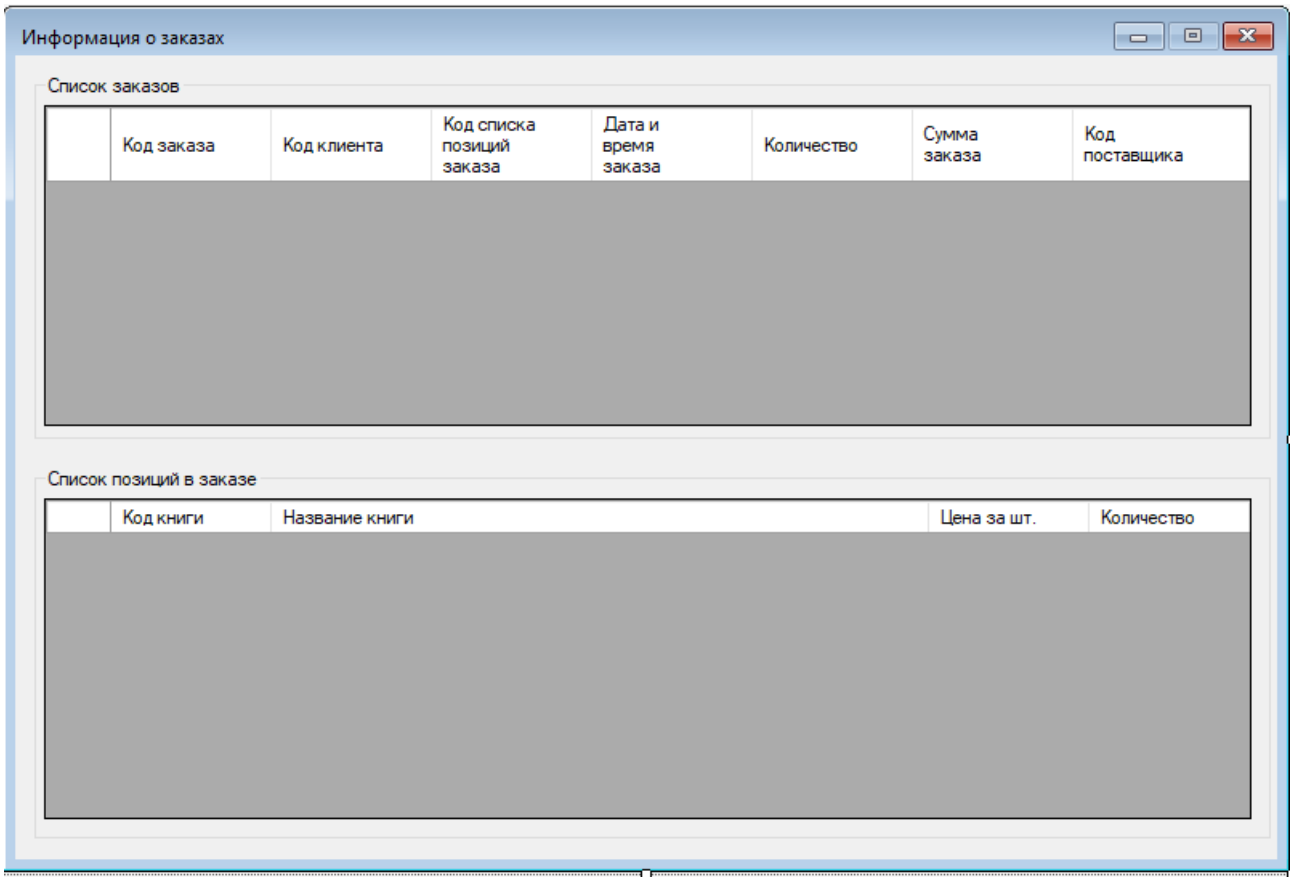


Рисунок 25 – Форма «Информация о заказах» <FrmViewOrdersInfo>

15.7 Настройка объекта «DataSet»

В обозревателе решений найдите и кликните дважды, созданный ранее <DataSet1> (рисунок 26).

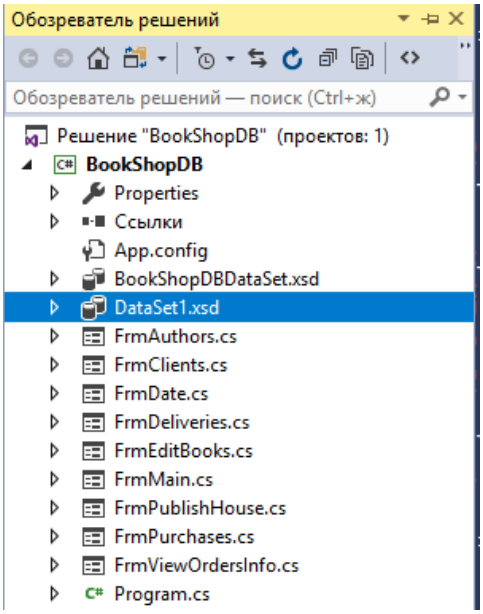


Рисунок 26 – Выбор <DataSet1> в обозревателе решений

Добавьте все созданные элементы из базы данных, для этого нажмите правой клавишей мыши в конструкторе диаграмм и выберите пункт «Добавить-TableAdapter» (рисунок 27).

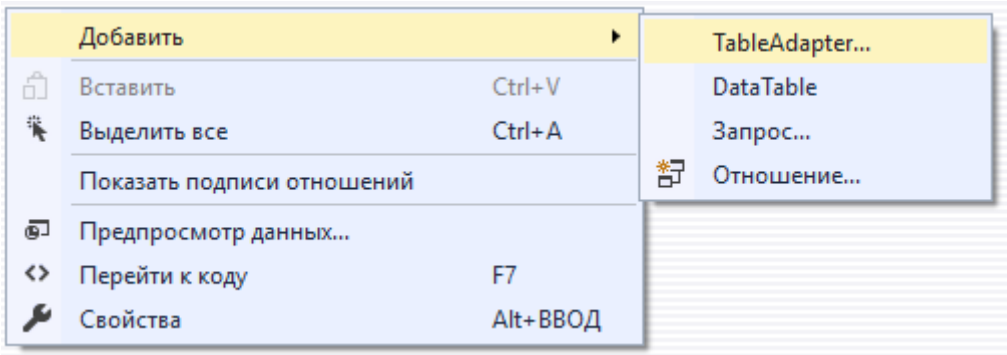


Рисунок 27 – Добавление нового <TableAdapter>

После добавления всех объектов окно конструктора должно принять такой вид (содержать следующие объекты таблиц и процедур из базы данных) как на рисунке 28.

