Федеральное агентство по образованию
Государственное образовательное учреждение высшего профессионального образования
Ульяновский государственный технический университет

РАБОТА С БАЗАМИ ДАННЫХ НА ЯЗЫКЕ C#

ТЕХНОЛОГИЯ ADO .NET

Учебное пособие

для студентов, обучающихся по специальности 08080165

Составители О. Н. Евсеева
А. Б. Шамшев

Ульяновск
2009

УДК 681.3(075)
ББК 22.18я 7
Р 13

Рецензенты:

кафедра «Информационные технологии» Ульяновского государственного университета (зав. кафедрой канд. физ.-мат. наук, доцент М. А. Волков);

доктор технических наук, профессор кафедры «Информационные технологии» Ульяновского государственного университета И. В. Семушин.

Утверждено редакционно-издательским советом университета в качестве учебного пособия

Р 13 **РАБОТА** С БАЗАМИ ДАННЫХ НА ЯЗЫКЕ C#. ТЕХНОЛОГИЯ ADO .NET: учебное пособие / сост. О. Н. Евсеева, А. Б. Шамшев. – Ульяновск: УлГТУ, 2009. – 170 с.

ISBN 978-5-9795-0475-9

Пособие содержит введение в технологию разработки баз данных средствами ADO (ActiveX Data Object) на платформе .NET с использованием языка программирования C#. В книге представлены работа с базами данных на примере настольного приложения СУБД Microsoft Access и серверного приложения Microsoft SQL Server; основы языка SQL, создание и использование объектов ADO .NET.

Пособие предназначено для студентов, изучающих высокоуровневые методы информатики и программирования (специальность 080801 «Прикладная информатика (по областям)») и базы данных, а также для студентов других специальностей, связанных с программированием.

УДК 681.3(075)
ББК 22.18я 7

ISBN 978-5-9795-0475-9

© Евсеева О. Н., Шамшев А. Б., составление, 2009
© Оформление. УлГТУ, 2009

ОГЛАВЛЕНИЕ

Предисловие **5**
1. Проектирование баз данных **7**
1.1. Реляционная база данных и ее структура 7
1.2. Этапы проектирования реляционной базы данных 8
1.2.1. Определение требований 9
1.2.2. Логическая модель 10
ER-диаграммы 10
Объекты, атрибуты и ключи 11
Нормализация 13
1.2.3. Физическая модель 15
1.3. Создание БД в СУБД Microsoft Access 15
1.3.1. Таблицы 15
1.3.2. Ключи 17
1.3.3. Связи 18
1.4. Создание базы данных в среде Microsoft SQL Server 22
1.4.1. Определение структуры базы данных 23
1.4.2. Перенос файла БД Microsoft SQL на другой компьютер 28
1.5. Контрольные вопросы и задания к разделу 1 30
2. Основы языка SQL **32**
2.1. Базовая конструкция SQL-запроса 33
2.2. Агрегирующие функции языка SQL 36
2.3. Оператор сравнения записей like 37
2.4. Команды определения данных языка SQL 38
2.5. Команды изменения данных языка DML 39
2.6. Контрольные вопросы и задания к разделу 2 41
3. Создание приложений баз данных **42**
3.1. Пример простейшего приложения баз данных 42
3.2. Обзор объектов ADO .NET 53
3.2.1. Источник данных DataSet 53
3.2.2. Таблицы и поля (объекты DataTable и DataColumn) 53
3.2.3. Объекты DataRelation 53
3.2.4. Строки (объект DataRow) 53
3.2.5. DataAdapter 54
3.2.6. Объекты DBConnection и DBCommand 54
3.3. Server Explorer 54
3.4. Пример создания приложения БД «вручную» 56
3.5. Контрольные вопросы и задания к разделу 3 58
4. Объекты ADO .NET **59**
4.1. Соединение с базой данных 59
4.1.1. Командная строка соединения ConnectionString 59
4.1.2. Управление соединением. Объект Connection 60
4.1.3. События объекта Connection 61
4.1.4. Обработка исключений 64
При работе с MS SQL 64
При работе с MS Access 65
4.1.5. Работа с пулом соединений 65
4.2. Хранимые процедуры 67
4.2.1. Стандартные запросы к БД 68
4.2.2. Простые запросы к БД 70

4.2.3. Параметризованные запросы к БД ... 74
4.2.4. Создание хранимых процедур в Management Studio ... 80
4.2.5. Создание хранимых процедур в Visual Studio 2008 ... 82
4.3. Запросы к базе данных ... 85
4.3.1. Командная строка SQL-запроса CommandText ... 85
4.3.2. Объект Command ... 86
Создание и инициализация ... 86
Свойства CommandType и CommandText ... 88
Метод ExecuteNonQuery ... 89
Метод ExecuteScalar ... 93
Метод ExecuteReader ... 95
4.3.3. Параметризированные запросы ... 97
Использование метода ExecuteNonQuery ... 98
Использование метода ExecuteScalar ... 104
Использование метода ExecuteReader ... 104
4.3.4. Вызов хранимых процедур ... 108
Хранимые процедуры с входными параметрами ... 108
Хранимые процедуры с входными и выходными параметрами ... 112
Хранимые процедуры из нескольких SQL-конструкций ... 113
4.3.5. Транзакции ... 115
Транзакции в ADO .NET ... 119
4.4. Работа с таблицами данных ... 123
4.4.1. Объекты DataSet, DataTable и DataColumn ... 123
Программное создание объектов DataTable и DataColumn ... 124
Свойство PrimaryKey ... 126
Ограничения UniqueConstraint и ForeignKeyConstraint ... 127
Создание столбцов, основанных на выражении ... 128
Отслеживание изменений в базе данных ... 130
Обработка исключений ... 132
4.4.2. Объект DataRow ... 133
Программное создание и изменение записей таблицы данных ... 133
Свойство RowState ... 137
Свойство RowVersion ... 138
События объекта DataTable ... 143
4.4.3. Объект DataGridView ... 145
Вывод двух связанных таблиц данных в один элемент DataGridView ... 145
Вывод связанных таблиц данных в два элемента DataGridView ... 149
4.4.4. Объект DataView ... 149
Фильтрация данных ... 150
Сортировка данных ... 151
Поиск данных ... 152
4.4.5. Вспомогательные классы ... 157
Класс HashTable ... 157
Класс ArrayList ... 158
4.5. Контрольные вопросы и задания к разделу 4 ... 160
Заключение ... 162
Предметный указатель ... 167
Библиографический список ... 170

ПРЕДИСЛОВИЕ

Одними из базовых дисциплин в программе подготовки информатиков-экономистов по специальности 080801065 «Прикладная информатика (в экономике)» являются курсы «Высокоуровневые методы информатики и программирования» и «Базы данных», содержание которых определяется выпиской из государственного образовательного стандарта высшего профессионального образования (ГОС ВПО).

Выписка из ГОС ВПО

Индекс	*Наименование дисциплины и ее основные разделы*
ОПД.Ф.03	**БАЗЫ ДАННЫХ** Базы данных (БД). **Принципы построения**. Жизненный цикл БД. Типология БД. Документальные БД. Фактографические БД. Гипертекстовые и мультимедийные БД. XML-серверы. Объектно-ориентированные БД. Распределенные БД. Коммерческие БД. **Организация процессов обработки данных в БД. Ограничения целостности. Технология оперативной обработки транзакции (OLTP-технология).** Информационные хранилища. OLAP-технология. Проблема создания и сжатия больших информационных массивов, информационных хранилищ и складов данных. Основы фракталов. Фрактальная математика. Фрактальные методы в архивации. Управление складами данных.
ОПД.Ф.04	**ВЫСОКОУРОВНЕВЫЕ МЕТОДЫ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ** Новейшие направления в области создания технологий программирования. Законы эволюции программного обеспечения. Программирование в средах современных информационных систем: создание модульных программ, элементы теории модульного программирования, объектно-ориентированное проектирование и программирование. **Объектно-ориентированный подход к проектированию и разработке программ**: сущность объектно-ориентированного подхода; объектный тип данных; переменные объектного типа; инкапсуляция; наследование; полиморфизм; классы и объекты. **Конструкторы и деструкторы. Особенности программирования в оконных операционных средах. Основные стандартные модули, обеспечивающие работу в оконной операционной среде. Среда разработки; система окон разработки; система меню.** Отладка и тестирование программ. **Основы визуального программирования. Размещение нового компонента. Реакция на события. Компоненты; использование компонентов**.

Настоящее пособие затрагивает вопросы курса «Базы данных» в разделе проектирования реляционных баз данных (РБД), организации процессов обработки данных в РБД, проблемы ограничения целостности и дает введение в технологию оперативной обработки транзакций (OLTP-технология) на базе объектов ADO (ActiveX Data Objects) .NET и языка C#.

А также оно практически полностью посвящено раскрытию содержания третьего раздела «Объектно-ориентированный подход к проектированию и разработке программ» требований дисциплины «Высокоуровневые методы информатики и программирования». Изложение выделенных тем данного раздела построено на примерах использования системы классов ADO .NET для разработки приложений баз данных. Весь материал иллюстрируется примерами программ, написанных на языке C# под Windows на платформе .NET.

Целью данного учебного пособия является ознакомление студентов с современными технологиями проектирования и разработки приложений баз данных на языке C# под Windows на платформе .NET.

Задачей учебного пособия является раскрытие выделенных дидактических единиц на простых и конкретных примерах использования библиотек классов в Windows-приложениях на основе платформы .NET на языке C#, что позволит студентам освоить базовые принципы и методы технологии программирования на современном уровне и поможет применить их в курсовом и дипломном проектировании.

Пособие состоит из 4-х разделов и заключения.

Первый раздел посвящен введению в проектирование РБД. Здесь даются определения основных понятий РБД: отношение (таблица РБД), схема, атрибут, ключ, кортеж, и т. п., рассматриваются основные этапы проектирования РБД, где уделяется особое внимание вопросам нормализации отношений в БД. Приводится пример использования СУБД для создания РБД.

Второй раздел дает введение в язык запросов к БД SQL (Structured Query Language) с демонстрацией на примерах с использованием утилиты SQL Management Studio.

Третий раздел посвящен возможностям среды Visual Studio 2008 по разработке приложений баз данных. Приводятся примеры разработки приложений с использованием мастеров, предоставляемых средой, и «вручную».

Четвертый раздел полностью посвящен описанию объектов ADO .NET.

Заключение содержит краткий обзор рассмотренных возможностей технологии ADO .NET, ее двух основополагающих частей: модели классов подключений и модели классов для автономной работы с данными.

Для успешного изучения материала достаточно знание основ программирования и желательны начальные навыки программирования на языке C#. Советуем ознакомиться с материалом пособий [11, 8].

Для усвоения материала рекомендуется самостоятельно воспроизвести учебные примеры, приведенные в пособии, развивать и дополнять их новыми функциями, а также применять изученные положения в практической работе (в курсовом и дипломном проектировании).

Для проверки степени усвоения материала необходимо ответить на контрольные вопросы и выполнить предлагаемые упражнения.

1. ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

1.1. Реляционная база данных и ее структура

Базой данных (БД) называется организованная в соответствии с определенными правилами и поддерживаемая в памяти компьютера совокупность сведений об объектах, процессах, событиях или явлениях, относящихся к некоторой предметной области, теме или задаче. Она организована таким образом, чтобы обеспечить информационные потребности пользователей, а также удобное хранение этой совокупности данных, как в целом, так и любой ее части.

Реляционная база данных представляет собой множество взаимосвязанных таблиц, каждая из которых содержит информацию об объектах определенного вида. Каждая строка таблицы содержит данные об одном объекте (например, автомобиле, компьютере, клиенте), а столбцы таблицы содержат различные характеристики этих объектов – *атрибуты* (например, номер двигателя, марка процессора, телефоны фирм или клиентов).

Строки таблицы называются *записями*. Все записи таблицы имеют одинаковую структуру – они состоят из *полей* (элементов данных), в которых хранятся атрибуты объекта (рис. 1). Каждое поле записи содержит одну характеристику объекта и представляет собой заданный тип данных (например, текстовая строка, число, дата). Для идентификации записей используется первичный ключ. *Первичным ключом* называется набор полей таблицы, комбинация значений которых однозначно определяет каждую запись в таблице.

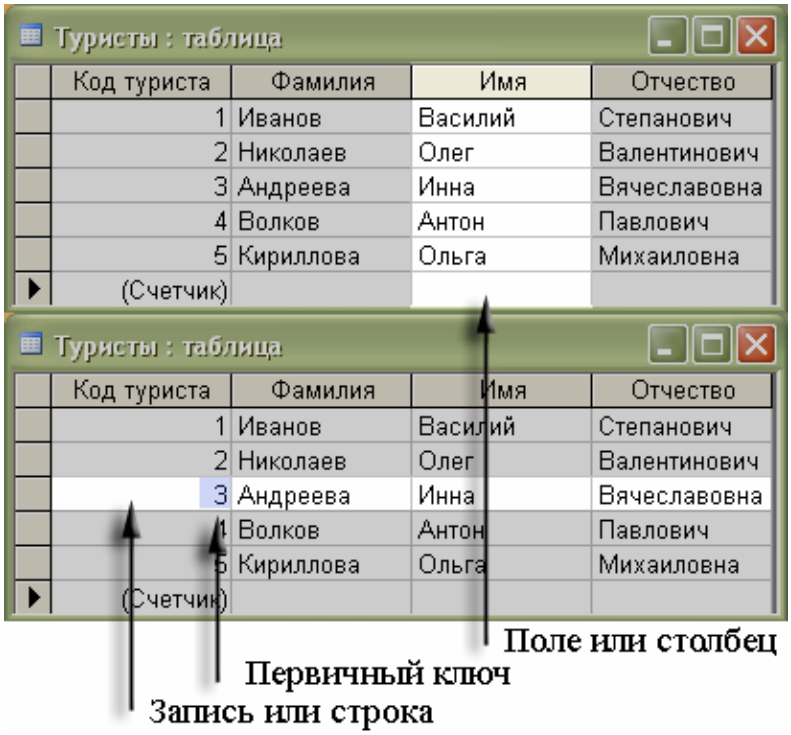


Рис. 1. Названия объектов в таблице

Для работы с данными используются системы управления базами данных (СУБД). Основные функции СУБД:

– определение данных (описание структуры баз данных);
– обработка данных;
– управление данными.

Разработка структуры БД – важнейшая задача, решаемая при проектировании БД. Структура БД (набор, форма и связи ее таблиц) – это одно из основных проектных решений при создании приложений с использованием БД. Созданная разработчиком структура БД описывается на языке определения данных СУБД.

Любая СУБД позволяет выполнять следующие операции с данными:

– добавление записей в таблицы;
– удаление записей из таблицы;
– обновление значений некоторых полей в одной или нескольких записях в таблицах БД;
– поиск одной или нескольких записей, удовлетворяющих заданному условию.

Для выполнения этих операций применяется механизм запросов. Результатом выполнения запросов является либо отобранное по определенным критериям множество записей, либо изменения в таблицах. Запросы к базе формируются на специально созданном для этого языке, который так и называется «язык структурированных запросов» (SQL – Structured Query Language).

Под управлением данными обычно понимают защиту данных от несанкционированного доступа, поддержку многопользовательского режима работы с данными и обеспечение целостности и согласованности данных.

1.2. Этапы проектирования реляционной базы данных

Основная причина сложности проектирования базы данных заключается в том, что объекты реального мира и взаимосвязи между ними вовсе не обязаны иметь и, как правило, не имеют структуры, согласованной с реляционной моделью данных. Разработчик при проектировании должен *придумать* представление для реальных объектов и их связей в терминах таблиц, полей, атрибутов, записей и т. п., то есть в терминах абстракций реляционной модели данных. Поэтому в данном контексте термин «проектирование» можно понимать и как процесс, результатом которого является *проект*, и как процесс, результатом которого является *проекция*.

Разработка эффективной базы данных состоит из нескольких этапов. Процесс разработки БД начинается с анализа требований. Проектировщик на этом этапе разработки должен найти ответы на следующие вопросы: какие элементы данных должны храниться, кто и как будет к ним обращаться.

На втором этапе создается логическая структура БД. Для этого определяют, как данные будут сгруппированы логически. Структура БД на этом этапе выражается в терминах прикладных объектов и отношений между ними.

На заключительном (третьем) этапе логическая структура БД преобразуется в физическую с учетом аспектов производительности. Элементы данных на этом этапе получают атрибуты и определяются как столбцы в таблицах выбранной для реализации БД СУБД.

Рассмотрим применение концепции реляционных баз данных на практике. Представим себе деятельность туристической фирмы. Очевидно, что для ее работы необходимо хранить и отслеживать определенный набор информации о клиентах данной турфирмы (туристах), о предлагаемых им турах, об оформлении и оплате путевок. Это можно делать в обычной бумажной тетради, но со временем поиск нужных записей и финансовая отчетность будут представлять собой довольно рутинную, длительную работу.

1.2.1. Определение требований

Требования к приложению с БД обычно составляются с помощью опросов и бесед с конечными пользователями. Это – итерационный процесс, в ходе которого разработчики определяют структуру пользовательских диалогов, критерии поиска документов и возможные реакции пользователей.

Общая методика определения и документирования требований к БД заключается в составлении словаря данных. *Словарь данных* перечисляет и определяет отдельные элементы данных, которые должны храниться в базе. Начальный проект словаря данных для менеджера турфирмы приведен в таблице 1.

Таблица 1

Словарь данных для приложения БД менеджера турфирмы

Элемент данных	**Описание**
Фамилия	Фамилия туриста
Имя	Имя туриста
Отчество	Отчество туриста
Паспорт	Серия и номер паспорта туриста
Телефон	Контактный телефон туриста
Город	Город проживания туриста
Страна	Страна проживания туриста
Индекс	Почтовый индекс адреса туриста
Тур	Название туристической поездки
Цена	Цена туристической поездки
Дата начала	Время начала туристической поездки
Дата конца	Время завершения туристической поездки
Информация	Дополнительная информация о туре
Дата оплаты	Дата оплаты путевки
Сумма	Сумма оплаты

Составление словаря – хороший способ, чтобы начать определять требования к базе данных. Но одного словаря не достаточно для определения структуры БД, так как словарь данных не описывает, как связаны элементы, как данные создаются, обновляются и выбираются, кто и как будет использовать БД.

Необходима *функциональная спецификация*, отражающая информацию о количестве одновременно работающих пользователей, о том, как часто записи будут вставляться и обновляться, и каким образом информация будет выбираться из БД.

Функциональное описание для приложения БД менеджера турфирмы могло бы включать, например, следующие требования:

- Приложением будут пользоваться руководитель турфирмы, 2 менеджера по продажам, бухгалтер, кассир и 2 офисных сотрудника турфирмы – всего 7 пользователей. Предполагается, что одновременно с БД будут работать не более 3 сотрудников. Персоналу бухгалтерии для работы достаточно иметь доступ только к данным по оплате путевок.
- Все пользователи в любое время могут добавлять информацию в БД. При добавлении информации или ее изменении, пользователь, который сделал изменение, а также дата и время изменения, должны быть зарегистрированы.
- Один из офисных сотрудников будет назначен системным администратором. Только он должен вести учетные записи пользователей.

Спецификация функций и словарь данных, как правило, разрабатываются одновременно, так как эти документы информационно дополняют друг друга.

Важная часть анализа требований – предупредить потребности пользователей, поскольку они не всегда способны полностью и четко объяснить их собственные требования к системе. Практически функциональное описание должно представлять систему как можно более полно и подробно.

1.2.2. Логическая модель

ER-диаграммы

Общим способом представления логической модели БД является построение ER-диаграмм (Entity-Relationship – сущность-связь). В этой модели сущность определяется как дискретный объект, для которого сохраняются элементы данных, а связь описывает отношение между двумя объектами.

В примере менеджера турфирмы имеются 5 основных объектов:

- Туристы
- Туры
- Путевки
- Сезоны
- Оплаты

Отношения между этими объектами могут быть определены простыми терминами:

- Каждый турист может купить одну или несколько (много) путевок.
- Каждой путевке соответствует ее оплата (оплат может быть и несколько, если путевка, например, продана в кредит).
- Каждый тур может иметь несколько сезонов.
- Путевка продается на один сезон одного тура.

Эти объекты и отношения могут быть представлены ER-диаграммой, как показано на рис. 2.

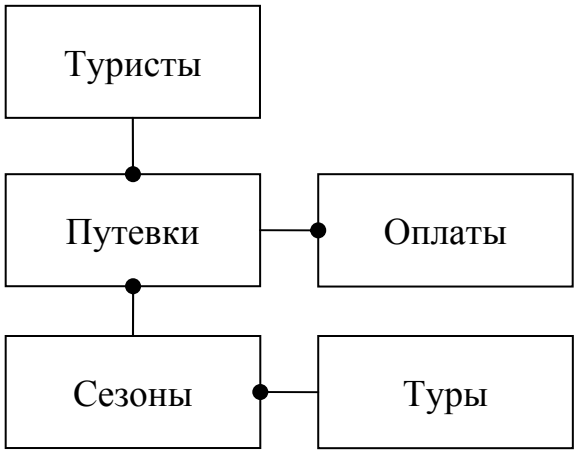


Рис. 2. ER-диаграмма для приложения БД менеджера турфирмы

Объекты, атрибуты и ключи

Далее модель развивается путем определения атрибутов для каждого объекта. *Атрибуты* объекта – это элементы данных, относящиеся к определенному объекту, которые должны сохраняться. Анализируем составленный словарь данных, выделяем в нем объекты и их атрибуты, расширяем словарь при необходимости. Атрибуты для каждого объекта в рассматриваемом примере представлены в таблице 2.

Таблица 2

Объекты и атрибуты БД

Объект	**Туристы**	**Путевки**	**Туры**	**Сезоны**	**Оплаты**
Атрибуты	Фамилия	Турист	Название	Дата начала	Дата оплаты
	Имя	Сезон	Цена	Дата конца	Сумма
	Отчество		Информация	Тур	Путевка
	Паспорт			Места	
	Телефон				
	Город				
	Страна				
	Индекс				

Следует обратить внимание, что несколько элементов отсутствуют. Опущена регистрационная информация, упомянутая в функциональной спецификации. Как ее учесть, вы подумаете самостоятельно и доработаете предложенный пример. Но более важно то, что пока отсутствуют атрибуты, необходимые для связи объектов друг с другом. Эти элементы данных в ER-модели не представ-

ляются, так как не являются, собственно, «натуральными» атрибутами объектов. Они обрабатываются по-другому и будут учтены в реляционной модели данных.

Реляционная модель характеризуется использованием ключей и отношений. Существует отличие в контексте реляционной базы данных терминов relation (отношение) и relationship (схема данных). *Отношение* рассматривается как неупорядоченная, двумерная таблица с несвязанными строками. *Схема данных* формируется между отношениями (таблицами) через общие атрибуты, которые являются *ключами*.

Существует несколько типов ключей, и они иногда отличаются только с точки зрения их взаимосвязи с другими атрибутами и отношениями. *Первичный ключ* уникально идентифицирует строку в отношении (таблице), и каждое отношение может иметь только один первичный ключ, даже если больше чем один атрибут является уникальным. В некоторых случаях требуется более одного атрибута для идентификации строк в отношении. Совокупность этих атрибутов называется *составным ключом*. В других случаях первичный ключ должен быть специально создан (сгенерирован). Например, в отношение «Туристы» имеет смысл добавить уникальный идентификатор туриста (код туриста) в виде первичного ключа этого отношения для организации связей с другими отношениями БД.

Другой тип ключа, называемый *внешним* ключом, существует только в терминах схемы данных между двумя отношениями. Внешний ключ в отношении – это атрибут, который является первичным ключом (или частью первичного ключа) в другом отношении. Это – распределенный атрибут, который формирует схему данных между двумя отношениями в БД.

Для проектируемой БД расширим атрибуты объектов кодовыми полями в качестве первичных ключей и используем эти коды в отношениях БД для ссылки на объекты БД следующим образом (табл. 3).

Построенную схему БД еще рано считать законченной, так как требуется ее нормализация. Процесс, известный как *нормализация* реляционной БД, используется для группировки атрибутов специальными способами, чтобы минимизировать избыточность и функциональную зависимость.

Таблица 3

Объекты и атрибуты БД с расширенными кодовыми полями

Объект	**Туристы**	**Путевки**	**Туры**	**Сезоны**	**Оплаты**
Атрибуты	Код туриста	Код путевки	Код тура	Код сезона	Код оплаты
	Фамилия	Код туриста	Название	Дата начала	Дата оплаты
	Имя	Код сезона	Цена	Дата конца	Сумма
	Отчество		Информация	Код тура	Код путевки
	Паспорт			Места	
	Телефон				
	…				

Нормализация

Функциональные зависимости проявляются, когда значение одного атрибута может быть определено из значения другого атрибута. Атрибут, который может быть определен, называется *функционально зависимым* от атрибута, который является детерминантом. Следовательно, по определению, все неключевые (без ключа) атрибуты будут функционально зависеть от первичного ключа в каждом отношении (так как первичный ключ уникально определяет каждую строку). Когда один атрибут отношения уникально не определяет другой атрибут, но ограничивает его набором предопределенных значений, это называется *многозначной* зависимостью. *Частичная* зависимость имеет место, когда атрибут отношения функционально зависит от одного атрибута составного ключа. Транзитивные зависимости наблюдаются, когда неключевой атрибут функционально зависит от одного или нескольких других неключевых атрибутов в отношении.

Процесс нормализации состоит в пошаговом построении БД в нормальной форме (НФ).

- Первая нормальная форма (1НФ) очень проста. Все таблицы БД должны удовлетворять единственному требованию – каждая ячейка в таблицах должна содержать атомарное значение, другими словами, хранимое значение в рамках предметной области приложения БД не должно иметь внутренней структуры, элементы которой могут потребоваться приложению.
- Вторая нормальная форма (2НФ) создается тогда, когда удалены все частичные зависимости из отношений БД. Если в отношениях не имеется никаких составных ключей, то этот уровень нормализации легко достигается.
- Третья нормальная форма (3НФ) БД требует удаления всех транзитивных зависимостей.
- Четвертая нормальная форма (4НФ) создается при удалении всех многозначных зависимостей.

БД нашего примера находится в 1НФ, так как все поля таблиц БД атомарные по своему содержанию. Наша БД также находится и во 2НФ, так как мы искусственно ввели в каждую таблицу уникальные коды для каждого объекта (Код Туриста, Код Путевки и т. д.), за счет чего и добились 2НФ для каждой из таблиц БД и всей базы данных в целом. Осталось разобраться с третьей и четвертой нормальными формами.

Обратите внимание, что они существуют только относительно различных видов зависимостей атрибутов БД. Есть зависимости – нужно стоить НФ БД, нет зависимостей – БД и так находится в НФ. Но последний вариант практически не встречается в реальных приложениях.

Итак, какие же транзитивные и многозначные зависимости присутствуют в нашем примере БД менеджера турфирмы?

Давайте проанализируем отношение «Туристы». Рассмотрим зависимости между атрибутами «Код туриста», «Фамилия», «Имя», «Отчество» и «Паспорт» (рис. 3). Каждый турист, представленный в отношении сочетанием «Фамилия-Имя-Отчество», имеет на время поездки только один паспорт, при этом полные тезки должны иметь разные номера паспортов. Поэтому атрибуты «Фамилия-Имя-Отчество» и «Паспорт» образуют в отношении туристы составной ключ.

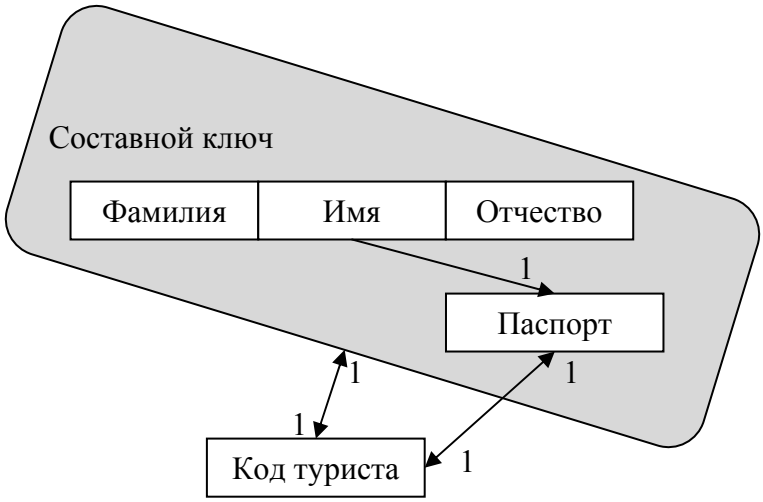


Рис. 3. Пример транзитивной зависимости

Как видно из рисунка, атрибут «Паспорт» транзитивно зависит от ключа «Код туриста». Поэтому, чтобы исключить данную транзитивную зависимость, разобьем составной ключ отношения и само отношение на 2 по связям «один-к-одному». В первое отношение, оставим ему имя «Туристы», включаются атрибуты «Код туриста» и «Фамилия», «Имя», «Отчество». Второе отношение, назовем его «Информация о туристах», образуют атрибуты «Код туриста» и все оставшиеся атрибуты отношения «Туристы»: «Паспорт», «Телефон», «Город», «Страна», «Индекс». Эти два новых отношения уже не имеют транзитивной зависимости и находятся в 3НФ.

Многозначные зависимости в нашей упрощенной БД отсутствуют. Для примера предположим, что для каждого туриста должны храниться несколько контактных телефонов (домашний, рабочий, сотовый и пр., что весьма характерно на практике), а не один, как в примере. Получаем многозначную зависимость ключа – «Код туриста» и атрибутов «Тип телефона» и «Телефон», в этой ситуации ключ перестает быть ключом. Что делать? Проблема решается также путем разбиения схемы отношения на 2 новые схемы. Одна из них должна представлять информацию о телефонах (отношение «Телефоны»), а вторая о туристах (отношение «Туристы»), которые связываются по полю «Код туриста». «Код туриста» в отношении «Туристы» будет первичным ключом, а в отношении «Телефоны» – внешним.

1.2.3. Физическая модель

Физическая модель данных зависит от выбранной СУБД. Например, если вы планируете использовать СУБД Oracle, то физическая база данных будет состоять из файлов данных, областей таблиц, сегментов отката, таблиц, столбцов и индексов.

В данном пособии будут рассмотрено создание физической модели БД средствами СУБД Microsoft Access и сервера баз данных Microsoft SQL Server 2005 Express Edition.

1.3. Создание БД в СУБД Microsoft Access

1.3.1. Таблицы

Для создания таблицы в СУБД Microsoft Access используем режим конструктора (рис. 4).

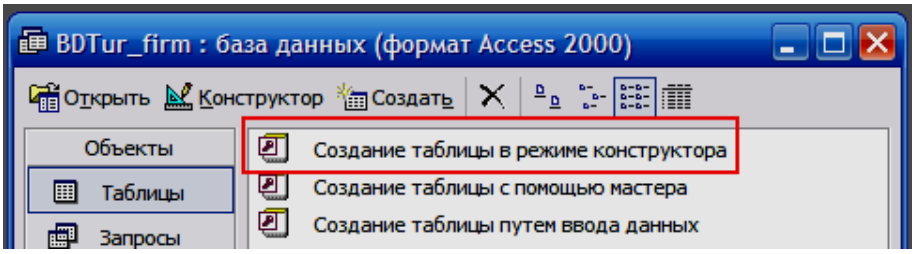


Рис. 4. Выбор режима конструктора

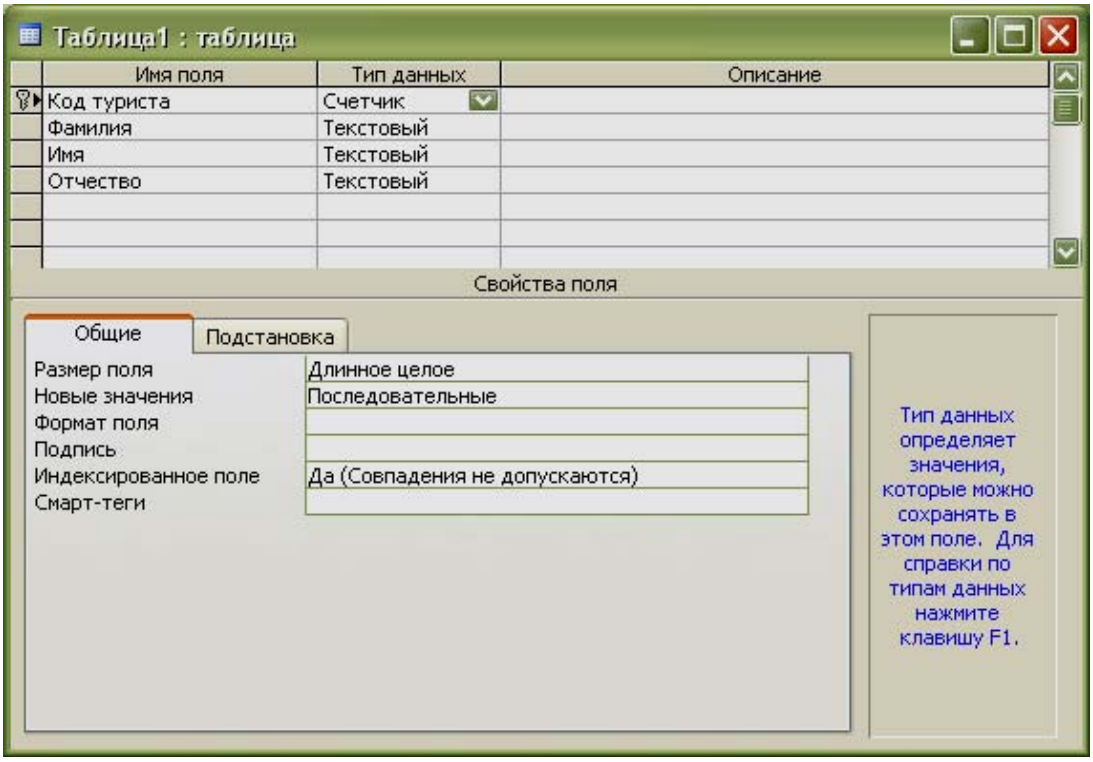


Рис. 5. Полный список полей таблицы

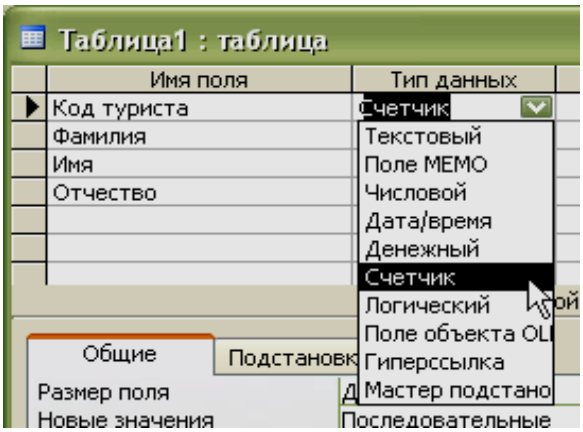


Рис. 6. Определение типа данных поля

В появившемся окне «Таблица1: таблица» предстоит определить названия полей, которые и станут заголовками в этой таблице. Введем следующие названия полей (рис. 5).

При вводе названия поля, для него по умолчанию определяется тип данных «текстовый». Для изменения типа следует выбрать нужное значение из выпадающего списка (рис. 6).

Описания возможных типов данных Microsoft Access приводятся в таблице 4.

Таблица 4

Типы данных Microsoft Access

Тип данных	**Описание**
Текстовый	Текст или комбинация текста и чисел, например, адреса, а также числа, не требующие вычислений, например, номера телефонов, инвентарные номера или почтовые индексы. Сохраняет до 255 знаков. Свойство «Размер поля» (FieldSize) определяет максимальное количество знаков, которые можно ввести в поле
Поле MEMO	Предназначено для ввода текстовой информации, по объему превышающей 255 символов. Такое поле может содержать до 65 535 символов. Этот тип данных отличается от типа Текстовый (Text) тем, что в таблице даются не сами данные, а ссылки на блоки данных, хранящиеся отдельно. За счет этого ускоряется обработка таблиц (сортировка, поиск и т. п.). Поле типа MEMO не может быть ключевым или проиндексированным
Числовой	Данные, используемые для математических вычислений, за исключением финансовых расчетов (для них следует использовать тип «Денежный»). Сохраняет 1, 2, 4 или 8 байтов. Конкретный тип числового поля определяется значением свойства Размер поля (FieldSize)
Дата/время	Значения дат и времени. Сохраняет 8 байтов
Денежный	Используется для денежных значений и для предотвращения округления во время вычислений. Сохраняет 8 байтов
Счетчик	Автоматическая вставка уникальных последовательных (увеличивающихся на 1) или случайных чисел при добавлении записи. Сохраняет 4 байта
Логический	Данные, принимающие только одно из двух возможных значений, таких, как «Да/Нет», «Истина/Ложь», «Вкл./Выкл.». Значения Null не допускаются. Сохраняет 1 бит.
Поле объекта OLE	Объекты OLE (такие, как документы Microsoft Word, электронные таблицы Microsoft Excel, рисунки, звукозапись или другие данные в двоичном формате) (ограничивается объемом диска)

Окончание таблицы 4

Тип данных	Описание
Гиперссылка	Гиперссылки. Гиперссылка может указывать на расположение файла на локальном компьютере либо адреса URL. Сохраняет до 64 000 знаков
Мастер подстановок	Создает поле, позволяющее выбрать значение из другой таблицы или из списка значений, используя поле со списком. При выборе данного параметра в списке типов данных запускается мастер для автоматического определения этого поля. Обычно сохраняет 4 байта

Применение определенного типа данных позволяет избежать ошибок в работе с таблицами – в поле с форматом даты невозможно ввести значение суммы, а в поле с денежным форматом невозможно ввести дату. Кроме того, для различных данных требуется разный объем памяти, и резервирование полей с однородным составом позволяет значительно уменьшить общий размер базы данных.

1.3.2. Ключи

Около поля «Код туриста» на рис. 5 находится изображение ключа. Это означает, что указанное поле будет первичным ключом для записей в таблице. Для того чтобы сделать данное поле ключевым, следует выделить его, щелкнуть на нем правой кнопкой мыши, а затем в появившемся контекстном меню выбрать команду «Ключевое поле» (рис. 7).

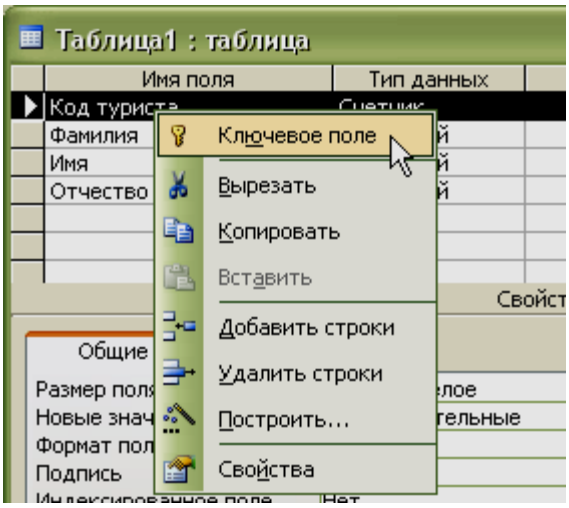


Рис. 7. Установка первичного ключа

Первая таблица готова. Сохраняем ее под названием «Туристы» и закрываем. Аналогичным образом создаем таблицы «Информация о туристах», «Туры», «Сезоны», «Путевки» и «Оплата» (таблица 5).

Таблица 5

Структура и описание таблиц базы данных BDTur_firm.mdb

№	Название	Структура в режиме конструктора		Описание
		Имя поля	**Тип данных**	
1	Туристы	Код туриста	Счетчик	Содержит основные сведения о туристе
		Фамилия	Текстовый	
		Имя	Текстовый	
		Отчество	Текстовый	
2	Информация о туристах	Код туриста	Числовой	Содержит дополнительные сведения о туристе, которые были вынесены в отдельную таблицу – для избегания повторяющихся записей
		Серия паспорта	Текстовый	
		Город	Текстовый	
		Страна	Текстовый	
		Телефон	Текстовый	
		Индекс	Числовой	
3	Туры	Код тура	Счетчик	Содержит общие сведения о странах для туров
		Название	Текстовый	
		Цена	Денежный	
		Информация	Поле МЕМО	
4	Сезоны	Код сезона	Счетчик	Содержит сведения о сезонах – некоторые туры доступны лишь в определенный период
		Код тура	Числовой	
		Дата начала	Дата/время	
		Дата конца	Дата/время	
		Сезон закрыт	Логический	
		Количество мест	Числовой	
5	Путевки	Код путевки	Числовой	Содержит сведения о путевках, реализованных туристам
		Код туриста	Числовой	
		Код сезона	Числовой	
6	Оплата	Код оплаты	Счетчик	Содержит сведения об оплате за путевки
		Код путевки	Числовой	
		Дата оплаты	Дата/время	
		Сумма	Денежный	

Теперь в окне базы данных есть несколько таблиц. Обратите внимание на наличие в нескольких таблицах одинаковых полей, например, в таблицах «Туристы» и «Информация о туристах» поле «Код туриста».

1.3.3. Связи

Приступим к связыванию таблиц.

В окне базы данных щелкаем правой кнопкой мыши на чистом месте и в появившемся меню выбираем «Схема данных» (или в главном меню выбираем «Сервис \ Схема данных»).

В появившемся окне «Добавление таблицы» выделяем все таблицы и нажимаем кнопки «Добавить» и «Закрыть». В окне «Схема данных» добавленные таблицы можно перетаскивать, располагая удобным способом.

Выделив поле «Код туриста» в таблице «Туристы» и не отпуская левой кнопки мыши, перетащим его на поле «Код туриста» таблицы «Информация о туристах» (рис. 8).

Рис. 8. Создание связи между таблицами

После отпускания кнопки мыши появляется окно «Изменение связей», в котором отмечаем галочки «Обеспечение целостности данных», «Каскадное обновление связанных полей» и «Каскадное удаление связанных записей», а затем нажимаем кнопку «Создать» (рис. 9).

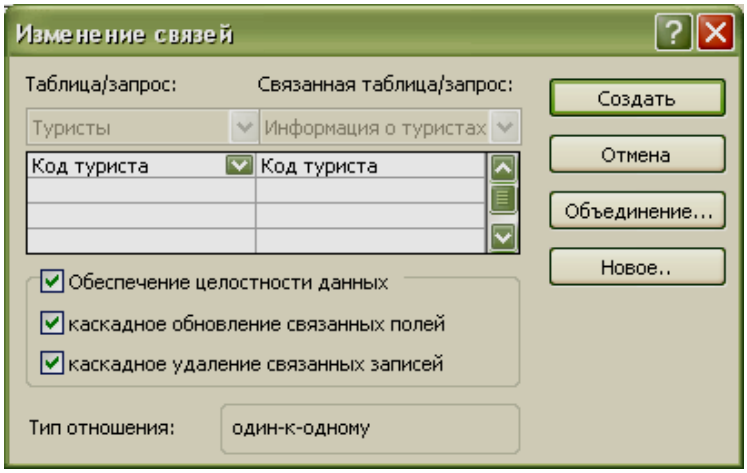


Рис. 9. Определение параметров связи

Определение этих параметров позволит автоматически обновлять связанные записи в таблицах при их изменении. В окне появилась связь между таблицами, которая была создана Microsoft Access (рис. 10).

Эта связь была создана автоматически – так происходит тогда, когда две таблицы имеют одинаковые названия связанных полей и *согласованные типы* данных, причем хотя бы в одной из таблиц связанное поле является ключевым.

Под согласованным типом данных понимается следующее: если ключевое поле имеет тип данных «Счетчик», то соответствующее ему поле в другой таблице должно иметь тип «Числовой». В остальных случаях типы данных должны просто совпадать.

Рис. 10. Связь между таблицами

Около полей «Код туриста» обеих таблиц на связи расположено изображение единицы, указывающее на принадлежность связи к отношению «один-к-одному». Это означает, что одной записи в таблице «Туристы» будет соответствовать одна запись в таблице «Информация о туристах». Существуют также другие типы отношений – «один-ко-многим» и «многие-ко-многим». Отношение «один-ко-многим» далее появится у нас между таблицами «Информация о туристах» и «Путевки» – один турист может приобрести несколько путевок, что и находит логическое отражение в связи между таблицами.

Другой возможный тип – «многие-ко-многим» в нашей базе данных отсутствует, но его примером может служить связь между таблицами с преподавателями и предметами: один преподаватель может вести несколько предметов, но и один предмет могут вести несколько преподавателей.

Продолжая определение связей между таблицами, получим следующую схему базы данных (рис. 11).

Для дальнейшей работы с базой данных заполним ее данными. В окне базы данных дважды щелкнем на таблице «Туристы». Щелкая на значок «+» этой записи, можно отобразить и непосредственно вносить изменения в дочерних таблицах. *Дочерние таблицы* образуют группу, определенную в структуре базы данных.

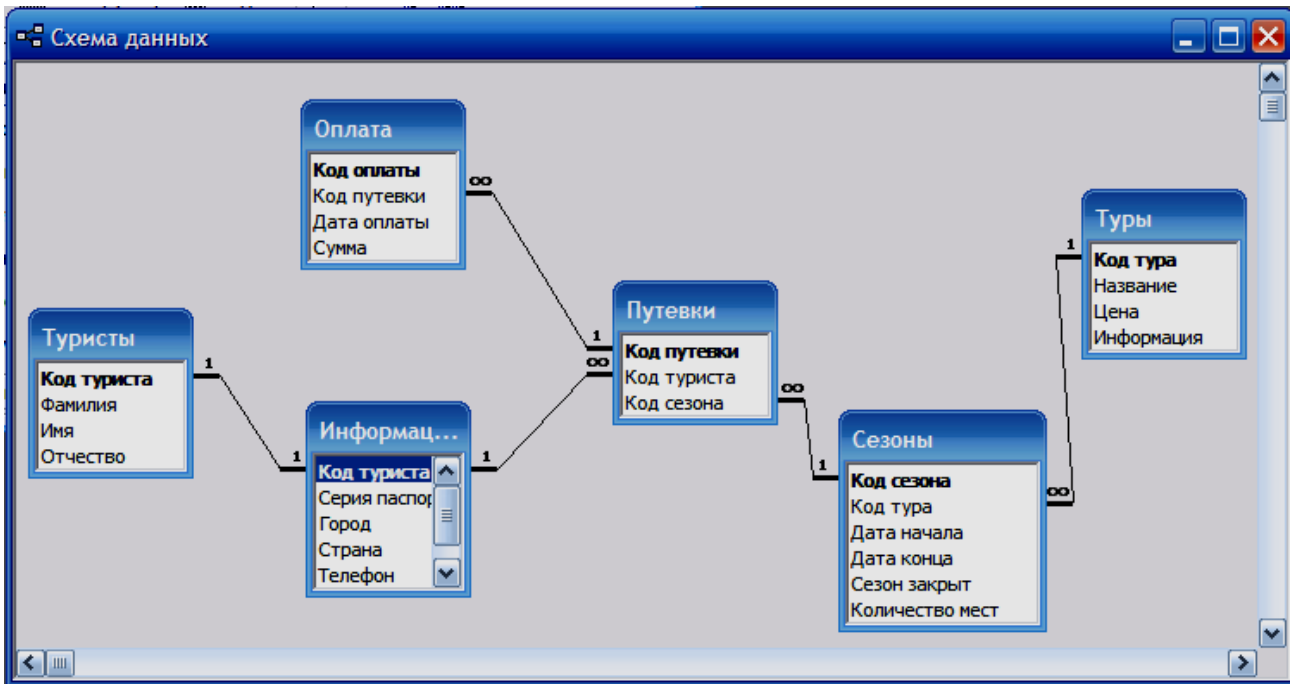
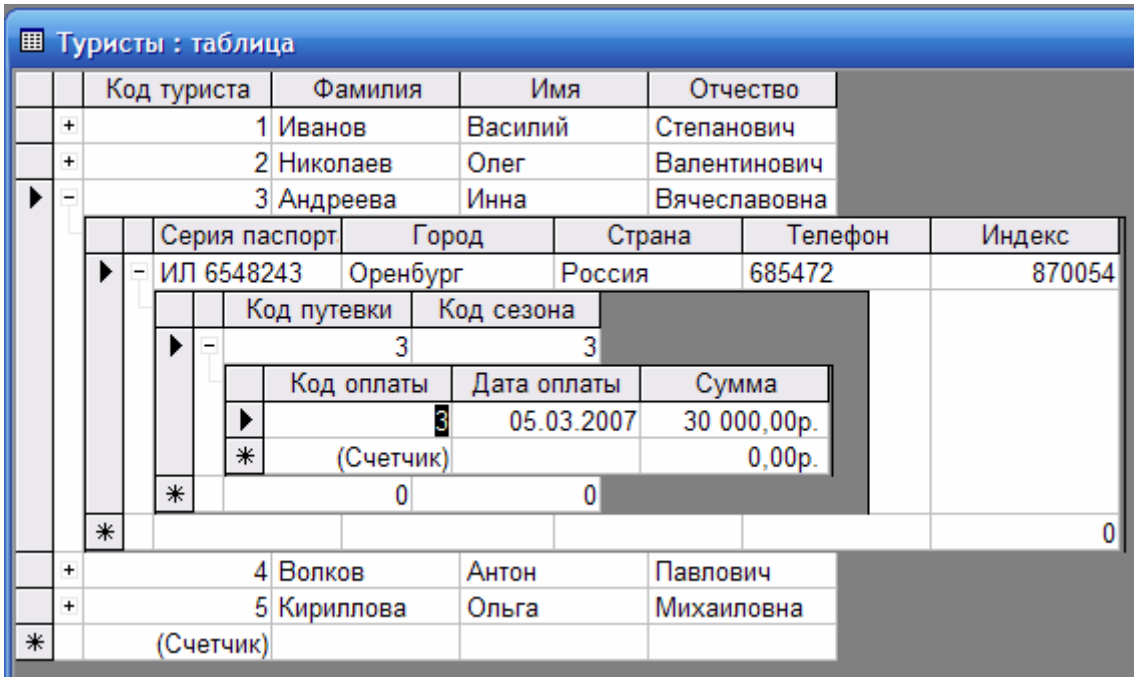


Рис. 11. Схема данных базы BDTur_firm.mdb

На рис. 12 приведена раскрытая группа таблиц – «Туристы» – «Информация о туристах» – «Путевки» – «Оплата». По схеме данных несложно определить таблицы, представленные на данном рисунке.



Туристы : таблица

Код туриста	Фамилия	Имя	Отчество
1	Иванов	Василий	Степанович
2	Николаев	Олег	Валентинович
3	Андреева	Инна	Вячеславовна
4	Волков	Антон	Павлович
5	Кириллова	Ольга	Михаиловна
(Счетчик)			

Серия паспорт	Город	Страна	Телефон	Индекс
ИЛ 6548243	Оренбург	Россия	685472	870054
				0

Код путевки	Код сезона
3	3
0	0

Код оплаты	Дата оплаты	Сумма
3	05.03.2007	30 000,00р.
(Счетчик)		0,00р.

Рис. 12. Группа вложенных таблиц

Таблица «Туры» также содержит вложенную группу дочерних таблиц «Туры» – «Сезоны» – «Путевки» – «Оплата» (рис. 13).

Туры : таблица

Код тура	Название	Цена	Информация
1	Кипр	25 000,00р.	В стоимость двух взрослых путевок входит цена одной детской (до 7лет)
2	Греция	32 000,00р.	В августе и сентябре действуют специальные скидки
3	Таиланд	30 000,00р.	Не включая стоимость авиабилета
4	Италия	26 000,00р.	Завтрак в отеле включен в стоимость путевки

Код сезона	Дата начала	Дата конца	Сезон закрыт	Количество мест
4	01.03.2007	01.09.2007	☐	12

Код путевки	Код туриста
4	4

Код оплаты	Дата оплаты	Сумма
4	02.02.2007	26 000,00р.
(Счетчик)		0,00р.

0	0
(Счетчик)	0

5	Франция	27 000,00р.	Дополнительные экскурсии не входят в стоимость путевки
(Счетчик)		0,00р.	

Рис. 13. Вложенная группа таблиц «Туры» – «Сезоны» – «Путевки» – «Оплата»

В результате проделанной работы была создана база данных Microsoft Access, которая может применяться для управления туристической фирмой. Непосредственное использование таблиц – простое их открытие и внесение данных – встречается крайне редко: отсутствие интерфейса, а главное – отсутствие защиты от случайных ошибок делает всю работу весьма ненадежной. Тем не менее, саму базу данных можно считать готовой серверной частью двухуровневого приложения «клиент-сервер». СУБД Microsoft Access содержит все средства для эффективной разработки клиентской части приложения (форм, отчетов, страниц).

Программа Microsoft Access с самого начала создавалась как средство управления и проектирования баз данных для офисной работы и задач небольших организаций. Ограничение максимального количества одновременно работающих пользователей (всего 255) делает невозможным использование базы данных даже для управления среднего по размерам Интернет-магазина или форума. Для обслуживания крупных проектов используются более мощные системы, например, Microsoft SQL Server.

1.4. Создание базы данных в среде Microsoft SQL Server

В составе Microsoft Visual Studio 2008 находится сервер баз данных Microsoft SQL Server 2005 Express Edition. От полнофункционального сервера данных он отличается только ограничением размера базы данных в 2 гигабайта, что позволяет производить разработку и тестирование приложений баз данных.

Для работы по созданию базы данных и таблиц будем использовать Microsoft SQL Server Management Studio Express. Данный программный продукт является свободнораспространяемым и доступен для скачивания по следующему адресу: http://www.microsoft.com/downloads/details.aspx?familyid=C243A5AE-4BD1-4E3D-94B8-5A0F62BF7796&displaylang=ru

1.4.1. Определение структуры базы данных

Внешний вид окна программы Microsoft SQL Server Management Studio Express приведен на рис. 14.

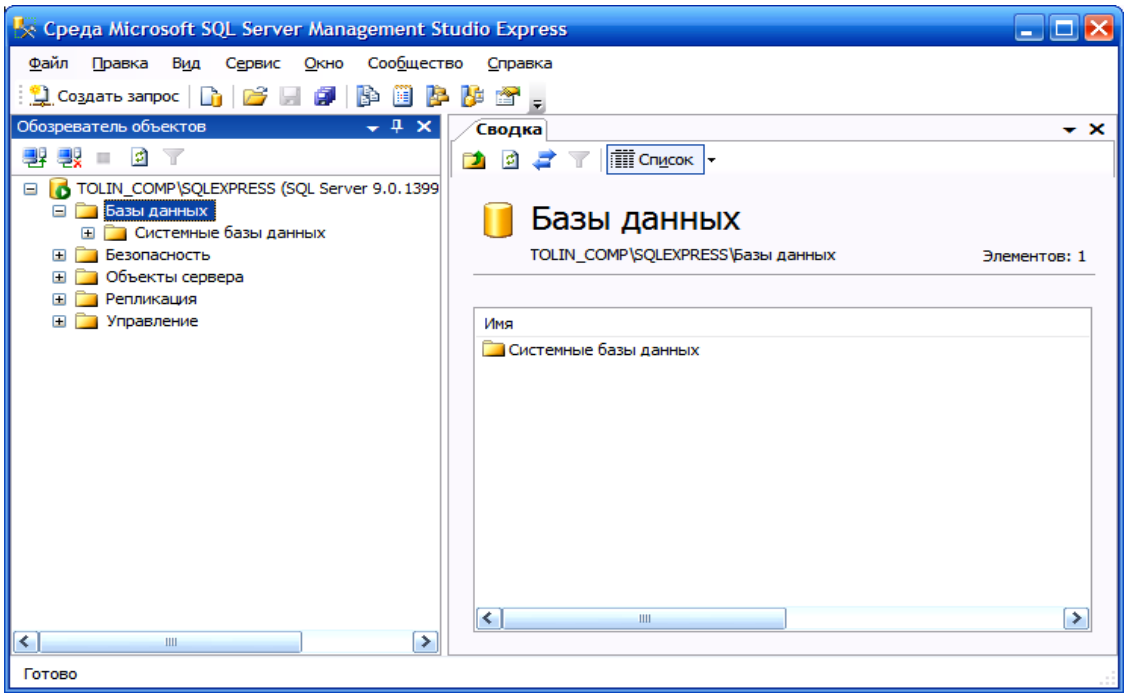


Рис. 14. Внешний вид окна программы Microsoft SQL Server Management Studio Express

Для создания базы данных необходимо кликнуть правой кнопкой мыши на пункте «Базы данных» и выбрать пункт меню «Создать базу данных». Окно создания БД представлено на рис. 15.

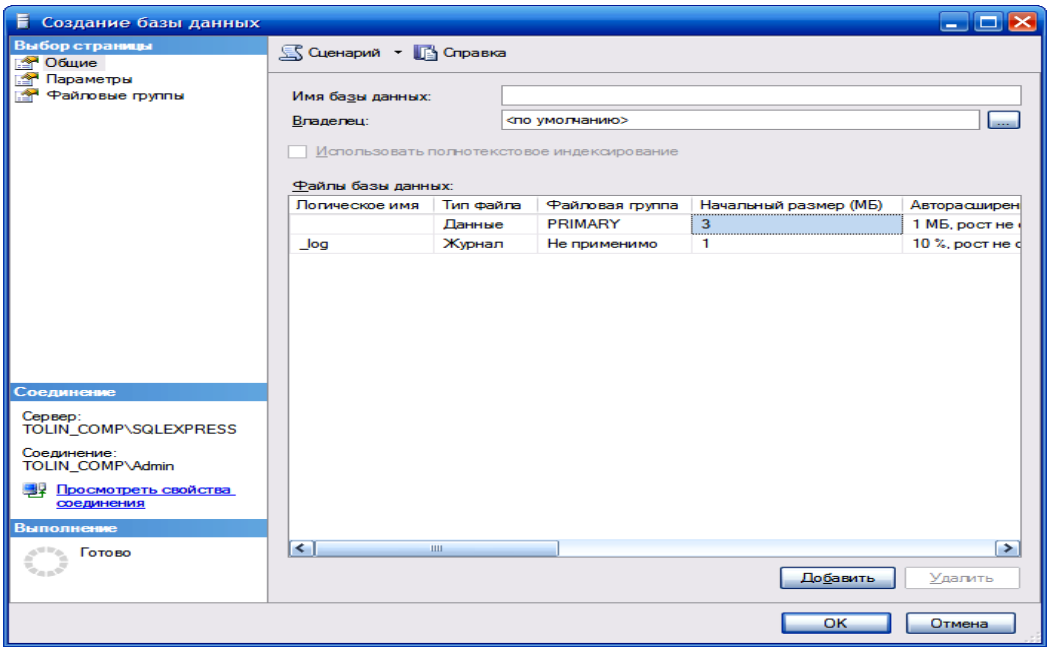


Рис. 15. Окно создания БД

В данном окне задается имя базы данных, имена и пути к файлам базы данных, начальный размер файлов и шаг увеличения размера БД в случае необходимости. После нажатия кнопки «ОК» созданная БД появляется в списке баз данных (рис. 16).

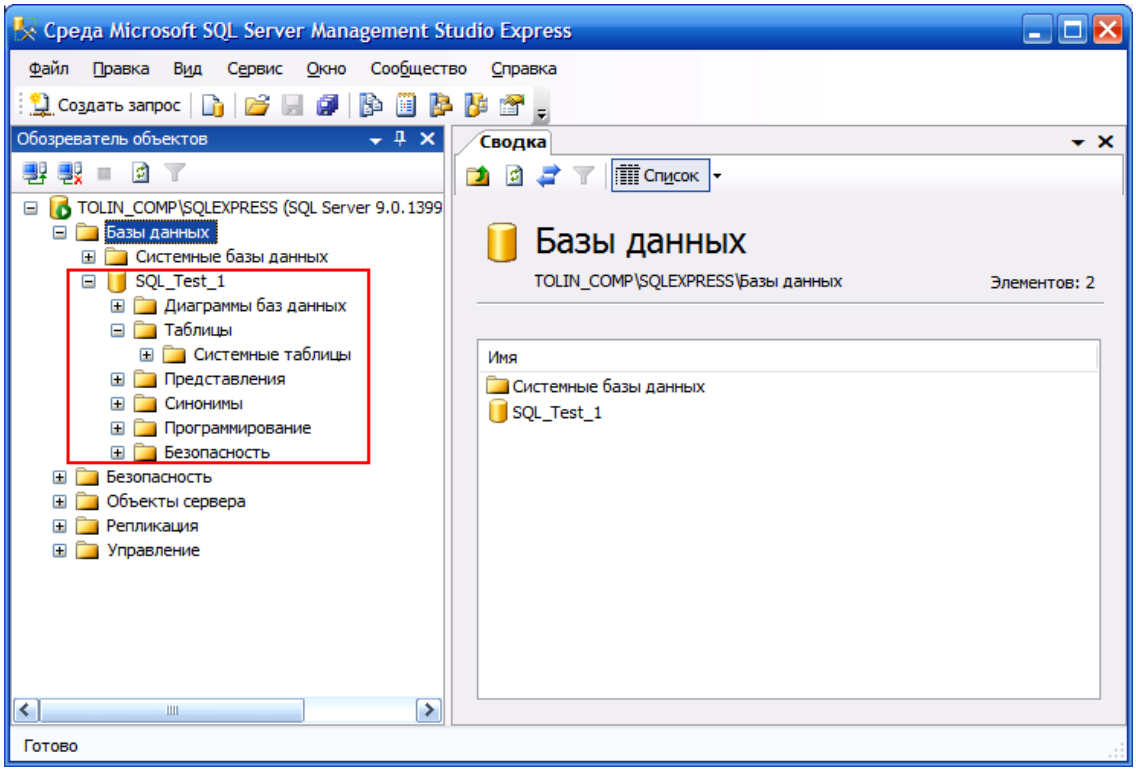


Рис. 16. Вид Management Studio с созданной базой данных

Созданная база данных пуста, т. е. не содержит ни одной таблицы. Поэтому следующей задачей является создание таблиц, структура которых аналогична таблицам из базы данных Access. При создании таблиц необходимо обратить внимание на соотношения типов Access и SQL Server, представленные в таблице 6.

Таблица 6

Соответствие типов данных Microsoft Access и Microsoft SQL

№	Тип данных Microsoft Access	Тип данных Microsoft SQL	Описание типа данных Microsoft SQL
1	Текстовый	nvarchar	Тип данных для хранения текста до 4000 символов
2	Поле MEMO	ntext	Тип данных для хранения символов в кодировке Unicode до $2^{30} - 1$ (1 073 741 823) символов
3	Числовой	int	Численные значения (целые) в диапазоне от -2^{31} (–2 147 483 648) до $2^{31} - 1$ (+2 147 483 647)

Окончание таблицы 6

№	Тип данных Microsoft Access	Тип данных Microsoft SQL	Описание типа данных Microsoft SQL
4	Дата/время	smalldatetime	Дата и время от 1 января 1900 г. до 6 июня 2079 года с точностью до одной минуты
5	Денежный	money	Денежный тип данных, значения которого лежат в диапазоне от -2^{63} (–922 337 203 685 477.5808) до $2^{63}-1$ (+922 337 203 685 477.5807), с точностью до одной десятитысячной
6	Счетчик	int	См. пункт 3
7	Логический	bit	Переменная, способная принимать только два значения – 0 или 1
8	Поле объекта OLE	image	Переменная для хранения массива байтов от 0 до $2^{31}-1$ (2 147 483 647) байт
9	Гиперссылка	ntext	См. пункт 2
10	Мастер подстановок	nvarchar	См. пункт 1

Для создания таблиц необходимо выбрать в контекстном меню ветки «Таблицы» пункт «Создать таблицу». Среда Management Studio принимает следующий вид (рис. 17).

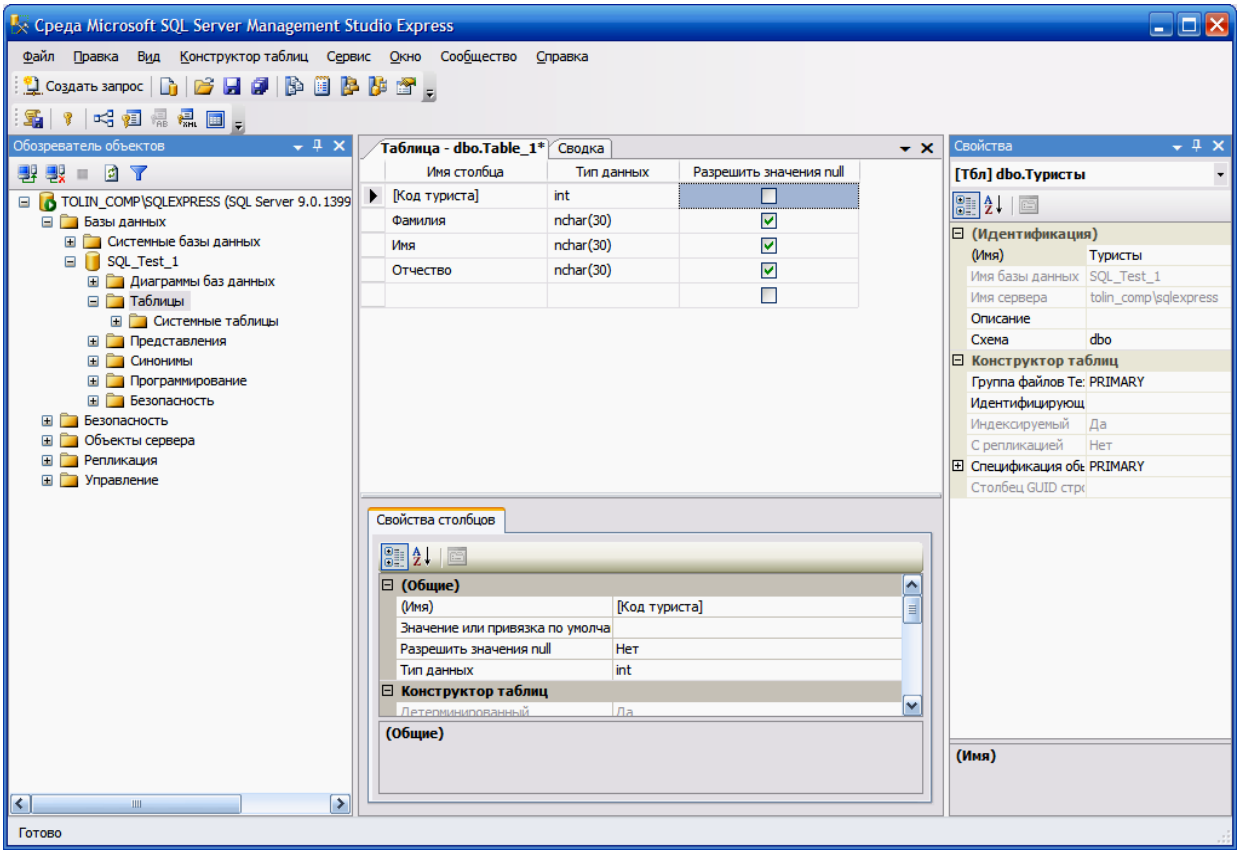


Рис. 17. Среда Management Studio в режиме создания таблицы

Для определения связей между таблицами необходимо задать первичные ключи таблиц. Для этого в контекстном меню соответствующего поля выбрать пункт «Задать первичный ключ» (рис. 18).

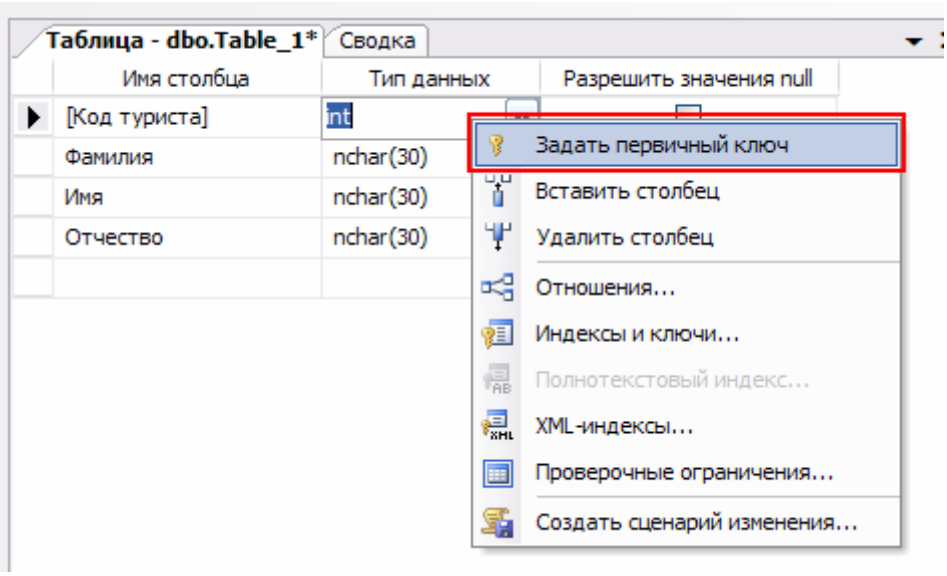


Рис. 18. Задание первичного ключа

Для создания связей между таблицами и схемы базы данных необходимо создать новую диаграмму базы данных, выбрав соответствующий пункт в контекстном меню ветви «Диаграммы баз данных». Добавив в появившемся окне необходимые таблицы в диаграмму, получаем следующий вид среды Management Studio (рис. 19).

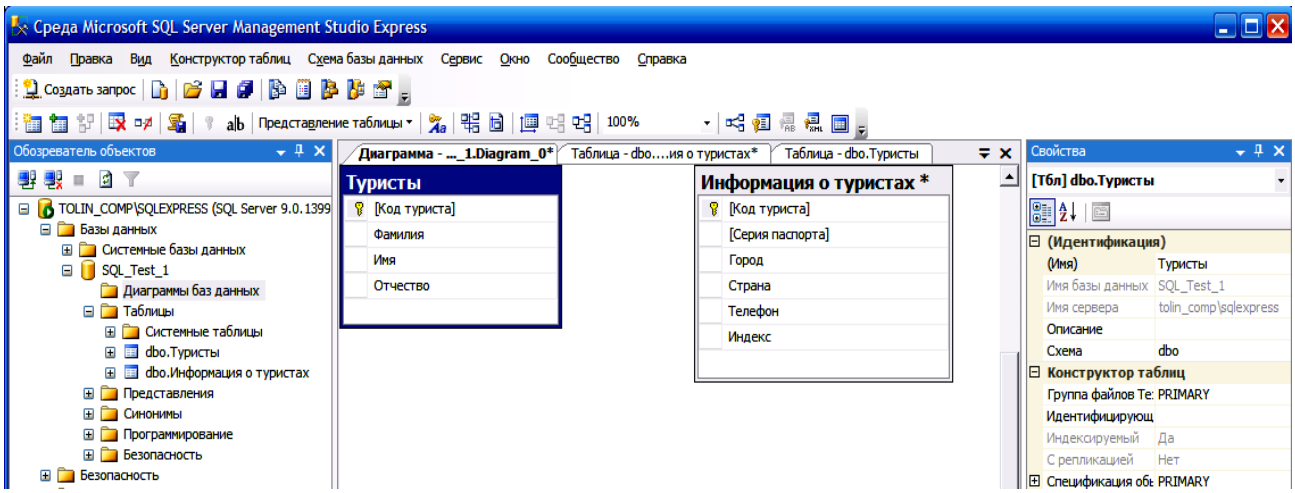


Рис. 19. Начало построения диаграммы БД

Создание связей происходит путем совмещения связываемых полей. Результатом становится появление окна создания отношения (рис. 20).

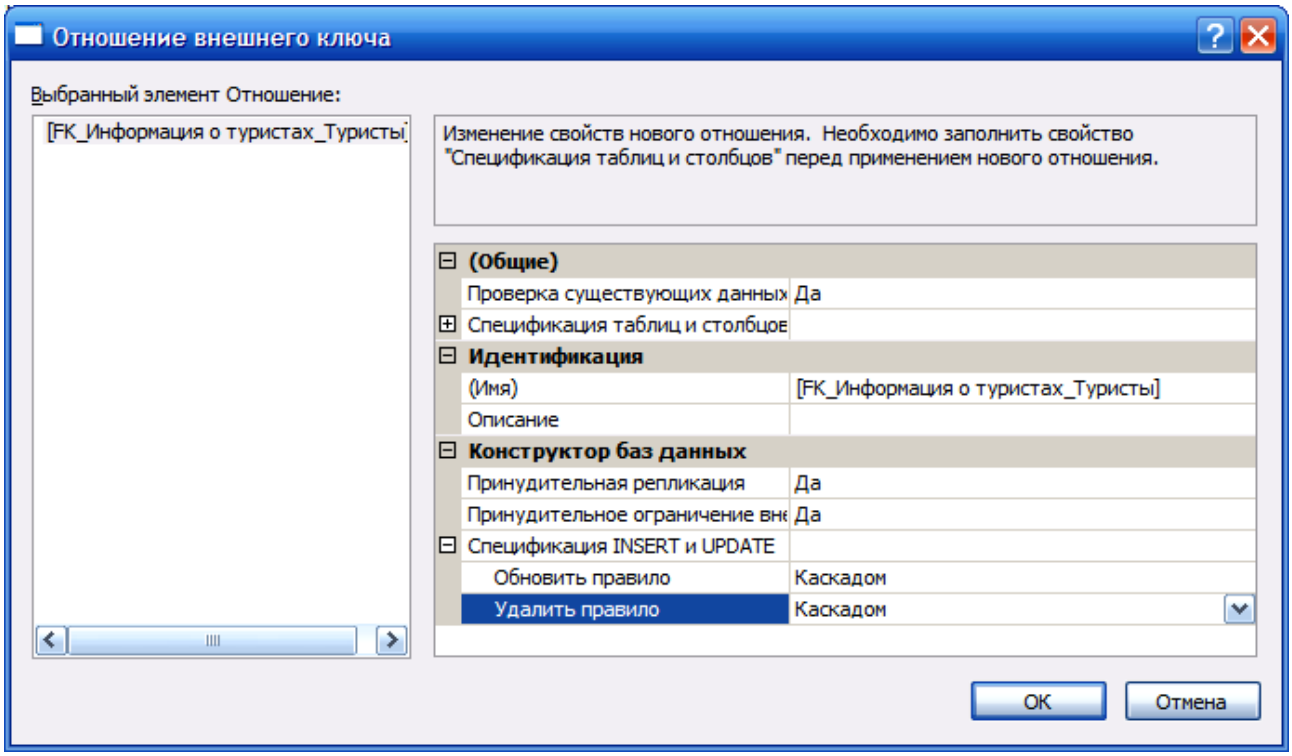


Рис. 20. Создание отношения между таблицами БД

Особо отметим пункт «Спецификация INSERT и UPDATE», задающий правила обновления и удаления связанных данных в таблицах.

После создания остальных таблиц и их связей схема данных будет выглядеть следующим образом (рис. 21).

В отличие от схемы данных Microsoft Access, здесь линии, отображающие связи по умолчанию, не привязываются графически к первичным и вторичным полям. Однако при щелчке левой кнопкой на любой связи в панели свойств появляется информация о выбранном отношении.

Завершив работу со схемой данных, сохраняем ее. Отметим, что в SQL Management Studio, в отличие от Access, для одной базы данных может быть создано несколько диаграмм (рис. 22).

Данная возможность является полезной для баз данных с очень большим количеством таблиц, так как одна общая диаграмма была бы слишком нагруженной.

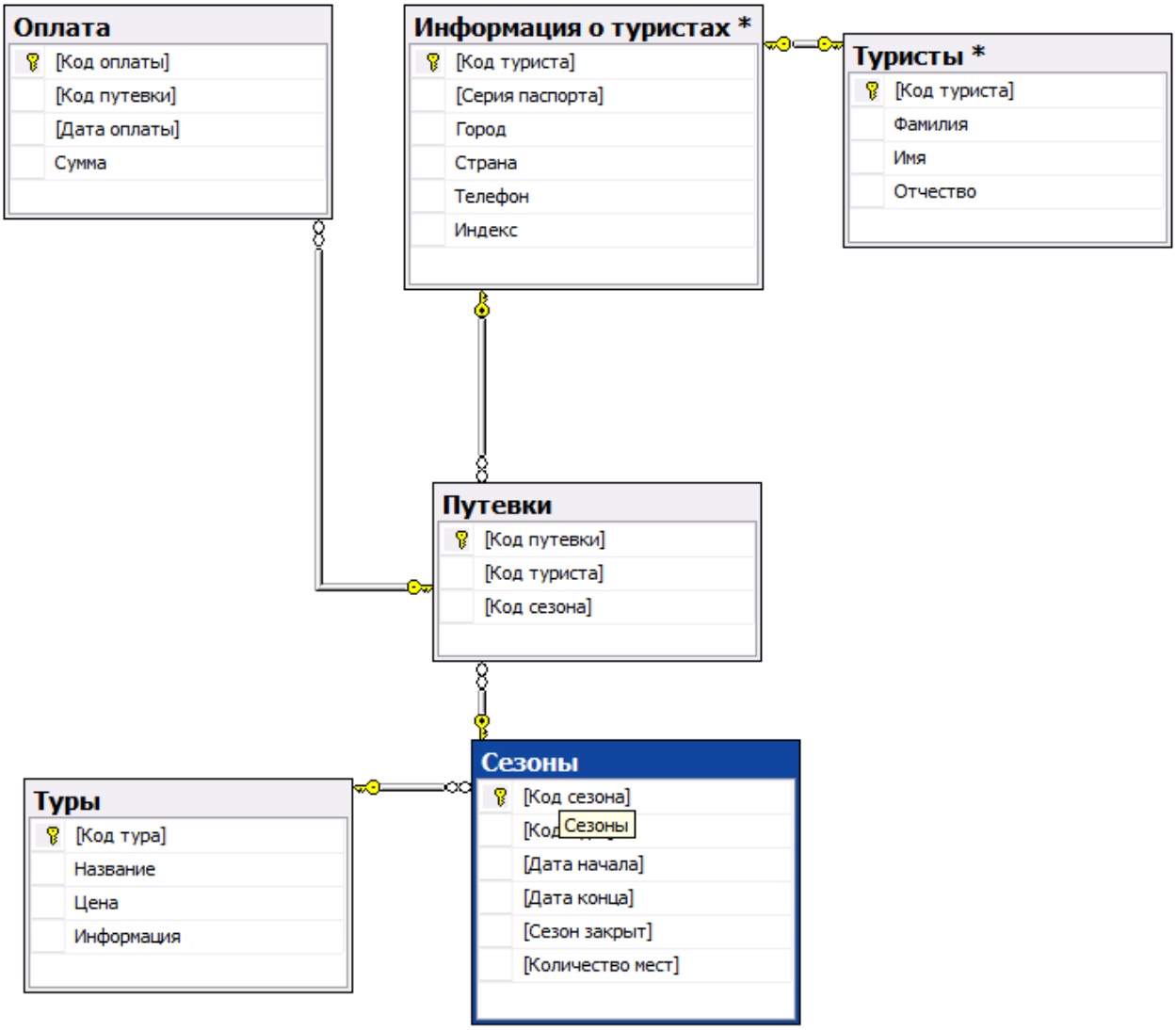


Рис. 21. Схема базы данных BDTur_firmSQL

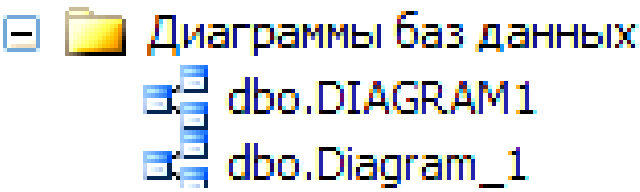


Рис. 22. Несколько диаграмм для одной БД

1.4.2. Перенос файла БД Microsoft SQL на другой компьютер

В большинстве случаев необходимо разрабатывать приложения, использующие в качестве базы данных Microsoft SQL Server. Наиболее рациональным решением является разработка базы данных в формате Microsoft SQL на рабо-

чем компьютере с установленной локальной версией Microsoft SQL Server. При сдаче проекта заказчику возникает необходимость переноса базы данных с локального компьютера. Для переноса на другой компьютер нам потребуется скопировать два файла – саму базу данных BDTur_firmSQL.mdf и файл отчетов о транзакциях BDTur_firmSQL.ldf. Однако непосредственное копирование данных файлов невозможно, так как данные файлы используются сервером баз данных. Для того чтобы сделать файлы доступными для копирования, базу данных необходимо отсоединить от сервера (рис. 23).

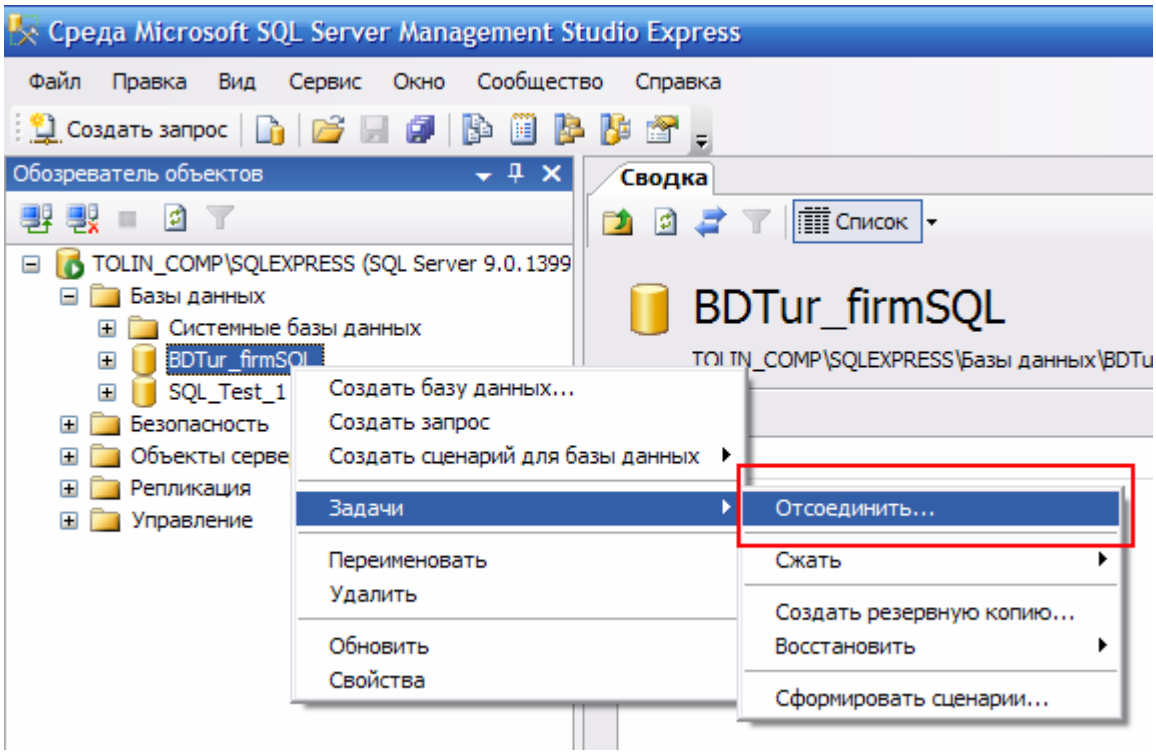


Рис. 23. Отсоединение выбранной базы данных от сервера

Появляется диалоговое окно «Отсоединение базы данных». Подтверждаем отсоединение, нажимая кнопку «ОК», – и база отсоединена. Теперь нужные файлы доступны для копирования.

Для присоединения базы данных на другом компьютере запускаем SQL Management Studio, выделяем ветку «Базы данных» и в контекстном меню выбираем «Присоединить» (рис. 24).

В появившемся окне указываем расположение файла базы данных BDTur_firmSQL.mdf – файл отчетов присоединится автоматически – и нажимаем «ОК». Присоединившаяся база данных немедленно отображается в папке «Базы данных». Следует отметить, что после присоединения БД может потребоваться настройка пользователей БД и прав доступа.