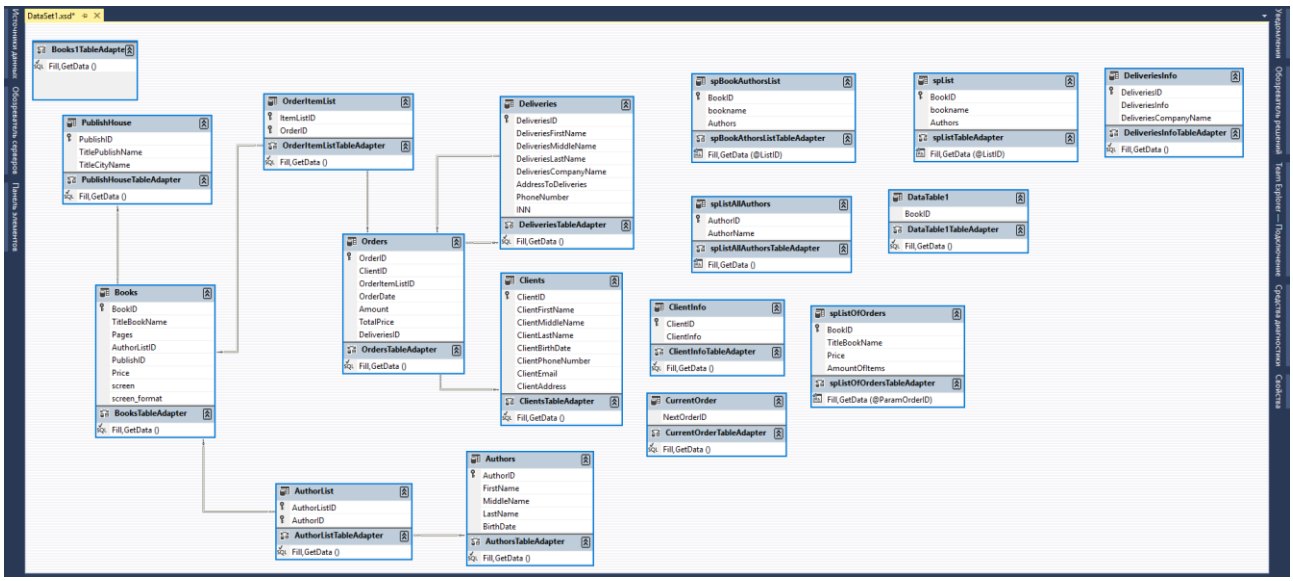


Рисунок 28 – Окно конструктора со схемой таблиц

Обратите внимание, что на схему автоматически добавились также и отношения между таблицами.

Выберите пункт «Сохранить все» в основном меню Visual Studio, после этого будут автоматически сгенерированы объекты для взаимодействия с данными из <DataSet1>. Закройте окно конструктора.

15.8 Настройка элементов главной формы «FrmMain»

Далее перейдите на главную форму <FrmMain>, выделите заголовок окна формы и нажмите клавишу F4, для того чтобы открыть свойства:

– найдите свойство <Text> и присвойте ему значение «Книжный магазин» – это заголовок окна;

– найдите свойство <FormBorderStyle> и установите ему значение <FixedDialog> из выпадающего списка;

– найдите свойство <StartPosition> и установите ему значение <CenterScreen>;

– у свойства <Name> установите имя формы как <FrmMain>;

– у свойства <MainMenuStrip> должно было автоматически установиться свойство <menuStrip1> – это меню, созданное ранее, если это свойство пустое, установите его значение вручную.

– у свойства <MaximizeBox> установите значение <False>;

– проделайте соответствующие операции для всех остальных форм.

15.9 Добавление обработчика события «Form_Load» для главной формы

Откройте главную форму <FrmMain> и дважды щелкните по заголовку окна формы, либо выбрав свойства формы (клавиша F4) прейдите на вкладку «События», найдите событие <FormLoad> и щелкните по нему левой кнопкой мыши дважды (рисунок 29) (это одно и тоже действие), среда Visual Studio автоматически сгенерирует обработчик события загрузки формы и откроет редактор кода, добавьте следующий код в этот обработчик:

```
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Height = 510;
            this.dataTable1TableAdapter.Fill(this.dataSet1.DataTable1);
            this.booksTableAdapter.Fill(this.dataSet1.Books);
            pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
            toolTip1 = new ToolTip();
            LoadAuthorsFromProcToTheList();
            this.dataGridView1.ColumnHeadersHeightSizeMode =
DataGridViewColumnHeadersHeightSizeMode.AutoSize;
       }
```

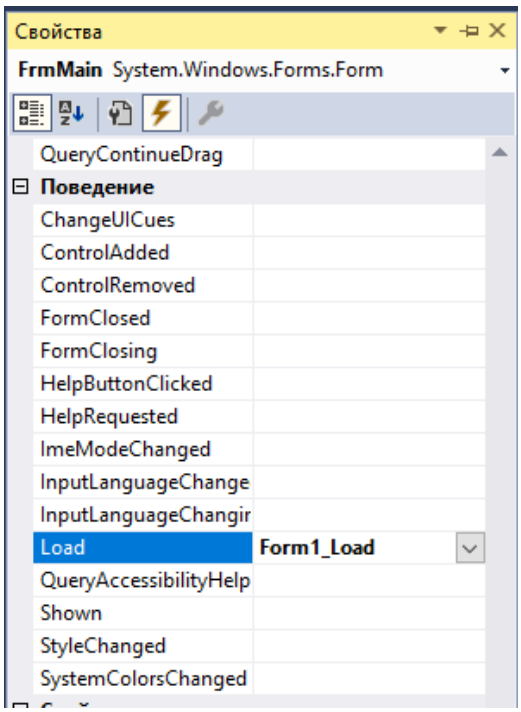


Рисунок 29 – Обработчики событий формы <FrmMain>

15.10 Настройка объекта «dataGtidView1»

Для объекта <dataGtidView1> установите следующие свойства:

– <DataSource> = booksBindingSource, для этого, выберите пункт DataSource и установите связь с соответствующей таблицей из созданного выше <DataSet1> (рисунок 30):

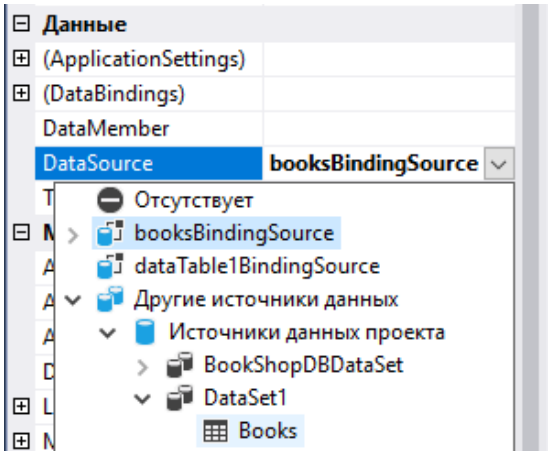


Рисунок 30 – Добавление источника данных

После этого будут сгенерированы экземпляры специальных объектов типа <DataBinding>, служащих для привязки данных.