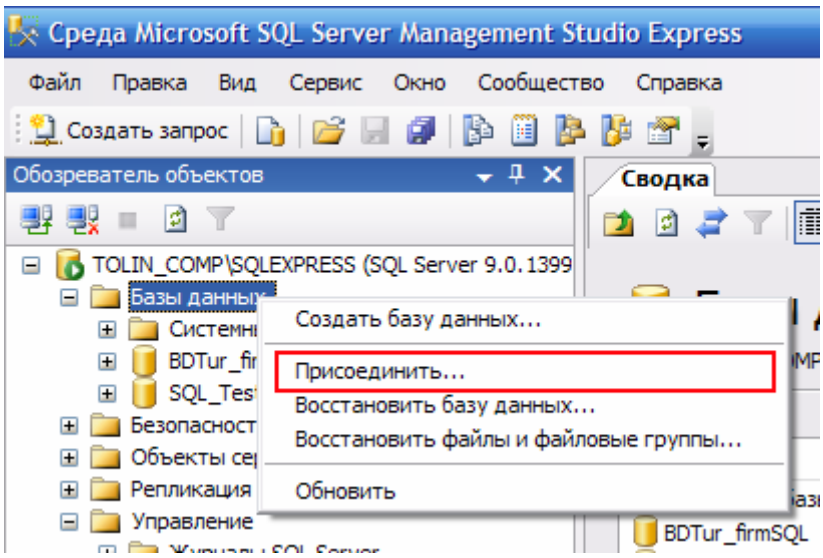


Рис. 24. Присоединение базы данных

1.5. Контрольные вопросы и задания к разделу 1

1. Является ли схемой реляционной БД следующий набор из пяти абстрактных атрибутов: A B C A M? Почему?
2. Является ли отношением следующий набор записей со схемой [ABC]: (a, в, c), (a1, в, c), (a1, в1, c), (a2, в, c1), (a, в, c), (a1, в, c1), (a2, в1, c)? Почему?
3. Пусть дана следующая схема отношения [СЛУЖАЩИЙ, РУКОВОДИТЕЛЬ, ДОЛЖНОСТЬ, ЗАРПЛАТА, СТАЖ], где атрибуты СЛУЖАЩИЙ и РУКОВОДИТЕЛЬ имеют в качестве значений фамилии, ДОЛЖНОСТЬ – названия должностей, ЗАРПЛАТА – числа, выражающие размер месячного оклада в рублях, СТАЖ – количество полных лет, которые проработал служащий на должности. Выберите первичный ключ для данного отношения. Какие зависимости могут быть выделены в данном отношении?
4. Пусть результатами предварительного анализа данных о служащих некоторой авиакомпании является следующая информация:
   a. Иванов, Петров и Сидоров – агенты по продаже билетов;
   b. Савельев принимает багаж;
   c. Кашин – авиамеханик;
   d. Леонов руководит всеми агентами по продаже билетов;
   e. У Павлова в подчинении Савельев;
   f. Кораблев отвечает за работу Кашина, Леонова, Павлова и самого себя;
   g. Павлов – начальник наземных служб, а Кораблев – начальник по эксплуатации.

h. Каждый служащий получает надбавку 10% за каждый полный проработанный год.
i. Иванов, Петров, Сидоров и Савельев начали с оклада в 12 000 руб., Иванов только приступил к работе, Петров и Савельев работают полтора года, а Сидоров – 2 года.
j. Кашин начал с оклада 18 000 руб. и сейчас зарабатывает 21 780.
k. Леонов и Павлов начали с оклада 16 000 руб., и оба работают 3 года.
l. Кораблев начал с оклада 20 000 руб. и проработал на 2 года больше, чем остальные.

Создайте в БД отношение со схемой, описанной в п. 3, и занесите в него данную информацию.

5. Выделите ключи для следующей схемы отношения [№ РЕЙСА, АЭРОПОРТ НАЗНАЧЕНИЯ, ВЫХОД, ДАТА, ВРЕМЯ], которая означает, что посадка на рейс № РЕЙСА, вылетающий в город АЭРОПОРТ НАЗНАЧЕНИЯ, осуществляется через выход номер ВЫХОД, дата отправления рейса – ДАТА, время вылета – ВРЕМЯ.
6. Может ли объединение двух ключей быть ключом?
7. Обязательно ли пересечение двух наборов атрибутов, содержащих ключи отношения, является ключом?
8. Каково максимальное число простых первичных ключей может иметь отношение со схемой [A1, A2, .., AN]? Каково максимальное число составных ключей может иметь это отношение?
9. Постройте словарь данных для отношения из заданий 3,4.
10. Придумайте порядок регистрации документов для приложения-примера БД менеджера турфирмы, определите регистрационную информацию и доработайте этот пример с указанными добавлениями.
11. Пусть дано отношение со схемой [СТУДЕНТ, ИМЯ, ДЕНЬ_РОЖДЕНИЯ, ВОЗРАСТ, КОНСУЛЬТАНТ, ФАКУЛЬТЕТ, СЕМЕСТР, КУРС, ГРУППА], на котором выполняются следующие функциональные зависимости:
    a. СТУДЕНТ→ИМЯ, ДЕНЬ_РОЖДЕНИЯ, ВОЗРАСТ, КОНСУЛЬТАНТ, ФАКУЛЬТЕТ;
    b. ДЕНЬ_РОЖДЕНИЯ→ВОЗРАСТ;
    c. КОНСУЛЬТАНТ→ФАКУЛЬТЕТ.

    Постройте для данного отношения схему БД в 3НФ.
12. Расширьте возможности приложения-примера БД менеджера турфирмы, изменив разработанную схему БД таким образом, чтобы можно было хранить несколько контактных телефонов для каждого туриста.

2. ОСНОВЫ ЯЗЫКА SQL

Постепенная унификация баз данных привела к необходимости создания стандартного языка доступа к данным БД, который мог бы использоваться для функционирования в большом количестве различных видов компьютерных сред. Стандартный язык позволит пользователям, знающим один набор команд, применять их, чтобы создавать, отыскивать, изменять и передавать информацию независимо от используемого сервера баз данных и его местоположения.

SQL (Structured Query Language), или Структурированный Язык Запросов, – это язык, который дает возможность работать с данными в реляционных базах данных. Стандарт SQL определяется ANSI (Американским Национальным Институтом Стандартов), а также ISO (Международной организацией по стандартизации). Однако большинство коммерческих программ баз данных расширяют SQL, добавляя разные особенности в этот язык, которые, как они считают, будут полезны. Эти дополнения являются не стандартизированными и часто приводят к сложностям при переходе от одного сервера данных к другому.

Для обращения к базе данных используются запросы, написанные на языке SQL. *Запросом* называется команда, которая передается серверу базы данных, и которая сообщает ему, что нужно вывести определенную информацию из таблиц в память. Эта информация обычно посылается непосредственно на экран компьютера или терминала, хотя в большинстве случаев ее можно также передать на принтер, сохранить в файле (как объект в памяти компьютера) или представить как вводную информацию для другой команды или процесса.

Несмотря на большое количество разновидностей этого языка, существующих в настоящее время, логика его работы проста. Достаточно освоить основные команды хотя бы в одной из его версий, чтобы впоследствии без труда разобраться в любой другой его реализации.

Для выполнения SQL-запросов будем использовать SQL Management Studio. При запуске среды Management Studio появляется следующее окно (рис. 25).

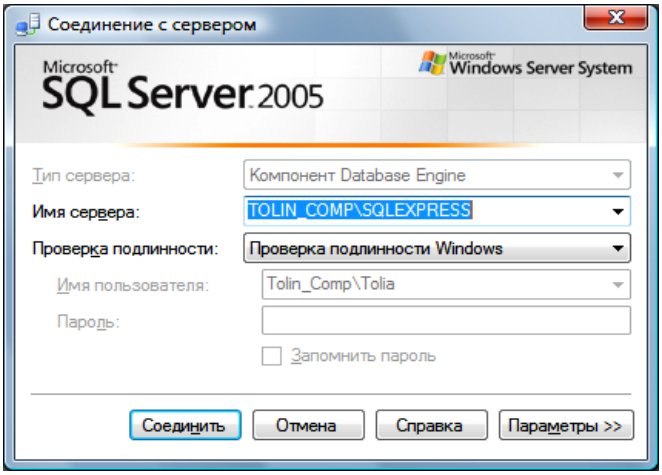


Рис. 25. Задание параметров подключения

Задействуем подключение к серверу, находящемуся на локальном компьютере. Параметр «Проверка подлинности» задает аутентификацию при подключении к серверу – при выбранном значении «Проверка подлинности Windows» (Windows authentification) в качестве имени пользователя и пароля будут использованы системные параметры.

Если все сделано правильно, то появляется главное окно программы. Для перехода в режим запросов необходимо нажать кнопку «Создать запрос» (рис. 26).

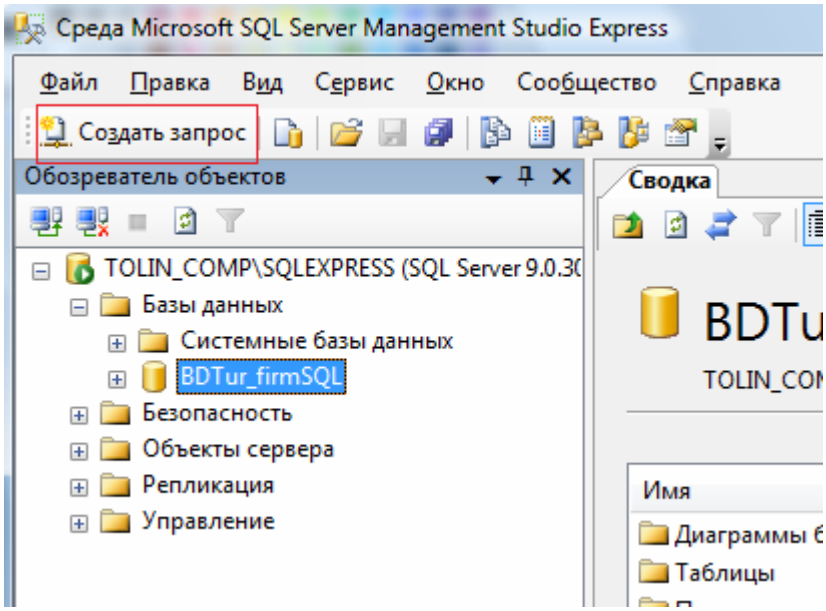


Рис. 26. Переход в режим создания запросов

Отметим, что будут создаваться запросы, работающие с выделенной базой данных. После нажатия кнопки «Создать запрос» среда SQL Management Studio принимает вид, как показано на рисунке (рис. 27). Обратите внимание на кнопку «Выполнить», которая выполняет запросы, введенные в правом текстовом поле, и выводит результат их выполнения.

2.1. Базовая конструкция SQL-запроса

Основной операцией для описания запроса к БД в языке SQL является конструкция вида:

```
Select   <список атрибутов>
From   <список отношений>
Where  <условие>
```

Эта операция представляет собой композицию реляционных операторов проекции, соединения и выбора. Проекция берется для указанного списка атрибутов, соединение выполняется для указанного списка отношений, выбор определяется условием отбора записей where.

В результате выполнения операции соединения данные из указанных в списке отношений представляются одной таблицей. В этой таблице из всех имеющихся столбцов исходных отношений списка отношений остаются только те столбцы, которые указаны в списке атрибутов, и только те строки, которые удовлетворяют условию where.

Итак, напишем первый запрос и нажмем клавишу F5 (пункт меню Запрос – Выполнить):

select * from Туристы;

В результате возвращаются все записи из таблицы «Туристы» базы данных BDTur_firmSQL.

Главное окно программы принимает вид (рис. 27).

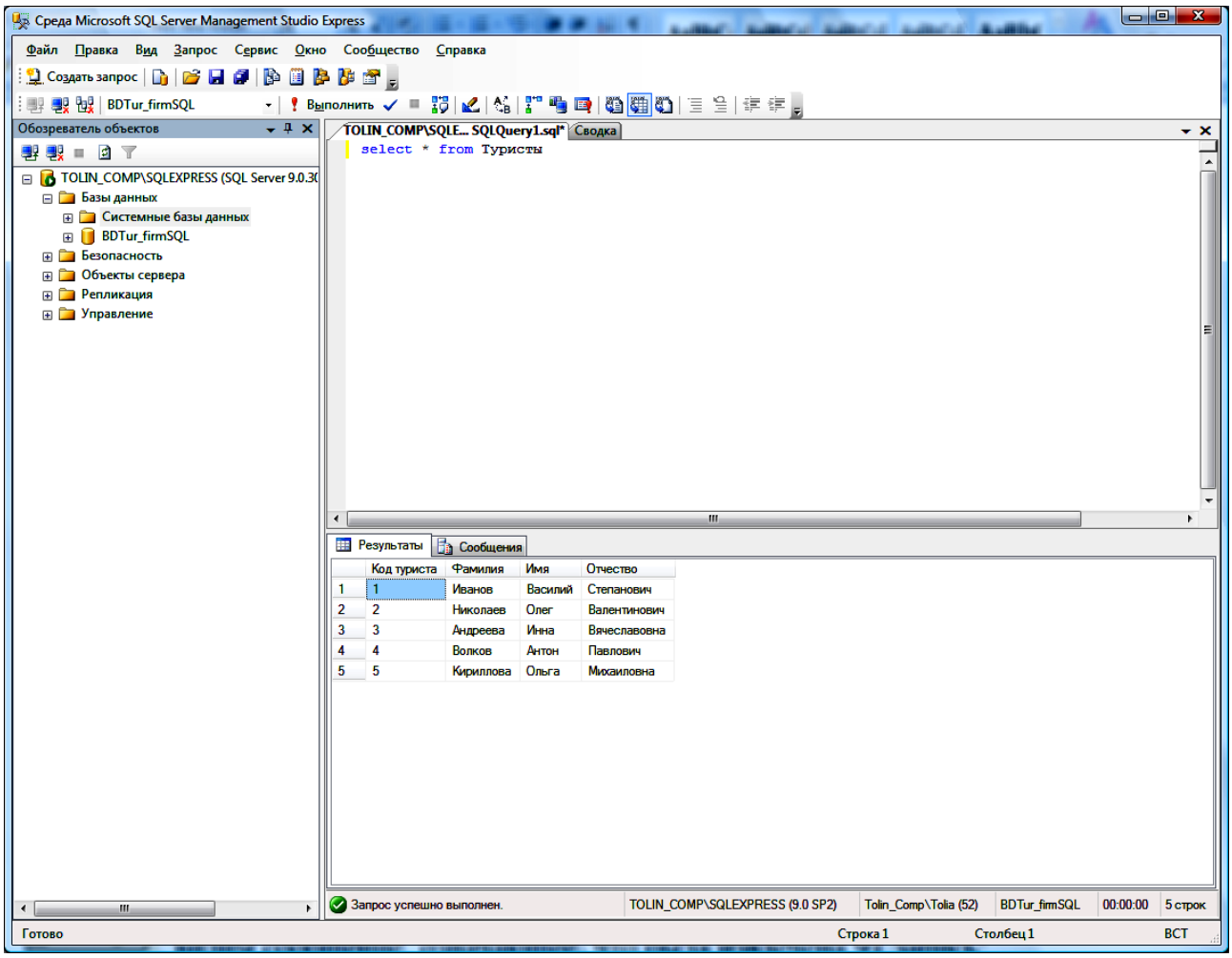


Рис. 27. Извлечение строк из таблицы «Туристы»

Данный запрос извлекал все столбцы таблицы. Если необходимо выбрать только столбец «Фамилия», запрос необходимо модифицировать следующим образом:

select Фамилия from Туристы;

Результат выполнения данного запроса представлен на рисунке 28.

Для вывода определенного количества записей используем следующий запрос (рис. 29):

```
select top 3 Фамилия from Туристы;
```

Результаты | Сообщения

	Фамилия
1	Иванов
2	Николаев
3	Андреева

Рис. 29. Извлечение заданного количества записей

Результаты | Сообщения

	Фамилия
1	Иванов
2	Николаев
3	Андреева
4	Волков
5	Кириллова

Рис. 28. Извлечение столбца «Фамилия»

Извлекаются первые три записи поля «Фамилия», расположенные в самой таблице «Туристы». Обратим внимание на то, что фамилии расположены не в алфавитном порядке, а в порядке, в котором они были сохранены в базе данных.

Добиться алфавитного порядка можно с помощью предложения order by, содержащего список атрибутов, после каждого из которых стоит либо ключевое слово asc (сортировка по возрастанию), либо ключевое слово dec (сортировка по убыванию). Теперь предыдущий запрос может выглядеть так:

```
select top 3 Фамилия from Туристы order by Фамилия asc;
```

Вводя оператор percent, можем получить указанный процент записей от общего числа:

```
select top 25 percent Фамилия from Туристы;
```

Результат выполнения запроса представлен на рисунке 30.

Результаты | Сообщения

	Фамилия
1	Иванов
2	Николаев

Рис. 30. Извлечение нескольких записей

Для отбора записей, отвечающих заданному условию, используем оператор where:

```
select * from Туры where Цена > 27000;
```

Этот запрос возвращает все записи из таблицы «Туры», в которых поле «Цена» имеет значение, большее 27000 (рис. 31).

Результаты | Сообщения

	Код тура	Назван...	Цена	Информация
1	2	Греция	32000,00	В августе и сентябре действуют специальные скидки
2	3	Таиланд	30000,00	Не включая стоимость авиабилета

Рис. 31. Отбор записей со всеми полями по заданному значению

Оператор where поддерживает работу со знаками сравнения <, >, >=, <=.

Точную выборку только из заданного множества значений осуществляет оператор in, в следующем примере извлекаются лишь те записи, в которых значение поля «Цена» в точности равно либо 10 000, либо 20 000, либо 30 000 (рис. 32):

select * from Туры where Цена in (10000, 20000, 30000);

Результаты | Сообщения

	Код тура	Назван...	Цена	Информация
1	3	Таиланд	30000,00	Не включая стоимость авиабилета

Рис. 32. Отбор записей по точному совпадению значений поля Цена

Выборка значений, лежащих в указанном интервале, осуществляется оператором between <первое_значение> and <второе_значение> (рис. 33):

Select * from Туры where Цена between 10000 and 30000;

Результаты | Сообщения

	Код тура	Назван...	Цена	Информация
1	1	Кипр	25000,00	В стоимость двух взрослых путевок входит цена одн...
2	3	Таиланд	30000,00	Не включая стоимость авиабилета
3	4	Италия	26000,00	Завтрак в отеле включен в стоимость путевки
4	5	Франция	27000,00	Дополнительные экскурсии не входят в стоимость п...

Рис. 33. Отбор записей по значениям в указанном интервале поля Цена

2.2. Агрегирующие функции языка SQL

При статистическом анализе баз данных необходимо получать такую информацию, как общее количество записей, наибольшее и наименьшее значения заданного поля записи, усредненное значение поля и т. д. Данная задача выполняется с помощью запросов, содержащих так называемые агрегирующие функции.

Агрегирующие функции производят вычисление одного «собирающего» значения (суммы, среднего, максимального, минимального значения и т. п.) для заданных групп строк таблицы. Группы строк определяются различными значениями заданного поля (полей) таблицы. Разбиение на группы выполняется с помощью предложения group by.

Рассмотрим перечень агрегирующих функций.

- count определяет количество записей данного поля в группе строк.

- sum вычисляет арифметическую сумму всех выбранных значений данного поля.
- avg рассчитывает арифметическое среднее (усреднение) всех выбранных значений данного поля.
- max находит наибольшее из всех выбранных значений данного поля.
- min находит наименьшее из всех выбранных значений данного поля.

Для определения общего числа записей в таблице Туристы используем запрос
select count (*) from Туристы;

Результат выполнения запроса представлен на рисунке 34.

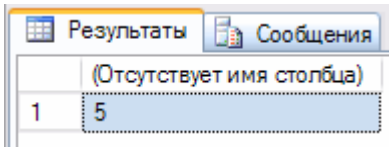


Рис. 34. Результат запроса с функцией count

Отметим, что результатом запроса является одно число, содержащееся в поле с отсутствующим именем.

А если мы захотим посчитать однофамильцев (то есть разбить набор записей-результатов запроса на группы с одинаковыми фамилиями), то запрос будет выглядеть так:
select Фамилия, count (Фамилия) from Туристы group by Фамилия;

Синтаксис использования других операторов одинаков – следующие запросы извлекают сумму, арифметическое среднее, наибольшее и наименьшее значения поля «Цена» таблицы «Туры» (здесь заданной группой записей, как и в первом примере с функцией count, являются все записи таблицы).
select sum(Цена) from Туры
select avg(Цена) from Туры
select max(Цена) from Туры
select min(Цена) from Туры

Если значение поля может быть незаполненным, то для обращения к таким полям необходимо использовать оператор null. Отметим, что величина null не означает, что в поле стоит число 0 (нуль) или пустая текстовая строка. Существует два способа образования таких значений:

1) Microsoft SQL Server автоматически подставляет значение null, если в значение поля не было введено никаких значений и тип данных для этого поля не препятствует присвоению значения null;
2) или если пользователь явным образом вводит значение null.

2.3. Оператор сравнения записей like

Оператор сравнения like нужен для поиска записей по заданному шаблону. Эта задача является одной из наиболее часто встречаемых задач – например, поиск клиента с известной фамилией в базе данных.

Результаты | Сообщения

	Фамил...	Имя	Отчество
1	Иванов	Василий	Степанович

Рис. 35. Запрос с оператором like

Предположим, что в таблице «Туристы», содержащей поля «Фамилия», «Имя» и «Отчество», требуется найти записи клиентов с фамилиями, начинающимися на букву «И».

```
select Фамилия, Имя, Отчество from Туристы
where Фамилия Like 'И%'
```

Результатом этого запроса будет таблица, представленная на рисунке 35.

Оператор like содержит шаблоны, позволяющие получать различные результаты (таблица 7).

Таблица 7

Шаблоны оператора like

Шаблон	**Значение**
like '5[%]'	5%
like '[_]n'	_n
like '[a–cdf]'	a, b, c, d, или f
like '[–acdf]'	–, a, c, d, или f
like '[[ ]'	[
like ']'	]
like 'abc[_]d%'	abc_d и abc_de
like 'abc[def]'	abcd, abce, и abcf

2.4. Команды определения данных языка SQL

Пока мы познакомились только с работой некоторых команд языка SQL по извлечению таблиц и данных из таблиц, предполагая, что сами таблицы были созданы кем-то ранее. Это наиболее реальная ситуация, когда небольшая группа людей (проектировщики баз данных) создает таблицы, которые затем используются другими людьми. Эти команды относятся к области SQL, называемой DML (Data Manipulation Language, или Язык Манипулирования Данными). Тем не менее существует специальная область SQL, называемая DDL (Data Definition Language, или Язык Определения Данных), которая специально работает над созданием объектов данных.

Таблицы создаются командой create table. Эта команда создает пустую таблицу. Команда create table задает имя таблицы, столбцы таблицы в виде описания набора имен столбцов, указанных в определенном порядке, а также она может определять главный и вторичные ключи таблицы. Кроме того, она указывает типы данных и размеры столбцов. Каждая таблица должна содержать, по крайней мере, один столбец.

Пример команды create table:

```
create table ClientInfo (
FirstName varchar(20), LastName varchar(20), Address varchar(20), Phone varchar(15)
);
```

Тип varchar предназначен для хранения символов не в кодировке Unicode. Число, указываемое в скобках, определяет максимальный размер поля и может принимать значение от 1 до 8000. Если введенное значение поля меньше зарезервированного, при сохранении будет выделяться количество памяти, равное длине значения. После выполнения этого запроса в окне «Сообщения» появляется сообщение:

Команды выполнены успешно.

После перезапуска Management Studio в списке таблиц появилась новая таблица (рис. 36).

Итак, была создана таблица, состоящая из четырех полей типа varchar, причем для трех полей была определена максимальная длина 20 байт, а для одного – 15. Значения полей не заполнены – на это указывает величина Null.

Можно удалить созданную таблицу непосредственно в интерфейсе Management Studio, щелкнув правой кнопкой и выбрав «Удалить».

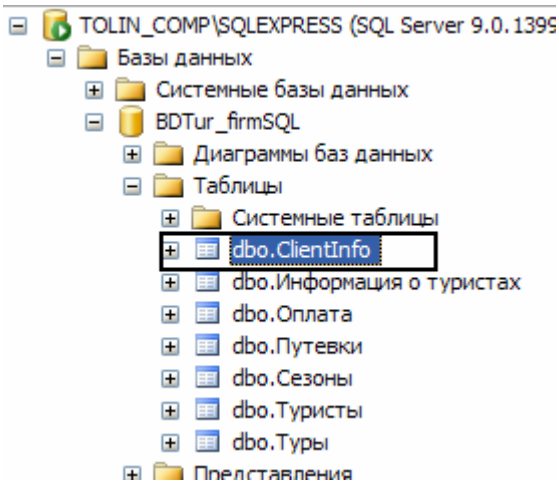


Рис. 36. Созданная таблица в базе данных

2.5. Команды изменения данных языка DML

Значения могут быть помещены и удалены из полей тремя командами языка DML (Язык Манипулирования Данными):

- insert (вставить),
- update (изменить),
- delete (удалить).

Команда insert имеет свои особенности.

1) При указании значений конкретных полей вместо использования каких-либо значений можно применить ключевое слово DEFAULT.
2) Вставка пустой строки приводит к добавлению пробела, а не значения NULL.
3) Строки и даты задаются в апострофах.
4) Можно задавать NULL явно, а можно задавать DEFAULT.

Например:

```
insert into ClientInfo (FirstName, LastName, Address, Phone)
values('Petr','Petrov','Chehova 13','1234567');
```

Однократное выполнение этого запроса (нажатие клавиши F5 один раз) приводит к добавлению одной записи. Добавим еще несколько записей, изменяя значения values:

```
insert into ClientInfo (FirstName, LastName, Address, Phone)
values('Ivan','Ivanov','Naberejnaya 13','1234568');
insert into ClientInfo (FirstName, LastName, Address, Phone)
values(null,'Sidorov','Naberejnaya 25','1234569');
```

Извлечем все записи созданной таблицы (рис. 37):

```
select * from ClientInfo;
```

Результаты | Сообщения

	FirstName	LastName	Address	Phone
1	Petr	Petrov	Chehova 13	1234567
2	Ivan	Ivanov	Naberejnaya 13	1234568
3	NULL	Sidorov	Naberejnaya 25	1234569

Рис. 37. Все записи таблицы ClientInfo

Отметим, что третья строка содержит значение null, а не текстовую строку «null».

Команда update позволяет изменять заданные значения записей:

```
update ClientInfo set FirstName = 'Andrey' where FirstName = 'Petr';
```

В этом случае в первой записи поля FirstName значение Petr изменится на Andrey (рис. 38).

Результаты | Сообщения

	FirstName	LastName	Address	Phone
1	Andrey	Petrov	Chehova 13	1234567
2	Ivan	Ivanov	Naberejnaya 13	1234568
3	NULL	Sidorov	Naberejnaya 25	1234569

Рис. 38. Изменение одной записи

Отметим, что если не указывать условие, определяющее значение, которое необходимо изменить, команда update затронет все записи.

Команда delete удаляет записи из таблицы.

```
delete from ClientInfo where LastName like 'Petrov';
```

Результатом этого запроса будет удаление первой записи из таблицы ClientInfo.

Если не задавать условие, определяющее данные, которые необходимо удалить, то будут удалены все данные таблицы.

Запросы с командами insert, update и delete могут содержать в себе все прочие конструкции языка SQL.

2.6. Контрольные вопросы и задания к разделу 2

1. Напишите SQL-запросы для вывода на экран содержания всех таблиц БД (для каждой таблицы свой запрос, см. пример из п. 2.1. select * from Туристы;).
2. Добавьте к соответствующим запросам задания п. 1 сортировку по фамилиям и именам.
3. Что будет выведено на экран в результате выполнения следующего запроса: select Фамилия, Имя, Отчество from Туристы order by Имя dec, Отчество asc; ?
4. Напишите SQL-запрос, который позволит отобрать всех туристов, проживающих в заданном городе. Используйте сортировку при выводе.
5. Посчитайте туристов с одинаковыми именами.
6. А как посчитать туристов с одинаковыми отчествами? Ведь слова «Иванович» и «Ивановна» одинаковые отчества, но не одинаковые строки, которые хранятся в базе данных.
7. Как определить среднюю цену по турам в Париж, (например, цена может меняться в зависимости от сезона)?
8. Как будет выглядеть таблица «Туристы» после выполнения следующей SQL-команды: update Туристы set Имя = 'Владимир'?
9. Что произойдет с таблицей «Туристы» после выполнения SQL-команды: delete from Туристы where Отчество like 'Иван'?
10. Что произойдет с таблицей «Туры» после выполнения SQL-команды: delete from Туры?
11. Выясните с помощью SQL-запроса к БД, кто из туристов еще не оплатил свои путевки? Подсчитайте их количество и общую сумму недоплаты.
12. Распечатайте все предлагаемые турфирмой туры в алфавитном порядке.
13. Составьте с помощью оператора update SQL-команду для переименования города Ульяновска в Симбирск в информации о туристах.
14. Распечатайте все предлагаемые турфирмой туры с сезонами.
15. Выведите полную информацию о туристах, выкупивших путевки на какой-нибудь определенный тур и сезон.
16. С учетом внесенных изменений в структуру БД по заданию п.12 раздела 1.5. постройте SQL-запрос, выводящий полную контактную информацию о туристах, имеющих долги по оплате своих путевок.
17. С помощью операторов добавления в БД перенесите соответствующую информацию из отношений «Туристы» и «Информация о туристах» в отношение ClientInfo.
18. Сформулируйте на естественном языке содержание следующих SQL-запросов к БД:

select * **from** Туристы **where** Имя **in** ('Владимир', 'Иван');
select * **from** Сезоны **order by** [Количество мест] **dec**;
update ClientInfo **set** FirstName = 'Andrey' **where** FirstName = NULL;

3. СОЗДАНИЕ ПРИЛОЖЕНИЙ БАЗ ДАННЫХ

3.1. Пример простейшего приложения баз данных

Создадим простое приложение баз данных, которое выводит на экранную форму информацию из таблицы «Туристы» и связанную с текущей записью таблицы «Туристы» запись таблицы «Информация о туристах» из базы данных Microsoft Access.

Для этого создадим пустое Windows-приложение[1]. Внешний вид среды разработки приведен на рисунке 39[2].

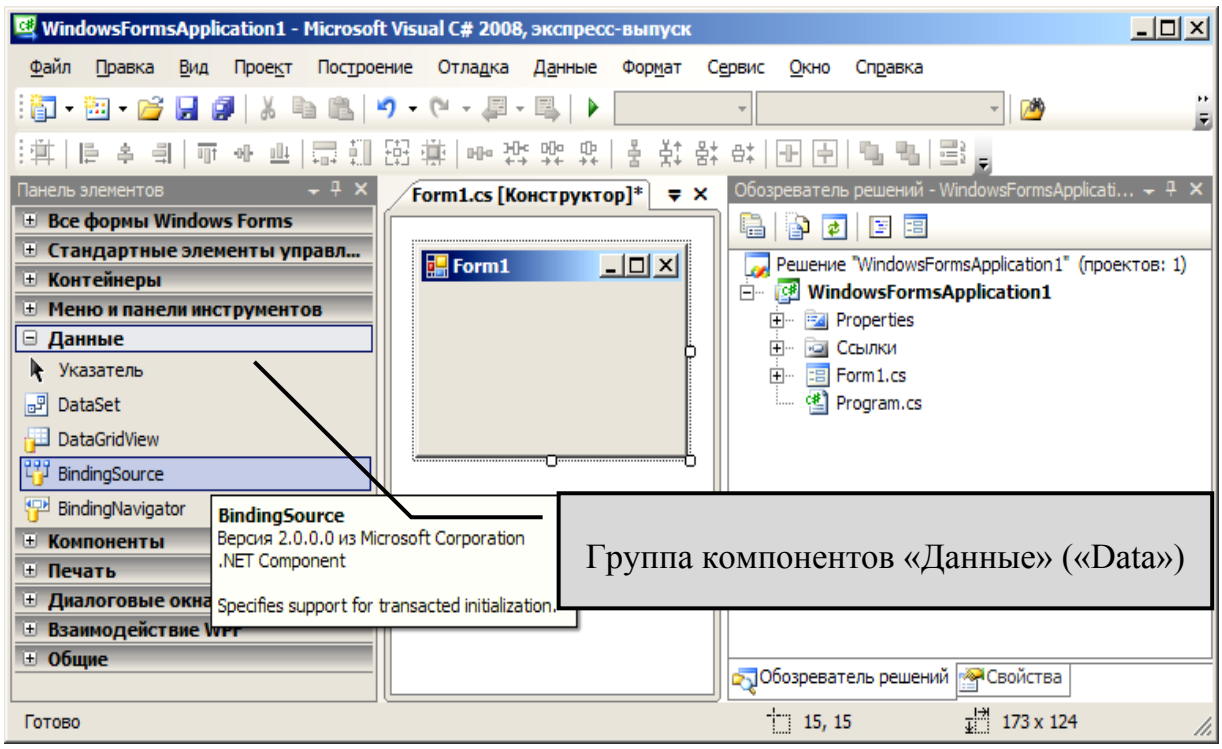


Рис. 39. Пустое приложение

На рисунке 39 выделена группа компонентов «Данные» («Data»), которая содержит компоненты для доступа к данным и манипулирования ими.

Привязку данных БД к форме осуществляет компонент «Binding Source». Перенесем его на форму. После размещения его на форме среда разработки принимает следующий вид (рис. 40).

---

[1] Меню «Файл» - команда «Создать» - «Проект», или сочетание клавиш Ctrl+Shift+N, или первая кнопка на панели инструментов «Создать проект».

[2] Здесь и далее в примерах используется русскоязычная версия среды MS Visual C# 2008, правда, часть окон приводится либо только англоязычной версии, либо обеих версий совместно. Надеемся, что такой подход не внесет путаницы, а позволит читателям ориентироваться в среде разработки независимо от ее языка.

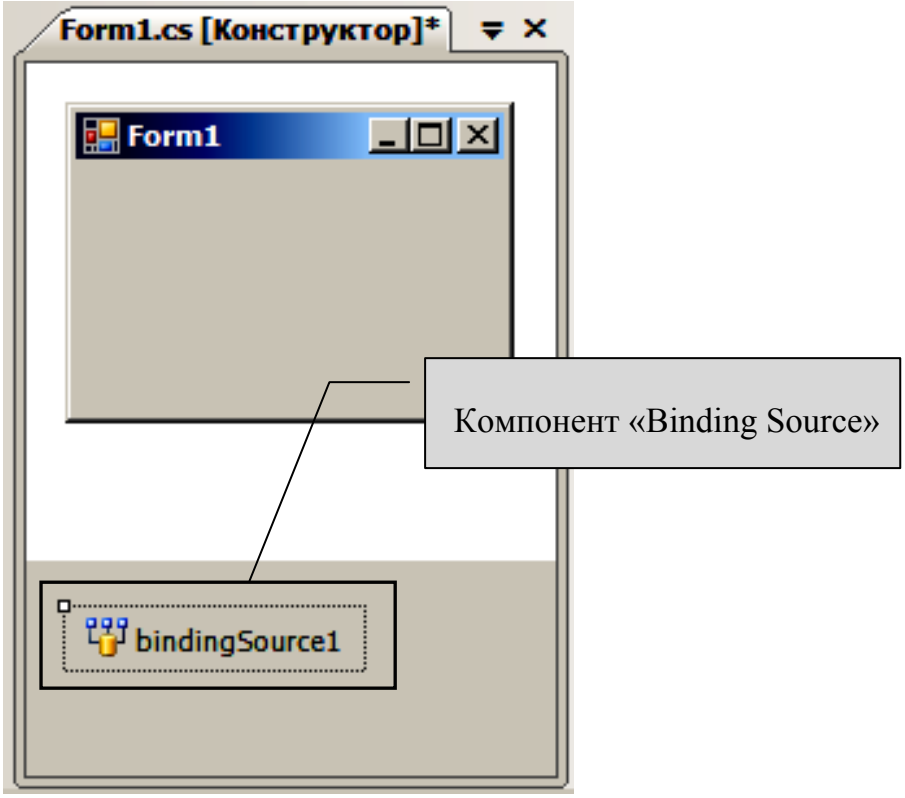


Рис. 40. Компонент Binding Source на форме

Компонент является не визуальным, поэтому он отображается на дополнительной панели. Основным свойством компонента является свойство DataSource, указывающее на источник данных. По умолчанию свойство является пустым, поэтому необходимо сформировать его значение. При выборе данного свойства в окне свойств появляется следующее окно (рис. 41).

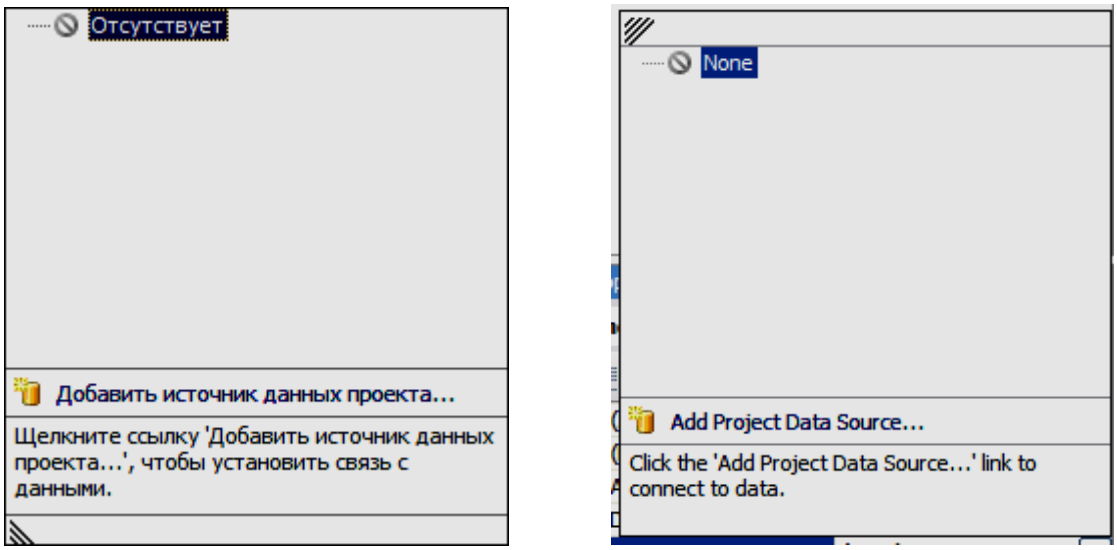


Рис. 41. Список источников данных

В настоящий момент список пуст, поэтому необходимо создать новый источник данных, выбрав команду «Add Project Data Source» для создания нового источника данных и соединения с ним. Появляется следующее окно диалога (рис. 42).

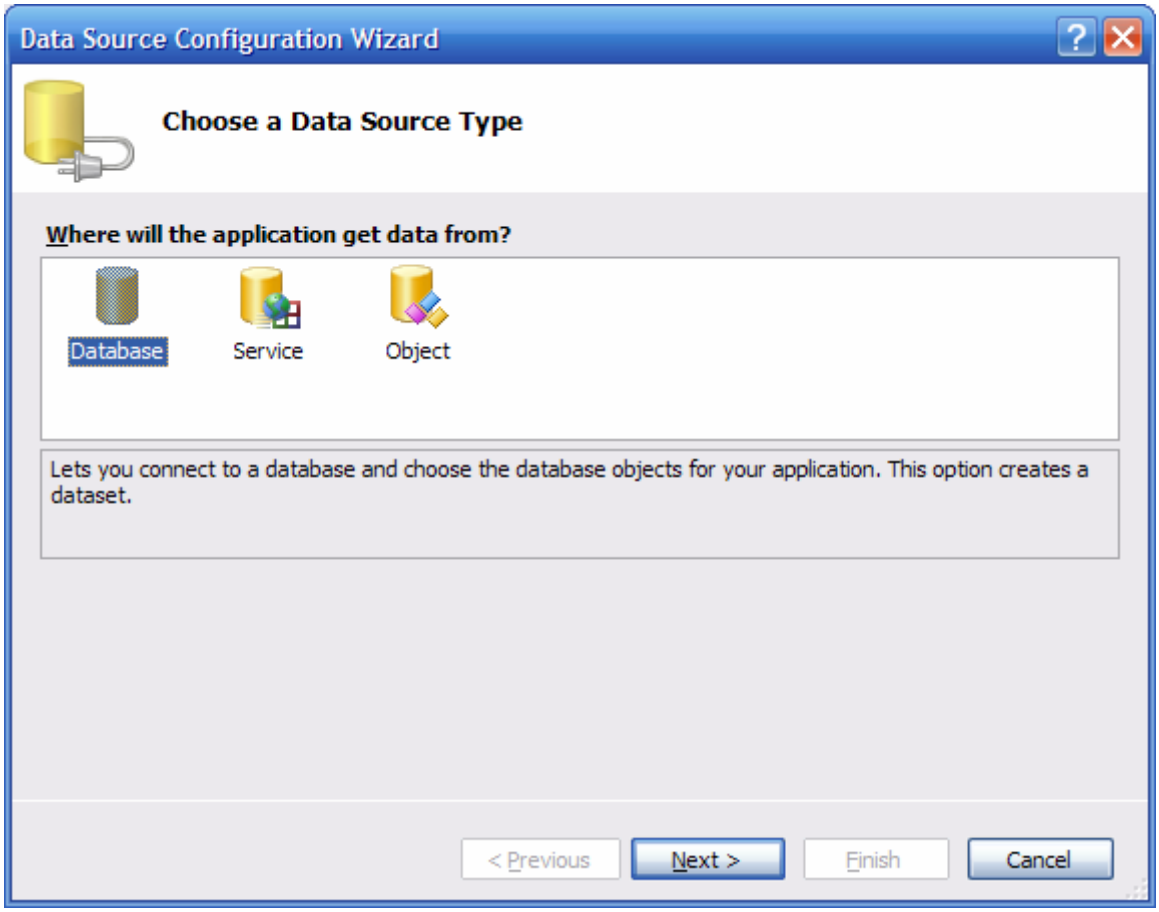


Рис. 42. Список источников данных

Данный диалог предоставляет следующий выбор источников данных:

- Database – База данных;
- Service – Служба, это некоторый сервис, предоставляющий данные. Чаще всего это Web-сервис;
- Object – Объект для выбора объекта, который будет генерировать данные и объекты для работы с ними.

В нашем случае необходимо выбрать пункт «База данных» («Database»). Появляется окно выбора соединения с данными (рис. 43).