– Щелкните по <dataGtidView1> правой клавишей мыши и выберите пункт «Правка столбцов», в появившемся окне выберите те элементы, которые должны будут отображаться в таблице, для этого установите свойство <Visible> в соответствующее положение (True или False). Затем измените заголовки столбцов таблицы, введя кириллицей соответствующие имена каждому столбцу для свойства <HeaderText>. А также установите ширину столбцов, оперируя значением свойства <Width> и свойства автоматической установки размера ширины столбца <AutoSizeMode> (рисунок 31).

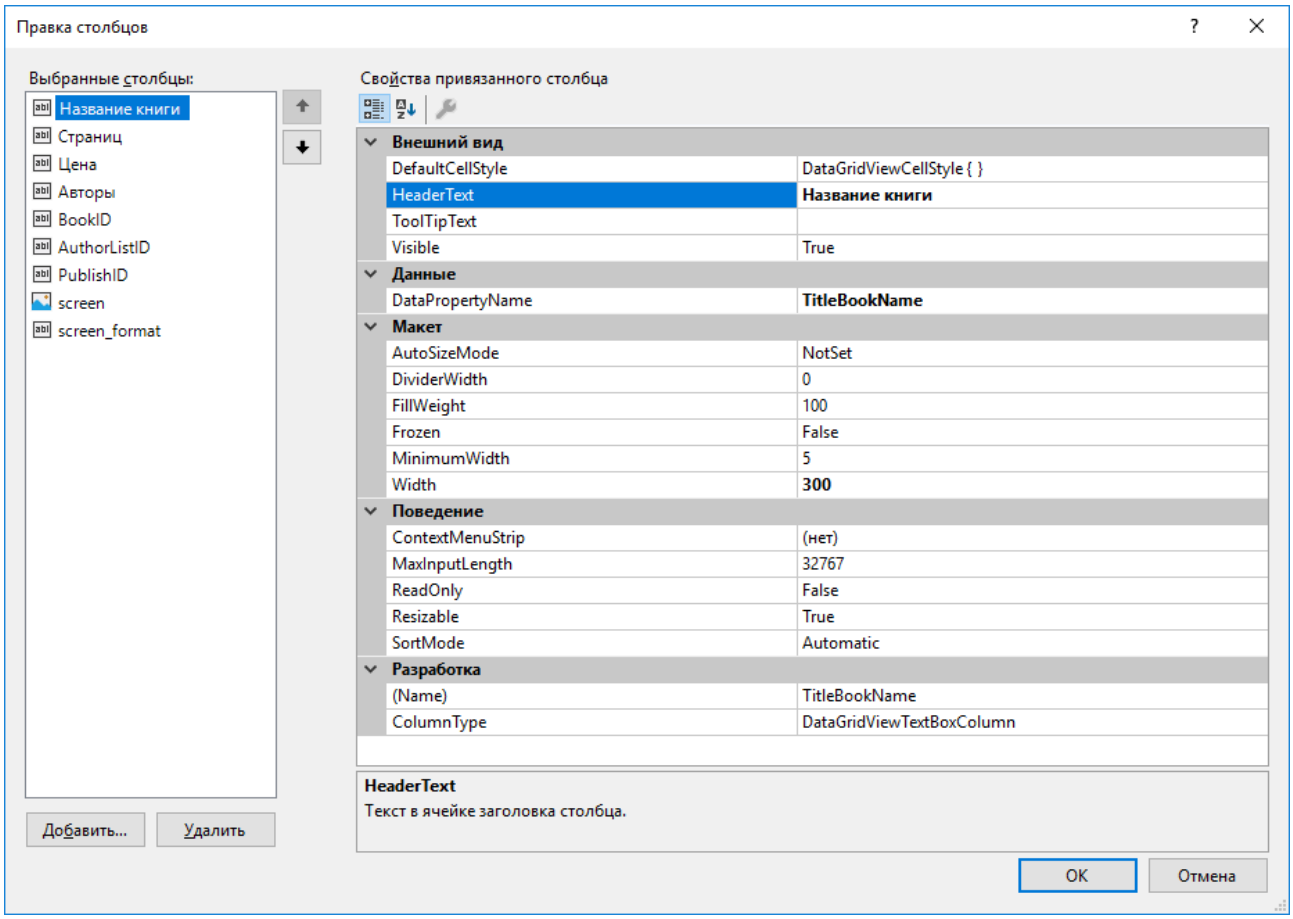


Рисунок 31 – Окно «Правка столбцов»

После изменения заголовков столбцов, имена в левой части окна будут заменены на русские.

Для <dataGtidView1> установите следующие свойства:

– <Dock> = <Top> – это привязка к текущему объекту, мы устанавливали <dataGtidView1> внутри панели <panel1>, соответственно теперь этот элемент был привязан к верхней части панели.

– <AllowUserToAddRows> = <False>, установка данного свойства говорит о том, что теперь пользователь не может автоматически добавлять новые строки нажатием горячих клавиш;

– <AllowUserToDeleteRows> = <False>, установка данного свойства говорит о том, что теперь пользователь не может автоматически удалять строки нажатием горячих клавиш;

– <ReadOnly> = <True>, установка данного свойства говорит о том, что теперь элементы таблицы доступны только для чтения, но не для редактирования;

– <SelectionMode> = <FullRowSelect>, установка данного свойства говорит о том, что теперь будет выделяться вся текущая строка целиком, а не одна ячейка таблицы;

Для объекта <bindingNavigator1> на форме <FrmMain> установите следующие свойства:

– <bindingSource> = <booksBindingSource> – это привязка объекта навигатора ко всем использующим этот же dataSource источник;

– <Dock> = <Top>.

Добавьте метод изменения сортировки по столбцам в событие «ColumnSortModeChanged»:

```
        private void dataGridView1_ColumnSortModeChanged(object sender,
DataGridViewColumnEventArgs e)
        {
            this.dataTable1TableAdapter.Fill(this.dataSet1.DataTable1);
            this.booksTableAdapter.Fill(this.dataSet1.Books);
            LoadAuthorsFromProcToTheList();
        }
```

Добавьте обработчик события «CellContentClick»:

```
        private void dataGridView1_CellContentClick(object sender, DataGridViewCellEventArgs
e)
        {
            this.dataTable1TableAdapter.Fill(this.dataSet1.DataTable1);
```

```
            this.booksTableAdapter.Fill(this.dataSet1.Books);
            LoadAuthorsFromProcToTheList();
        }
```

15.11 Настройка объекта «PictureBox»

Для объекта <pictureBox1> установите следующие свойства:

– <(DataBinding) Image> = <booksBindingSource>-<screen> – это привязка объекта <pictureBox1> к конкретному изображению с именем <screen>;

– Добавьте событие, которое по щелчку левой кнопки мыши будет вызывать диалоговое окно с просьбой указать файл картинки, которую следует загрузить в базу данных:

```
        private void pictureBox1_Click(object sender, EventArgs e)
        {
            string address = "";
            address = SelectOpenFileDialogFileName(address);

            if (address != "Canceld")
            {
                PutImageBinaryIntoDb(@address); // запись изображения в БД
                this.booksTableAdapter.Fill(this.dataSet1.Books);
            }
        }
```

Добавьте метод для загрузки изображения в базу данных «PutImageBinaryIntoDb», код которого приведен ниже:

```
        private void PutImageBinaryIntoDb(string iFile)
        {
            // конвертация изображения в байты
            byte[] imageData = null;
            FileInfo fInfo = new FileInfo(iFile);
            long numBytes = fInfo.Length;
            FileStream fStream = new FileStream(iFile, FileMode.Open, FileAccess.Read);
            BinaryReader br = new BinaryReader(fStream);
            imageData = br.ReadBytes((int)numBytes);

            // получение расширения файла изображения не забыв удалить точку перед
расширением
            string iImageExtension = (Path.GetExtension(iFile)).Replace(".", "").ToLower();

            // запись изображения в БД
            using (SqlConnection sqlConnection =
                new SqlConnection
                (
                    @"Data Source=.\SQLEXPRESS;
                    Initial Catalog=BookShopDB;
                    Integrated Security = True"
                )) // строка подключения к БД
            {
```

```
                string commandText = "UPDATE Books SET screen = @screen, screen_format =
@screen_format WHERE (BookID = @BookID)";

                SqlCommand command = new SqlCommand(commandText, sqlConnection);
                command.Parameters.AddWithValue("@screen", (object)imageData); // записываем
само изображение
                command.Parameters.AddWithValue("@screen_format", iImageExtension); //
записываем расширение изображения

                command.Parameters.AddWithValue("@BookID",
this.dataGridView1.Rows[this.dataGridView1.CurrentRow.Index].Cells["bookIDDataGridViewTextBo
xColumn"].Value); // записываем расширение изображения

                sqlConnection.Open();
                command.ExecuteNonQuery();
                sqlConnection.Close();
            }
        }
```

15.12 Кнопка «Фильтры сортировки и поиска»

Для кнопки <button5> установите следующие свойства:

– <Text> = «Фильтры сортировки и поиска»;

– Введите дополнительную переменную flagHeight и событию <Click> добавьте следующий код:

```
bool flagHeight = false;
private void button5_Click(object sender, EventArgs e)
{
    if (flagHeight) this.Height -= 200; else this.Height += 200;
    flagHeight = !flagHeight;
}
```

– Событию <MouseHover> добавьте следующий код:

```
private void pictureBox1_MouseHover(object sender, EventArgs e)
{
    toolTip1.SetToolTip(pictureBox1, "Нажмите для изменения картинки");
}
```

15.13 Метод загрузки списка авторов из базы данных с помощью хранимой процедуры

Добавьте метод, загружающий список авторов из базы данных с помощью хранимой процедуры, код метода приведен ниже:

```
private void LoadAuthorsFromProcToTheList()
{
    List<string> listOfAuthors = new List<string>();
    using (SqlConnection sqlConnection =
```

```
            new SqlConnection
            (
                @"Data Source=.\SQLEXPRESS;
                Initial Catalog=BookShopDB;
                Integrated Security = True"
             )) // строка подключения к БД
            {
                sqlConnection.Open();
                SqlCommand sqlCommand = new SqlCommand();
                sqlCommand.Connection = sqlConnection;
                int BookID = 0;
                SqlDataReader sqlReader;
                sqlCommand.CommandText = @"exec spBookAuthorsList @BookID";
                string iTrimText = null;
                for (int i = 0; i < dataGridView1.Rows.Count; i++)
                {
                    BookID =
Convert.ToInt32(this.dataGridView1.Rows[i].Cells["authorListIDDataGridViewTextBoxColumn"].Va
lue);
                    sqlCommand.Parameters.AddWithValue("@BookID", BookID);
                    sqlReader = sqlCommand.ExecuteReader();
                    iTrimText = null;
                    while (sqlReader.Read()) // считываем и вносим в лист результаты
                    {
                        iTrimText += sqlReader["Authors"].ToString();
                        iTrimText += "; ";
                    }
                    try
                    {
                        iTrimText = iTrimText.Remove(iTrimText.Length - 2, 2);
                        this.dataGridView1.Rows[i].Cells["Authors"].Value = iTrimText;
                    }
                    catch (Exception ex)
                    {

                    }

                    sqlReader.Close();
                    sqlCommand.Parameters.Clear();

                }
                sqlConnection.Close();
            }

            return;
        }
```

15.14 Кнопка «Поиск»

Для кнопки <button1> установите следующие свойства:

– <Text> = «Поиск»;

– Событию <Click> добавьте следующий код:

```
 private void button1_Click(object sender, EventArgs e)
 {
     switch (textBox1.Text)
     {
```

```
            case "Название книги": filterString = "TitleBookName"; break;
            case "Страниц": filterString = "Pages"; break;
            case "Цена": filterString = "Price"; break;
        }
        if (String.IsNullOrEmpty(textBox2.Text)) booksBindingSource.Filter = "";
        else
        {
            try
            {
                booksBindingSource.Filter = "Convert ([" + filterString + "]," +
                                            "'System.String') LIKE '" + textBox2.Text +
"%'";
            }
            catch (Exception ex)
            {
                MessageBox.Show("Невозможно выполнить данный поиск");
            }
        }
        this.dataTable1TableAdapter.Fill(this.dataSet1.DataTable1);
        this.booksTableAdapter.Fill(this.dataSet1.Books);
        LoadAuthorsFromProcToTheList();
    }
```

15.15 Кнопка «Снять фильтр»

Для кнопки <button2> установите следующие свойства:

– <Text> = «Снять фильтр»;