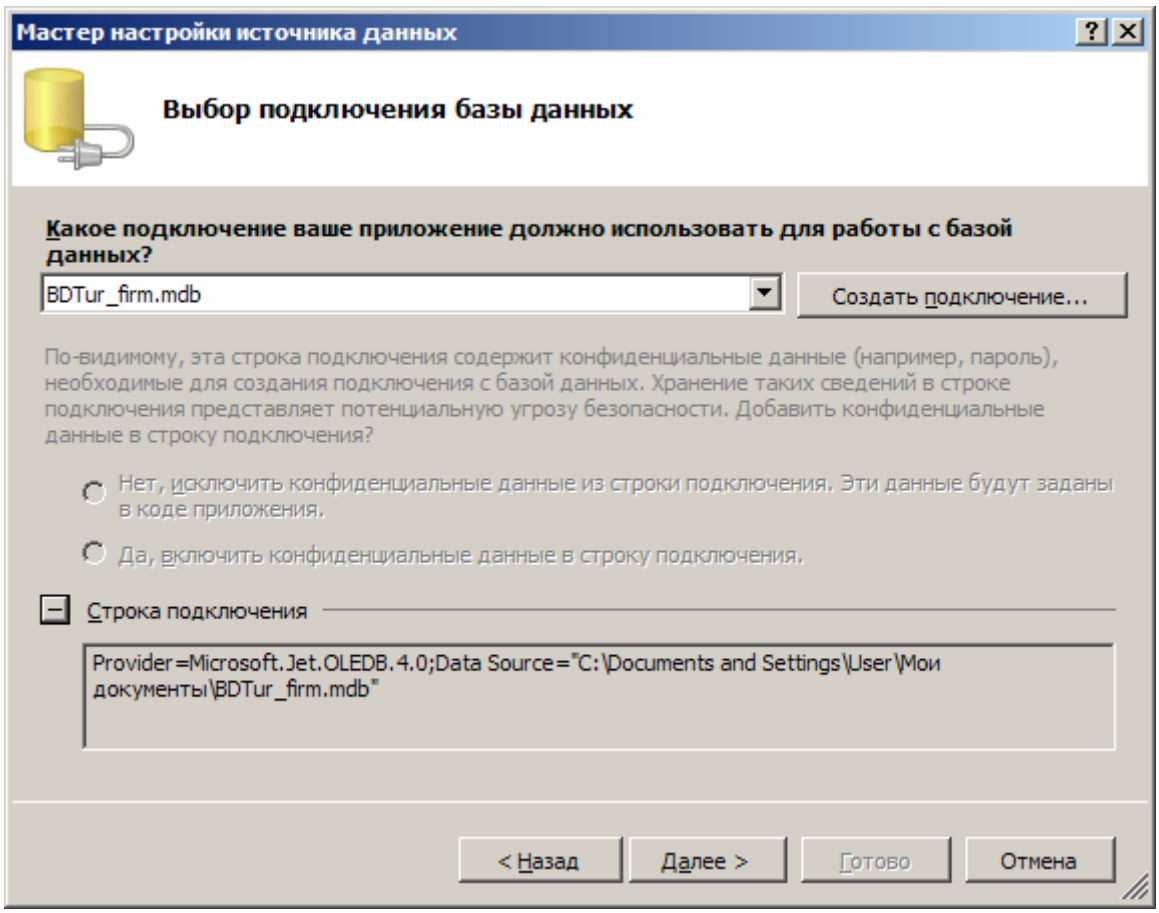


Рис. 43. Выбор соединения с данными

Целью данного диалога является создание строки соединения, в которой будут описаны параметры соединения для механизма ADO, такие как тип базы данных, ее местонахождение, имена пользователей, средства безопасности и пр.

В выпадающем списке диалога находятся все создаваемые ранее соединения. Если необходимого соединения в списке нет, то следует использовать кнопку «Создать подключение» («New connection»). Нажатие кнопки приводит к появлению следующего диалога (рис. 44).

В данном диалоге выбирается тип источника данных (в данном случае Microsoft Access), имя базы данных (в данном случае имя и местоположение файла базы данных), имя пользователя и пароль, используемые для подключения к базе данных. Кнопка «Дополнительно» («Advanced») позволяет задать большое количество параметров, относящихся к различным деталям механизма ADO. Использование кнопки «Проверить подключение» («Test Connection») позволит убедиться в правильности введенных параметров и работоспособности соединения.

Следующий шаг диалога предлагает сохранить полученную строку соединения в файле настроек приложения. Рекомендуется принять данный выбор для упрощения последующего размещения и поддержки программного продукта.

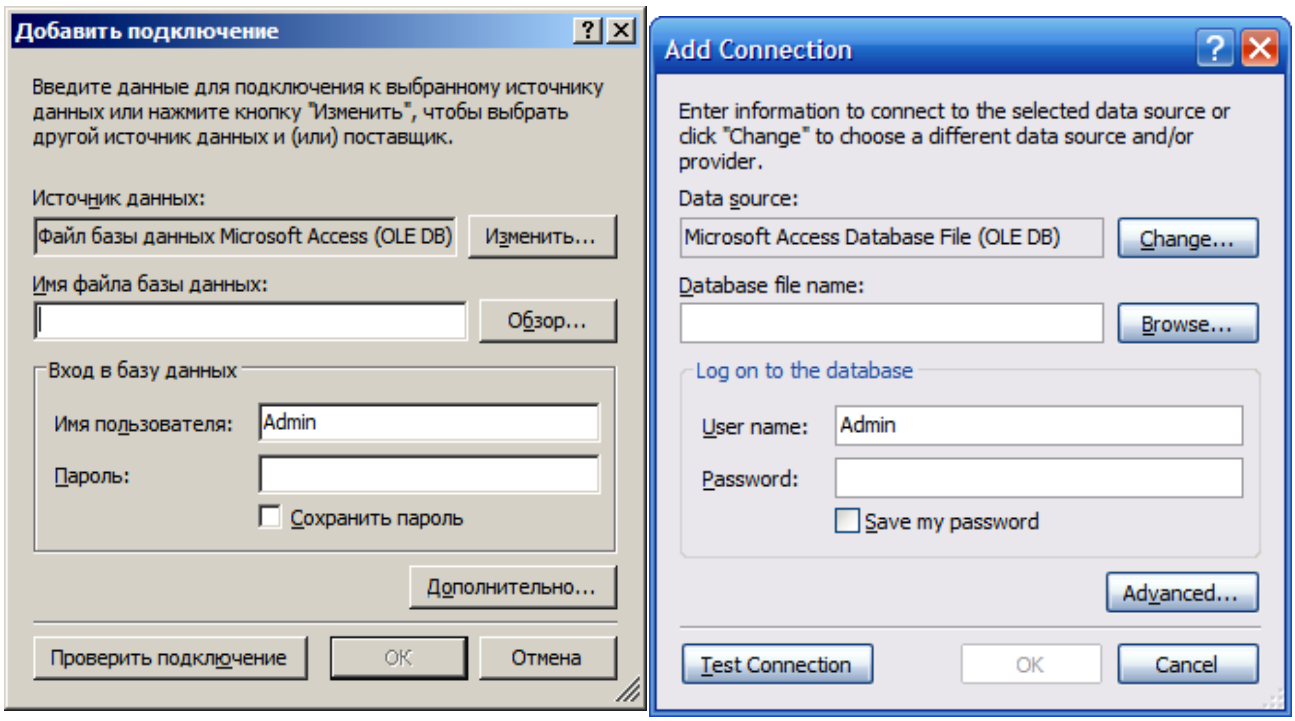


Рис. 44. Создание нового соединения

Последний шаг диалога – выбор тех таблиц или иных объектов базы данных, которые необходимы в данном источнике данных. Окно выбора представлено на рисунке 45.

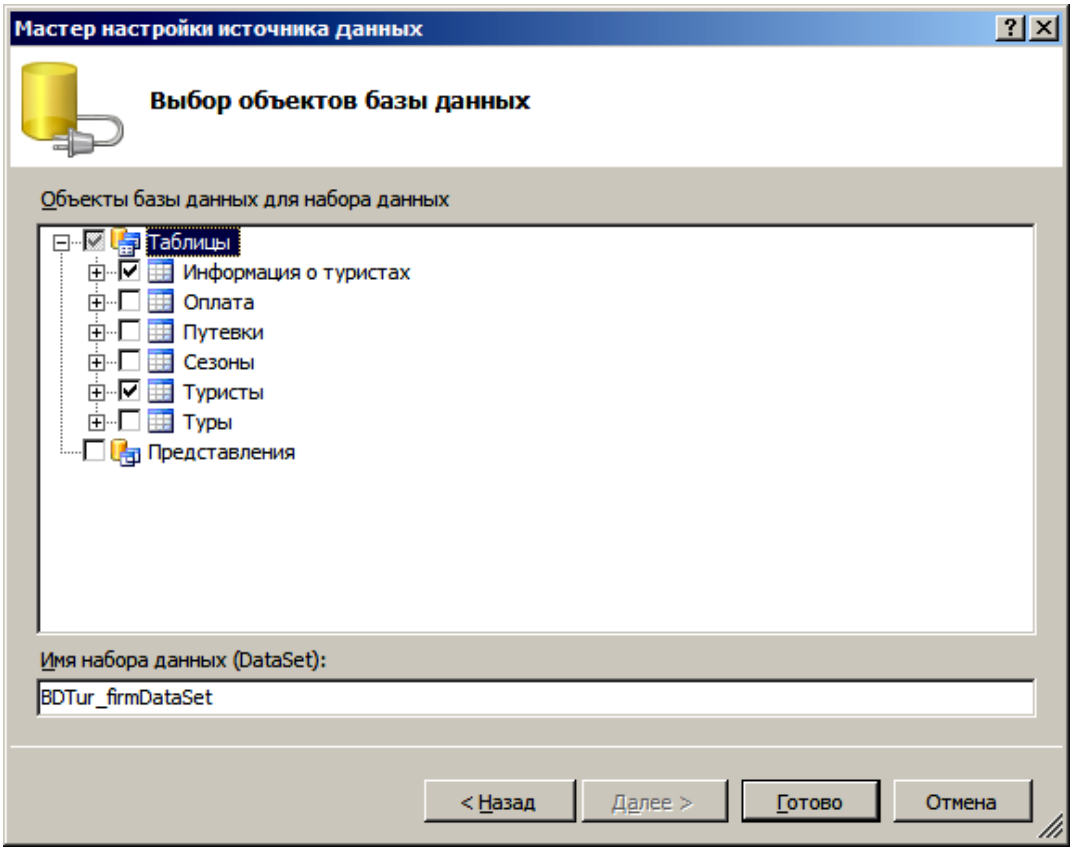


Рис. 45. Выбор необходимых таблиц

В данном окне выбраны таблицы «Туристы» и «Информация о туристах». Поскольку иных объектов, кроме таблиц, в базе данных не было создано, на рисунке 45 отображаются только таблицы. На этом создание источника данных завершено. После нажатия кнопки «Готово» («Finish») рядом с компонентом BindingSource на форме появляется компонент DataSet.

Теперь данные, подключенные выше, необходимо отобразить на форме. Простейшим способом отображения данных является использование компонента DataGridView из группы компонентов Data. Компонент является визуальным и на форме выглядит следующим образом (рис. 46).

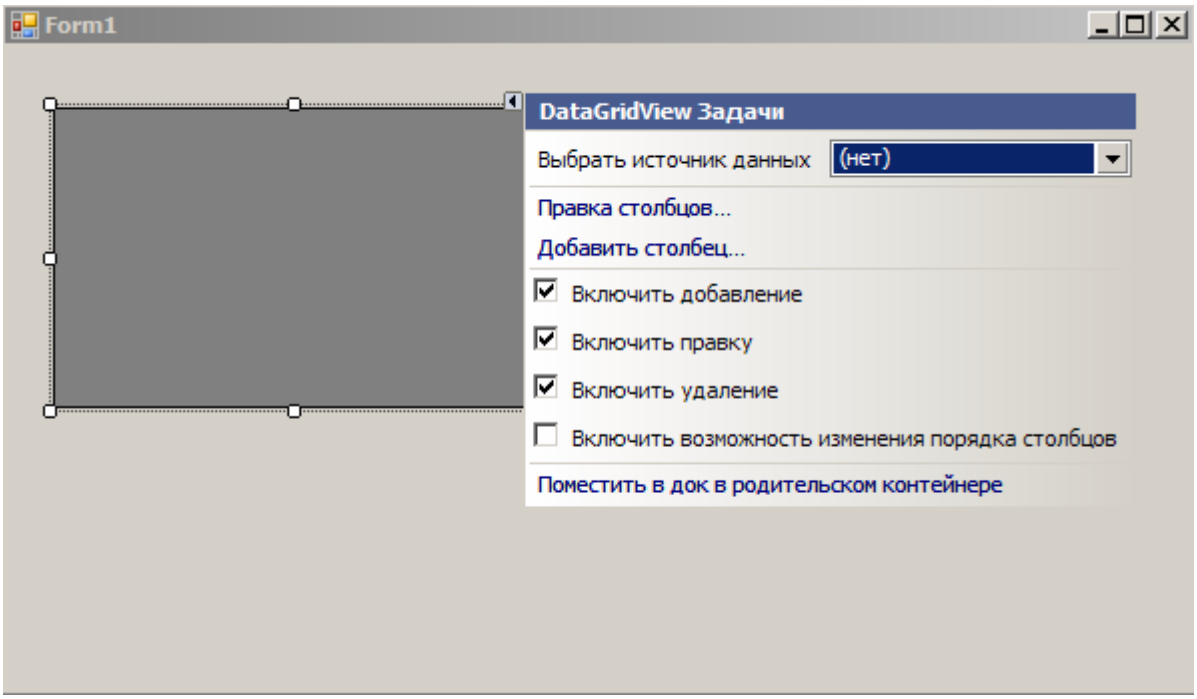


Рис. 46. Компонент DataGridView

Сразу же возникает окно настройки компонента, которое определяет его возможности по редактированию данных: «Включить редактирование» («Enable Adding»), «Включить правку» («Enable Editing»), «Включить удаление» («Enable Deleting»); возможность изменения последовательности столбцов: «Включить возможность изменения порядка столбцов» («Enable Column Reordering»); а также возможность закрепления в контейнере-родителе.

Для того чтобы компонент мог отображать данные, необходимо выбрать источник данных в выпадающем списке. Выбор выпадающего списка приводит к появлению следующего диалога (рис. 47).

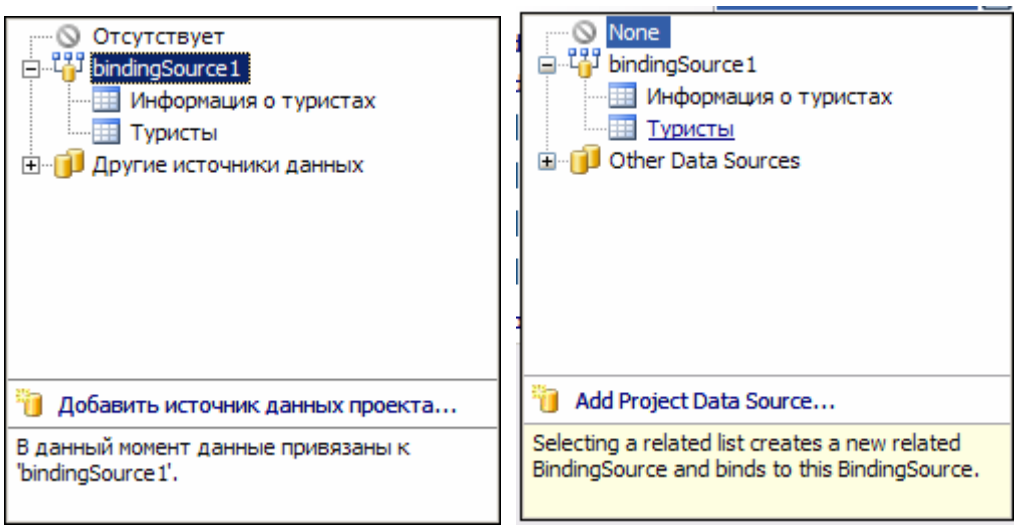


Рис. 47. Выбор источника данных для DataGridView

В данном случае мы выбрали в качестве источника данных таблицу «Туристы». Данный выбор изменяет экранную форму следующим образом (рис. 48).

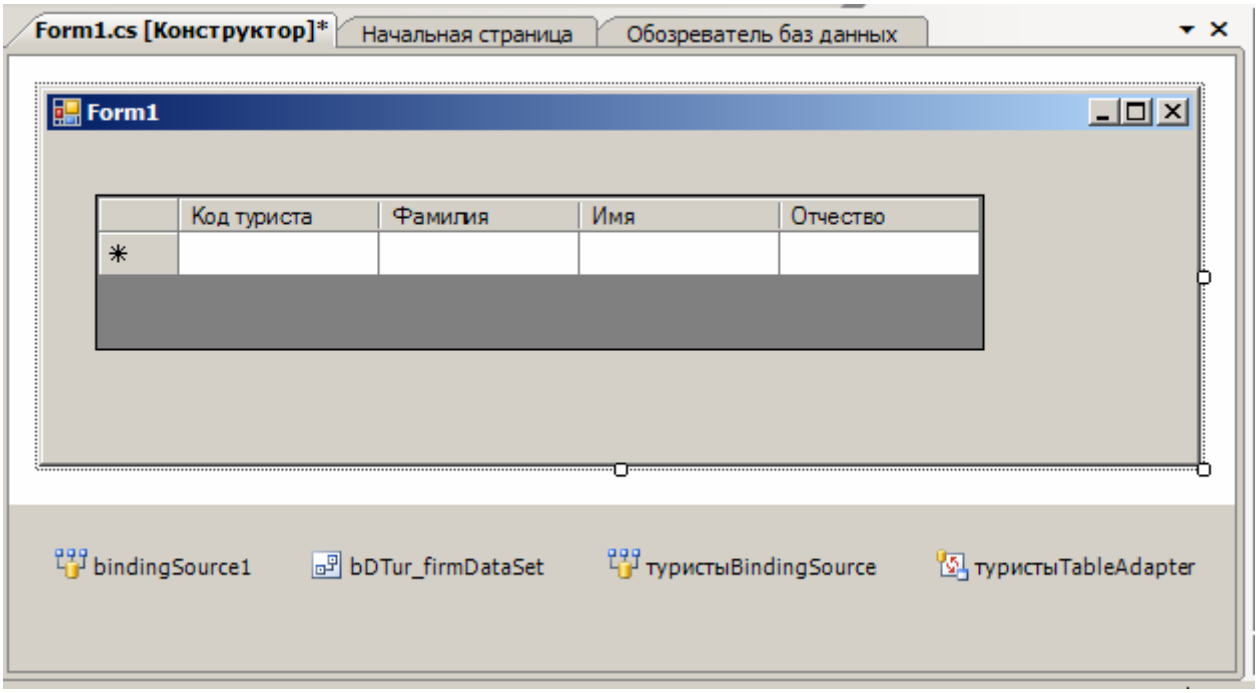


Рис. 48. Компонент DataGridView отображает структуру таблицы

На рисунке видно, что появился еще один компонент BindingSource и компонент TableAdapter, работающий с таблицей «Туристы». Обратите внимание, что в design-time или в процессе разработки данные из таблицы не отображаются.

Теперь необходимо отобразить данные из связанной таблицы «Информация о туристах». Для этого разместим на форме еще один компонент DataGridView и в качестве источника данных выберем следующее (рис. 49).

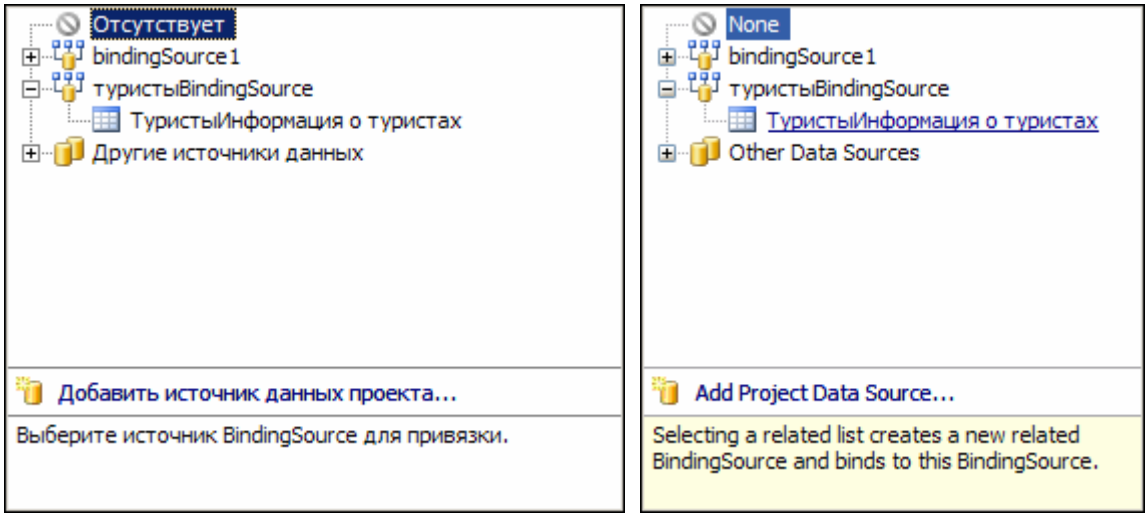


Рис. 49. Выбор источника данных для второго DataGridView

Здесь в качестве источника данных выступает не сама таблица «Информация о туристах», а связь (Binding Source) между таблицами «Туристы» и «Информация о туристах». Такой выбор гарантирует выбор из таблицы «Информация о туристах» только тех строк, которые связаны с текущей строкой в таблице «Туристы». Также такой выбор гарантирует правильность обновления и удаления связанных данных. Работа полученного приложения показана на рисунке 50.

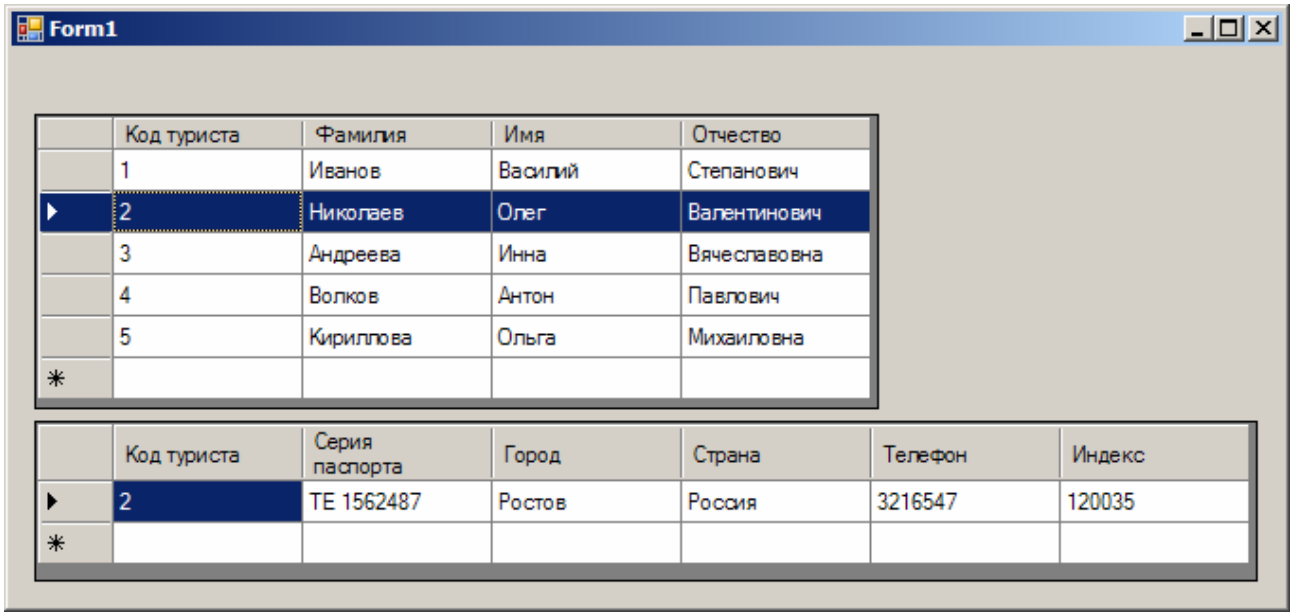


Рис. 50. Приложение базы данных в работе

Перемещение по данным при помощи стрелочных клавиш является неудобным. Для упрощения навигации по данным существует компонент BindingNavigator. Поместим его на форме (рис. 51).

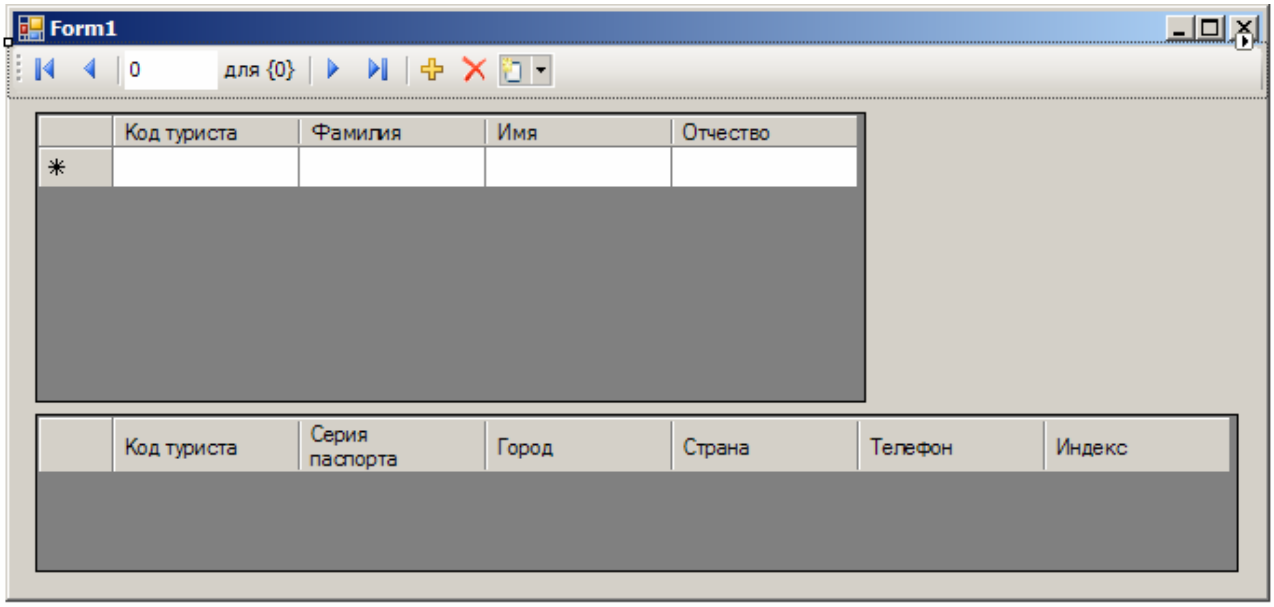


Рис. 51. Компонент BindingNavigator на форме

Данный компонент позволяет осуществлять навигацию между записями таблицы, добавлять и удалять строки таблицы. Возможности и внешний вид компонента можно настраивать, так как он представляет собой полосу меню ToolStripContainer.

Свойством, определяющим таблицу, по которой производится навигация, является свойство BindingSource. Установим значение этого свойства равным «туристыBindingSource». В работе компонент выглядит следующим образом (рис. 52).

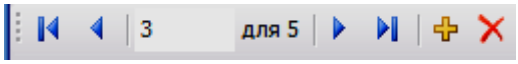


Рис. 52. Компонент BindingNavigator в работе

Редактирование данных в ячейках компонента DataGridView при соответствующих настройках возможно, но неудобно и не рационально. В частности, трудно проверять введенные значения на ошибки. Поэтому для таблицы «Туристы» сделаем экранную форму, позволяющую отображать данные в компонентах TextBox и редактировать их. Для этого разместим на форме контейнер типа Panel, а на нем три компонента TextBox следующим образом (рис. 53).

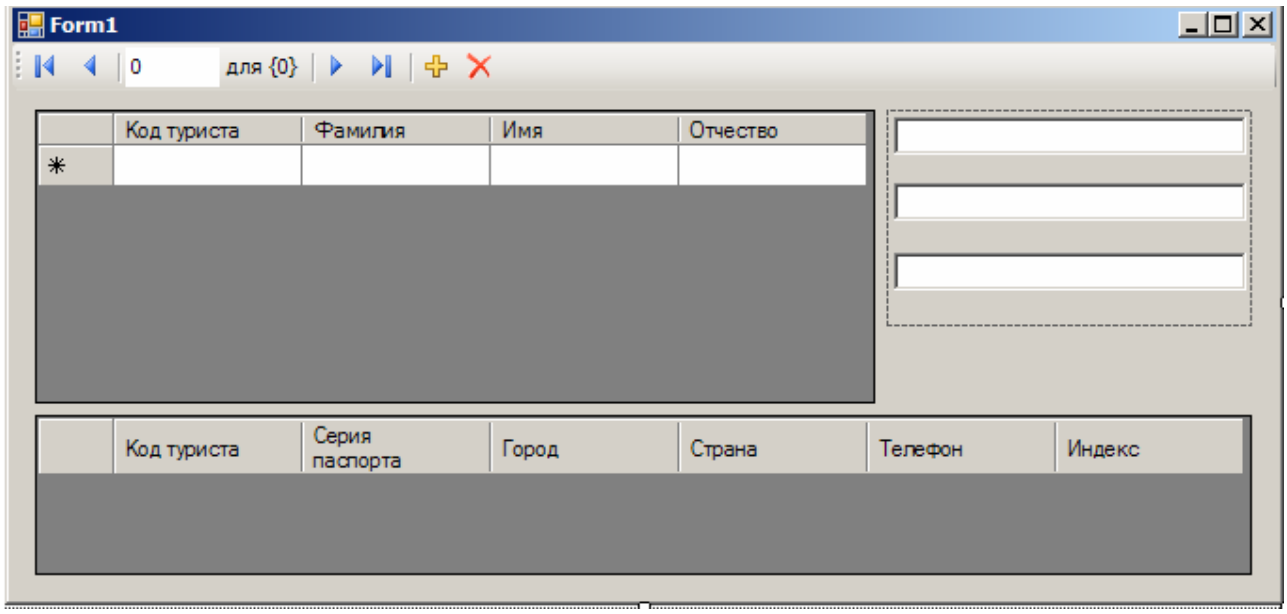


Рис. 53. Экранная панель для редактирования записей таблицы «Туристы»

Теперь необходимо осуществить привязку компонентов TextBox к соответствующим полям таблицы «Туристы». Для этого используем свойство из группы DataBindings – Advanced, показанное на рисунке 54.

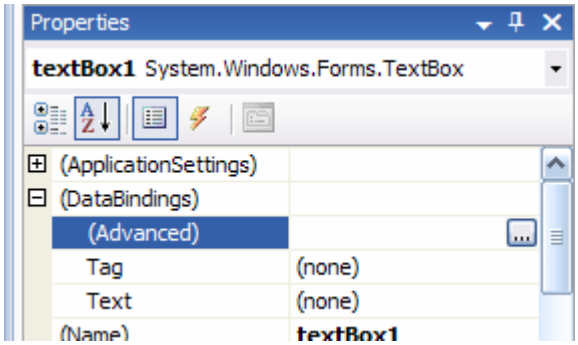


Рис. 54. Свойство «DataBindings – Advanced»

Выбор данного свойства приводит к появлению диалога, показанного на рисунке 55. Данный диалог позволяет осуществить не только привязку данных, но также задать событие, в рамках которого будет проводиться обновление данных, а также форматирование данных при их выводе.

Для верхнего компонента TextBox в выпадающем списке Binding выберем источником данных «туристыBindingSource» и поле источника – «Фамилия». Для среднего и нижнего компонентов TextBox выберем тот же источник данных и поля «Имя» и «Отчество» соответственно.

Разработанное приложение в работе выглядит следующим образом (рис. 56).

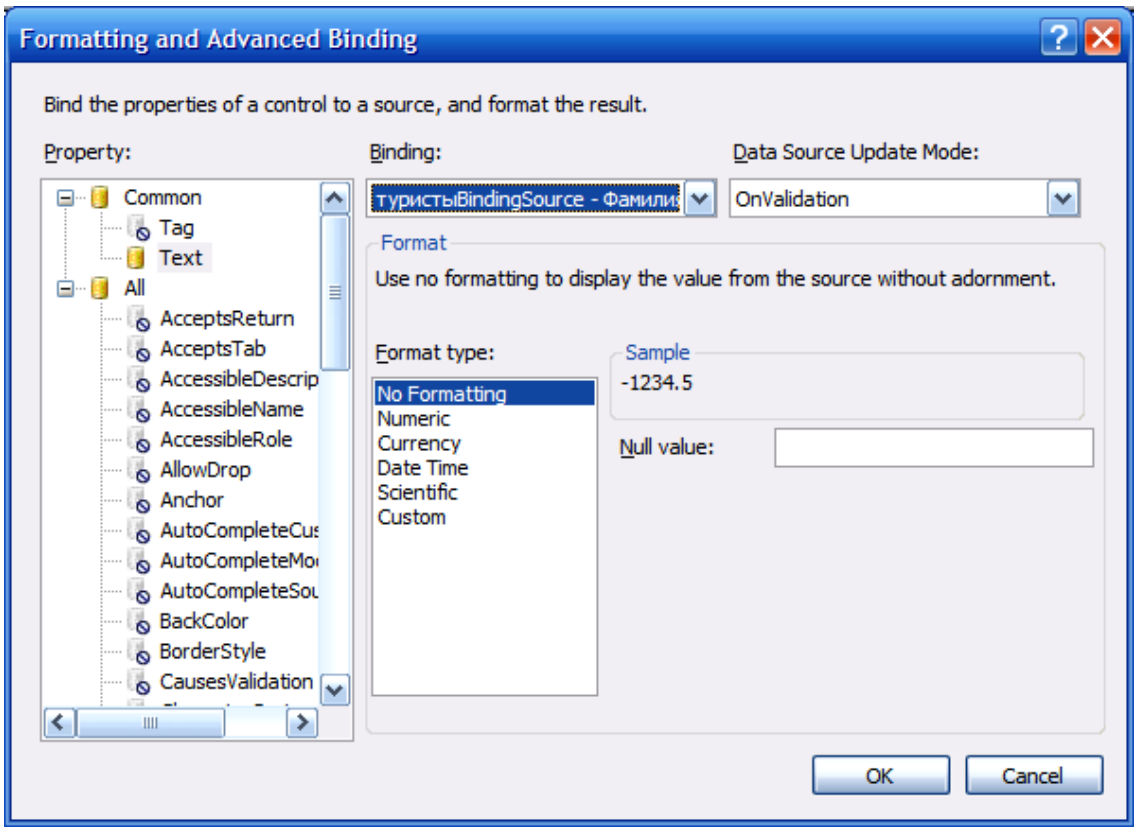


Рис. 55. Окно диалога для свойства «DataBindings – Advanced»

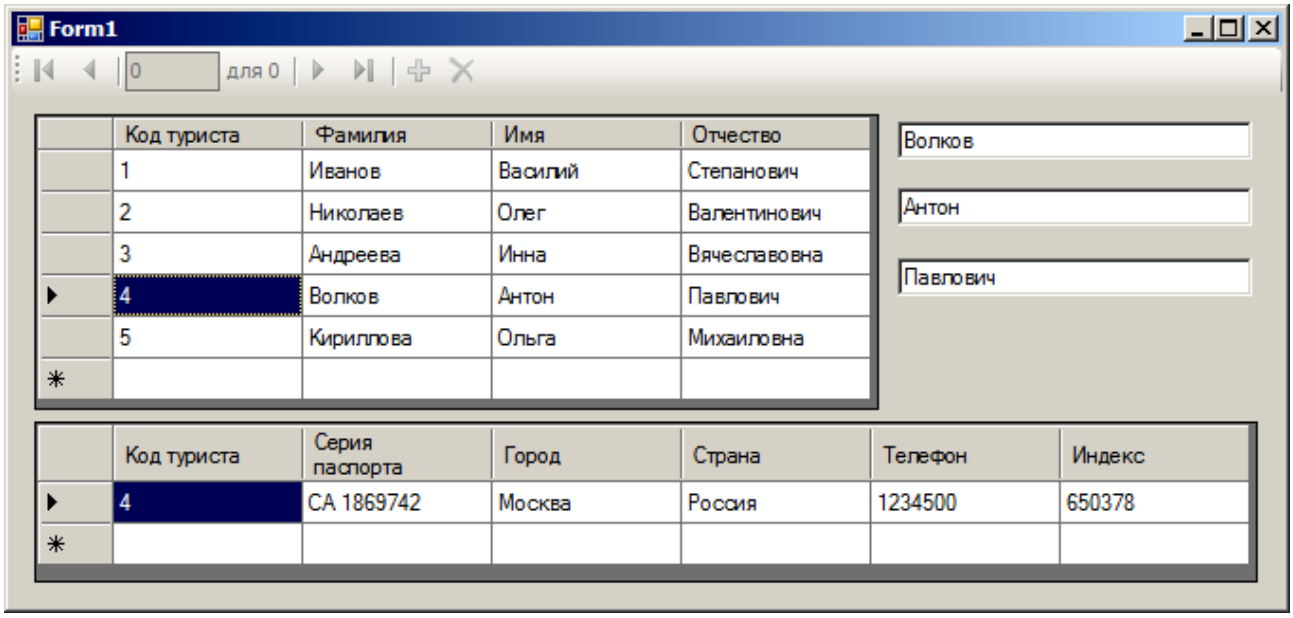


Рис. 56. Привязка данных к визуальным компонентам

Однако при внесении изменений все новые данные остаются только на форме. В базе данных они не сохраняются, и при повторном вызове приложения, конечно же, будут отсутствовать. Это происходит потому, что данные были загружены в объект DataSet, который представляет собой копию таблицы в памяти. Все действия выполняются с этой копией. Для того чтобы изменения

отобразились в базе данных, необходимо выполнить метод Update класса TableAdapter. Таким образом, в разрабатываемом приложении необходимо разместить кнопку «Обновить» и записать в обработчик события Click следующий программный код:

```
туристыTableAdapter.Update(bDTur_firmDataSet);
информация_о_туристахTableAdapter.Update(bDTur_firmDataSet);
```

Данный код обновляет информацию в таблицах «Туристы» и «Информация о туристах», предоставляемых источником данных. Отметим, что данный метод является перегруженным, и его варианты позволяют обновлять как отдельную строку таблицы, так и группу строк.

3.2. Обзор объектов ADO .NET

3.2.1. Источник данных DataSet

Основным объектом ADO является *источник данных*, представленный объектом DataSet. DataSet состоит из объектов типа DataTable и объектов DataRelation. В коде к ним можно обращаться как к свойствам объекта DataSet, то есть, используя точечную нотацию. Свойство Tables возвращает объект типа DataTableCollection, который содержит все объекты DataTable используемой базы данных.

3.2.2. Таблицы и поля (объекты DataTable и DataColumn)

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

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

3.2.3. Объекты DataRelation

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

3.2.4. Строки (объект DataRow)

Коллекция Rows объекта DataTable возвращает набор строк (записей) заданной таблицы. Эта коллекция используется для изучения результатов запроса

к базе данных. Мы можем обращаться к записям таблицы как к элементам простого массива.

3.2.5. DataAdapter

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

3.2.6. Объекты DBConnection и DBCommand

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

3.3. Server Explorer

В состав Visual Studio 2008 входит замечательный инструмент управления и обзора подключениями к базам данных – «Обозреватель баз данных» («Server Explorer»).

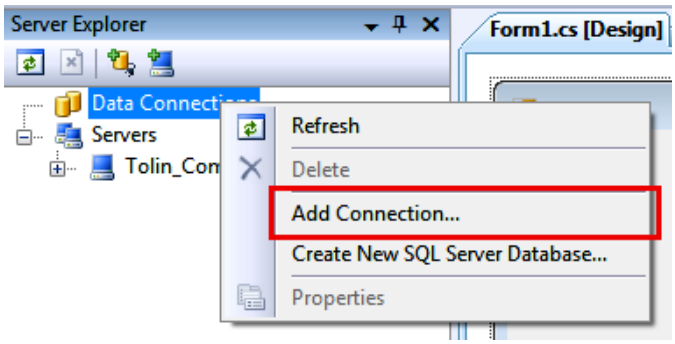


Рис. 57. Добавление соединения

Создадим новый проект. Выберем в меню «Вид» (View) пункт «Обозреватель баз данных» («Server Explorer»). Появится окно «Обозреватель баз данных» («Server Explorer»). Щелкнем на «Подключения данных» («Data Connections») правой кнопкой мыши и выберем пункт «Добавить подключение» («Add Connection») (рис. 57).

Появляется окно мастера создания соединения, который был рассмотрен выше (п. 3.1. рис. 43, 44). Создадим подключение к базе данных туристической фирмы. После создания соединение появляется в списке всех подключений (рис. 58). На рисунке 58 показаны все таблицы, существующие в базе данных.

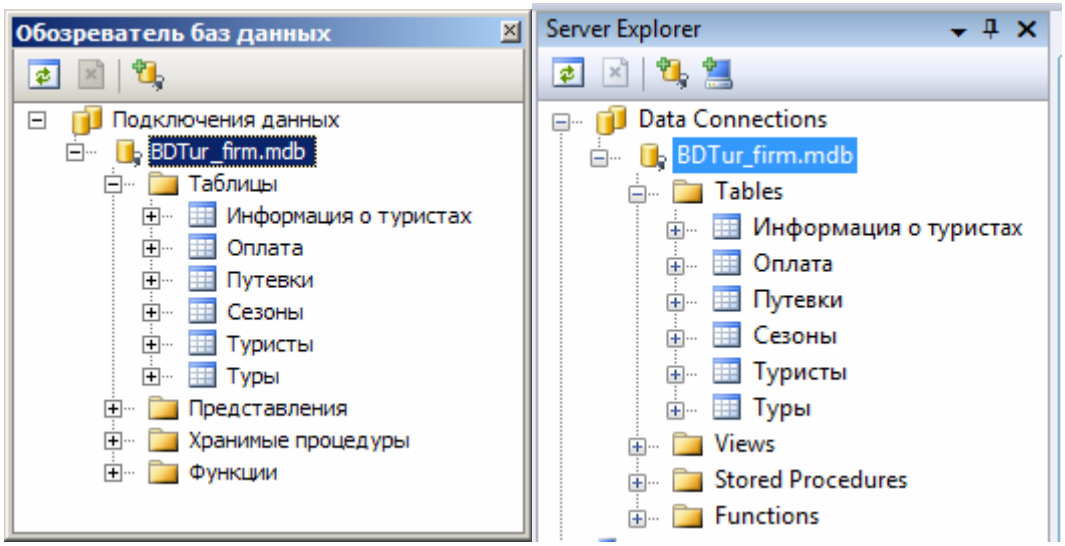


Рис. 58. В окне Server Explorer выводятся все подключения к базам данных

Также можно использовать окно «Обозреватель баз данных» («Server Explorer») для быстрого просмотра содержимого баз данных и – если подключение было создано с правами администратора – изменения их. Откроем нужную таблицу и в выпадающем меню выберем пункт «Загрузить данные» («Retrieve Data») (рис. 59).