– Событию <Click> добавьте следующий код:

```
 private void button2_Click(object sender, EventArgs e)
 {
     this.booksBindingSource.Filter = "";
     this.dataTable1TableAdapter.Fill(this.dataSet1.DataTable1);
     this.booksTableAdapter.Fill(this.dataSet1.Books);
     LoadAuthorsFromProcToTheList();
 }
```

15.16 Кнопка «ОК»

Для кнопки <button3> установите следующие свойства:

– <Text> = «ОК»;

– Событию <Click> добавьте следующий код:

```
 private void button3_Click(object sender, EventArgs e)
 {
     switch (textBox4.Text)
     {
         case "Название книги": filterString = "TitleBookName"; break;
         case "Страниц": filterString = "Pages"; break;
         case "Цена": filterString = "Price"; break;
     }
```

```
            try
            {
                this.booksBindingSource.Filter = "(" + filterString + " LIKE '" +
textBox3.Text + "*')";
            }
            catch (Exception ex)
            {
                MessageBox.Show("Не могу установить данный фильтр");
            }
            this.dataTable1TableAdapter.Fill(this.dataSet1.DataTable1);

            this.booksTableAdapter.Fill(this.dataSet1.Books);
            LoadAuthorsFromProcToTheList();
        }
```

15.17 Кнопка «Сортировка»

Для кнопки <button4> установите следующие свойства:

– <Text> = «Сортировка»;

– Событию <Click> добавьте следующий код:

```
        private void button4_Click(object sender, EventArgs e)
        {
            int field = 0;
            switch (textBox6.Text)
            {
                case "Название книги": field = 0; break;
                case "Страниц": field = 1; break;
                case "Цена": field = 2; break;
                case "Авторы": field = 3; break;
            }
            if (field != 3)
            {
                switch (comboBox1.SelectedIndex)
                {
                    case 0: dataGridView1.Sort(dataGridView1.Columns[field],
ListSortDirection.Ascending); break;
                    case 1: dataGridView1.Sort(dataGridView1.Columns[field],
ListSortDirection.Descending); break;
                }
            }
            else
                MessageBox.Show("Не могу отсортировать столбец авторов");
            this.dataTable1TableAdapter.Fill(this.dataSet1.DataTable1);
            this.booksTableAdapter.Fill(this.dataSet1.Books);
            LoadAuthorsFromProcToTheList();
        }
```

15.18 Кнопки «Корзина» и «Добавить в корзину»

Добавьте две кнопки и установите их свойства <Text> равными «Добавить в корзину» и «Корзина». Добавьте <label> и <textBox> и установите их свойства <Text> равными «Количество» (для label) и «1» (для textBox).

Добавьте обработчик события «Click» для кнопки «Корзина»:

```
public FrmPurchases FormPurchases;
private void button7_Click(object sender, EventArgs e)
{
    if (amountGoods == 0)
    {
        MessageBox.Show("Корзина пуста!");
        return;
    }
    FormPurchases = new FrmPurchases
    {
        Owner = this
    };
    FormPurchases.Width = 908;
    FormPurchases.ShowDialog();
}
```

Добавьте обработчик события «Click» для кнопки «Добавить в корзину»:

```
public Hashtable bucketGoods = new Hashtable();
public int amountGoods = 0;
private void button6_Click(object sender, EventArgs e)
{
    try
    {
bucketGoods[Convert.ToInt32(this.dataGridView1.Rows[this.dataGridView1.CurrentRow.Index].Cel
ls["bookIDDataGridViewTextBoxColumn"].Value)] = Convert.ToInt32(textBox5.Text);

        amountGoods = 0;
        foreach (DictionaryEntry entry in bucketGoods)
        {
            amountGoods += Convert.ToInt32(entry.Value);
        }
    }
    catch (Exception ex)
    {

    }
    if (amountGoods != 0) label10.Text = amountGoods.ToString() + " шт.";
    else label10.Text = "Пусто";
    textBox5.Text = "1";
}
```

15.19 Настройка событий «Click» на кнопках главного меню

Введите переменные классов для каждой созданной формы, как в коде, приведенном ниже:

```
private FrmAuthors FormAuthors;
```

Введите переменные с атрибутом public, показанные ниже:

```
public int stateForm;
public int bookID;
```

Нажмите на форме <Меню> → <Работа с таблицами> → <Авторы>. В описании метода <Click> введите следующий код:

```
private void авторыToolStripMenuItem_Click(object sender, EventArgs e)
{
    stateForm = 10;
    FormAuthors = new FrmAuthors
    {
        Owner = this
    };
    FormAuthors.ShowDialog();
}
```

Создайте обработчики событий для остальных кнопок главного меню, при этом объявите переменные остальных форм с атрибутом public.

Настройте остальные формы и расположенные на них элементы, при этом добавляйте соответствующие ссылки на источники данных в «DataSet».

15.20 Добавление обработчиков событий для формы «Авторы»

В обозревателе решений перейдите к форме <FrmAuthors>. Добавьте обработчик события на загрузку формы:

```
private void Authors_Load(object sender, EventArgs e)
{
    // TODO: данная строка кода позволяет загрузить данные в таблицу
"dataSet1.Authors". При необходимости она может быть перемещена или удалена.
    this.authorsTableAdapter.Fill(this.dataSet1.Authors);
}
```

Добавьте обработчик события на закрытие формы:

```
private void FrmAuthors_FormClosing(object sender, FormClosingEventArgs e)
{
    this.authorsTableAdapter.Update(this.dataSet1.Authors);
}
```

Добавьте обработчик события для открытия специальной формы для выбора даты:

```
int currentCol, currentRow;
public FrmDate FormDate;
public string date;
public int state;
private void toolStripButton1_Click(object sender, EventArgs e)
{
    this.authorsTableAdapter.Update(this.dataSet1.Authors);
```

```
        }

        private void dataGridView1_Click(object sender, EventArgs e)
        {
            FrmMain global = this.Owner as FrmMain;
            state = global.stateForm;
            if (this.dataGridView1.CurrentCell.ColumnIndex == 4)
            {
                currentCol = this.dataGridView1.CurrentCell.ColumnIndex;
                currentRow = this.dataGridView1.CurrentCell.RowIndex;

                FormDate = new FrmDate
                {
                    Owner = this
                };
                FormDate.ShowDialog();
                try
                {
                    this.dataGridView1[currentCol, currentRow].Value =
Convert.ToDateTime(date);
                }
                catch(Exception ex)
                {

                }
            }
        }
```

15.21 Добавление обработчиков событий для формы «Клиенты»

В обозревателе решений перейдите к форме <FrmClients>. Добавьте обработчик события на загрузку формы:

```
        private void FrmClients_Load(object sender, EventArgs e)
        {
            this.clientsTableAdapter.Fill(this.dataSet1.Clients);
        }
```

Добавьте обработчик события на закрытие формы:

```
        private void FrmClients_FormClosing(object sender, FormClosingEventArgs e)
        {
            this.clientsTableAdapter.Update(this.dataSet1.Clients);
        }
```

Добавьте обработчик события для открытия специальной формы для выбора даты:

```
        int currentCol, currentRow;
        public FrmDate FormDate;
        public string date;
        public int state;
        private void dataGridView1_Click(object sender, EventArgs e)
        {
            FrmMain global = this.Owner as FrmMain;
            state = global.stateForm;
            if (this.dataGridView1.CurrentCell.ColumnIndex == 4)
            {
```

```
            currentCol = this.dataGridView1.CurrentCell.ColumnIndex;
            currentRow = this.dataGridView1.CurrentCell.RowIndex;
            FormDate = new FrmDate
            {
                Owner = this
            };
            FormDate.ShowDialog();
            this.dataGridView1[currentCol, currentRow].Value = Convert.ToDateTime(date);
        }
    }
```

15.22 Добавление обработчиков событий для формы «Выбор даты»

В обозревателе решений перейдите к форме <FrmDate>.

Добавьте обработчик события «DateSelected» на элемент «MonthCalendar»:

```
private void monthCalendar1_DateSelected(object sender, DateRangeEventArgs e)
{
    var dateTime = monthCalendar1.SelectionRange.Start;
    textBox1.Text = dateTime.ToShortDateString();
}
```

Добавьте обработчик события «Click» на кнопку «Подтвердить выбор даты»:

```
private void button1_Click(object sender, EventArgs e)
{
    FrmClients main = this.Owner as FrmClients;
    if ((main != null) && (main.state == 11))
    {
        main.date = textBox1.Text;
        this.Close();
        return;
    }
    FrmAuthors main2 = this.Owner as FrmAuthors;
    if ((main2 != null) && (main2.state == 10))
    {
        main2.date = textBox1.Text;
        this.Close();
        return;
    }
}
```

15.23 Добавление обработчиков событий для формы «Поставщики»

В обозревателе решений перейдите к форме <FrmDeliveries>. Добавьте обработчик события на загрузку формы:

```
private void FrmDeliveries_Load(object sender, EventArgs e)
{
    this.deliveriesTableAdapter.Fill(this.dataSet1.Deliveries);
```

```
            this.dataSet1.Deliveries.Columns[0].AutoIncrementSeed =
this.dataSet1.Deliveries.Rows.Count + 1;
            this.dataSet1.Deliveries.Columns[0].AutoIncrementStep = 1;
        }
```

Добавьте обработчик события на закрытие формы:

```
        private void FrmDeliveries_FormClosing(object sender, FormClosingEventArgs e)
        {
            try
            {
                this.deliveriesTableAdapter.Update(this.dataSet1.Deliveries);
            }
            catch (Exception ex)
            {

            }
        }
```

15.24 Добавление обработчиков событий для формы «Редактирование книги»

В обозревателе решений перейдите к форме <FrmEditBooks>.

Объявите следующие перменные:

```
        public DataSet dsForOrder;
        public BindingSource binSrcForOrder;
        public int? ListID;
        public string cmb;
```

Добавьте обработчик события на загрузку формы:

```
        private void FrmEditBooks_Load(object sender, EventArgs e)
        {
            this.publishHouseTableAdapter.Fill(this.dataSet1.PublishHouse);
            this.spListAllAuthorsTableAdapter.Fill(this.dataSet1.spListAllAuthors);
            this.authorsTableAdapter.Fill(this.dataSet1.Authors);
            this.authorListTableAdapter.Fill(this.dataSet1.AuthorList);
            FrmMain main = this.Owner as FrmMain;
            dsForOrder = new DataSet();
            dsForOrder = main.dsFromMain;
            binSrcForOrder = new BindingSource();
            binSrcForOrder = main.bsFromMain;
            this.textBox1.DataBindings.Add(new System.Windows.Forms.Binding("Text",
this.binSrcForOrder, "TitleBookName", true));
            this.textBox2.DataBindings.Add(new System.Windows.Forms.Binding("Text",
this.binSrcForOrder, "Pages", true));
            this.textBox3.DataBindings.Add(new System.Windows.Forms.Binding("Text",
this.binSrcForOrder, "AuthorListID", true));
            this.label7.DataBindings.Add(new System.Windows.Forms.Binding("Text",
this.binSrcForOrder, "PublishID", true));
            cmb = label7.Text;
            try { this.comboBox1.SelectedValue = Convert.ToInt32(cmb); } catch (Exception
ex) { }
            this.label4.DataBindings.Add(new System.Windows.Forms.Binding("Text",
this.binSrcForOrder, "BookID", true));
            this.textBox4.DataBindings.Add(new Binding("Text", this.binSrcForOrder, "Price",
true));
```

```
            try { ListID = Convert.ToInt32(this.textBox3.Text); } catch ( Exception ex ) {
}
this.spBookAuthorsListTableAdapter.Fill(this.bookShopDBDataSet1.spBookAuthorsList, ref
ListID);
        }
```

Добавьте обработчик события «Click» на кнопку «Добавить автора»:

```
        private void button1_Click(object sender, EventArgs e)
        {
            using (SqlConnection sqlConnection =
            new SqlConnection
            (
                    @"Data Source=.\SQLEXPRESS;
                    Initial Catalog=BookShopDB;
                    Integrated Security = True"
            )) // строка подключения к БД
            {
                string commandText = "INSERT AuthorList (AuthorListID, AuthorID) VALUES
(@AuthorListID, @AuthorID)";
                SqlCommand command = new SqlCommand(commandText, sqlConnection);
                command.Parameters.AddWithValue("@AuthorListID", textBox3.Text);
                command.Parameters.AddWithValue("@AuthorID",
this.dataGridView3.Rows[this.dataGridView3.CurrentRow.Index].Cells["AuthorID"].Value);
                sqlConnection.Open();
                try
                {
                    command.ExecuteNonQuery();
                }
                catch (Exception ex)
                {

                }
                sqlConnection.Close();
            }
this.spBookAuthorsListTableAdapter.Fill(this.bookShopDBDataSet1.spBookAuthorsList, ref
ListID);
        }
```