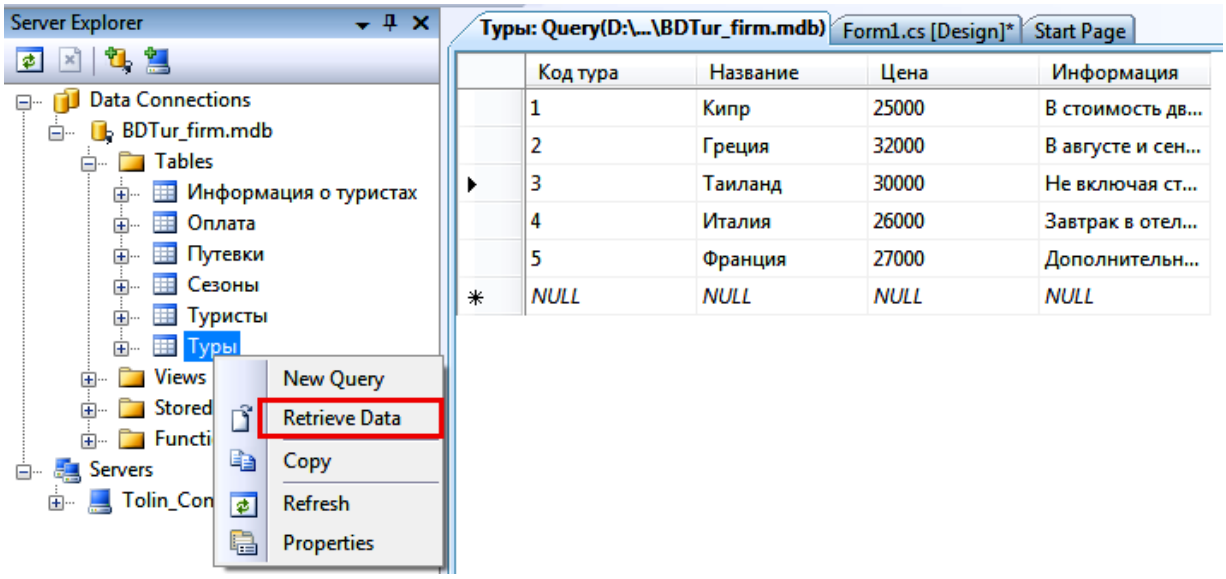


Рис. 59. Просмотр таблицы «Туры»

Выше было рассмотрено создание приложений для работы с базами данных с использованием различных компонентов визуальной среды. Однако разрабатывать такие приложения можно также без использования визуальной среды.

3.4. Пример создания приложения БД «вручную»

Создадим в параллель два приложения для работы с созданными ранее базами данных, аналогичные приложению, рассмотренному выше (см. п. 3.1). Работа будет проводиться, соответственно, со следующими таблицами:

- Microsoft Access – BDTur_firm.mdb (см. п. 1.3);
- Microsoft SQL – BDTur_firmSQL.mdf (см. п. 1.4).

Начнем с запуска Visual Studio 2008 и создания нового проекта Windows Application.

Размещаем на создавшейся форме элемент управления DataGridView, свойству Dock которого устанавливаем значение Fill. Переходим в код формы и подключаем соответствующие пространства имен:

- для MS Access – using System.Data.OleDb;
- для MS SQL – using System.Data.SqlClient;

В любом случае необходимо подключить пространство имен System.Data.

В конструкторе формы после InitializeComponent создаем объект DataAdapter.

В приложении, работающем с MS Access, соответствующий код будет выглядеть следующим образом:

```
public Form1()
{
//
// Required for Windows Form Designer support
//
InitializeComponent();
OleDbDataAdapter dataAdapter =
 new OleDbDataAdapter(CommandText, ConnectionString);
}
```

А в приложении, работающем с MS SQL:

```
public Form1()
{
//
// Required for Windows Form Designer support
//
InitializeComponent();
SqlDataAdapter dataAdapter =
 new SqlDataAdapter(CommandText, ConnectionString);
}
```

Как видно из приведенного кода, фрагменты отличаются только названиями объектов.

В качестве параметров DataAdapter передаются CommandText и ConnectionString. Переменная типа string CommandText представляет собой обыч-

ный SQL-запрос на выборку из таблицы «Туристы», а переменная типа ConnectionString – это так называемая строка подключения, в которой указываются расположение базы данных, ее название, параметры авторизации и пр.

Воспользуемся следующими строками подключения и командами:

```
// MS Access
CommandText: "SELECT [Код туриста], Фамилия, Имя, Отчество FROM Туристы";
ConnectionString: "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="D:\ВМИ\For ADO\
BDTur_firm.mdb"

// MS SQL:
CommandText: "SELECT CustomerID, CompanyName, ContactName, ContactTitle, Address,
City, Region, PostalCode, Country, Phone,  Fax FROM Customers"
ConnectionString: “Data Source=.\SQLEXPRESS;AttachDbFilename="D:\ВМИ\For ADO\
BDTur_firmSQL.mdf";Integrated Security=True;Connect Timeout=30;User Instance=True”
```

Обратите внимание на названия переменных CommandText и ConnectionString. Когда создается объект DataAdapter, в качестве параметров можно передать названия строк, таких как cmdText и conString, или даже cmt и cns – совершенно равноправно, не забыв, конечно же, назвать также эти переменные в классе Form1. Но сама среда Visual Studio 2008 генерирует эти строки именно с такими названиями – CommandText и ConnectionString, поэтому такое название переменных облегчает поддержку и сопровождение разработанного программного продукта.

Продолжим создание программы. Дальнейший код одинаков для обоих вариантов.

Создаем объект DataSet:

```
DataSet  ds = new DataSet();
```

Заполняем таблицу «Туристы» объекта ds данными из базы:

```
dataAdapter.Fill(ds, "Туристы");
```

Связываем источник данных объекта dataGridView1 с таблицей «Туристы» объекта ds:

```
dataGrid1.DataSource = ds.Tables["Туристы"].DefaultView;
```

Теперь запустим созданное приложение. Если все сделано правильно, то на экранной форме отобразится содержимое таблицы «Туристы».

Теперь изменим код следующим образом:

```
dataAdapter.Fill(ds, "Туристы2");
dataGridView1.DataSource = ds.Tables["Туристы2"].DefaultView;
```

Таблицы «Туристы2» в БД нет, однако код по-прежнему работает. Это связано с тем, что таблица, которую мы называем «Туристы», при вызове метода Fill объекта dataAdapter может быть названа как угодно – ее содержимое будет представлять собой извлекаемую таблицу из базы данных. При указании источника данных (DataSource) для объекта dataGridView1 мы ссылаемся именно на

таблицу «Туристы», которая была создана при вызове метода Fill. Таким образом, при заполнении таблиц их можно называть произвольным образом. Однако для реальных проектов рекомендуется использовать настоящие названия таблиц, чтобы избежать путаницы и трудностей в сопровождении.

3.5. Контрольные вопросы и задания к разделу 3

1. Соберите так, как описано в п. 3.1, простейшее приложение для работы с базой данных менеджера турфирмы.
2. Постройте диаграмму классов UML-модели для данного приложения.
3. Каково назначение объектов классов DataSet, DataAdapter, DataTable, DataRelation?
4. Нарисуйте схему связей классов DataSet, DataTable, DataRelation; DataTable и DataRelation; DataColumn и DataTable; DataRow и DataTable.
5. Соберите в соответствии с описанием п. 3.4 «ручной» вариант приложения для работы с базой данных Microsoft Access.
6. Соберите в соответствии с описанием п. 3.4 «ручной» вариант приложения для работы с базой данных Microsoft SQL – NorthwindCS.
7. Постройте диаграммы классов и диаграммы взаимодействия UML-модели для данных приложений.
8. Для чего нужен компонент BindingNavigator и как его использовать?
9. В чем различие визуальных и не визуальных компонент? Приведите пример их использования.
10. Как связываются объекты DataGridView и BindingNavigator с базой данных?
11. Как связываются текстовые поля TextBox с полями базы данных?
12. Для чего в приложениях использованы строки ConnectionString и CommandText?
13. Соберите в соответствии с описанием п. 3.1 приложение для работы с базой данных служащих авиакомпании (см. п. 1.5, задание 4).
14. Соберите «ручной» вариант данного приложения.
15. Создайте и заполните базу данных студентов (см. п. 1.5, задание 11). Соберите в соответствии с описанием п. 3.1 приложение для работы с этой БД.
16. Соберите «ручной» вариант примера приложения для работы с базой данных студентов.

4. ОБЪЕКТЫ ADO .NET

4.1. Соединение с базой данных

4.1.1. Командная строка соединения ConnectionString

Строка соединения ConnectionString определяет параметры, необходимые для установления соединения с источником данных. Строка соединений при использовании мастеров генерируется средой, но можно (и желательно – во избежание неточностей и ошибок) писать эту строчку вручную.

Рассмотрим еще раз строки соединения, которые были созданы при подключении к базам данных BDTur_firm.mdb и BDTur_firmSQL.mdf.

```
// База данных BDTur_firm:
Provider=Microsoft.Jet.OLEDB.4.0;
Data Source="D:\ВМИ\For ADO\BDTur_firm.mdb" // путь к БД

// База данных NorthwindCS:
Data Source=.\SQLEXPRESS;
AttachDbFilename="D:\ВМИ\For ADO\ BDTur_firmSQL.mdf";
Integrated Security=True;
Connect Timeout=30;
User Instance=True
```

В этих строках через точку с запятой просто перечисляются параметры соединения. В таблице 8 приводятся основные значения этих параметров.

Таблица 8

Основные параметры строки соединения

Параметр	**Описание**
Provider (Поставщик)	Свойство применяется для установки или возврата имени поставщика для соединения, используется только для объектов OleDbConnection
Connection Timeout или Connect Timeout (Время ожидания связи)	Длительность времени ожидания связи с сервером перед завершением попытки и генерацией исключения в секундах. По умолчанию 15
Initial Catalog (Исходный каталог)	Имя базы данных
Data Source (Источник данных)	Имя используемого SQL-сервера, когда установлено соединение, или имя файла базы данных Microsoft Access
Password (Пароль)	Пользовательский пароль для учетной записи SQL Server
User ID (Пользовательский ID)	Пользовательское имя для учетной записи SQL Server
Workstation ID	Имя рабочей станции или компьютера

Окончание таблицы 8

Параметр	Описание
Integrated Security или Trusted Connection (Интегрированная безопасность, или Доверительное соединение)	Параметр, который определяет, является ли соединение защищенным. True, False и SSPI – возможные значения (SSPI – эквивалент True)
Persist Security Info (Удержание защитной информации)	Когда установлено False, нуждающаяся в защите информация, такая как пароль, не возвращается как часть соединения, если связь установлена или когда-либо была установленной. Выставление этого свойства в True может быть рискованным в плане безопасности. По умолчанию False

4.1.2. Управление соединением. Объект Connection

Большинство источников данных поддерживает ограниченное количество соединений. Так, например, база данных MS Access может поддерживать одновременную работу не более чем с 255 пользователями. При попытке обращения к базе данных, лимит соединений которой исчерпан, пользователь не получит нужной ему информации и будет вынужден ждать освобождения соединения. Задача разработчика заключается в минимизации времени связи с базой данных, поскольку соединение занимает полезные системные ресурсы.

Когда вызывается метод объекта DataAdapter (например, метод Fill), то он сам проверяет, открыто ли соединение. Если соединения нет, то DataAdapter открывает соединение, выполняет задачи и затем закрывает соединение.

Явное управление соединением обладает рядом преимуществ:

- дает более чистый и удобный для чтения код;
- помогает при отладке приложения;
- является более эффективным.

Для явного управления соединением используется объект Connection. Для того чтобы разобраться с возможностями этого объекта, создадим новый проект и, перетащив элемент управления DataGridView, установим свойству Dock значение Fill. Перейдем в код формы и подключим пространство имен:

```
using System.Data.OleDb;
```

В конструкторе Form1 после InitializeComponent создим объект Connection:

```
OleDbConnection conn = new OleDbConnection(ConnectionString);
```

В качестве параметра объекту conn передается строка подключения ConnectionString.

Можно также устанавливать строку подключения через свойство созданного объекта conn:

```
OleDbConnection conn = new OleDbConnection();
conn.ConnectionString = ConnectionString;
```

В качестве строки соединения можно использовать одну из строк из примеров для приложения, разработанного выше (см. п. 3.4). Например,

```
string ConnectionString = @"Provider=Microsoft.Jet.OLEDB.4.0; " +
                "Data Source=D:\ВМИ\For ADO\BDTur_firm.mdb";
```

Теперь можно устанавливать соединение, вызывая метод Open объекта Connection:

```
OleDbConnection conn = new OleDbConnection();
conn.ConnectionString = ConnectionString;
conn.Open();
```

4.1.3. События объекта Connection

Класс Connection поддерживает несколько событий, позволяющих отслеживать статус соединения и получать уведомляющие сообщения для экземпляра этого класса. Описание событий приводится в таблице 9.

Таблица 9

События объекта Connection

Событие	**Описание**
Disposed	Возникает при вызове метода Dispose экземпляра класса
InfoMes-sage	Возникает при получении информационного сообщения от поставщика данных
StateChange	Возникает при открытии или закрытии соединения. Поддерживается информация о текущем и исходном состояниях

При вызове метода Dispose объекта Connection происходит освобождение занимаемых ресурсов и «сборка мусора». При этом неявно вызывается метод Close.

Создадим новое Windows-приложение, работающее с базой данных BDTur_firmSQL.mdf.

На форме разместим объект dataGridView со свойством Dock=Top, кнопку btnFill с надписью «Заполнить» справа под ним, и две метки (объекты типа Label). Свойству AutoSize каждой метки назначим значение False, свойству Dock второй метки (label2) значение Top, а свойству Dock первой (label1) None, и поместим ее под второй меткой слева, увеличив ее размеры (рис. 61).

Подключаем пространство имен для работы с базой в файле Form1.cs:

```
using System.Data.SqlClient;
```

В классе формы создаем строки connectionString и commandText:

```
string connectionString = @"Data Source=.\SQLEXPRESS; AttachDbFilename=" +
        @"D:\ВМИ\For ADO\BDTur_firmSQL.mdf" +
        ";Integrated Security=True;Connect Timeout=30";

string commandText = "SELECT * FROM Туристы";
```

Объекты ADO будем создавать в обработчике события Click кнопки «Заполнить»:

```
private void btnFill_Click(object sender, System.EventArgs e)
{
	SqlConnection  conn = new SqlConnection();
	conn.ConnectionString = connectionString;
//Делегат EventHandler связывает метод-обработчик conn_Disposed
//с событием Disposed объекта conn
	conn.Disposed+=new EventHandler(conn_Disposed);
//Делегат StateChangeEventHandler связывает метод-обработчик conn_StateChange
//с событием StateChange объекта conn
	conn.StateChange+= new StateChangeEventHandler(conn_StateChange);
	SqlDataAdapter dataAdapter = new SqlDataAdapter(commandText, conn);
	DataSet ds = new DataSet();
	dataAdapter.Fill(ds);
	dataGrid1.DataSource = ds.Tables[0].DefaultView;
//Метод Dispose, включающий в себя метод Close,
//разрывает соединение и освобождает ресурсы.
	conn.Dispose();
}
```

Для создания методов-обработчиков дважды нажимаем клавишу TAB при вводе соответствующей строки как на рис. 60.

В методе conn_Disposed просто будем выводить текстовое сообщение в надпись label2:

```
private void conn_Disposed(object sender, EventArgs e)
{
  label2.Text+="Событие Dispose";
}
```

При необходимости в этом методе могут быть определены соответствующие события.

В методе conn_StateChange будем получать информацию о текущем и исходном состояниях соединения:

```
private void conn_StateChange(object sender, StateChangeEventArgs e)
{
  label1.Text+="\nИсходное состояние: "+e.OriginalState.ToString() +
                "\nТекущее состояние: "+ e.CurrentState.ToString();
}
```

Запускаем приложение. До открытия соединения состояние объекта conn было закрытым (Closed). В момент открытия текущим состоянием становится Open, а предыдущим – Closed. Этому соответствуют первые две строки, выведенные в надпись (рис. 61). После закрытия соединения (вызова метода Dispose) текущим состоянием становится закрытое (Closed), а предыдущим – открытое (Open). Этому соответствуют последние две строки, выводимые в надпись.

```
private void btnFill_Click(object sender, System.EventArgs e)
{
    SqlConnection  conn = new SqlConnection();
    conn.ConnectionString = connectionString;
    //Делегат EventHandler связывает метод-обработчик conn_Disposed
    //с событием Disposed объекта conn
    conn.Disposed+=
                    new EventHandler(conn_Disposed);  (Press TAB to insert)
```

```
private void btnFill_Click(object sender, System.EventArgs e)
{
    SqlConnection  conn = new SqlConnection();
    conn.ConnectionString = connectionString;
    //Делегат EventHandler связывает метод-обработчик conn_Disposed
    //с событием Disposed объекта conn
    conn.Disposed+=new EventHandler(conn_Disposed);
                                    Press TAB to generate handler 'conn_Disposed' in this class
```

```
private void conn_Disposed(object sender, EventArgs e)
{
    |
}
```

Рис. 60. Автоматическое создание методов-обработчиков при помощи Intelligence

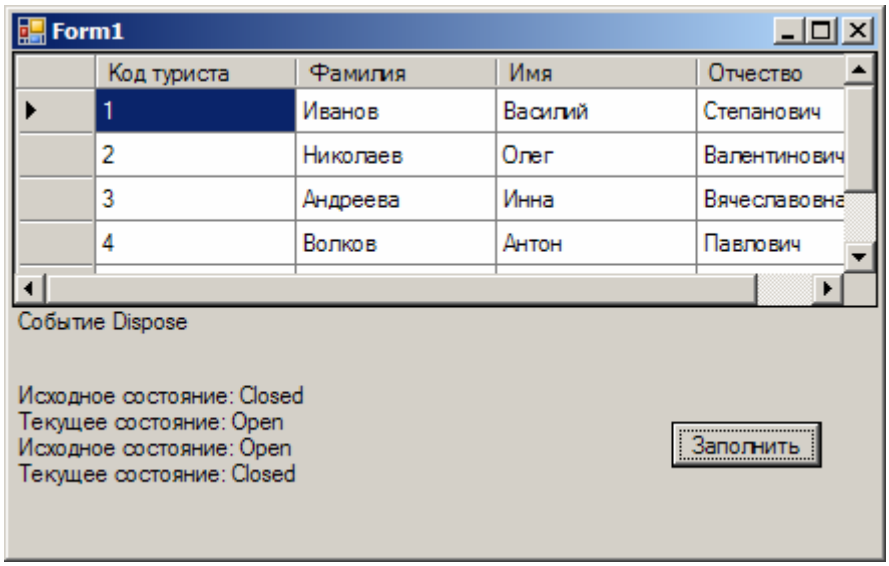


Рис. 61. Изменения состояния соединения с базой данных

Конечно, в таком предельно простом приложении статус соединения очевиден. Но сама идея может применяться в любых, сколь угодно сложных приложениях, когда необходимо определять статус одного из нескольких подключений.

4.1.4. Обработка исключений

При работе с MS SQL

Подключение к базе данных представляет собой одно из слабых мест в работе программы. В силу самых разных причин клиент может не получить доступ к базе данных. Поэтому при создании приложения следует обязательно включать обработку исключений и возможность предоставления пользователю информации о них.

Для получения специализированных сообщений при возникновении ошибок подключения к базе данных Microsoft SQL Server используются классы SqlException и SqlError. Объекты этих классов можно также применять для перехвата номеров ошибок, возвращаемых базой данных (таблица 10).

Таблица 10

Ошибки SQL Server

Номер ошибки	**Описание**
17	Неверное имя сервера
4060	Неверное название базы данных
18456	Неверное имя пользователя или пароль

Дополнительно вводятся уровни ошибок SQL Server, позволяющие охарактеризовать причину проблемы и ее сложность (таблица 11).

Таблица 11

Уровни ошибок SQL Server

Интервал возвращаемых значений	**Описание**	**Действие**
11–16	Ошибка, созданная пользователем	Пользователь должен повторно ввести верные данные
17–19	Ошибки программного обеспечения или оборудования	Пользователь может продолжать работу, но некоторые запросы будут недоступны. Соединение остается открытым
20–25	Ошибки программного обеспечения или оборудования	Сервер закрывает соединение. Пользователь должен открыть его снова

При возникновении исключительной ситуации при соединении с БД возникает исключение типа SqlException. Информация о возникших ошибках содержится в свойстве Errors, представляющем собой коллекцию объектов типа SqlError. Ошибка определяется целочисленным свойством Number, представляющим номер ошибки. Анализируя значение, можно определить причину возникновения ошибки и действия, которые необходимо предпринять для ее устранения.

При работе с MS Access

При возникновении исключительной ситуации при соединении с БД MS Access возникает исключение типа OleDbException. Собственно информация об ошибках хранится в массиве Errors. У каждого элемента массива есть два свойства:

1) Свойство Message возвращает причину ошибки. Язык сообщения определяется языком MS Access, т. е. если установлена русская версия, то сообщение будет на русском языке.
2) Свойство NativeError (внутренняя ошибка) возвращает номер исключения, генерируемый самим источником данных.

Вместе или по отдельности со свойством SQL State их можно использовать для создания кода, предоставляющего пользователю расширенную информацию об ошибке и действиях, необходимых для ее устранения.

4.1.5. Работа с пулом соединений

Подключение к базе данных требует затрат времени, в частности, на установление соединения по каналам связи, прохождение аутентификации, и лишь после этого можно выполнять запросы и получать данные.

Клиентское приложение, взаимодействующее с базой данных и закрывающее каждый раз соединение при помощи метода Close, будет не слишком производительным: значительная часть времени и ресурсов будет тратиться на установку повторного соединения. Решением данного затруднения может быть использование трехуровневой модели, при которой клиентское соединение будет взаимодействовать с базой данных через промежуточный сервер.

В этой модели клиентское приложение открывает соединение через промежуточный сервер (рис. 62, А). После завершения работы соединение закрывается приложением, но промежуточный сервер продолжает удерживать его в течение заданного промежутка времени, например, 60 секунд. По истечении этого времени промежуточный сервер закрывает соединение с базой данных (рис. 62, Б). Если в течение этой минуты, например, после 35 секунд, клиентское приложение снова требует связи с базой данных, то сервер просто предоставляет уже готовое соединение, причем после завершения работы обнуляет счет времени и готов снова минуту ждать обращения (рис. 62, В).

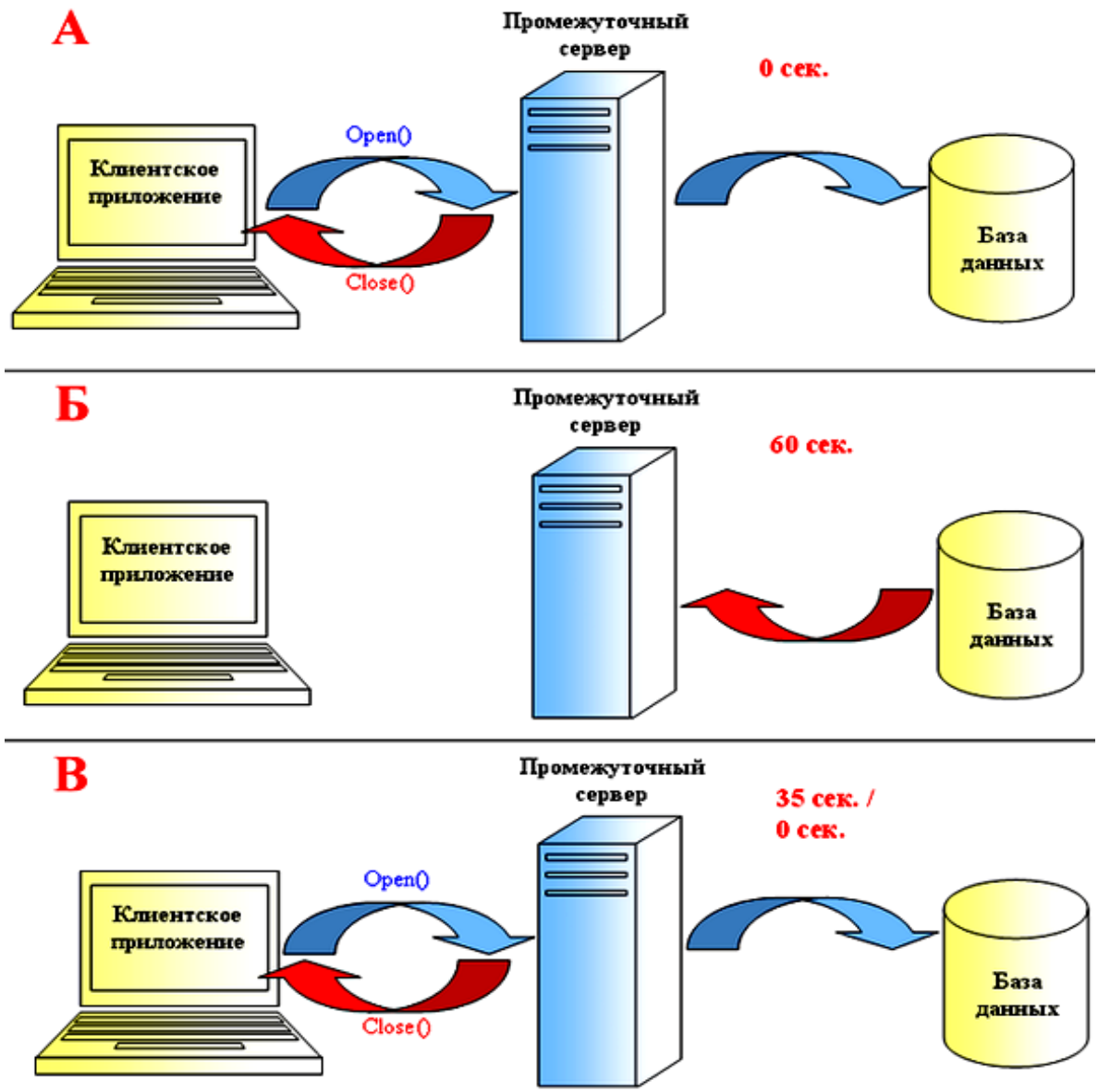


Рис. 62. Трехуровневая модель соединения с базой данных

Использование этой модели позволяет сократить время, необходимое для установки связи с удаленной базой данных, и повысить эффективность работы приложения. Промежуточный сервер будет выступать в качестве пула соединений. Более того, если к нему будут обращаться несколько клиентских приложений, использующих одинаковую базу данных и параметры авторизации, то выигрыш времени будет еще более заметным.

При создании подключения с использованием поставщиков данных .NET автоматически создается *пул соединений*. При вызове метода Close соединение не разрывается, а по умолчанию помещается в пул. В течение 60 секунд соединение остается открытым, и если оно не используется повторно, поставщик данных закрывает его. Если же по каким-либо причинам необходимо закрывать соединение, не помещая его в пул, в строке соединения ConnectionString нужно вставить дополнительный параметр.

Для поставщика OLE DB: OLE DB Services=–4;

Для поставщика SQL Server: Pooling=False;

Теперь при вызове метода Close соединение действительно будет разорвано.

Поставщик данных Microsoft SQL Server предоставляет также дополнительные параметры управления пулом соединений (таблица 12).

Таблица 12

Параметры пула соединения поставщика MS SQL Server

Параметр	**Описание**	**Значение по умолчанию**
Connection Lifetime	Время (в секундах), по истечении которого открытое соединение будет закрыто и удалено из пула. Сравнение времени создания соединения с текущим временем проводится при возвращении соединения в пул. Если соединение не запрашивается, а время, заданное параметром, истекло, соединение закрывается. Значение 0 означает, что соединение будет закрыто по истечении максимального предусмотренного таймаута (60 с.)	0
Enlist	Необходимость связывания соединения с контекстом текущей транзакции потока	True
Max Pool Size	Максимальное число соединений в пуле. При исчерпании свободных соединений клиентское приложение будет ждать освобождения свободного соединения	100
Min Pool Size	Минимальное число соединений в пуле в любой момент времени	0
Pooling	Использование пула соединений	True

Для задания значения параметра, отличного от принятого по умолчанию, следует явно включить его в строку ConnectionString.

Помещение соединений в пул, включенное по умолчанию, – одно из средств повышения производительности приложений. Не следует без надобности отключать эту возможность.

4.2. Хранимые процедуры

Хранимой процедурой называется одна или несколько SQL-конструкций, которые записаны в базе данных.

Задача администрирования базы данных включает в себя в первую очередь распределение уровней доступа к БД. Разрешение выполнения обычных SQL-запросов большому числу пользователей может стать причиной неисправ-

ностей из-за неверного запроса или их группы. Чтобы этого избежать, разработчики базы данных могут создать ряд хранимых процедур для работы с данными и полностью запретить доступ для обычных запросов.

Такой подход при прочих равных условиях обеспечивает большую стабильность и надежность работы. Это одна из главных причин создания собственных хранимых процедур. Другие причины – быстрое выполнение, разбиение больших задач на малые модули, уменьшение нагрузки на сеть – значительно облегчают процесс разработки и обслуживания архитектуры «клиент – сервер».

4.2.1. Стандартные запросы к БД

Создадим в среде Management Studio новый SQL-запрос, содержащий следующий текст:

```
exec sp_databases
```

В результате выполнения выводится список всех баз, созданных на данном локальном сервере (рис. 63):

Результаты | Сообщения

	DATABASE_NAME	DATABASE_SIZE	REMARKS
1	BDTur_fimSQL	4096	NULL
2	master	5376	NULL
3	model	3264	NULL
4	msdb	7616	NULL
5	Northwind	7936	NULL
6	pubs	6016	NULL
7	tempdb	2560	NULL

Рис. 63. Выполнение запроса exec sp_databases

Запрос запустил на выполнение одну из системных хранимых процедур, которая находится в базе master. Ее можно найти в списке «Хранимые процедуры» («Stored Procedures») базы. Все системные хранимые процедуры имеют префикс «sp». Обратите внимание на то, что системные процедуры в редакторе выделяются бордовым цветом и для многих из них не нужно указывать в выпадающем списке конкретную базу.

Выполним еще один запрос:

```
exec sp_monitor
```

В результате ее выполнения выводится статистика текущего SQL-сервера (рис. 64).

Результаты | Сообщения

	last_run	current_run	seconds
1	2005-10-14 01:53:53.740	2008-08-06 12:26:11.843	88770738

	cpu_busy	io_busy	idle
1	2(1)-0%	1(0)-0%	29570(29329)-0%

	packets_received	packets_sent	packet_errors
1	170(142)	490(462)	0(0)

	total_read	total_write	total_errors	connections
1	3081(3081)	242(242)	0(0)	73(61)

Рис. 64. Статистика Microsoft SQL Server

Для вывода списка хранимых процедур в базе Northwind[3] используем следующую процедуру:

```
USE Northwind
exec sp_stored_procedures
```

База Northwind содержит 1312 хранимых процедур (рис. 67), большая часть из которых системные. Для просмотра списка хранимых процедур в других базах следует вызвать для них эту же процедуру.

```
USE Northwind
exec sp_stored_procedures
```

Результаты | Сообщения

	PROCEDURE_QUALIFIER	PROCEDURE_OWNER	PROCEDURE_NAME	NUM_INPUT_PARAMS	NUM_OUTPUT
1	Northwind	dbo	CustOrderHist;1	-1	-1
2	Northwind	dbo	CustOrdersDetail;1	-1	-1
3	Northwind	dbo	CustOrdersOrders;1	-1	-1
4	Northwind	dbo	Employee Sales by Country;1	-1	-1
5	Northwind	dbo	Sales by Year;1	-1	-1
6	Northwind	dbo	SalesByCategory;1	-1	-1
7	Northwind	dbo	Ten Most Expensive Products;1	-1	-1
8	Northwind	sys	dm_db_index_operational_stats;0	-1	-1
9	Northwind	sys	dm_db_index_physical_stats;0	-1	-1

Запрос успешно выполнен. | COREDUO\SQLEXPRESS (9.0 RTM) | COREDUO\User (52) | Northwind | 00:00:05 | 1312 строк

Строка 6 Столбец 1 ВСТ

Рис. 65. Вывод списка хранимых процедур базы данных Northwind

[3] NorthWind – это пример базы данных для SQL Server 2000, файл установки для которого можно скачать по адресу http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=06616212-0356-46a0-8da

4.2.2. Простые запросы к БД

Перейдем к созданию своих собственных процедур. Создадим новый бланк запросов и введем следующий запрос:

create procedure proc1 **as**
select [Код туриста], Фамилия, Имя, Отчество **from** Туристы

Здесь create procedure – оператор, указывающий на создание хранимой процедуры, proc1 – ее название, далее после оператора as следует обычный SQL-запрос. Квадратные скобки необходимы для указания поля таблицы, в названии которого содержится пробел. После выполнения запроса появится сообщение:

Выполнение команд успешно завершено.
The COMMAND(s) completed successfully.

Данное сообщение означает, что все сделано правильно и команда создала процедуру proc1. Убедиться в этом можно, развернув ветку «Программирование – Хранимые процедуры» в среде Management Express (рис. 66)[4].

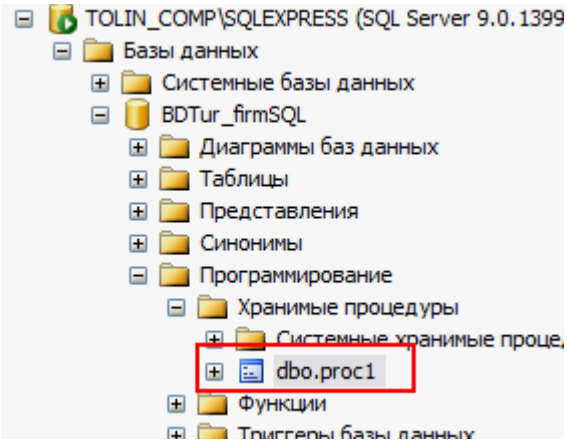


Рис. 66. Созданная хранимая процедура

Для просмотра результата вызываем ее:

exec proc1

Появляется уже знакомое извлечение всех записей таблицы «Туристы» со всеми записями (рис. 67).

Из полученного результата видно, что создание содержимого хранимой процедуры не отличается ничем от создания обычного SQL-запроса.

В таблице 13 приведены примеры хранимых процедур.

[4] Перед просмотром не забудте обновить данные в среде Management Express (меню Вид – Обновить, или F5).

Результаты | Сообщения

	Код туриста	Фамилия	Имя	Отчество
1	1	Иванов	Василий	Степанович
2	2	Николаев	Олег	Валентинович
3	3	Андреева	Инна	Вячеславовна
4	4	Волков	Антон	Павлович
5	5	Кириллова	Ольга	Михаиловна

Рис. 67. Результат запуска процедуры proc1

Таблица 13

Примеры хранимых процедур

№	SQL-конструкция для создания	Команда для извлечения	Описание
1	create procedure proc1 as select [Код туриста], Фамилия, Имя, Отчество from Туристы	exec proc1	Вывод всех записей таблицы «Туристы»
	Результат запуска Результаты | Сообщения Код туриста / Фамилия / Имя / Отчество 1 1 Иванов Василий Степанович 2 2 Николаев Олег Валентинович 3 3 Андреева Инна Вячеславовна 4 4 Волков Антон Павлович 5 5 Кириллова Ольга Михаиловна		
2	create procedure proc2 as select top 3 Фамилия from туристы	exec proc2	Вывод первых трех значений поля «Фамилия» таблицы «Туристы»
	Результат запуска Результаты Фамилия 1 Иванов 2 Николаев 3 Андреева		
3	create procedure proc3 as select * from туристы where Фамилия = 'Андреева'	exec proc3	Вывод всех полей таблицы «Туристы», содержащих в поле «Фамилия» значение «Андреева»
	Результат запуска Результаты | Сообщения Код туриста / Фамилия / Имя / Отчество 1 3 Андреева Инна Вячеславовна		

№	SQL-конструкция для создания	Команда для извлечения	Описание
4	create procedure proc4 as select count (*) from Туристы	exec proc4	Подсчет числа записей таблицы «Туристы»
	Результат запуска Результаты / Сообщения (Отсутствует имя столбца) 1 5		
5	create procedure proc5 as select sum(Сумма) from Оплата	exec proc5	Подсчет значений поля Сумма таблицы Оплата
	Результат запуска Результаты / Сообщения (Отсутствует имя столбца) 1 140000,00		
6	create procedure proc6 as select max(Цена) from Туры	exec proc6	Вывод максимального значения поля Цена таблицы Туры
	Результат запуска Результаты / Сообщения (Отсутствует имя столбца) 1 32000,00		
7	create procedure proc7 as select min(Цена) from Туры	exec proc7	Вывод минимального значения поля Цена таблицы Туры
	Результат запуска Результаты / Сообщения (Отсутствует имя столбца) 1 25000,00		
8	create procedure proc8 as select * from Туристы where Фамилия like '%и%'	exec proc8	Вывод всех записей таблицы Туристы, содержащих в значении поля Фамилия букву «и» (в любой части слова)
	Результат запуска Результаты / Сообщения Код туриста / Фамилия / Имя / Отчество 1 1 Иванов Василий Степанович 2 2 Николаев Олег Валентинович 3 5 Кириллова Ольга Михаиловна		

Окончание таблицы 13

№	SQL-конструкция для создания	Команда для извлечения	Описание
9	create procedure proc9 as select * from Туристы inner join [Информация о туристах] on Туристы.[Код Туриста]= [Информация о туристах].[Код Туриста]	exec proc9	Соединение двух таблиц «Туристы» и «Информация о туристах» по полю «Код туриста» и вывод полной связанной информации из обеих таблиц

Результат запуска

Результаты | Сообщения

	Код туриста	Фамилия	Имя	Отчество	Код туриста	Серия паспорта	Город	Страна	Теле
1	1	Иванов	Василий	Степанович	1	СА 1341548	Екатеринбург	Россия	1234
2	2	Николаев	Олег	Валентинович	2	ТЕ 1562487	Ростов	Россия	3216
3	3	Андреева	Инна	Вячеславовна	3	ИЛ 6548243	Оренбург	Россия	6854
4	4	Волков	Антон	Павлович	4	СА 1869742	Москва	Россия	1234
5	5	Кириллова	Ольга	Михаиловна	5	ПО 4567891	Санкт-Петербург	Россия	3245

10	create procedure proc10 as select * from Туристы left join [Информация о туристах] on Туристы.[Код Туриста]= [Информация о туристах].[Код Туриста]	exec proc10	Прежде чем создать эту процедуру и затем ее выполнить, добавим в таблицу «Туристы» базы данных BDTur_firm новую строчку с произвольными значениями. В результате в таблице «Туристы» у нас получится 6 записей, а в связанной с ней таблице «Информация о туристах» – 5. Создаем хранимую процедуру и запускаем ее

Результат запуска

Результаты | Сообщения

	Код туриста	Фамилия	Имя	Отчество	Код туриста	Серия паспорта	Город	Страна	Теле
1	1	Иванов	Василий	Степанович	1	СА 1341548	Екатеринбург	Россия	1234
2	2	Николаев	Олег	Валентинович	2	ТЕ 1562487	Ростов	Россия	3216
3	3	Андреева	Инна	Вячеславовна	3	ИЛ 6548243	Оренбург	Россия	6854
4	4	Волков	Антон	Павлович	4	СА 1869742	Москва	Россия	1234
5	5	Кириллова	Ольга	Михаиловна	5	ПО 4567891	Санкт-Петербург	Россия	3245
6	6	Проверка	Проверка	Проверка	NULL	NULL	NULL	NULL	NULL

Операция inner join объединяет записи из двух таблиц, если поле (поля), по которому связаны эти таблицы, содержат одинаковые значения. Общий синтаксис выглядит следующим образом:

from таблица1 **inner join** таблица2
  on таблица1.поле1 <оператор сравнения> таблица2.поле2

Операция left join используется для создания так называемого левого внешнего соединения. С помощью этой операции выбираются все записи пер-

вой (левой) таблицы, даже если они не соответствуют записям во второй (правой) таблице. Общий синтаксис имеет вид:

```
from таблица1 left join таблица2
   on таблица1.поле1  <оператор сравнения>  таблица2.поле2.
```

4.2.3. Параметризованные запросы к БД

На практике часто возникает потребность получить информацию из БД по некоторому заданному значению «входных» данных (параметру) запроса. Такие запросы называются *параметризированными*, а соответствующие процедуры создаются с параметрами.

Например, для получения записи в таблице «Туристы» по заданной фамилии создадим следующую процедуру:

```
create proc proc_p1
 @Фамилия nvarchar(50)
as
select *
from Туристы
where Фамилия=@Фамилия
```

После знака @ указывается название параметра и его тип. Таким образом, для параметра с именем Фамилия был выбран тип nvarchar с количеством символов 50, поскольку в самой таблице для поля «Фамилия» установлен этот тип. Попытка запустить процедуру командой exec proc_p1 приводит к появлению сообщения об отсутствующем параметре.

Процедуру необходимо запускать следующим образом:

```
exec proc_p1 'Андреева'
```

В результате выводится запись, соответствующая фамилии «Андреева» (рис. 68).

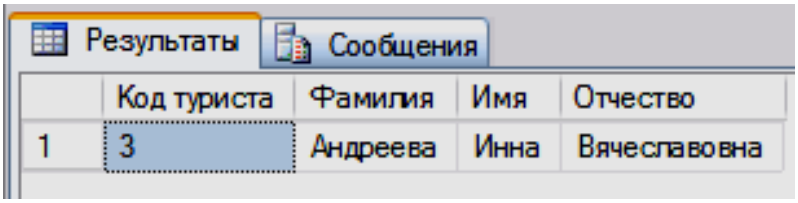


Рис. 68. Запуск процедуры proc_p1

Очевидно, что если в качестве параметра указать строку, отсутствующую в таблице, результатом выполнения процедуры будет пустая строка.

В таблице 14 приводятся примеры хранимых процедур с параметрами.