Добавьте обработчик события «Click» на кнопку «Удалить автора»:

```
        private void button2_Click(object sender, EventArgs e)
        {
            if (this.dataGridView2.Rows.Count != 0)
            {
                using (SqlConnection sqlConnection =
                new SqlConnection
                (
                     @"Data Source=.\SQLEXPRESS;
                    Initial Catalog=BookShopDB;
                    Integrated Security = True"
                )) // строка подключения к БД
                {
                    string commandText = "DELETE FROM AuthorList WHERE (AuthorListID =
@AuthorListID AND AuthorID = @AuthorID)";
                    SqlCommand command = new SqlCommand(commandText, sqlConnection);
                    command.Parameters.AddWithValue("@AuthorListID", textBox3.Text);
                    string aID =
Convert.ToString(this.dataGridView2.Rows[this.dataGridView2.CurrentRow.Index].Cells[1].Value
);
                    command.Parameters.AddWithValue("@AuthorID",
this.dataGridView2.Rows[this.dataGridView2.CurrentRow.Index].Cells[1].Value);
                    sqlConnection.Open();
                    try
                    {
```

```
                    command.ExecuteNonQuery();
                }
                catch (Exception ex)
                {

                }
                sqlConnection.Close();
            }
this.spBookAuthorsListTableAdapter.Fill(this.bookShopDBDataSet1.spBookAuthorsList, ref
ListID);
        }
    }
```

Добавьте обработчик события «Click» на кнопку «Отмена»:

```
    private void button4_Click(object sender, EventArgs e)
    {
        FrmMain main = this.Owner as FrmMain;
        if (main.stateForm == 1) this.Close();
        if (main.stateForm == 2)
        {
                using (SqlConnection sqlConnection =
                new SqlConnection
                (
                    @"Data Source=.\SQLEXPRESS;
                    Initial Catalog=BookShopDB;
                    Integrated Security = True"
                )) // строка подключения к БД
                {
                    string commandText = "DELETE FROM Books WHERE (BookID = @BookID)";
                    SqlCommand command = new SqlCommand(commandText, sqlConnection);
                    command.Parameters.AddWithValue("@BookID",
Convert.ToInt32(label4.Text));
                    sqlConnection.Open();
                    try
                    {
                        command.ExecuteNonQuery();
                    }
                    catch (Exception ex)
                    {

                    }

                    sqlConnection.Close();
            }
            main.UpdateFormChanges();
            this.Close();
        }
    }
```

Добавьте обработчик события «Click» на кнопку «Сохранить изменения и выйти»:

```
    private void button3_Click(object sender, EventArgs e)
    {
        using (SqlConnection sqlConnection =
        new SqlConnection
        (
            @"Data Source=.\SQLEXPRESS;
            Initial Catalog=BookShopDB;
            Integrated Security = True"
        )) // строка подключения к БД
        {
```

```
                string commandText = "UPDATE Books SET TitleBookName = @TitleBookName, Pages
= @Pages, Price = @Price, PublishID = @PublishID WHERE (BookID = @BookID)";
                SqlCommand command = new SqlCommand(commandText, sqlConnection);
                Decimal Price;
                try
                {
                    Price = Convert.ToDecimal(textBox4.Text);
                }
                catch (Exception ex)
                {
                    Price = 0;
                }
                command.Parameters.AddWithValue("@TitleBookName", textBox1.Text);
                command.Parameters.AddWithValue("@Pages", Convert.ToInt32(textBox2.Text));
                command.Parameters.AddWithValue("@BookID", Convert.ToInt32(label4.Text));
                command.Parameters.AddWithValue("@Price", Price);
                command.Parameters.AddWithValue("@PublishID",
Convert.ToInt32(comboBox1.SelectedValue));
                sqlConnection.Open();
                try
                {
                    command.ExecuteNonQuery();
                    FrmMain main = this.Owner as FrmMain;
                    main.UpdateFormChanges();
                }
                catch (Exception ex)
                {

                }
                sqlConnection.Close();
            }
            this.Close();
        }
```

15.25 Добавление обработчиков событий для формы «Издательства»

В обозревателе решений перейдите к форме <FrmPublishHouse>. Добавьте обработчик события на загрузку формы:

```
        private void FrmPublishHouse_Load(object sender, EventArgs e)
        {
            this.publishHouseTableAdapter.Fill(this.dataSet1.PublishHouse);
        }
```

Добавьте обработчик события на закрытие формы:

```
        private void FrmPublishHouse_FormClosing(object sender, FormClosingEventArgs e)
        {
            try
            {
                this.publishHouseTableAdapter.Update(this.dataSet1.PublishHouse);
            }
            catch (Exception ex)
            {

            }
        }
```

15.26 Добавление обработчиков событий для формы «Оформление заказа»

В обозревателе решений перейдите к форме <FrmPurchases>. Добавьте обработчик события на загрузку формы:

```
        private void FrmPurchases_Load(object sender, EventArgs e)
        {
            this.currentOrderTableAdapter.Fill(this.dataSet1.CurrentOrder);
            this.deliveriesInfoTableAdapter.Fill(this.dataSet1.DeliveriesInfo);
            this.clientInfoTableAdapter.Fill(this.dataSet1.ClientInfo);
            main = this.Owner as FrmMain;
            SourceDataToBindingBucketGoods();
            this.label2.Text = main.amountGoods.ToString() + " шт.";
            this.label4.Text = CalcTotalPrice().ToString();
            this.comboBox1.SelectedIndex = -1;
            this.comboBox2.SelectedIndex = -1;
        }
```

Добавьте метод расчета итоговой суммы заказа:

```
        private int CalcTotalPrice()
        {
            int totalSum = 0;
            for (int i = 0; i < this.dataGridView1.Rows.Count; i++)
                totalSum += Convert.ToInt32(this.dataGridView1.Rows[i].Cells[3].Value) *
Convert.ToInt32(this.dataGridView1.Rows[i].Cells[4].Value);
            return totalSum;
        }
```