Таблица 14

Хранимые процедуры с параметрами

№	SQL-конструкция для создания	Команда для извлечения
1	create proc proc_p1 @Фамилия nvarchar(50) as select * from Туристы where Фамилия=@Фамилия	exec proc_p1 'Андреева'
	Описание Извлечение записи из таблицы «Туристы» с заданной фамилией	
	Результат запуска Результаты / Сообщения Код туриста: 3; Фамилия: Андреева; Имя: Инна; Отчество: Вячеславовна	
№	**SQL-конструкция для создания**	**Команда для извлечения**
2	create proc proc_p2 @nameTour nvarchar(50) as select * from Туры where Название=@nameTour	exec proc_p2 'Франция'
	Описание Извлечение записи из таблицы «Туры» с заданным названием тура.	
	Результат запуска Результаты / Сообщения Код тура: 5; Назван...: Франция; Цена: 27000,00; Информация: Дополнительные экскурсии не входят в стоимость п...	
№	**SQL-конструкция для создания**	**Команда для извлечения**
3	create procedure proc_p3 @Фамилия nvarchar(50) as select * from Туристы inner join [Информация о туристах] on Туристы.[Код Туриста] = [Информация о туристах].[Код Туриста] where Туристы.Фамилия = @Фамилия	exec proc_p3 'Андреева'
	Описание Вывод родительской и дочерней записей с заданной фамилией из таблиц «Туристы» и «Информация о туристах»	
	Результат запуска Результаты / Сообщения Код туриста: 3; Фамилия: Андреева; Имя: Инна; Отчество: Вячеславовна; Код туриста: 3; Серия паспорта: ИП 6548243; Город: Оренбург; Страна: Россия; Тел: 685	

№	SQL-конструкция для создания	Команда для извлечения
4	create procedure proc_p4 @nameTour nvarchar(50) as select * from Туры inner join Сезоны on Туры.[Код тура]=Сезоны.[Код тура] where Туры.Название = @nameTour	exec proc_p4 'Франция'

Описание

Вывод родительской и дочерней записей с заданным названием тура из таблиц «Туры» и «Сезоны»

Результат запуска (изображение разрезано)

Результаты | Сообщения

	Код тура	Назван...	Цена	Информация	Код сезо...	Код
1	5	Франция	27000,00	Дополнительные экскурсии не входят в стоимость п...	5	5

Дата начала	Дата конца	Сезон закр...	Количество мест
2007-03-01 00:00:00	2007-10-01 00:00:00	0	12

№	SQL-конструкция для создания	Команда для извлечения
5	create proc proc_p5 @nameTour nvarchar(50), @Курс float as update Туры set Цена=Цена/(@Курс) where Название=@nameTour	exec proc_p5 'Франция', 26 или exec proc_p5 @nameTour = 'Франция', @Курс= 26 Просматриваем изменения простым SQL – запросом: select * from Туры

Описание

Процедура с двумя входными параметрами – названием тура и курсом валюты. При извлечении процедуры они последовательно указываются. Поскольку в самом запросе используется оператор update, не возвращающий данных, то для просмотра результата следует извлечь измененную таблицу оператором select

Результат запуска

(1 строка обработана)

После запуска оператора select

Результаты | Сообщения

	Код тура	Назван...	Цена	Информация
1	1	Кипр	25000,00	В стоимость двух взрослых путевок входит цена одн...
2	2	Греция	32000,00	В августе и сентябре действуют специальные скидки
3	3	Таиланд	30000,00	Не включая стоимость авиабилета
4	4	Италия	26000,00	Завтрак в отеле включен в стоимость путевки
5	5	Франция	1038,4615	Дополнительные экскурсии не входят в стоимость п...

Окончание таблицы 14

№	SQL-конструкция для создания	Команда для извлечения
6	create proc proc_p6 @nameTour nvarchar(50), @Курс float = 26 as update Туры set Цена=Цена/(@Курс) where Название=@nameTour	exec proc_p6 'Таиланд' или exec proc_p6 'Таиланд', 28
	Описание Процедура с двумя входными параметрами, причем один их них – @Курс имеет значение по умолчанию. При запуске процедуры достаточно указать значение первого параметра – для второго параметра будет использоваться его значение по умолчанию. При указании значений двух параметров будет использоваться введенное значение	
	Результат запуска Запускаем процедуру с одним входным параметром: exec proc_p6 'Таиланд' Для просмотра используем оператор select:	

Результаты | Сообщения

	Код тура	Назван...	Цена	Информация
1	1	Кипр	25000,00	В стоимость двух взрослых путевок входит цена одн...
2	2	Греция	32000,00	В августе и сентябре действуют специальные скидки
3	3	Таиланд	1153,8462	Не включая стоимость авиабилета
4	4	Италия	26000,00	Завтрак в отеле включен в стоимость путевки
5	5	Франция	1038,4615	Дополнительные экскурсии не входят в стоимость п...

Процедуры с выходными параметрами позволяют возвращать значения, получаемые в результате обработки SQL-конструкции при подаче определенного параметра. Представим, что нам нужно получать фамилию туриста по его коду (полю «Код туриста»). Создадим следующую процедуру:

```
create proc proc_po1
@TouristID int,
@LastName nvarchar(60) output
as
select @LastName = Фамилия from Туристы where [Код туриста] = @TouristID
```

Оператор output указывает на то, что выходным параметром здесь будет @LastName. Запустим эту процедуру для получения фамилии туриста, значение поля «Код туриста» которого равно «4»:

```
declare @LastName nvarchar(60)
exec proc_po1 '4',
@LastName output
select @LastName
```

Оператор declare нужен для объявления поля, в которое будет выводиться значение. Получаем фамилию туриста (рис. 69):

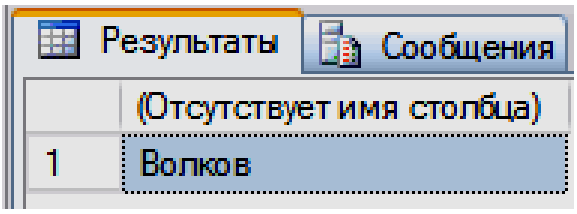


Рис. 69. Результат запуска процедуры proc_po1

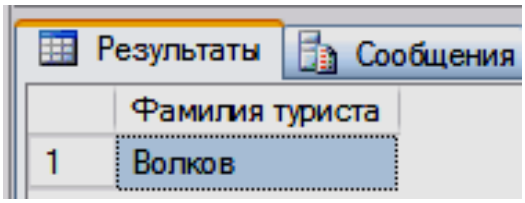


Рис. 70. Выполнение процедуры proc_po1. Применение псевдонима

Для задания названия столбца можно применить псевдоним:

```
declare @LastName nvarchar(60)
exec proc_po1 '4',
@LastName output
select @LastName as 'Фамилия туриста'
```

Теперь столбец имеет заголовок (рис. 70).

В таблице 15 приводятся примеры хранимых процедур с входными и выходными параметрами.

Таблица 15

Хранимые процедуры с входными и выходными параметрами

№	SQL-конструкция для создания	Команда для извлечения
1	create proc proc_po1 @TouristID int, @LastName nvarchar(60) output as select @LastName = Фамилия from Туристы where [Код туриста] = @TouristID	declare @LastName nvarchar(60) exec proc_po1 '4', @LastName output select @LastName as 'Фамилия туриста'
	Описание Извлечение фамилии туриста по заданному коду	
	Результат запуска Результаты Сообщения Фамилия туриста 1 Волков	
№	**SQL-конструкция для создания**	**Команда для извлечения**
2	create proc proc_po2 @CountCity int output as select @CountCity = count([Код туриста]) from [Информация о туристах] where Город like '%рг%'	declare @CountCity int exec proc_po2 @CountCity output select @CountCity as 'Количество туристов, проживающих в городах %рг%'
	Описание Подсчет количества туристов из городов, имеющих в своем названии сочетание букв «рг». Следует ожидать число три (Екатеринбург, Оренбург, Санкт–Петербург)	

№	SQL-конструкция для создания	Команда для извлечения
	Результат запуска Результаты / Сообщения Количество туристов, проживающих в городах %pr% 1 3	
№	**SQL–конструкция для создания**	**Команда для извлечения**
3	create proc proc_po3 @TouristID int, @CountTour int output as select @CountTour = count(Туры.[Код тура]) from Путевки inner join Сезоны on Путевки.[Код сезона]= Сезоны.[Код сезона] inner join Туры on Туры.[Код тура] = Сезоны.[Код тура] inner join Туристы on Путевки.[Код туриста] = Туристы.[Код туриста] where Туристы.[Код туриста] = @TouristID	declare @CountTour int exec proc_po3 '1', @CountTour output select @CountTour AS 'Количество туров, которые турист посетил'
	Описание Подсчет количества туров, которых посетил турист с заданным значением поля «Код Туриста»	
	Результат запуска Результаты / Сообщения Количество туров, которые турист посетил 1 1	
№	**SQL–конструкция для создания**	**Команда для извлечения**
4	create proc proc_po4 @TouristID int, @BeginDate smalldatetime, @EndDate smalldatetime, @SumMoney money output as select @SumMoney = sum(Сумма) from Оплата inner join Путевки on Оплата.[Код путевки] = Путевки.[Код путевки] inner join Туристы on Путевки.[Код туриста] = Туристы.[Код туриста] where [Дата оплаты] between(@BeginDate) and (@EndDate) and Туристы.[Код туриста] = @TouristID	declare @TouristID int, @BeginDate smalldatetime, @EndDate smalldatetime, @SumMoney money exec proc_po4 '1', '1/20/2007', '1/20/2008', @SumMoney output select @SumMoney as 'Общая сумма за период'

Окончание таблицы 15

№	SQL-конструкция для создания	Команда для извлечения
	Описание Подсчет общей суммы, которую заплатил данный турист за определенный период. Турист со значением «1» поля «Код Туриста» внес оплату 4/13/2007	
	Результат запуска Результаты / Сообщения Общая сумма 1 25000,00	
№	**SQL–конструкция для создания**	**Команда для извлечения**
5	create proc proc_po5 @CodeTour int, @ChisloPutevok int output as select @ChisloPutevok = count(Путевки.[Код сезона]) from Путевки inner join Сезоны on Путевки.[Код сезона] = Сезоны.[Код сезона] inner join Туры on Туры.[Код тура] = Сезоны.[Код тура] where Сезоны.[Код тура] = @CodeTour	declare @ChisloPutevok int exec proc_po5 '1', @ChisloPutevok output select @ChisloPutevok AS 'Число путевок, проданных в этом туре'
	Описание Подсчет количества путевок, проданных по заданному туру	
	Результат запуска Результаты / Сообщения Число путевок, проданных в этом туре 1 1	

Для удаления хранимой процедуры используется оператор drop:

```
drop proc proc1
```

Здесь proc1 – название процедуры (см. табл. 13 под № 1).

4.2.4. Создание хранимых процедур в Management Studio

Программа SQL Management Studio позволяет создавать хранимые процедуры при помощи встроенного шаблона. Для этого необходимо выбрать пункт выпадающего меню «Создать хранимую процедуру» для ветки «Программирование – Хранимые процедуры».

Генерируемый шаблон имеет следующий вид:

```
-- ==============================================
-- Template generated from Template Explorer using:
-- Create Procedure (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
```

```
-- command (Ctrl–Shift–M) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the procedure.
-- ============================================
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- ===========================================
-- Author:          <Author,,Name>
-- Create date:     <Create Date,,>
-- Description:     <Description,,>
-- ===========================================
CREATE PROCEDURE <Procedure_Name, sysname, ProcedureName>
        -- Add the parameters for the stored procedure here
<@Param1,sysname,@p1> <Datatype_For_Param1, ,int>=<Default_Value_For_Param1, , 0>,
<@Param2,sysname,@p2> <Datatype_For_Param2, ,int>=<Default_Value_For_Param2, , 0>
AS
BEGIN
        -- SET NOCOUNT ON added to prevent extra result sets from
        -- interfering with SELECT statements.
        SET NOCOUNT ON;

   -- Insert statements for procedure here
        SELECT <@Param1, sysname, @p1>, <@Param2, sysname, @p2>
END
GO
```

В данном шаблоне выделяются следующие части:

- Общий комментарий, объясняющий происхождение шаблона
- Служебные команды
- Комментарий к процедуре, определяющий автора, дату создания и описание процедуры
- Оператор создания процедуры
- Определение параметров процедуры
- Определение текста процедуры

Созданные хранимые процедуры можно использовать при создании источника данных (рис. 45). Вид окна выбора местоположения данных для БД с одной хранимой процедурой приведен на рисунке 71.

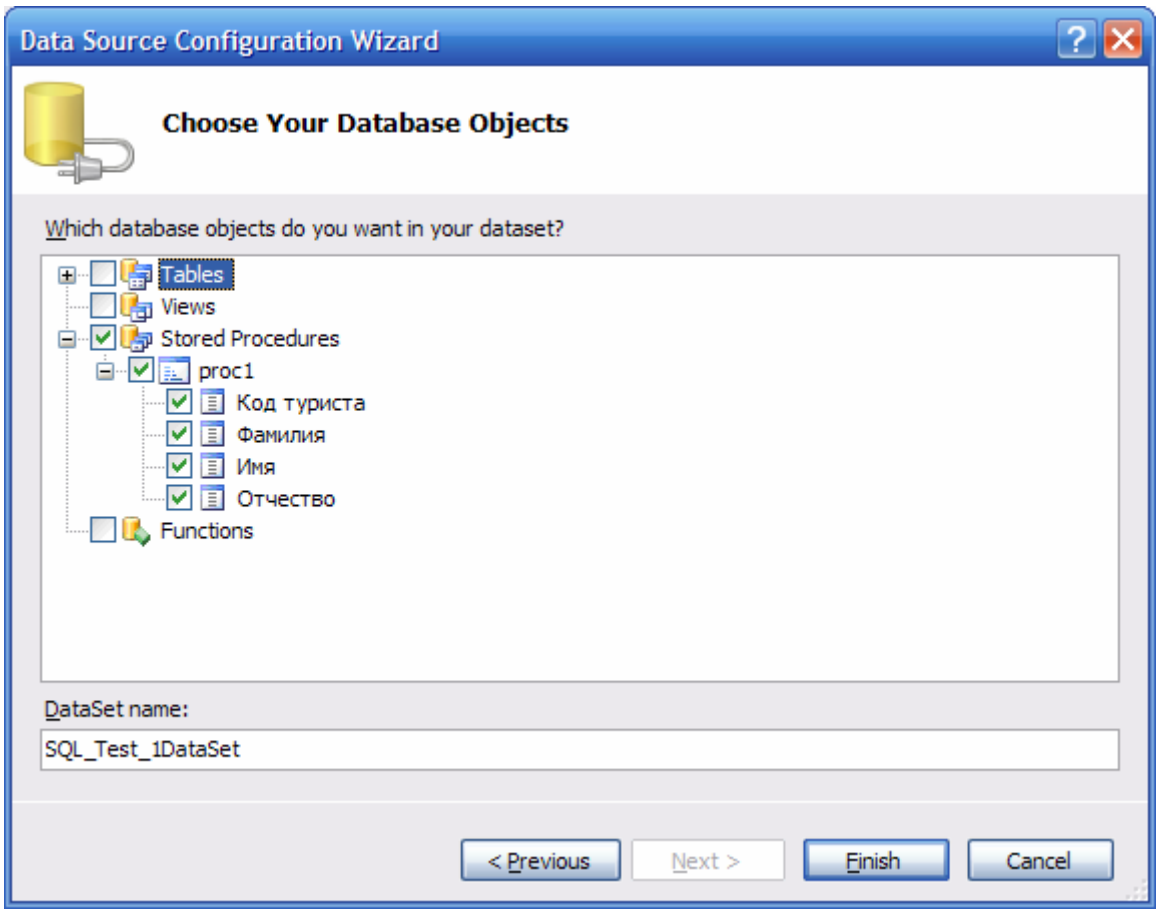


Рис. 71. Использование хранимой процедуры в качестве источника данных

4.2.5. Создание хранимых процедур в Visual Studio 2008

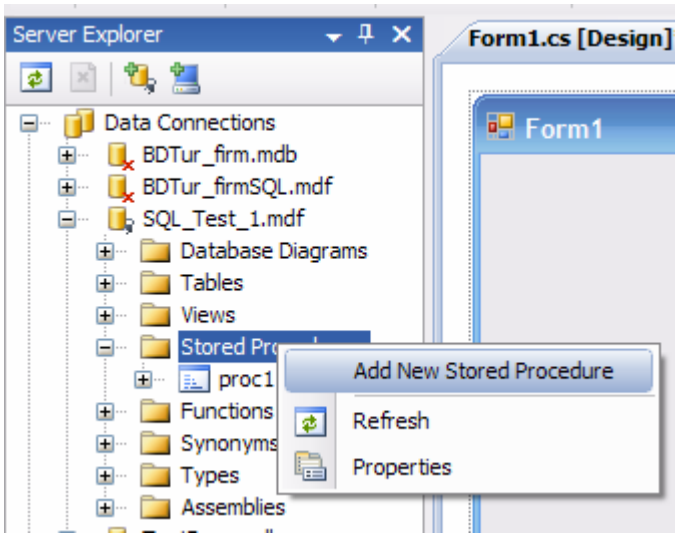


Рис. 72. Создание новой процедуры в окне «Server Explorer»

Среда Visual Studio 2008 предоставляет интерфейс для создания хранимых процедур в базе данных при наличии подключения к ней.

Запустим Visual Studio (даже нет необходимости создавать какой-либо проект), перейдем на вкладку «Обозреватель баз данных» («Server Explorer»), раскрываем подключение к базе данных, затем на узле «Хранимые процедуры» («Stored Procedures») щелкаем правой кнопкой и выбираем пункт «Добавить новую хранимую процедуру» («New Stored Procedure») (рис. 72).

Появляется шаблон структуры, сгенерированный мастером:

```
CREATE PROCEDURE dbo.StoredProcedure1
	/*
	(
	@parameter1 int = 5,
	@parameter2 datatype OUTPUT
	)
	*/
AS
	/* SET NOCOUNT ON */
	RETURN
```

Для того чтобы приступить к редактированию, достаточно убрать знаки комментариев «/*», «*/».

Команда NOCOUNT со значением ON отключает выдачу сообщений о количестве строк таблицы, получающейся в качестве запроса. Необходимость данного оператора заключается в том, что при использовании более чем одного оператора (SELECT, INSERT, UPDATE или DELETE) в начале запроса надо поставить команду SET NOCOUNT ON, а перед последним оператором SELECT – команду SET NOCOUNT OFF.

Например, хранимую процедуру proc_po1 (см. таблицу 15) можно переписать следующим образом:

```
CREATE PROCEDURE dbo.proc_vs1
  (
    @TouristID int,
    @LastName nvarchar(60) OUTPUT
  )
AS
  SET NOCOUNT ON
  SELECT    @LastName = Фамилия
  FROM      Туристы
  WHERE     ([Код туриста] = @TouristID)
RETURN
```

После завершения редактирования SQL-конструкция будет обведена синей рамкой. Щелкнув правой кнопкой в этой области и выбрав пункт меню «Разработать блок SQL» («Design SQL Block»), можно перейти к построителю запросов (Query Builder) (рис. 73).

Для выполнения процедуры необходимо в выпадающем меню процедуры, представленном на рисунке 74, выбрать пункт Execute.

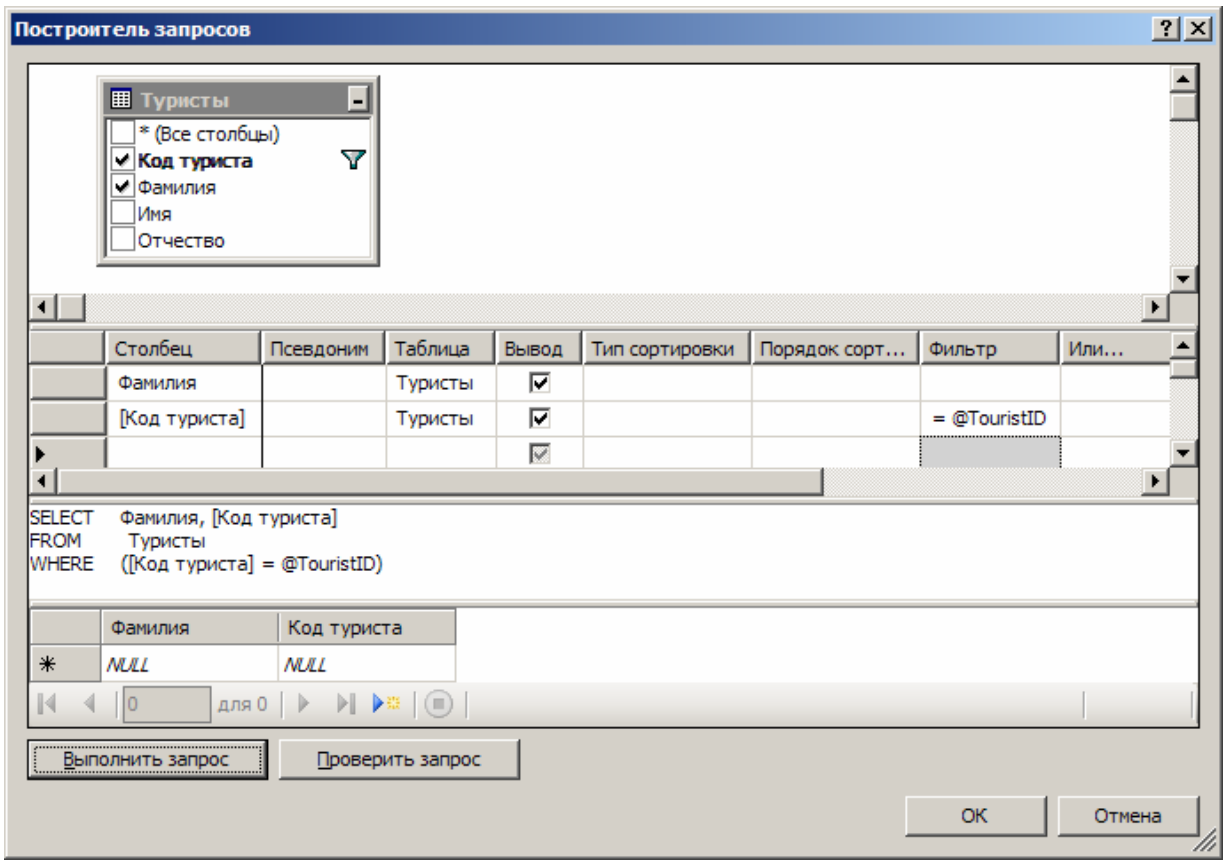


Рис. 73. Построитель запросов (Query Builder)

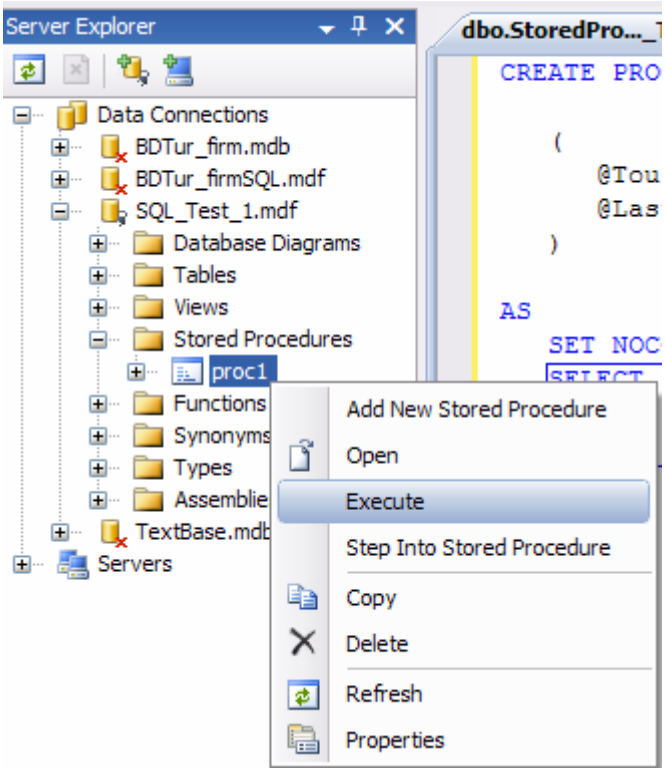


Рис. 74. Выполнение хранимой процедуры

При попытке выполнить хранимую процедуру в окно сообщений будет выведена информация о том, что данная процедура требует значение параметра.

Для записи процедуры в базу данных достаточно ее сохранить. Происходит синхронизация с базой данных, и процедура "proc_vs1" появляется в списке. Двойной щелчок открывает ее для редактирования, причем заголовок имеет следующий вид:

ALTER PROCEDURE dbo.proc_vs1

Оператор ALTER позволяет производить действия (редактирование) с уже имеющимся объектом базы данных.

4.3. Запросы к базе данных

4.3.1. Командная строка SQL-запроса CommandText

Для извлечения таблиц и содержащихся в них данных используются SQL-запросы. Как уже было показано выше, переменная CommandText содержит в себе SQL-запрос, синтаксис которого адаптирован для данного поставщика данных. Мы можем управлять извлечением данных, изменяя строку CommandText. Например, если на экранной форме приложения столбец «Код туриста» отображать не нужно, то SQL-запрос будет выглядеть следующим образом:

```
string CommandText = "SELECT Фамилия, Имя, Отчество FROM Туристы";
```

В окне работающего приложения теперь будут выводиться только соответствующие три поля (рис. 75).

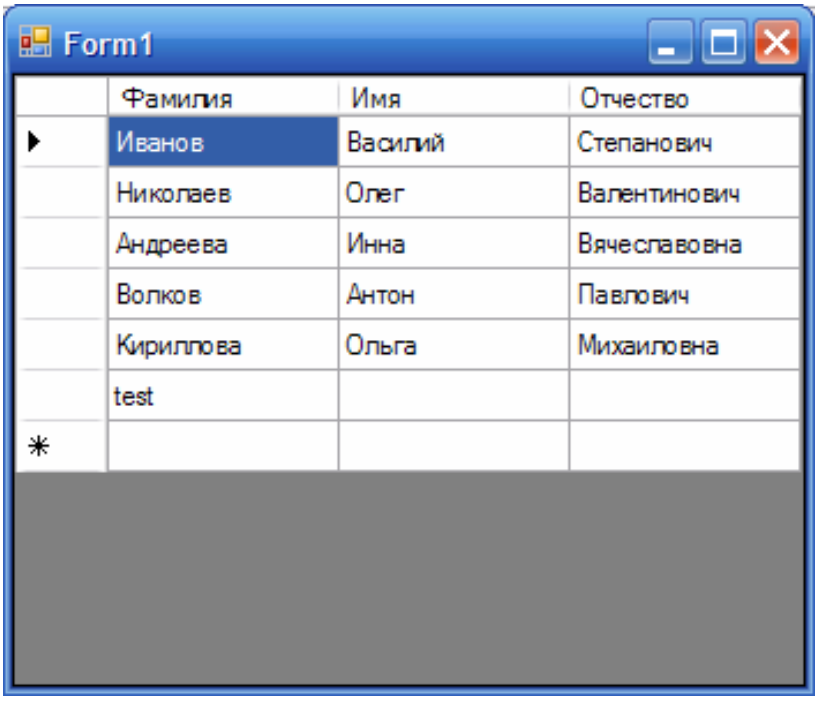


Рис. 75. Ограничение выводимых полей

Выведем теперь все записи клиентов, имена которых начинаются на "О":

```
string CommandText = "SELECT Фамилия, Имя, Отчество
                      FROM Туристы  where  Имя  like 'О%'";
```

Окно запущенного приложения показано на рисунке 76.

Рис. 76. Ограничение выводимых полей и записей

Можно использовать все возможности языка манипулирования данными (DML) SQL для отбора данных и модификации строки CommandText для получения нужного результата.

4.3.2. Объект Command

Создание и инициализация

При определении объектов ADO .NET DataAdapter был назван адаптером, преобразователем, предназначенным для взаимодействия с базой данных. Это действительно так, однако если рассматривать взаимодействие с базой данных более глубоко, то выясняется, что в ADO .NET есть специализированный объект для выполнения запросов, называемый Command. Под запросами понимается выполнение SQL-конструкций или запуск хранимых процедур. Этот объект среда создает неявным образом в методе InitializeComponent. Данный объект обладает следующими свойствами.

Connection – подключение к базе данных. Можно использовать как существующее подключение, так и создавать новое.

CommandType – тип команды (запроса), который будет направляться к базе данных. Возможны следующие значения:

- Text. Текстовая команда состоит из SQL-конструкции, направляемой к базе данных. Это значение используется по умолчанию.
- StoredProcedure. Текстовая команда состоит из названия хранимой процедуры.
- TableDirect. Текстовая команда состоит из названия таблицы базы данных. В результате извлекается все содержимое таблицы. Эта команда аналогична текстовой команде SELECT * FROM Название_таблицы. Данная команда поддерживается только управляемым поставщиком OLE DB.

CommandText – собственно сам текст запроса.

Чтобы выполнить запрос, свойству Connection объекта Command следует задать объект – имя созданного объекта Connection:

```
// создание, инициализация и открытие объекта Connection
OleDbConnection conn = new OleDbConnection();
conn.ConnectionString = ConnectionString;        // содержание строки см. раздел 4.1
conn.Open();

// создание объекта Command и связывание его с объектом Connection
OleDbCommand myCommand = new OleDbCommand();
myCommand.Connection = conn;
```

Объект Connection также предоставляет метод CreateCommand, позволяющий упростить данный процесс – этот метод возвращает новый объект Command, уже инициализированный для использования созданного объекта Connection:

```
OleDbConnection conn = new OleDbConnection();
conn.ConnectionString = ConnectionString;
conn.Open();
OleDbCommand myCommand = conn.CreateCommand();
```

Эти два способа создания и инициализации объекта Command эквивалентны. Теперь следует определить SQL-запрос, который будет извлекать данные. Как и для строки соединения, будем использовать запрос из предыдущего приложения.

Для хранения текста команды объект myCommand имеет свойство CommandText. Чтобы избежать путаницы, изменим название переменной – CommandText на commandText (с маленькой буквы):

```
MyCommand.CommandText = commandText;
```

Создаем объект OleDbDataAdapter:

```
OleDbDataAdapter dataAdapter = new OleDbDataAdapter();
```

Объект dataAdapter имеет свойство SelectCommand, в котором мы и будем указывать объект myCommand:

```
dataAdapter.SelectCommand = myCommand;
```

Создаем объект DataSet:

```
DataSet ds = new DataSet();
```

Заполняем ds данными из dataAdapter:

```
dataAdapter.Fill(ds, "Туристы");
```

Указываем источник данных DataSource для dataGrid1:

```
dataGrid1.DataSource = ds.Tables["Туристы"].DefaultView;
```

Закрываем соединение явным образом:

```
conn.Close();
```

Теперь можно запускать приложение. Получим результат, что и на рисунке 60, но теперь управление всеми объектами, работающими с данными, происходит явно.

Полный листинг конструктора формы выглядит так:

```
public Form1()
{
   InitializeComponent();
   string commandText = "SELECT Фамилия, Имя, Отчество FROM Туристы";
   string ConnectionString = @"Provider=Microsoft.Jet.OLEDB.4.0; " +
         "Data Source=D:\ВМИ\For ADO\BDTur_firm.mdb";
   OleDbConnection conn=new OleDbConnection(ConnectionString);
   conn.Open();
   OleDbCommand MyCommand=new OleDbCommand();
   MyCommand.Connection = conn;
   //или OleDbCommand MyCommand=conn.CreateCommand();
   MyCommand.CommandText = commandText;
   OleDbDataAdapter dataAdapter = new OleDbDataAdapter();
   dataAdapter.SelectCommand = MyCommand;
   DataSet ds = new DataSet();
   dataAdapter.Fill(ds, "Туристы");
   dataGridView1.DataSource = ds.Tables["Туристы"].DefaultView;
   conn.Close();
}
```

Работа с базой данных SQL аналогична рассмотренному примеру. Отличие заключается в использовании вместо классов OleDbConnection, OleDbCommand, OleDbDataAdapter классов SqlConnection, SqlCommand, SqlDataAdapter соответственно.

Отметим, что оба подхода (визуальный и ручной) к созданию и использованию объектов ADO обладают своими областями применения. Для упрощения разработки и сопровождения визуальных форм проще использовать визуальный подход. При создании приложений, от которых требуется надежность, гибкость или отсутствие экранных форм (например, Web-сервис) лучше создавать объекты ADO вручную.

При визуальном подходе наряду с представлением данных в виде таблицы часто приходится использовать традиционные элементы интерфейса, такие как TextBox, Label и пр. Привязка к элементам интерфейса и средство навигации по записям были рассмотрены выше (см. п. 3.1).

Свойства CommandType и CommandText

Для демонстрации свойств создадим новое Windows-приложение баз данных. Перетаскиваем на форму элемент управления DataGridView, его свойству Dock устанавливаем значение Fill. В классе формы создаем строки connectionString и commandText:

```
private string connectionString = @"Data Source = .\SQLEXPRESS; AttachDbFilename = " +
      @"D:\ВМИ\For ADO\BDTur_firmSQL2.mdf" +
      ";Integrated Security=True; Connect Timeout=30; User Instance=True";
```

```
private string commandText = "SELECT * FROM Туры";
```

Конструктор формы будет иметь следующий вид:

```
private string connectionString = @"Data Source = .\SQLEXPRESS; AttachDbFilename = " +
        @"D:\ВМИ\For ADO\BDTur_firmSQL2.mdf" +
        ";Integrated Security=True; Connect Timeout=30; User Instance=True";
private string commandText = "SELECT * FROM Туры";

    public Form1()
    {
        InitializeComponent();
        SqlConnection conn = new SqlConnection(connectionString);
        conn.Open();
        SqlCommand myCommand = conn.CreateCommand();
        //myCommand.Connection = conn;
        myCommand.CommandText = commandText;
        //myCommand.ExecuteNonQuery();
        SqlDataAdapter dataAdapter = new SqlDataAdapter();
        dataAdapter.SelectCommand = myCommand;
        DataSet ds = new DataSet();
        dataAdapter.Fill(ds, "Туры");
        conn.Close();
        dataGridView1.DataSource = ds.Tables["Туры"].DefaultView;
    }
```

Запустим приложение. На форму выводится содержимое таблицы «Туры». Здесь используется всего один объект DataAdapter, который сам открывает и закрывает соединение для получения данных. Конструктор объекта SqlCommand является перегруженным. Полное описание конструктора приведено в MSDN.

Метод ExecuteNonQuery

Для выполнения запросов на выборку простейших процедур достаточно просто указать тип и передать название запроса или процедуры. Однако этого явно недостаточно для серьезных приложений. Поэтому рассмотрим методы объекта Command.

Метод ExecuteNonQuery применяется для выполнения запросов, не возвращающих данные, таких как UPDATE, INSERT и DELETE – они вносят изменения в таблицу базы данных, не возвращая ничего назад в результате выполнения.

Создадим новое консольное приложение. Ниже приведен его полный листинг:

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
```

```
using System.Text;

namespace ConsoleApplication3
{
   class Program
   {
      static void Main(string[] args)
      {
         string connectionString = @"Data Source=.\SQLEXPRESS;AttachDbFilename="+
                        @"D:\ВМИ\For ADO\BDTur_firmSQL2.mdf" +
                        ";Integrated Security=True;Connect Timeout=30;User Instance=True";
         string commandText = "SELECT * FROM Туры";
         SqlConnection conn = new SqlConnection(connectionString);
         conn.Open();
         SqlCommand myCommand = conn.CreateCommand();
         myCommand.CommandText = "UPDATE Туристы " +
                        "SET Фамилия = 'Сергеева' WHERE [Код туриста] = 3";
         myCommand.ExecuteNonQuery();
         conn.Close();
      }
   }
}
```

В свойстве CommandText указывается непосредственно текст запроса, который устанавливает значение «Сергеева» поля «Фамилия» для записи с полем «Код Туриста» = 3. Для выполнения запроса просто вызываем метод ExecuteNonQuery:

```
myCommand.ExecuteNonQuery();
```

Запускаем приложение, нажимая F5. При успешном выполнении запроса консольное окно приложения появляется и тут же исчезает. Запускаем Management Studio, открываем таблицу «Туристы» и убеждаемся в том, что запись изменилась (рис. 77).

Таблица - dbo.Туристы | Таблица - dbo.Туристы | Сводка

Код туриста	Фамилия	Имя	Отчество
1	Иванов	Василий	Степанович
2	Николаев	Олег	Валентинович
3	Сергеева	Инна	Вячеславовна
4	Волков	Антон	Павлович
5	Кириллова	Ольга	Михаиловна
NULL	NULL	NULL	NULL

Рис. 77. Таблица «Туристы», изменение записи

Метод ExecuteNonQuery неявно возвращает результат выполнения запроса в виде количества измененных записей. Это значение может применяться для проверки правильности выполнения запроса, например, следующим образом:

```
int Uspeshnoelzmenenie = myCommand.ExecuteNonQuery();
    if (Uspeshnoelzmenenie !=0)
    {
      Console.WriteLine ("Изменения внесены");
    }
    else
    {
      Console.WriteLine("Не удалось внести изменения");
    }
```

Отметим, что данный вариант текста не позволяет отличить обновление одной строки и нескольких строк.

Закомментируем имеющееся свойство CommandText и добавим новое:

```
myCommand.CommandText =
        "INSERT  " +
        "INTO  Туристы ([Код туриста], Фамилия, Имя, Отчество) " +
        "VALUES (6, 'Тихомиров', 'Андрей', 'Борисович')";
```

Запускаем приложение, затем переходим в SQL Server Enterprise Manager – запрос добавил новую запись (рис. 78).

	Код туриста	Фамилия	Имя	Отчество
▶	1	Иванов	Василий	Степанович
	2	Николаев	Олег	Валентинович
	3	Сергеева	Инна	Вячеславовна
	4	Волков	Антон	Павлович
	5	Кириллова	Ольга	Михаиловна
	6	Тихомиров	Андрей	Борисович
*	NULL	NULL	NULL	NULL

Рис. 78. Таблица «Туристы», добавление записи

Снова закомментируем свойство CommandText, добавим теперь запрос на удаление записи:

```
myCommand.CommandText = "DELETE FROM Туристы WHERE [Код туриста] = 4";
```

Запускаем приложение – из таблицы удалена четвертая запись (рис. 79).

Метод ExecuteNonQuery применяется также для выполнения запросов, относящихся к категории DDL языка SQL. Язык определения данных (Data Defi-

nition Language, DDL) позволяет создавать и изменять структуру объектов базы данных, например, создавать и удалять таблицы. Основными операторами этого языка являются CREATE, ALTER, DROP. В результате выполнения запросов DDL не возвращаются данные – именно поэтому можно применять метод ExecuteNonQuery.

	Код туриста	Фамилия	Имя	Отчество
▶	1	Иванов	Василий	Степанович
	2	Николаев	Олег	Валентинович
	3	Сергеева	Инна	Вячеславовна
	5	Кириллова	Ольга	Михаиловна
	6	Тихомиров	Андрей	Борисович
*	NULL	NULL	NULL	NULL

Рис. 79. Таблица «Туристы», удаление записи

Закомментируем имеющееся свойство CommandText и напишем новое, создающее в базе BDTur_firm2 новую таблицу «Отзывы»:

```
myCommand.CommandText = "CREATE TABLE Отзывы (Кодотзыва INT NOT NULL, " +
                        "Кодтуриста INT NOT NULL, Комментарий VARCHAR(50))";
```

Запускаем приложение, затем переходим в Management Studio, нажимаем кнопку (обновить) на панели инструментов – в базе появляется новая таблица (рис. 80).

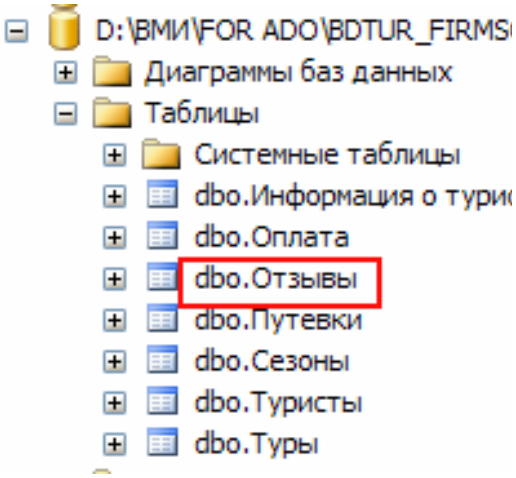


Рис. 81. База данных BDTur_firm2, новая таблица "Отзывы"

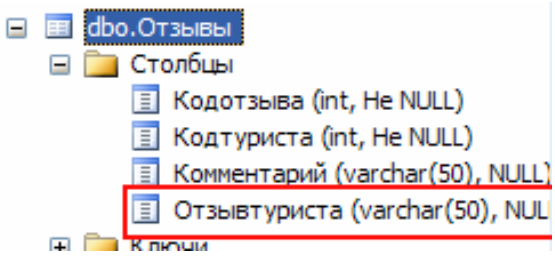


Рис. 80. Свойства таблицы "Отзывы"

Для добавления нового столбца «Отзыв туриста» строка CommandText должна иметь следующий вид:

```
myCommand.CommandText = "ALTER TABLE Отзывы ADD Отзывтуриста VARCHAR(50)";
```

В Management Studio в списке столбцов таблицы видим новое поле «Отзывтуриста» (рис. 81).

Для удаления таблицы «Отзывы» запускаем приложение, содержащее следующую строку CommandText:

```
myCommand.CommandText = "DROP TABLE Отзывы";
```

В Management Studio видно, что таблица полностью исчезла из базы данных. Если нужно лишь удалить данные из таблицы, сохранив структуру, то необходимо воспользоваться следующей командой:

```
myCommand.CommandText = "DELETE FROM Отзывы";
```

Объектами базы данных могут быть не только таблицы, но и хранимые процедуры, схемы, представления. В любом случае манипуляция с ними будет относиться к категории DDL.

Метод ExecuteNonQuery может применяться и для выполнения запросов, относящихся к категории DCL. Язык управления данными (Data Control Language, DCL) предназначен для управления доступом (определения полномочий) к объектам базы данных. Основными операторами этого языка являются GRANT, DENY, REVOKE. Данные запросы рассматриваться не будут – использование в данном случае объекта Commnad не отличается ничем от рассмотренного выше.

Метод ExecuteScalar

Метод ExecuteScalar объекта Command применяется для запросов, возвращающих одно значение. Такие запросы возникают при использовании агрегатных функций COUNT, MIN, MAX. Для демонстрации метода создадим новое консольное приложение. Полный листинг этого приложения:

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Text;

namespace ConsoleApplication3
{
   class Program
   {
      static void Main(string[] args)
      {
         string connectionString = @"Data Source=.\SQLEXPRESS;AttachDbFilename="+
                        @"D:\ВМИ\For ADO\BDTur_firmSQL2.mdf"+
                        ";Integrated Security=True;Connect Timeout=30;User Instance=True";
         string commandText = "SELECT * FROM Туры";
         SqlConnection conn = new SqlConnection(connectionString);
         conn.Open();
         SqlCommand myCommand = conn.CreateCommand();
         myCommand.CommandText = "SELECT COUNT (*) FROM Туры";
         string KolichestvoTurov = Convert.ToString(myCommand.ExecuteScalar());
```

```
            conn.Close();
            Console.WriteLine("Количество туров: " + KolichestvoTurov);
            Console.ReadKey();
        }
    }
}
```

Возвращаемый методом ExecuteScalar результат приводим к типу string для вывода в окно консоли. Запускаем приложение – запрос вернул число 5 (рис. 82).