Добавьте метод создания источника данных для корзины товаров:

```
        BindingSource bs = new BindingSource();
        private void SourceDataToBindingBucketGoods()
        {
            int BookID = 2;
            string conStr = @"Data Source=.\SQLEXPRESS; Initial Catalog=BookShopDB;
Integrated Security=True;"; // создание строки подключения
            SqlConnection connection = new SqlConnection(conStr);
            connection.Open();
            SqlCommand cmd = new SqlCommand(@"SELECT BookID, TitleBookName, Pages, Price
FROM Books WHERE BookID = @BookID", connection);
            foreach (DictionaryEntry entry in main.bucketGoods)
            {
                BookID = Convert.ToInt32(entry.Key);
                break;
            }
            cmd.Parameters.AddWithValue("@BookID", BookID);
            SqlDataReader reader = cmd.ExecuteReader();
            DataTable table = CreateSchemaFromReader(reader, "BooksToOrder"); // создание
новой таблицы на основе схемы, предоставляемой DataReader
            reader.Close();
            cmd.Parameters.Clear();
            foreach (DictionaryEntry entry in main.bucketGoods)
            {
                cmd.Parameters.Clear();
                BookID = Convert.ToInt32(entry.Key);
                cmd.Parameters.AddWithValue("@BookID", BookID);
                reader = cmd.ExecuteReader();
                WriteDataFromReader(table, reader, Convert.ToInt32(entry.Value)); // запись
данных  в таблицу с помощью DataReader
```

```
                reader.Close(); // Каждый раз закрываем DataReader
            }
            reader.Close();
            connection.Close();
            this.dataGridView1.DataSource = table;
            this.dataGridView1.Columns[0].HeaderCell.Value = "Код";
            this.dataGridView1.Columns[0].Width = 65;
            this.dataGridView1.Columns[1].HeaderCell.Value = "Название книги";
            this.dataGridView1.Columns[1].Width = 250;
            this.dataGridView1.Columns[1].AutoSizeMode =
DataGridViewAutoSizeColumnMode.Fill;
            this.dataGridView1.Columns[2].HeaderCell.Value = "Страниц";
            this.dataGridView1.Columns[3].HeaderCell.Value = "Цена за 1 шт.";
            this.dataGridView1.Columns[3].Width = 110;
            this.dataGridView1.Columns[4].HeaderCell.Value = "Количество шт.";
            this.dataGridView1.Columns[4].Width = 110;
            bs.DataSource = table;
            this.bindingNavigator1.BindingSource = bs;
            return;
        }
```

Добавьте метод создания таблицы «DataTable» по схеме базы данных:

```
        /// <summary>
        /// Метод создает новую таблицу DataTable по схеме SqlDataReader
        /// </summary>
        /// <param name="reader">Данные из DataReader</param>
        /// <param name="tableName">Имя новой таблицы DataTable</param>
        /// <returns>Возвращает таблицу DataTable, составленную по схеме
DataReader</returns>
        private static DataTable CreateSchemaFromReader(SqlDataReader reader, string
tableName)
        {
            DataTable table = new DataTable(tableName);
            for (int i = 0; i < reader.FieldCount; i++)
                table.Columns.Add(new DataColumn(reader.GetName(i),
reader.GetFieldType(i)));
            table.Columns.Add("AmountOf", typeof(int));
            return table;
        }
```

Добавьте метод записи данных в таблицу «DataTable» из базы данных:

```
        /// <summary>
        /// Метод записывает данные в таблицу DataTable из схемы DataReader
        /// </summary>
        /// <param name="table">Таблица DataTable куда будут записаны данные</param>
        /// <param name="reader">Данные из DataReader</param>
        private static void WriteDataFromReader(DataTable table, SqlDataReader reader)
        {
            while (reader.Read())
            {
                DataRow row = table.NewRow();

                for (int i = 0; i < reader.FieldCount; i++)
                    row[i] = reader[i];
                table.Rows.Add(row);
            }
        }
        private static void WriteDataFromReader(DataTable table, SqlDataReader reader, int
amount)
        {
            while (reader.Read())
            {
                DataRow row = table.NewRow();
```

```
            for (int i = 0; i < reader.FieldCount; i++)
                row[i] = reader[i];
            row[reader.FieldCount] = amount;
            table.Rows.Add(row);
        }
    }
```

Добавьте ручной обработчик события «bindingNavigatorDeleteItem_Click» для элемента «BindingNavigator»:

```
    private void bindingNavigatorDeleteItem_Click(object sender, EventArgs e)
    {
        if (this.dataGridView1.Rows.Count > 0)
            if (MessageBox.Show(this, "Эта книга будет удалена из заказа", "Вы уверены,
что хотите продолжить?", MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
System.Windows.Forms.DialogResult.Yes)
            {
                int temp =
Convert.ToInt32(this.dataGridView1.CurrentRow.Cells[4].Value);
main.bucketGoods.Remove(Convert.ToInt32(this.dataGridView1.CurrentRow.Cells[0].Value));
                main.amountGoods -= temp;
                bs.RemoveCurrent();
                this.label2.Text = main.amountGoods.ToString() + " шт.";
                this.label4.Text = CalcTotalPrice().ToString();
                main.UpdateFormChanges();
            }
    }
```

Добавьте обработчик события на закрытие формы:

```
    private void FrmPurchases_FormClosing(object sender, FormClosingEventArgs e)
    {
        main.UpdateFormChanges();
    }
```

Добавьте обработчик события «Click» для кнопки «Оформить заказ»:

```
    private void button2_Click(object sender, EventArgs e)
    {
        if (flagHeight) this.Width -= 540; else this.Width += 540;
        flagHeight = !flagHeight;
    }
```

Добавьте обработчик события «Click» для кнопки «Подтвердить»:

```
    private void button1_Click(object sender, EventArgs e)
    {
        DataTable dt = new DataTable();
        dt = (DataTable)dataGridView1.DataSource;
        WriteToOrderItemList();
    }
```

15.27 Добавление обработчиков событий для формы «Информация о заказах»

В обозревателе решений перейдите к форме <FrmViewOrdersInfo>. Добавьте обработчик события на загрузку формы:

```
        int? OrderListID = 0;

        private void FrmViewOrdersInfo_Load(object sender, EventArgs e)
        {
            this.ordersTableAdapter.Fill(this.dataSet1.Orders);
            OrderListID = Convert.ToInt32(this.dataGridView1.CurrentRow.Cells[0].Value);
            this.spListOfOrdersTableAdapter.Fill(this.dataSet1.spListOfOrders, ref
OrderListID);
        }
```

Добавьте обработчик события «SelectionChanged» для элемента «dataGridView1»:

```
        private void dataGridView1_SelectionChanged(object sender, EventArgs e)
        {
            OrderListID = Convert.ToInt32(this.dataGridView1.CurrentRow.Cells[0].Value);
            this.spListOfOrdersTableAdapter.Fill(this.dataSet1.spListOfOrders, ref
OrderListID);
        }
```

15.28 Контрольное задание

Создайте приложение-клиент к базе данных согласно выбранному вами варианту. В приложении должно быть реализовано:

– отображение таблиц;

– манипулирование данными;

– поиск, сортировка, фильтрация данных;

– не менее 3-х запросов;

– не менее 3-х хранимых процедур (для проведения затратного анализа).

15.29 Контрольные вопросы

1. Как организовать отображение данных из таблиц?

2. С помощью какого компонента осуществляется манипулирование данными?

3. При помощи какого метода осуществляется сортировка данных?

4. С помощью какого метода осуществляется поиск данных?

5. При помощи какого метода осуществляется фильтрация данных?

6. Каким образом организуется осуществление выполнения запросов?

7. Как организовать осуществление выполнения хранимых процедур?

Список использованных источников