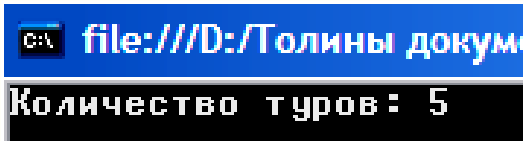


Рис. 82. Вывод количества туров

Можно несколько раз применять этот метод.

```
myCommand.CommandText = "SELECT COUNT (*) FROM Туры";
string KolichestvoTurov = Convert.ToString(myCommand.ExecuteScalar());
myCommand.CommandText = "SELECT MAX (Цена) FROM Туры";
string MaxPrice = Convert.ToString(myCommand.ExecuteScalar());
myCommand.CommandText = "SELECT MIN (Цена) FROM Туры";
string MinPrice = Convert.ToString(myCommand.ExecuteScalar());
myCommand.CommandText = "SELECT AVG (Цена) FROM Туры";
string AvgPrice = Convert.ToString(myCommand.ExecuteScalar());
conn.Close();
Console.WriteLine("Количество туров: " + KolichestvoTurov +
  "\nСамый дорогой тур, цена в руб. : " + MaxPrice +
  "\nСамый дешевый тур, цена в руб.: " + MinPrice +
  "\nСредняя цена туров: " + AvgPrice);
```

Запускаем приложение и получаем несколько значений из базы данных (рис. 83).

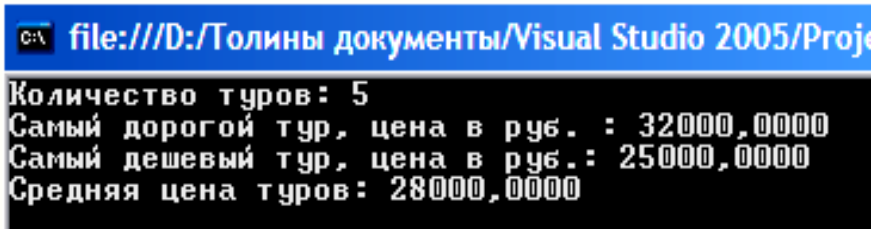


Рис. 83. Вывод нескольких значений

Когда требуется получать подобные одиночные значения, всегда следует применять метод ExecuteScalar. Такое решение позволяет значительно повысить производительность.

Метод ExecuteReader

Теперь перейдем к рассмотрению следующего метода – ExecuteReader. Он применяется для получения набора записей из базы данных. Особенностью этого метода является то, что он возвращает специальный объект DataReader, с помощью которого просматриваются записи. Для хранения данных, полученных из базы, ранее использовался объект DataSet. Объект DataReader, в отличие от DataSet, требует наличия постоянного подключения для извлечения и просмотра данных, кроме того, он открывает данные только для чтения.

Создадим новое консольное приложение. Полный листинг этого приложения:

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Text;

namespace ConsoleApplication3
{
   class Program
   {
      static void Main(string[] args)
      {
         string connectionString = @"Data Source=.\SQLEXPRESS;AttachDbFilename="+
                        @"D:\ВМИ\For ADO\BDTur_firmSQL2.mdf"+
                        ";Integrated Security=True;Connect Timeout=30;User Instance=True";
         string commandText = "SELECT * FROM Туры";
         SqlConnection conn = new SqlConnection(connectionString);
         conn.Open();
         SqlCommand myCommand = conn.CreateCommand();
         myCommand.CommandText = "SELECT * FROM Туристы";
         SqlDataReader dataReader = myCommand.ExecuteReader();
         while (dataReader.Read())
         {
            Console.WriteLine(dataReader["Фамилия"]);
         }
         dataReader.Close();
         conn.Close();
         Console.ReadKey();
      }
   }
}
```

Объект dataReader создается в результате вызова метода ExecuteReader объекта myCommand:

```
SqlDataReader dataReader = myCommand.ExecuteReader();
```

Перед считыванием первой записи происходит вызов метода Read объекта dataReader и вывод набора записей в консольное окно. Результат выполнения данного приложения представлен на рисунке 84.

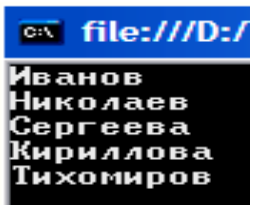


Рис. 84. Вывод поля «Фамилия»

Объект DataReader возвращает набор данных типа object, причем для обращения к содержимому поля таблицы вместо имени поля можно использовать индекс:

```
Console.WriteLine(dataReader[1]);
```

Перечислим несколько полей:

```
Console.WriteLine(dataReader[0]);
Console.WriteLine(dataReader[1]);
Console.WriteLine(dataReader[2]);
Console.WriteLine(dataReader[3]);
```

При выводе они будут располагаться в структурированном виде (рис. 85).

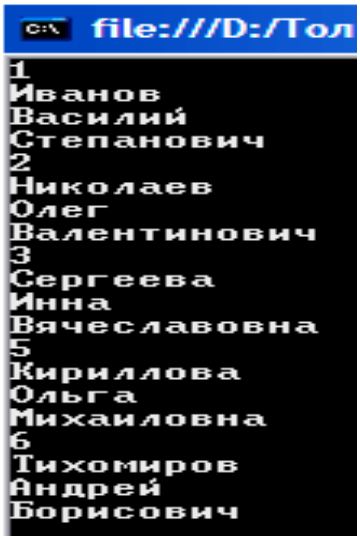


Рис. 85. Вывод содержимого всех полей

Поскольку мы имеем дело с объектами (тип данных object), для вывода записей в виде строк не применимо их простое объединение:

```
...
Console.WriteLine(dataReader[0] + dataReader[1] + dataReader[2] + dataReader[3]);
...
```

Преобразованные к типу string значения можно объединять:

```
Console.WriteLine(Convert.ToString(dataReader[0]) + " " +
                  Convert.ToString(dataReader[1]) + " " +
                  Convert.ToString(dataReader[2]) +" " +
                  Convert.ToString(dataReader[3]));
```

Теперь записи выводятся в более привычном виде (рис. 86).

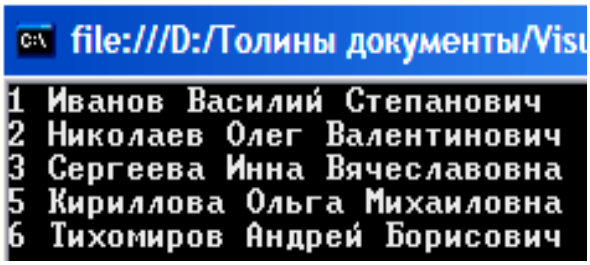


Рис. 86. Вывод содержимого всех полей в виде записей

4.3.3. Параметризированные запросы

Выше были рассмотрены основные методы объекта Command в консольных приложениях. Это дало возможность понять синтаксис самих методов, без привязки к какому-либо интерфейсу. Однако, после того как синтаксис стал ясен, возникает вопрос – как же использовать эти методы в реальных приложениях? Очевидно, что простое копирование кода в конструктор формы, по сути, не изменит пример. Следовательно, необходимо привязывать вводимые значения к элементам пользовательского интерфейса, например, к текстовым полям. Но это означает, что параметры строки запроса будут неизвестны до тех пор, пока пользователь не введет соответствующие значения. Например, для метода ExecuteNonQuery строка commandText имела следующий вид:

```
myCommand.CommandText =
        "UPDATE Туристы SET Фамилия = 'Сергеева' WHERE Кодтуриста = 3";
```

Если создадим приложение, где пользователь будет вводить фамилию и код туриста, то мы не можем заранее указать, какие это будут значения. Логически запрос можно представить примерно так:

```
myCommand.CommandText = " UPDATE Туристы SET
    Фамилия = 'Какая-то_фамилия,_которую_введет_пользователь'
    WHERE Кодтуриста = 'Какой-то_код,_который_введет_пользователь' ";
```

Для решения таких задач, которые возникли еще в самом начале разработки языка SQL, были придуманы параметризированные запросы. В них неизвестные значения заменяются параметрами следующим образом:

```
myCommand.CommandText =
    "UPDATE Туристы SET Фамилия = @Family WHERE Кодтуриста = @TouristID";
```

Здесь @Family – параметр для неизвестного значения фамилии, @TouristID – параметр для неизвестного значения кода туриста. Отметим, что параметры пишутся без кавычек.

Теперь можно выполнить привязку параметров к тексту, вводимому пользователем.

Использование метода ExecuteNonQuery

Для демонстрации привязки параметров создадим новое Windows-приложение и разместим на нем компоненты пользовательского интерфейса. Форма в режиме дизайна будет иметь следующий вид (рис. 87).

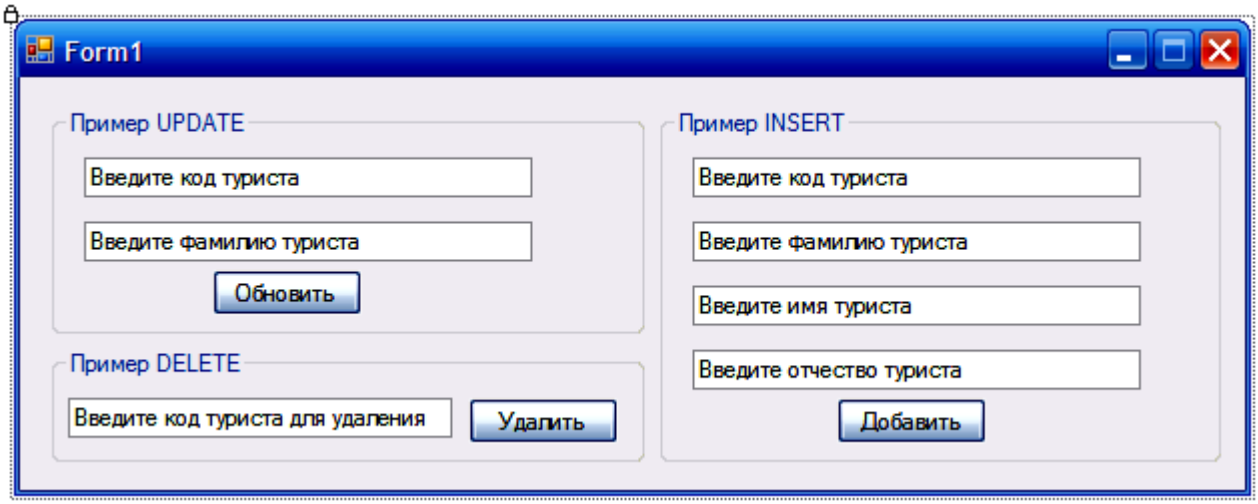


Рис. 87. Разработанное приложение, вид формы в режиме дизайна

Подключаем пространство имен для работы с базой данных:

```
using System.Data.SqlClient;
```

В классе формы создаем экземпляр conn:

```
SqlConnection  conn = null;
```

Обработчик кнопки btnUpdate будет иметь следующий вид:

```
private void btnUpdate_Click(object sender, System.EventArgs e)
{
    try
    {
        string  Family = Convert.ToString(this.txtFamilyUpdate.Text);
        int TouristID = int.Parse(this.txtTouristIDUpdate.Text);

        conn = new SqlConnection();
        conn.ConnectionString = @"Data Source=.\SQLEXPRESS;AttachDbFilename=" +
                        @"D:\ВМИ\For ADO\BDTur_firmSQL2.mdf" +
                        ";Integrated Security=True;Connect Timeout=30;User Instance=True";
        conn.Open();
```

```
        SqlCommand myCommand = conn.CreateCommand();
        myCommand.CommandText =
        "UPDATE Туристы SET  Фамилия = @Family  WHERE [Код туриста] = @TouristID";
        myCommand.Parameters.Add("@Family", SqlDbType.NVarChar, 50);
        myCommand.Parameters["@Family"].Value =  Family;
        myCommand.Parameters.Add("@TouristID", SqlDbType.Int, 4);
        myCommand.Parameters["@TouristID"].Value = TouristID;

        int UspeshnoeIzmenenie = myCommand.ExecuteNonQuery();

        if (UspeshnoeIzmenenie !=0)
        {
                MessageBox.Show("Изменения внесены", "Изменение записи");
        }
        else
        {
                MessageBox.Show("Не удалось внести изменения", "Изменение записи");
        }
        conn.Close();
    }
   catch(Exception ex)
   {
        MessageBox.Show(ex.ToString());
    }
    finally
    {
        conn.Close();
    }
}
```

Обратим внимание на то, что в блоке finally происходит закрытие соединения, это нужно сделать в любом случае, независимо от результата выполнения команды. Значения, введенные пользователем в текстовые поля txtFamilyUpdate и txtTouristIDUpdate, помещаются в переменные Family и TouristID. В запросе к базе данных используются два параметра – @Family и @TouristID. Они добавляются в коллекцию объекта Command с помощью метода Add свойства Parameters, а затем значения параметров устанавливаются равными переменным Family и TouristID. Метод Add перегружен. Использованный в данном фрагменте кода первый вариант этого метода принимает наибольшее количество свойств. Описание некоторых свойств метода Add приводится в таблице 16.

Конструкция метода Add, свойства Parameters объекта Command для поставщика данных OLE DB имеет в точности такую же структуру.

Таблица 16

Свойства метода Add

Свойство	**Описание**
parameterName	Название параметра
sqlDbType	Тип данных передаваемого параметра
size	Размер параметра
sourceColumn	Название имени столбца объекта DataSet, на который ссылается данный параметр

Добавим обработчик кнопки btnInsert.

```
private void btnInsert_Click(object sender, System.EventArgs e)
{
  try
  {
        int TouristID = int.Parse(this.txtTouristIDInsert.Text);
        string  Family = Convert.ToString(this.txtFamilyInsert.Text);
        string  FirstName = Convert.ToString(this.txtFirstNameInsert.Text);
        string  MiddleName = Convert.ToString(this.txtMiddleNameInsert.Text);

        conn = new SqlConnection();
        conn.ConnectionString = @"Data Source=.\SQLEXPRESS;AttachDbFilename=" +
                        @"D:\ВМИ\For ADO\BDTur_firmSQL2.mdf" +
                        ";Integrated Security=True;Connect Timeout=30;User Instance=True";
        conn.Open();
        SqlCommand myCommand = conn.CreateCommand();
        myCommand.CommandText = "INSERT INTO " +
                        "Туристы ([Код туриста], Фамилия, Имя, Отчество) " +
                        "VALUES (@TouristID, @Family, @FirstName, @MiddleName)";
        myCommand.Parameters.Add("@TouristID", SqlDbType.Int, 4);
        myCommand.Parameters["@TouristID"].Value = TouristID;
        myCommand.Parameters.Add("@Family", SqlDbType.NVarChar, 50);
        myCommand.Parameters["@Family"].Value =  Family;
        myCommand.Parameters.Add("@FirstName", SqlDbType.NVarChar, 50);
        myCommand.Parameters["@FirstName"].Value =  FirstName;
        myCommand.Parameters.Add("@MiddleName", SqlDbType.NVarChar, 50);
        myCommand.Parameters["@MiddleName"].Value =  MiddleName;
        int UspeshnoeIzmenenie = myCommand.ExecuteNonQuery();
        if (UspeshnoeIzmenenie !=0)
        {
                MessageBox.Show("Изменения внесены", "Изменение записи");
        }
        else
        {
```

```
                MessageBox.Show("Не удалось внести изменения", "Изменение записи");
        }
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
    finally
    {
        conn.Close();
    }
}
```

В запросе используются четыре параметра: @TouristID, @Family, @FirstName, @MiddleName. Тип данных создаваемых параметров соответствует типу данных полей таблицы «Туристы» в базе.

Добавим обработчик кнопки btnDelete.

```
private void btnDelete_Click(object sender, System.EventArgs e)
{
    try
    {
        int TouristID = int.Parse(this.txtTouristIDDelete.Text);

        conn = new SqlConnection();
        conn.ConnectionString = @"Data Source=.\SQLEXPRESS;AttachDbFilename=" +
                        @"D:\ВМИ\For ADO\BDTur_firmSQL2.mdf" +
                        ";Integrated Security=True;Connect Timeout=30;User Instance=True";
        conn.Open();

        SqlCommand myCommand = conn.CreateCommand();
        myCommand.CommandText = "DELETE FROM Туристы" +
                        "WHERE [Код туриста] = @TouristID";
        myCommand.Parameters.Add("@TouristID", SqlDbType.Int, 4);
        myCommand.Parameters["@TouristID"].Value = TouristID;

        int UspeshnoeIzmenenie = myCommand.ExecuteNonQuery();
        if (UspeshnoeIzmenenie !=0)
        {
                MessageBox.Show("Изменения внесены", "Изменение записи");
        }
        else
        {
```

```
                MessageBox.Show("Не удалось внести изменения", "Изменение записи");
            }
        }
        catch(Exception ex)
        {
            MessageBox.Show(ex.ToString());
        }
        finally
        {
            conn.Close();
        }
    }
```

Запускаем приложение. В каждой из групп заполняем поля, затем нажимаем на кнопки. Проверять результат можно, запуская Management Studio и просматривая каждый раз содержимое таблицы «Туристы» (рис. 88 – 90).

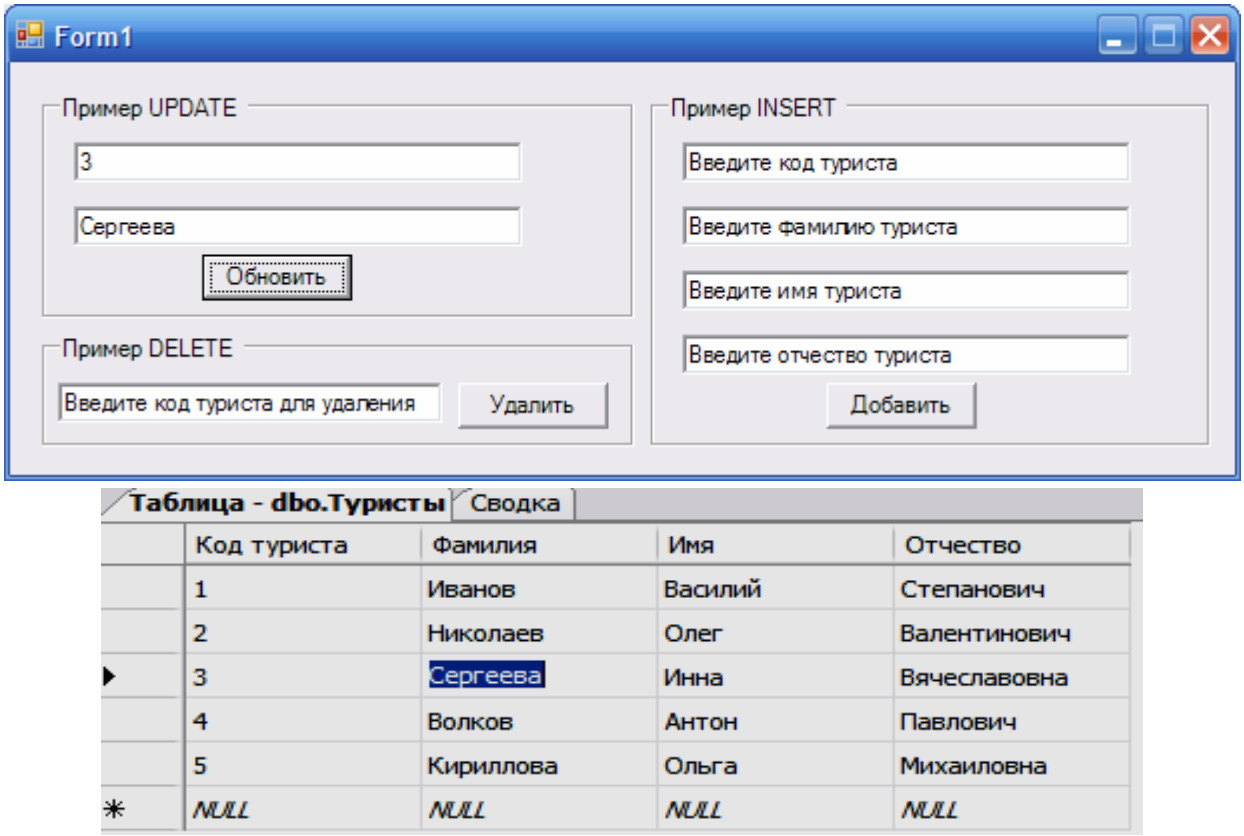


Рис. 88. Пример обновления

Form1

Пример UPDATE

Введите код туриста

Введите фамилию туриста

Обновить

Пример DELETE

4

Удалить

Пример INSERT

Введите код туриста

Введите фамилию туриста

Введите имя туриста

Введите отчество туриста

Добавить

Таблица - dbo.Туристы | Сводка

	Код туриста	Фамилия	Имя	Отчество
▶	1	Иванов	Василий	Степанович
	2	Николаев	Олег	Валентинович
	3	Сергеева	Инна	Вячеславовна
	5	Кириллова	Ольга	Михаиловна
	6	Тихомиров	Андрей	Борисович
*	NULL	NULL	NULL	NULL

Рис. 89. Пример удаления

Form1

Пример UPDATE

Введите код туриста

Введите фамилию туриста

Обновить

Пример DELETE

4

Удалить

Пример INSERT

7

Иванов

Иван

Иванович

Добавить

Таблица - dbo.Туристы | Сводка

	Код туриста	Фамилия	Имя	Отчество
▶	1	Иванов	Василий	Степанович
	2	Николаев	Олег	Валентинович
	3	Сергеева	Инна	Вячеславовна
	5	Кириллова	Ольга	Михаиловна
	6	Тихомиров	Андрей	Борисович
	7	Иванов	Иван	Иванович
*	NULL	NULL	NULL	NULL

Рис. 90. Пример добавления

Использование метода ExecuteScalar

Применять метод ExecuteScalar объекта Command в Windows-приложениях очень легко – достаточно указать элемент управления (текстовое поле, надпись) для вывода одиночного значения.

Использование метода ExecuteReader

Теперь рассмотрим метод ExecuteReader. Одна из главных задач при использовании этого метода – разместить возвращаемый набор данных в элементе управления на форме.

Создайте новое приложение. Расположите на форме элемент ListBox, его свойству Dock установите значение Bottom. Добавим элемент Splitter, свойству Dock которого также установим значение Bottom. Наконец, перетаскиваем элемент ListView, свойству Dock которого устанавливаем значение Fill.

Далее, нужно настроить внешний вид элемента ListView: в окне Properties в поле свойства Columns нажимаем на кнопку ...(...).

В редакторе «Column Header Collection Editor» добавляем следующие четыре элемента:

Name	Text
chTouristID	Код туриста
chFamily	Фамилия
chFirstName	Имя
chMiddleName	Отчество

Для отображения созданных столбцов свойству View элемента устанавливаем значение Details. Также включим режим отображения линий сетки – в свойстве GridLines выбираем значение True. Сделанные изменения немедленно отобразятся на элементе.

Подключаем пространство имен для работы с базой:

```
using System.Data.SqlClient;
```

В классе формы создаем объекты conn и dataReader:

```
SqlConnection    conn = null;
SqlDataReader  dataReader;
```

В конструкторе формы добавляем код для заполнения данными элементов управления:

```
public Form1()
{
  InitializeComponent();
  try
 {
        conn = new SqlConnection();
        conn.ConnectionString = @"Data Source=.\SQLEXPRESS;AttachDbFilename=" +
                        @"D:\ВМИ\For ADO\BDTur_firmSQL2.mdf" +
                        ";Integrated Security=True;Connect Timeout=30;User Instance=True";
```

```
        conn.Open();
        SqlCommand myCommand = conn.CreateCommand();
        myCommand.CommandText = "SELECT * FROM Туристы";
        dataReader = myCommand.ExecuteReader();
        while (dataReader.Read())
        {
        // Создаем переменные, получаем для них  значения из объекта dataReader,
        //используя метод  GetТипДанных
                int TouristID = dataReader.GetInt32(0);
                string Family = dataReader.GetString(1);
                string FirstName = dataReader.GetString(2);
                string MiddleName = dataReader.GetString(3);
        //Выводим данные в элемент listBox1
                listBox1.Items.Add("Код туриста: " + TouristID+ " Фамилия: " + Family +
                                        " Имя: "+ FirstName + " Отчество: " + MiddleName);
        //Создаем экземпляр item класса ListViewItem для записи в него
        //данных из dataReader
                ListViewItem item = new ListViewItem(new
                string[]{Convert.ToString(dataReader[0]), Convert.ToString(dataReader[1]),
                        Convert.ToString(dataReader[2]), Convert.ToString(dataReader[3])});
                listView1.Items.Add(item);
        }
  }
  catch(Exception ex)
  {
        MessageBox.Show(ex.ToString());
  }
  finally
  {
        dataReader.Close();
        conn.Close();
  }
}
```

Метод GetТипДанных позволяет приводить значения, возвращаемые объектом DataReader, если заранее известен их тип данных.

Запускаем приложение. На форму выводятся данные в виде списка в элементе ListBox и в виде таблицы в элементе ListView (рис. 91).

Вывод данных в элемент ListView приводит к достаточно удовлетворительному результату, однако более привычным является вывод в элемент DataGridView. Ранее, при использовании объекта DataSet, источник данных для элемента DataGridView указывался следующим образом:

```
dataGridView1.DataSource = dataset.Tables["Название_таблицы"].DefaultView;
```

Или так:

```
dataGridView1.DataSource = dataset;
```

Объект DataReader не поддерживает аналогичного вывода – мы не можем связать объекты таким простым образом:

```
dataGridView1.DataSource = datareader;
```

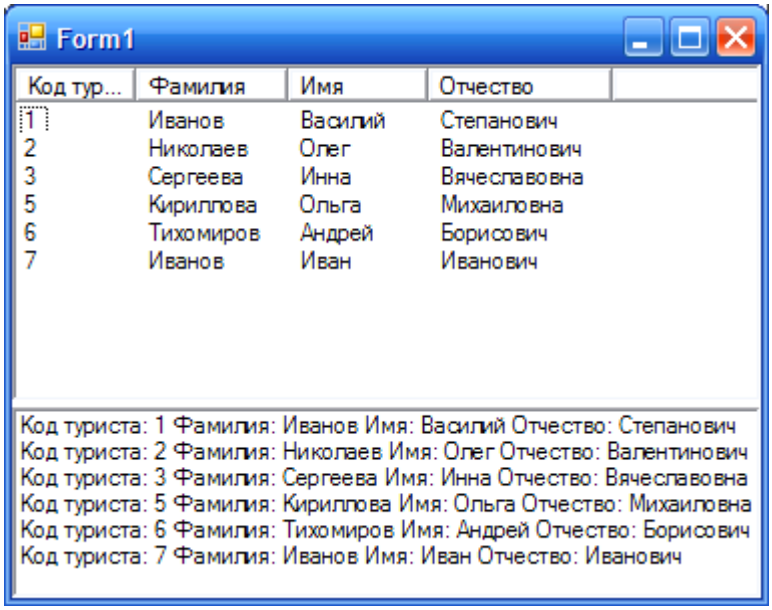


Рис. 91. Использование метода ExecuteReader

Одним из способов вывода является применение дополнительных объектов DataTable. Объект DataTable предназначен для хранения таблицы, полученной из базы данных.

Создадим новое приложение. Перетаскиваем на форму элемент управления DataGrid, его свойству Dock устанавливаем значение Fill.

Подключаем пространство имен для работы с базой:

```
using System.Data.SqlClient;
```

В классе формы создаем следующие объекты:

```
    SqlConnection  conn = null;
//Создаем экземпляр FullDataTable, в который будут помещаться данные
    DataTable FullDataTable = new DataTable();
//Создаем экземпляр FullDataTable для получения структуры таблицы из базы данных
    DataTable ShemaDataTable = new DataTable();
    SqlDataReader dataReader;
    SqlCommand myCommand;
//Создаем объект objectRow для получения информации о числе столбцов
    object[] objectRow;
//Создаем объект myDataRow  для помещения записей
    DataRow myDataRow;
```

Основной код помещаем в конструктор формы:

```
public Form1()
{
  InitializeComponent();
  try
  {
        conn = new SqlConnection();
        conn.ConnectionString = @"Data Source=.\SQLEXPRESS;AttachDbFilename=" +
                        @"D:\ВМИ\For ADO\BDTur_firmSQL2.mdf" +
                        ";Integrated Security=True;Connect Timeout=30;User Instance=True";
        conn.Open();
        myCommand = conn.CreateCommand();
        myCommand.CommandText = "SELECT * FROM Туристы";
        dataReader = myCommand.ExecuteReader();
//Вызываем метод GetSchemaTable, который получает схему таблицы из базы
//и передает ее объекту ShemaDataTable
        ShemaDataTable = dataReader.GetSchemaTable();
//Свойство FieldCount возвращает число столбцов для текущей записей.
//Передаем это значение объекту objectRow
        objectRow = new object[dataReader.FieldCount];
//Определяем структуру объекта FullDataTable
        for(int i =0; i <dataReader.FieldCount; i++)
        {
        FullDataTable.Columns.Add(ShemaDataTable.Rows[i]["ColumnName"].ToString(),
                                ((System.Type)ShemaDataTable.Rows[i]["DataType"]));
        }
//Добавляем записи в объект FullDataTable
        while(dataReader.Read())
        {
                dataReader.GetValues(objectRow);
                myDataRow = FullDataTable.Rows.Add(objectRow);
        }
//Определяем источник данных для элемента dataGrid1
                dataGrid1.DataSource = FullDataTable;
  }
  catch (Exception ex)
  {
        MessageBox.Show(ex.ToString());
  }
  finally
  {
        dataReader.Close();
        conn.Close();
  }
}
```

Запускаем приложение (рис. 92):

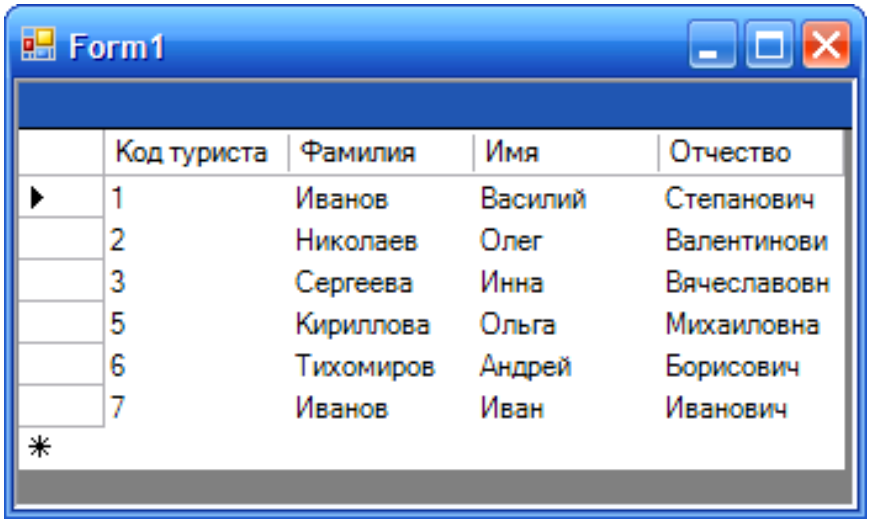


Рис. 92. Демонстрация работы приложения

4.3.4. Вызов хранимых процедур

Хранимые процедуры с входными параметрами

Теперь вернемся к работе с хранимыми процедурами. Выше уже применялись самые простые процедуры (они приводятся в таблице 13), содержимое которых представляло собой, по сути, элементарные запросы на выборку в Windows-приложениях. Применение хранимых процедур с параметрами (таблица 14), как правило, связано с интерфейсом приложения – пользователь имеет возможность вводить значение и затем на основании его получать результат.

Для демонстрации программной работы с хранимыми процедурами создадим новое Windows-приложение, главное окно которого представлено на рис. 93.

В классе формы создаем строку подключения:

```
string connectionString = @"Data Source=.\SQLEXPRESS;AttachDbFilename=" +
                          @"D:\ВМИ\For ADO\BDTur_firmSQL2.mdf" +
                          ";Integrated Security=True;Connect Timeout=30;User Instance=True";
```

В каждом из обработчиков кнопок создаем объекты Connection и Command, определяем их свойства, для последнего добавляем нужные параметры в набор Parameters:

```
private void btnRun_p1_Click(object sender, System.EventArgs e)     {
  SqlConnection  conn = new SqlConnection();
  conn.ConnectionString = connectionString;
  SqlCommand myCommand  = conn.CreateCommand();
  myCommand.CommandType = CommandType.StoredProcedure;
  myCommand.CommandText = "[proc_p1]";
  string  FamilyParameter  = Convert.ToString(txtFamily_p1.Text);
```

```
  myCommand.Parameters.Add("@Фамилия", SqlDbType.NVarChar, 50);
  myCommand.Parameters["@Фамилия"].Value = FamilyParameter;
  conn.Open();
  SqlDataReader dataReader = myCommand.ExecuteReader();
  while (dataReader.Read())
 {
// Создаем переменные, получаем для них  значения из объекта dataReader,
//используя метод  GetТипДанных
        int TouristID = dataReader.GetInt32(0);
        string Family = dataReader.GetString(1);
        string FirstName = dataReader.GetString(2);
        string MiddleName = dataReader.GetString(3);
//Выводим данные в элемент lbResult_p1
        lbResult_p1.Items.Add("Код туриста: " + TouristID+ " Фамилия: " + Family +
                        " Имя: "+ FirstName + " Отчество: " + MiddleName);
  }
  conn.Close();
}  // btnRun_p1_Click
private void btnRun_p5_Click(object sender, System.EventArgs e)
{
  SqlConnection  conn = new SqlConnection();
  conn.ConnectionString = connectionString;
  SqlCommand myCommand  = conn.CreateCommand();
  myCommand.CommandType = CommandType.StoredProcedure;
  myCommand.CommandText = "[proc_p5]";
  string  NameTourParameter  = Convert.ToString(txtNameTour_p5.Text);
  double KursParameter = double.Parse(this.txtKurs_p5.Text);
  myCommand.Parameters.Add("@nameTour", SqlDbType.NVarChar, 50);
  myCommand.Parameters["@nameTour"].Value = NameTourParameter;
  myCommand.Parameters.Add("@Курс", SqlDbType.Float, 8);
  myCommand.Parameters["@Курс"].Value = KursParameter;
  conn.Open();
  int UspeshnoeIzmenenie = myCommand.ExecuteNonQuery();
  if (UspeshnoeIzmenenie !=0)
  {
        MessageBox.Show("Изменения внесены", "Изменение записи");
  }
  else
 {
        MessageBox.Show("Не удалось внести изменения", "Изменение записи");
  }

  conn.Close();
}
```

```
private void btnRun_proc6_Click(object sender, System.EventArgs e)
{
  SqlConnection  conn = new SqlConnection();
  conn.ConnectionString = connectionString;
  SqlCommand myCommand  = conn.CreateCommand();
  myCommand.CommandType = CommandType.StoredProcedure;
  myCommand.CommandText = "[proc6]";
  conn.Open();
  string MaxPrice = Convert.ToString(myCommand.ExecuteScalar());
  lblPrice_proc6.Text = MaxPrice;
  conn.Close();
}
```

Результаты выполнения данных обработчиков представлены на рисунках 93–95.

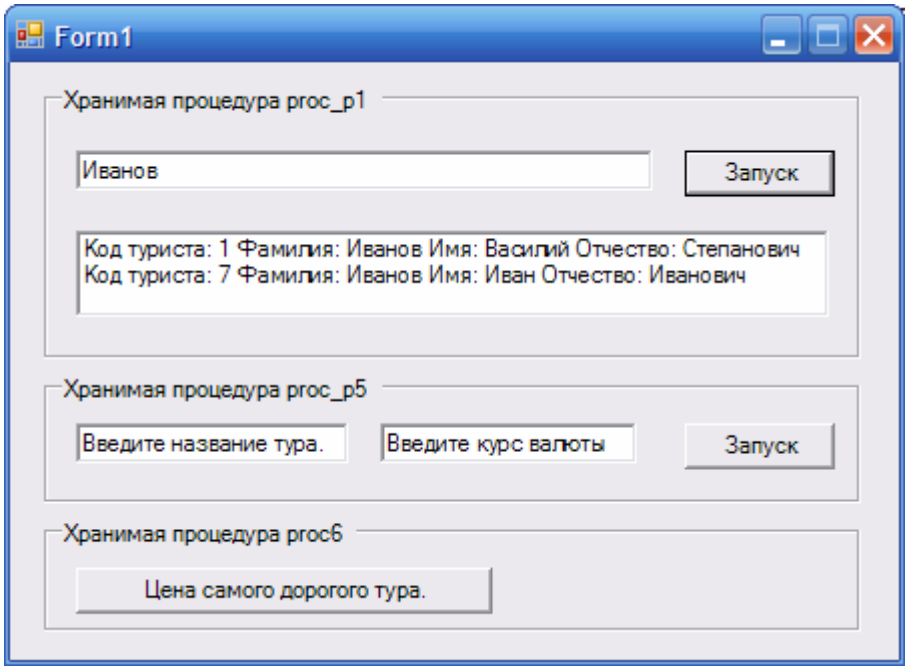


Рис. 93. Вызов хранимой процедуры proc_p1

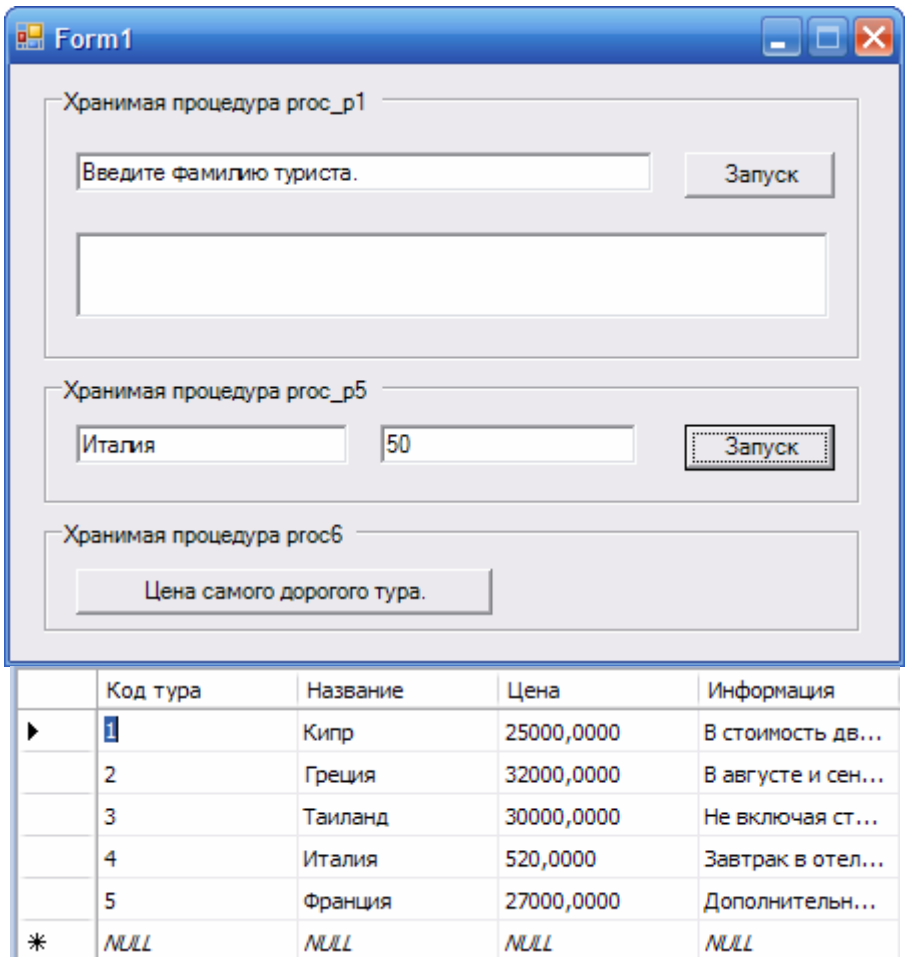


Рис. 94. Вызов хранимой процедуры proc_p5

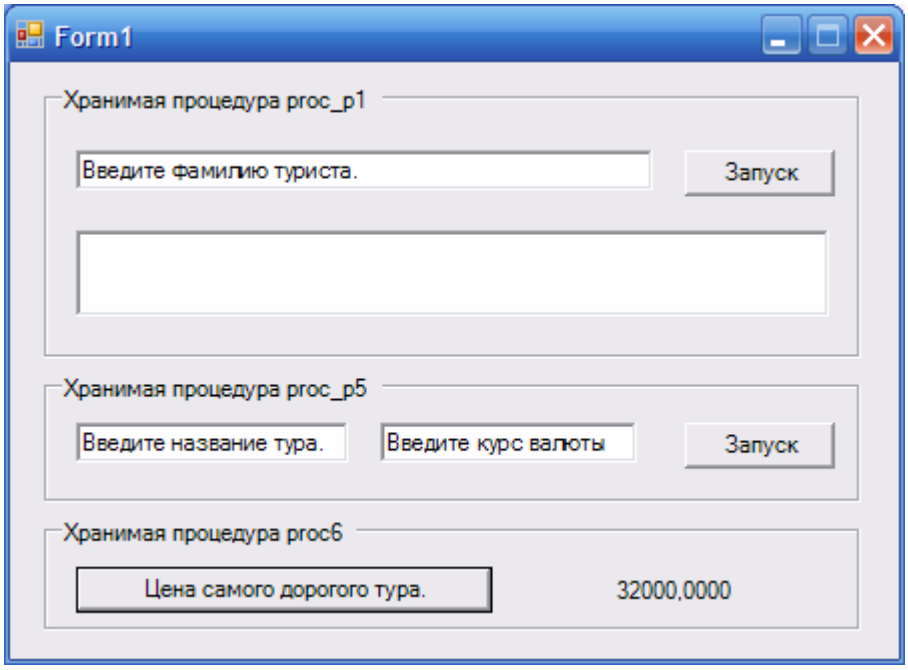


Рис. 95. Вызов хранимой процедуры proc_p6

Хранимые процедуры с входными и выходными параметрами

На практике наиболее часто используются хранимые процедуры с входными и выходными параметрами (см. таблицу 15). Создадим новое Windows-приложение, вид его формы в режиме дизайна показан на рисунке 96.

Рис. 96. Вид формы в режиме дизайна

В классе формы создаем объект Connection:

```
SqlConnection  conn = null;
```

Обработчик кнопки btnRun принимает следующий вид:

```
private void btnRun_Click(object sender, System.EventArgs e)
{
  try
  {
  conn = new SqlConnection();
  conn.ConnectionString = @"Data Source=.\SQLEXPRESS;AttachDbFilename=" +
                          @"D:\ВМИ\For ADO\BDTur_firmSQL2.mdf" +
                          ";Integrated Security=True;Connect Timeout=30;User Instance=True";
  SqlCommand myCommand  = conn.CreateCommand();
  myCommand.CommandType = CommandType.StoredProcedure;
  myCommand.CommandText = "[proc_po1]";
  int TouristID = int.Parse(this.txtTouristID.Text);
  myCommand.Parameters.Add("@TouristID", SqlDbType.Int, 4);
  myCommand.Parameters["@TouristID"].Value = TouristID;
//Необязательная строка, т.к. совпадает со значением по умолчанию.
//myCommand.Parameters["@TouristID"].Direction = ParameterDirection.Input;
  myCommand.Parameters.Add("@LastName", SqlDbType.NVarChar, 60);
  myCommand.Parameters["@LastName"].Direction = ParameterDirection.Output;
  conn.Open();
  myCommand.ExecuteScalar();
  lblFamily.Text = Convert.ToString(myCommand.Parameters["@LastName"].Value);
  }
  catch (Exception ex)
  {     MessageBox.Show(ex.ToString());   }
  finally
  {     conn.Close();   }
}
```

Параметр @LastName был добавлен в набор Parameters, причем его значение output указано в свойстве Direction:

```
myCommand.Parameters["@LastName"].Direction = ParameterDirection.Output;
```

Параметр @TouristID является исходным, поэтому для него в свойстве Direction указывается Input. Поскольку это является значением по умолчанию для всех параметров набора Parameters, указывать его явно не нужно. Для параметров, работающих в двустороннем режиме, устанавливается значение InputOutput, для параметров, возвращающих данные о выполнения хранимой процедуры, – ReturnValue.

Результат выполнения приложения показан на рисунке 97.

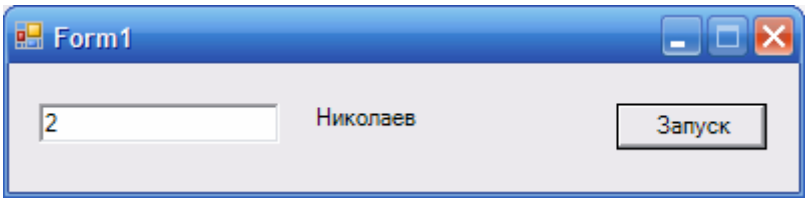


Рис. 97. Результат выполнения приложения

Хранимые процедуры из нескольких SQL-конструкций

Хранимая процедура может содержать несколько SQL-конструкций, определяющих работу приложения. При ее вызове возникает задача распределения данных, получаемых от разных конструкций. Создадим новую хранимую процедуру proc_NextResult. Процедура proc_NextResult будет состоять из двух конструкций: первая будет возвращать содержимое таблицы «Туристы», а вторая – содержимое таблицы «Туры»:

```
CREATE PROCEDURE proc_NextResult
AS
   SET NOCOUNT ON
  SELECT * FROM Туристы
  SELECT * FROM Туры
  RETURN
```

После сохранения процедуры создадим новое Windows-приложение.

Расположим на форме элемент управления ListBox, его свойству Dock установим значение Bottom. Добавим элемент Splitter (разделитель), свойству Dock которого также установим значение Bottom. Наконец, добавим еще один элемент ListBox, свойству Dock которого устанавливаем значение Fill.

Поставим следующую задачу: в первый элемент ListBox вывести несколько произвольных столбцов таблицы «Туры», а во второй – несколько столбцов таблицы «Туристы».

Конструктор формы примет следующий вид:

```
public Form1()
{
  InitializeComponent();
  SqlConnection conn = new SqlConnection();
  conn.ConnectionString = @"Data Source=.\SQLEXPRESS;AttachDbFilename=" +
                          @"D:\ВМИ\For ADO\BDTur_firmSQL2.mdf" +
                          ";Integrated Security=True;Connect Timeout=30;User Instance=True";
  SqlCommand myCommand  = conn.CreateCommand();
  myCommand.CommandType = CommandType.StoredProcedure;
  myCommand.CommandText = "[proc_NextResult]";
  conn.Open();
  SqlDataReader dataReader = myCommand.ExecuteReader();
  while(dataReader.Read())
  {
        listBox2.Items.Add(dataReader.GetString(1)+" "+dataReader.GetString(2));
  }
  dataReader.NextResult();
  while(dataReader.Read())
  {
        listBox1.Items.Add(dataReader.GetString(1) +
                ". Дополнительная информация:  "+dataReader.GetString(3));
  }
  dataReader.Close();
  conn.Close();
}
```

Результат выполнения данного приложения приведен на рисунке 98.

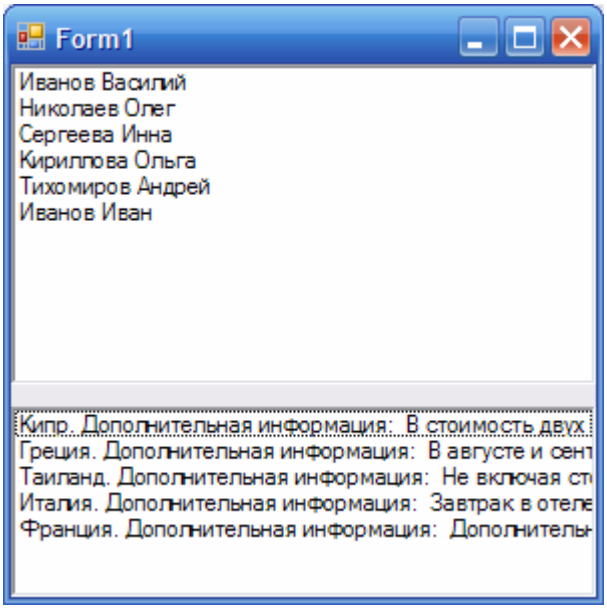


Рис. 98. Результат выполнения приложения

4.3.5. Транзакции

Транзакцией называется выполнение последовательности команд (SQL-конструкций) в базе данных, которая либо фиксируется при успешной реализации каждой команды, либо отменяется при неудачном выполнении хотя бы одной команды. Большинство современных СУБД поддерживают механизм транзакций, и подавляющее большинство клиентских приложений, работающих с ними, используют для выполнения своих команд транзакции.

Возникает вопрос – зачем нужны транзакции? Представим себе, что в базу данных BDTur_firm2 требуется вставить связанные записи в две таблицы – «Туристы» и «Информация о туристах». Если запись, вставляемая в таблицу «Туристы», окажется неверной, например, из-за неправильно указанного кода туриста, база данных не позволит внести изменения, но при этом в таблице «Информация о туристах» появится ненужная запись. Рассмотрим такую ситуацию на примере.

Запустим Management Studio, в новом бланке введем запрос для добавления двух записей:

```
INSERT INTO Туристы ([Код туриста], Фамилия, Имя, Отчество)
VALUES (8, 'Тихомиров', 'Андрей', 'Борисович');
INSERT INTO [Информация о туристах]([Код туриста], [Серия паспорта], Город, Страна,
Телефон, Индекс)
VALUES (8, 'СА 1234567', 'Новосибирск', 'Россия', 1234567, 996548);
```

Две записи успешно добавляются в базу данных:

```
(1 row(s) affected)    //или   (строк обработано: 1)

(1 row(s) affected)    //или   (строк обработано: 1)
```

Теперь спровоцируем ошибку – изменим код туриста только во втором запросе:

```
INSERT INTO Туристы ([Код туриста], Фамилия, Имя, Отчество)
VALUES (8, 'Тихомиров', 'Андрей', 'Борисович');
INSERT INTO [Информация о туристах]([Код туриста], [Серия паспорта], Город, Страна,
Телефон, Индекс)
VALUES (9, 'СА 1234567', 'Новосибирск', 'Россия', 1234567, 996548);
```

Появляется сообщение о невозможности вставки первой записи с уже имеющимся значением ключевого поля. Вторая запись, однако, добавляется в таблицу:

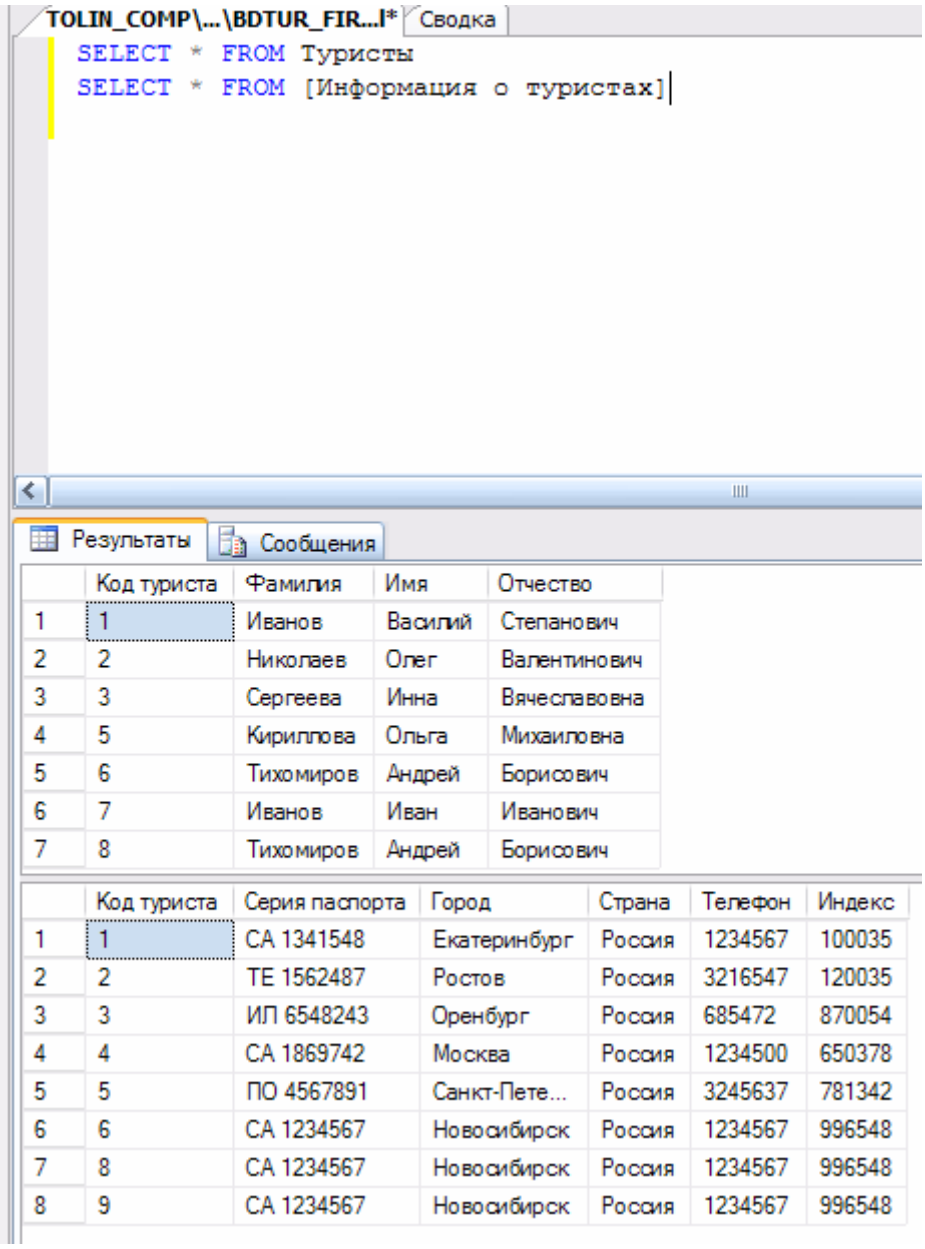
```
Сообщение 2627, уровень 14, состояние 1, строка 1
Violation of PRIMARY KEY constraint 'PK_Туристы'. Cannot insert duplicate key in object
'dbo.Туристы'.
The statement has been terminated.

(строк обработано: 1)
```

Извлечем содержимое обеих таблиц следующим двойным запросом:

```
SELECT * FROM Туристы
SELECT * FROM [Информация о туристах]
```

В таблице «Информация о туристах» последняя запись добавилась безо всякой связи с записью таблицы «Туристы» (рис. 99). Для того чтобы избегать подобных ошибок, нужно применить транзакцию.



	Код туриста	Фамилия	Имя	Отчество
1	1	Иванов	Василий	Степанович
2	2	Николаев	Олег	Валентинович
3	3	Сергеева	Инна	Вячеславовна
4	5	Кириллова	Ольга	Михаиловна
5	6	Тихомиров	Андрей	Борисович
6	7	Иванов	Иван	Иванович
7	8	Тихомиров	Андрей	Борисович

	Код туриста	Серия паспорта	Город	Страна	Телефон	Индекс
1	1	СА 1341548	Екатеринбург	Россия	1234567	100035
2	2	ТЕ 1562487	Ростов	Россия	3216547	120035
3	3	ИЛ 6548243	Оренбург	Россия	685472	870054
4	4	СА 1869742	Москва	Россия	1234500	650378
5	5	ПО 4567891	Санкт-Пете...	Россия	3245637	781342
6	6	СА 1234567	Новосибирск	Россия	1234567	996548
7	8	СА 1234567	Новосибирск	Россия	1234567	996548
8	9	СА 1234567	Новосибирск	Россия	1234567	996548

Рис. 99. Содержимое таблиц «Туристы» и «Информация о туристах» – нарушение связи

Удалим все внесенные записи из обеих таблиц и оформим исходные SQL-конструкции в виде транзакции:

```
BEGIN TRAN
DECLARE @OshibkiTabliciTourists int, @OshibkiTabliciInfoTourists int
INSERT INTO Туристы ([Код туриста], Фамилия, Имя, Отчество)
VALUES (8, 'Тихомиров', 'Андрей', 'Борисович');
SELECT @OshibkiTabliciTourists=@@ERROR
INSERT INTO [Информация о туристах]([Код туриста], [Серия паспорта],
 Город, Страна, Телефон, Индекс)
VALUES (8, 'СА 1234567', 'Новосибирск', 'Россия', 1234567, 996548);
SELECT @OshibkiTabliciInfoTourists=@@ERROR
IF @OshibkiTabliciTourists=0 AND @OshibkiTabliciInfoTourists=0
COMMIT TRAN
ELSE
ROLLBACK TRAN
```

Начало транзакции объявляется с помощью команды BEGIN TRAN. Далее создаются два параметра – @OshibkiTabliciTourists, @OshibkiTabliciInfoTourists для сбора ошибок. После первого запроса возвращаем значение, которое встроенная функция @@ERROR присваивает первому параметру:

```
SELECT @OshibkiTabliciTourists=@@ERROR
```

То же самое делаем после второго запроса для другого параметра:

```
SELECT @OshibkiTabliciInfoTourists=@@ERROR
```

Проверяем значения обоих параметров, которые должны быть равными нулю при отсутствии ошибок:

```
IF @OshibkiTabliciTourists=0 AND @OshibkiTabliciInfoTourists=0
```

В этом случае подтверждаем транзакцию (в данном случае внесение изменений) при помощи команды COMMIT TRAN. В противном случае – если значение хотя бы одного из параметров @OshibkiTabliciTourists и @OshibkiTabliciInfoTourists оказывается отличным от нуля, отменяем транзакцию при помощи команды ROLLBACK TRAN.

После выполнения транзакции появляется сообщение о добавлении двух строк:

```
(строк обработано: 1)

(строк обработано: 1)
```

Снова изменим код туриста во втором запросе:

```
BEGIN TRAN
DECLARE @OshibkiTabliciTourists int, @OshibkiTabliciInfoTourists int
INSERT INTO Туристы ([Код туриста], Фамилия, Имя, Отчество)
VALUES (8, 'Тихомиров', 'Андрей', 'Борисович');
SELECT @OshibkiTabliciTourists=@@ERROR
INSERT INTO [Информация о туристах]([Код туриста], [Серия паспорта],
 Город, Страна, Телефон, Индекс)
```

```
VALUES (9, 'CA 1234567', 'Новосибирск', 'Россия', 1234567, 996548);
SELECT @OshibkiTabliciInfoTourists=@@ERROR
IF @OshibkiTabliciTourists=0 AND @OshibkiTabliciInfoTourists=0
COMMIT TRAN
ELSE
ROLLBACK TRAN
```

Запускаем транзакцию – появляется в точности такое же сообщение, что и в случае применения обычных запросов:

```
Сообщение 2627, уровень 14, состояние 1, строка 3
Violation of PRIMARY KEY constraint 'PK_Туристы'. Cannot insert duplicate key in object
'dbo.Туристы'.
The statement has been terminated.

(строк обработано: 1)
```

Однако теперь изменения не были внесены во вторую таблицу (рис. 100).

Результаты | Сообщения

	Код туриста	Фамилия	Имя	Отчество
1	1	Иванов	Василий	Степанович
2	2	Николаев	Олег	Валентинович
3	3	Сергеева	Инна	Вячеславовна
4	5	Кириллова	Ольга	Михаиловна
5	6	Тихомиров	Андрей	Борисович
6	7	Иванов	Иван	Иванович
7	8	Тихомиров	Андрей	Борисович

	Код туриста	Серия паспорта	Город	Страна	Телефон	Индекс
1	1	СА 1341548	Екатеринбург	Россия	1234567	100035
2	2	ТЕ 1562487	Ростов	Россия	3216547	120035
3	3	ИЛ 6548243	Оренбург	Россия	685472	870054
4	4	СА 1869742	Москва	Россия	1234500	650378
5	5	ПО 4567891	Санкт-Пете...	Россия	3245637	781342
6	6	СА 1234567	Новосибирск	Россия	1234567	996548
7	8	СА 1234567	Новосибирск	Россия	1234567	996548

Рис. 100. Содержимое таблиц «Туристы» и «Информация о туристах» после выполнения неудачной транзакции

Сообщение (1 row(s) affected), указывающее на «добавление» одной записи, в данном случае оно всего лишь означает, что вторая SQL-конструкция была верной, и запись могла быть добавлена в случае успешного выполнения транзакции.

Таким образом, механизм транзакций поддерживает целостность данных двух таблиц, не позволяя ее нарушить добавлением неверных данных.

Транзакции в ADO .NET

Перейдем теперь к рассмотрению транзакций в ADO .NET. Создадим новое консольное приложение EasyTransaction. Поставим задачу: передать те же самые данные в две таблицы – «Туристы» и «Информация о туристах».

Полный листинг данного приложения выглядит следующим образом:

```
using System;
using System.Data.SqlClient;

namespace EasyTransaction
{
  class Class1
  {
        [STAThread]
        static void Main(string[] args)
  {
//Создаем соединение
  SqlConnection  conn = new SqlConnection();
  conn.ConnectionString = @"Data Source=.\SQLEXPRESS;AttachDbFilename=" +
                        @"D:\ВМИ\For ADO\BDTur_firmSQL2.mdf" +
                        ";Integrated Security=True;Connect Timeout=30;User Instance=True";
  conn.Open();
  SqlCommand myCommand = conn.CreateCommand();
//Создаем транзакцию
 myCommand.Transaction = conn.BeginTransaction(System.Data.IsolationLevel.Serializable);
 try
 {
  myCommand.CommandText =
        "INSERT INTO Туристы ([Код туриста], Фамилия, Имя, Отчество) " +
        "VALUES (9, 'Тихомиров', 'Андрей', 'Борисович')";
  myCommand.ExecuteNonQuery();
  myCommand.CommandText =
        "INSERT INTO [Информация о туристах]" +
              " ([Код туриста], [Серия паспорта], Город, Страна, Телефон, Индекс) " +
              "VALUES (9, 'СА 1234567', 'Новосибирск', 'Россия', 1234567, 996548)";
  myCommand.ExecuteNonQuery();
//Подтверждаем транзакцию
  myCommand.Transaction.Commit();
  Console.WriteLine("Передача данных успешно завершена");
  }
  catch(Exception ex)
 {
```

```
//Отклоняем транзакцию
        myCommand.Transaction.Rollback();
        Console.WriteLine("При передаче данных произошла ошибка: "+ ex.Message);

  }
  finally
 {
        conn.Close();
  }

 } end Main
} end Class
} end namespace
```

Перед запуском приложения снова удаляем все добавленные записи из таблиц. При успешном выполнении запроса появляется соответствующее сообщение, а в таблицы добавляются записи (рис. 101).

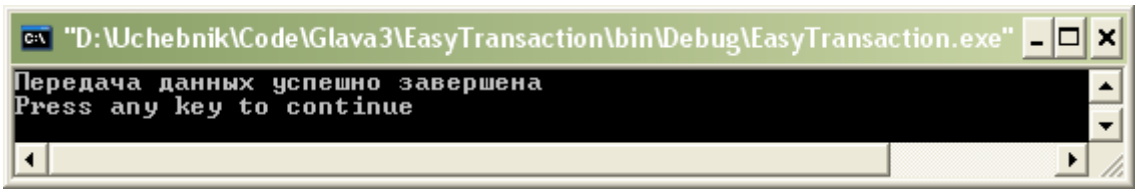


Рис. 101. Приложение EasyTransaction. Транзакция выполнена

Повторный запуск этого приложения приводит к отклонению транзакции – нельзя вставлять записи с одинаковыми значениями первичных ключей (рис. 102).

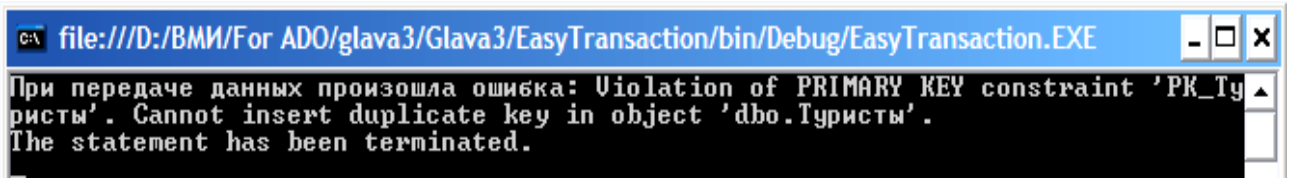


Рис. 102. Приложение EasyTransaction. Транзакция отклонена

В виде транзакции можно оформлять выполнение одной или нескольких хранимых процедур, – в самом деле, общая конструкция имеет следующий вид:

```
//Создаем соединение
… см. в примере приложения EasyTransaction

//Создаем транзакцию
myCommand.Transaction = conn.BeginTransaction();
```

```
try
  {
  //Выполняем команды, вызываем одну или несколько хранимых процедур
  //Подтверждаем транзакцию
    myCommand.Transaction.Commit();
  }
  catch(Exception ex)
  {
    //Отклоняем транзакцию
    myCommand.Transaction.Rollback();
  }
  finally
  {
    //Закрываем соединение
    conn.Close();
  }
```

При выполнении транзакций несколькими пользователями одной базы данных могут возникать следующие проблемы:

- Dirty reads – «грязное» чтение. Первый пользователь начинает транзакцию, изменяющую данные. В это время другой пользователь (или создаваемая им транзакция) извлекает частично измененные данные, которые не являются корректными.
- Non-repeatable reads – неповторяемое чтение. Первый пользователь начинает транзакцию, изменяющую данные. В это время другой пользователь начинает и завершает другую транзакцию. Первый пользователь при повторном чтении данных (например, если в его транзакцию входит несколько инструкций SELECT) получает другой набор записей.
- Phantom reads – чтение фантомов. Первый пользователь начинает транзакцию, выбирающую данные из таблицы. В это время другой пользователь начинает и завершает транзакцию, вставляющую или удаляющую записи. Первый пользователь получит другой набор данных, содержащий фантомы – удаленные или измененные строки.

Для решения этих проблем разработаны четыре уровня изоляции транзакции:

- Read uncommitted. Транзакция может считывать данные, с которыми работают другие транзакции. Применение этого уровня изоляции может привести ко всем перечисленным проблемам.
- Read committed. Транзакция не может считывать данные, с которыми работают другие транзакции. Применение этого уровня изоляции исключает проблему «грязного» чтения.

- Repeatable read. Транзакция не может считывать данные, с которыми работают другие транзакции. Другие транзакции также не могут считывать данные, с которыми работает эта транзакция. Применение этого уровня изоляции исключает все проблемы, кроме чтения фантомов.
- Serializable. Транзакция полностью изолирована от других транзакций. Применение этого уровня изоляции полностью исключает все проблемы.

По умолчанию установлен уровень Read committed. В справке Microsoft SQL Server 2005[5] приводится таблица, иллюстрирующая различные уровни изоляции (рис. 103).

Isolation level	Dirty read	Nonrepeatable read	Phantom
Read uncommitted	Yes	Yes	Yes
Read committed	No	Yes	Yes
Repeatable read	No	No	Yes
Serializable	No	No	No

Рис. 103. Уровни изоляции Microsoft SQL Server 2005

Использование наибольшего уровня изоляции (Serializable) означает наибольшую безопасность и вместе с тем наименьшую производительность – все транзакции выполняются в виде серии, последующая вынуждена ждать завершения предыдущей. И наоборот, применение наименьшего уровня (Read uncommitted) означает максимальную производительность и полное отсутствие безопасности. Впрочем, нельзя дать универсальных рекомендаций по применению этих уровней – в каждой конкретной ситуации решение будет зависеть от структуры базы данных и характера выполняемых запросов.

Для установки уровня изоляции применяется следующая команда:

```
SET TRANSACTION ISOLATION LEVEL
READ UNCOMMITTED
```

или READ COMMITTED

или REPEATABLE READ

или SERIALIZABLE

Например, в транзакции, добавляющей две записи, уровень изоляции указывается следующим образом:

```
BEGIN TRAN SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
DECLARE @OshibkiTabliciTourists int, @OshibkiTabliciInfoTourists int
...
ROLLBACK TRAN
```

[5] на вкладке «Указатель» нужно ввести «isolation levels» и затем выбрать заголовок «overview»

В ADO .NET уровень изоляции можно установить при создании транзакции:

```
myCommand.Transaction = conn.BeginTransaction(System.Data.IsolationLevel.Serializable);
```

Дополнительно поддерживаются еще два уровня (рис. 104):

- Chaos. Транзакция не может перезаписать другие непринятые транзакции с большим уровнем изоляции, но может перезаписать изменения, внесенные без использования транзакций. Данные, с которыми работает текущая транзакция, не блокируются;
- Unspecified. Отдельный уровень изоляции, который может применяться, но не может быть определен. Транзакция с этим уровнем может применяться для задания собственного уровня изоляции.

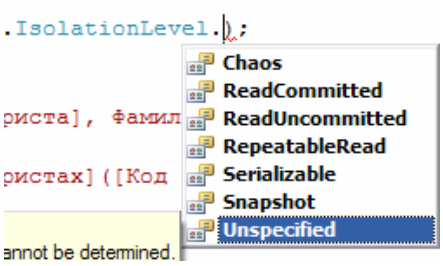


Рис. 104. Определение уровня транзакции

Транзакции обеспечивают целостность базы данных, при разработке многоуровневых приложений их применение является обязательным правилом.

4.4. Работа с таблицами данных

4.4.1. Объекты DataSet, DataTable и DataColumn

Объект DataSet представляет собой буфер для хранения данных из базы. Этот буфер предназначен для хранения структурированной информации, представленной в виде таблиц, поэтому первым, самым очевидным вложенным объектом DataSet является DataTable. Внутри одного объекта DataSet может храниться несколько загруженных таблиц из базы данных, помещенных в соответствующие объекты DataTable.

Всякая таблица состоит из столбцов (называемых также полями или колонками) и строк (записей). Для обращения к ним и для управления столбцами и строками в объекте DataTable предназначены специальные объекты – DataColumn и DataRow. Между таблицами могут быть связи – здесь они представлены объектом DataRelation. Наконец, в таблицах есть первичные и вторичные ключи – объект Constraint со своими двумя подклассами UniqueConstraint и ForeighKeyConstraint описывают их. Отметим, что объекты программы и соответствующие объекты базы данных не эквивалентны.

В загруженных таблицах не формируются автоматически все нужные объекты, в некоторых случаях необходимо сделать это самостоятельно. Сами объекты имеют также довольно тонкую и сложную структуру, поэтому это было бы довольно грубым приближением. Однако, на первых порах, для понимания сути полезно помнить следующие соотношения:

DataSet = <одна или несколько таблиц> = <один или несколько объектов DataTable>.
DataTable = <таблица>.

DataTable = <таблица> = <множество полей, столбцов, колонок> =
= <множество объектов DataColumn>.
DataTable = <таблица> = <множество строк> = <множество объектов DataRow>.
DataColumn = <столбец, поле, колонка>.
DataRow = <строка>.
DataRelation = <связь между таблицами>.

Возникает вопрос: для чего нужны эти объекты, если можно обходились и без них для вывода содержимого таблицы, например в элемент DataGridView? Дело в том, что для простого отображения информации создавать эти объекты не требуется, но в этом случае все данные будут однородными текстовыми переменными, подобно таблицам в документе Microsoft Word. DataSet не может сам сформировать структуру данных – тип переменных, первичные и вторичные ключи, связи между таблицами. Для управления такой структурой, для ее адекватного отображения (например, вывод информации с привязкой к элементам, создаваемым в режиме работы приложения) и нужно определение этих объектов.

Программное создание объектов DataTable и DataColumn

Как было сказано выше, все объекты ADO можно создать программно. Например, создание таблицы и столбцов:

```
DataSet dsTests = new System.Data.DataSet();
//
// dsTests
//
dsTests.DataSetName = "NewDataSet";
dsTests.Locale = new System.Globalization.CultureInfo("ru–RU");
```

Отметим, что в последней строке указывается информация о национальных настройках таблицы: ru–RU. Она необходима в том случае, если разрабатываемое приложение будет использоваться вне страны, в которой оно было разработано (в данном случае – России).

Создадим теперь объект DataTable для таблицы Questions (вопросы):

```
DataTable dtQuestions = dsTests.Tables.Add("Questions");
//Или
//DataTable dtQuestions = new DataTable("Questions");
//dsTests.Tables.Add(dtQuestions);
```

Здесь создается экземпляр dtQuestions объекта DataTable, затем вызывается метод Add свойства Tables объекта dsTests, которому передается название таблицы Questions. Далее создаем поля в объекте dtQuestions:

```
DataColumn dcQuestID = dtQuestions.Columns.Add("questID", typeof(Int32));
dcQuestID.Unique = true;
DataColumn dcQuestion = dtQuestions.Columns.Add("question");
DataColumn dcQuestType = dtQuestions.Columns.Add("questType", typeof(Int32));
```

Мы создаем поля, нужные для отражения соответствующих столбцов в таблице Questions. Перегруженный метод Add свойства Columns объекта dtQuestions позволяет задавать название столбца и его тип данных (рис. 105).

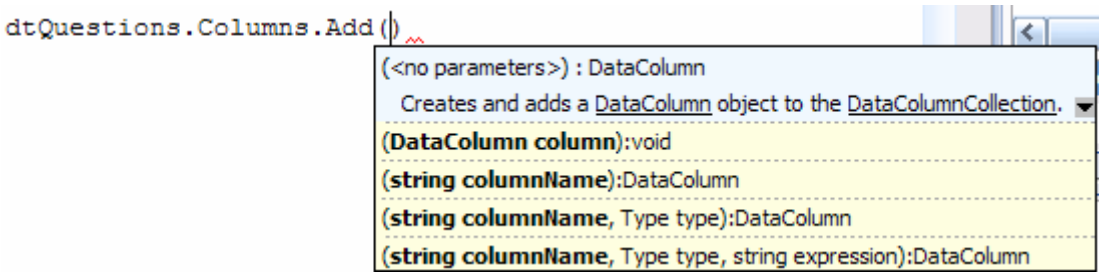


Рис. 105. Создание поля

Свойство Unique указывает, что в этом поле не должно быть повторяющихся значений, оно должно быть уникальным (здесь – поле questID является первичным ключом таблицы Questions).

Точно так же создаем объект DataTable для таблицы Variants (варианты ответов) и соответствующие поля:

```
//Создаем таблицу "Variants"
DataTable dtVariants = dsTests.Tables.Add("Variants");

//Заполняем поля таблицы "Variants"
DataColumn dcID = dtVariants.Columns.Add("id", typeof(Int32));
dcID.Unique = true;
dcID.AutoIncrement = true;
DataColumn dcVariantQuestID = dtVariants.Columns.Add("questID", typeof(Int32));
DataColumn dcVariant = dtVariants.Columns.Add("variant");
DataColumn dcIsRight = dtVariants.Columns.Add("isRight", typeof(Boolean));
```

Здесь мы дополнительно установили свойству AutoIncrement объекта dcID значение true. Свойство AutoIncrement (счетчик) позволяет создать счетчик для поля, аналогичный типу данных «Счетчик» в Microsoft Access.

Теперь приступим к созданию связи между таблицами. В базе данных между родительской таблицей Questions и дочерней Variants была бы установлена связь по полю questID, которое присутствует в обеих таблицах. При программном создании объектов для поля questID таблицы Questions был создан объект dcQuestID, для этого же поля таблицы Variants создан объект dcVariantQuestID. В коде создание связи между таблицами будет иметь следующий вид:

```
DataRelation drQuestionsVariants = new DataRelation("QuestionsVariants",
                                                    dcQuestID, dcVariantQuestID);
dsTests.Relations.Add(drQuestionsVariants);
```

Здесь в конструкторе объекта DataRelation имя drQuestionsVariants – название экземпляра объекта (класса) DataRelation, а строка "QuestionsVariants" – передаваемое конструктору свойство relationName – название связи, которая будет содержаться в создаваемом объекте drQuestionsVariants. Другими словами, drQuestionsVariants – название экземпляра DataRelation, которое будем использовать в коде, а свойство relationName – всего лишь название отражаемой связи, которую можно удалить или переименовать.

Таким образом, можно программно создавать все объекты для отображения таблиц, полей и даже связей между таблицами.

Свойство PrimaryKey

Мы рассмотрели способ конструирования структуры таблицы в объекте DataSet, а также как определять отношения между таблицами. Во всех случаях для выделения первичного ключа в таблице использовалось свойство Unique. Например, первичный ключ «Код туриста» для таблицы «Туристы» определялся так:

```
DataColumn dcTouristID = new DataColumn("Код туриста", typeof(int));
dcTouristID.Unique = true;
```

А для таблицы вариантов ответов «Variants»:

```
DataColumn dcID = dtVariants.Columns.Add("id", typeof(Int32));
dcID.Unique = true;
dcID.AutoIncrement = true;
```

Для вывода таблиц идентификации записей этого определения вполне хватает. Однако свойство Unique всего лишь указывает на уникальность заданного поля, т. е. на отсутствие повторяющихся записей.

В самом деле, в таблице может быть несколько полей, которые должны быть уникальными, и одно из них (или их комбинация) будут образовывать первичный ключ. Для указания именно первичного ключа используется свойство PrimaryKey объекта DataTable:

```
DataTable dtTourists = new DataTable("Туристы");
DataColumn dcTouristID = new DataColumn("Код туриста", typeof(int));
dtTourists.PrimaryKey = new DataColumn [] {dtTourists.Columns["Код туриста"]};
```

В сокращенной записи определение будет такое:

```
dtTourists.Columns.Add("Код туриста", typeof(int));
dtTourists.PrimaryKey = new DataColumn [] {dtTourists.Columns["Код туриста"]};
```

Можно применять комбинацию полей для задания первичного ключа:

```
DataTable dtTourists = new DataTable("Туристы");
DataColumn dcTouristID = new DataColumn("Код туриста", typeof(int));
DataColumn dcLastName = new DataColumn("Фамилия",typeof(string));
dtTourists.PrimaryKey =
    new DataColumn [] {dtTourists.Columns["Код туриста"],dtTourists.Columns["Фамилия"]};
```

Здесь первичным ключом будут значения поля «Код туриста» в сочетании со значением поля «Фамилия».

После определения первичного ключа объекта DataTable для свойства AllowDBNull (разрешение значений null) объектов DataColumn, формирующих ключ, будет установлено значение false.

В проектах, содержащих большое количество связанных таблиц, следует всегда определять свойство PrimaryKey. Это должно стать таким же правилом, как и задание первичного ключа при проектировании самой базы данных.

Ограничения UniqueConstraint и ForeignKeyConstraint

Теперь осталось разобраться, как можно определять некоторые свойства таблиц, называемые ограничениями. Свойство Constraint (ограничения) объекта DataTable бывает двух типов – UniqueConstraint и ForeignKeyConstraint.

Свойство UniqueConstraint определяет первичный ключ таблицы, например, в таблице Questions ключевым полем является questID. Объект dcQuestID представляет это поле:

```
DataColumn dcQuestID = dtQuestions.Columns.Add("questID", typeof(Int32));
```

Ограничение UniqueConstraint, налагаемое на объект dcQuestID, запрещает появление дублированных строк:

```
UniqueConstraint UC_dtQuestions = new UniqueConstraint(dcQuestID);
dtQuestions.Constraints.Add(UC_dtQuestions);
```

Однако при создании объекта dcQuestID мы ведь уже определяли его уникальность:

```
dcQuestID.Unique = true;
```

Действительно, последняя строка представляет собой неявный способ задания ограничения UniqueConstraint. Если уже определено уникальное поле или поля с помощью свойства Unique, то задавать ограничение UniqueConstraint не нужно.

Второе ограничение – ForeignKeyConstraint – определяет, как должны себя вести дочерние записи при изменении родительских записей и наоборот. ADO .NET требует точного описания объектов для управления ими, даже если изменение данных приложением не предусматривается. Ограничение ForeignKeyConstraint содержит следующие три правила:

- UpdateRule – применяется при изменении родительской строки;
- DeleteRule – применяется при удалении родительской строки;
- AcceptRejectRule – применяется при вызове метода AcceptChanges объекта DataTable, для которого определено ограничение.

Для каждого из этих правил, кроме последнего AcceptRejectRule, могут применяться следующие значения:

- Cascade – каскадное обновление связанных записей;

- None – изменения в родительской таблице не отражаются в дочерних записях;
- SetDefault – полю внешнего ключа в дочерних записях присваивается значение, заданное в свойстве DefaultValue этого поля;
- SetNull – полю внешнего ключа в дочерних записях присваивается значение Null.

Правило AcceptRejectRule принимает значения только Cascade или None. Значением по умолчанию для правил UpdateRule и DeleteRule является Cascade, для правила AcceptRejectRule – None.

Создадим ограничение для связи QuestionsVariants:

```
ForeignKeyConstraint   FK_QuestionsVariants =
        new ForeignKeyConstraint(dtQuestions.Columns["questID"],
                                 dtVariants.Columns["questID"]);
dtVariants.Constraints.Add(FK_QuestionsVariants);
```

Здесь задается вначале родительская колонка, а затем дочерняя (рис. 106). Добавлять созданное ограничение следует к объекту DataTable, представляющему дочернюю таблицу (в данном случае – объект dtVariants).

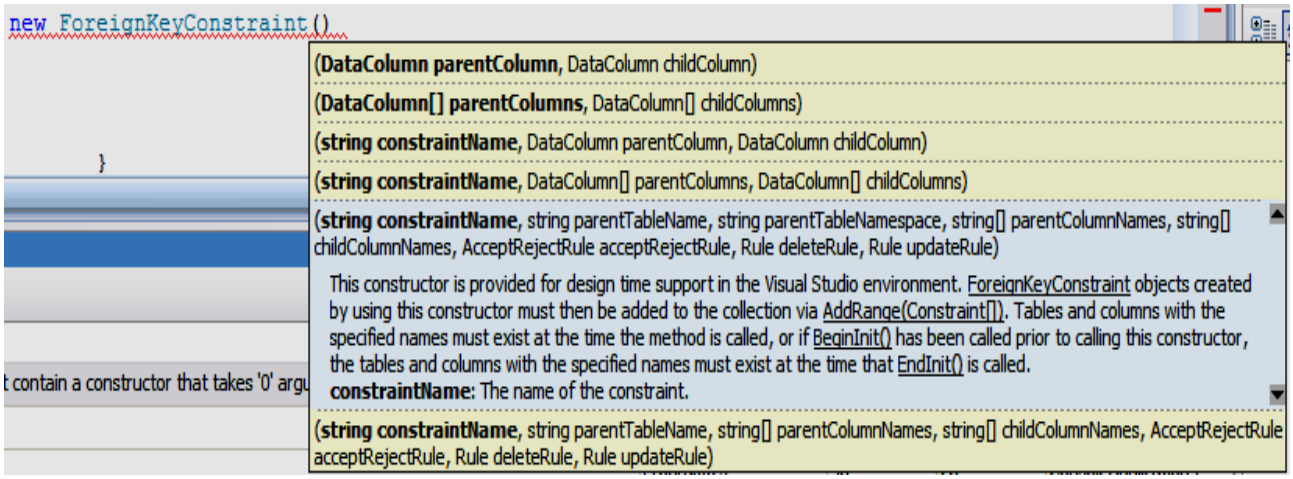


Рис. 106. Создание ограничения FK_QuestionsVariants

Этот фрагмент кода оставляет значения правил UpdateRule, DeleteRule и AcceptRejectRule заданными по умолчанию, т. е. Cascade и None, что соответствует значениям по умолчанию. Конструктор объекта является перегруженным.

Создание столбцов, основанных на выражении

При создании базы данных не следует помещать в нее значения, которые могут быть получены из уже имеющихся данных. Ранее была создана база данных BDTur_firm.mdb, в которой есть таблица «Туры». В этой таблице имеется поле «цена», где указывается стоимость туров. На практике может понадобить-

ся значение цены, выраженное в различных валютах, значение с учетом налогов, значение со скидкой и т. п. Для каждого конкретного случая можно получить дополнительное поле (*вычисляемое*), не вводя его в саму базу данных.

Технология ADO .NET позволяет создавать объекты DataColumn, основанные на выражении. Создадим новое Windows-приложение.

В коде формы, подключим пространство имен:

```
using System.Data.OleDb;
```

В классе формы определяем строки CommandText и ConnectionString:

```
string commandText  =  "SELECT Информация, [Код тура], Название, Цена FROM Туры";
string connectionString  =   @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
                             @"D:\ВМИ\For ADO\BDTur_firm.mdb";
```

В конструкторе формы программно создаем все объекты для вывода таблицы «Туры» в элемент управления DataGridView:

```
public Form1()
{
        InitializeComponent();
        OleDbConnection conn = new OleDbConnection(connectionString);
        OleDbCommand myCommand = new OleDbCommand();
        myCommand.Connection = conn;
        myCommand.CommandText = commandText;
        OleDbDataAdapter dataAdapter = new OleDbDataAdapter();
        dataAdapter.SelectCommand = myCommand;
        DataSet dsTours = new DataSet();
        DataTable dtTours = dsTours.Tables.Add("Туры");
        DataColumn dcIDtour = dtTours.Columns.Add("Код тура", typeof(Int32));
        dcIDtour.Unique = true;
        DataColumn dcName = dtTours.Columns.Add("Название");
        DataColumn dcPrice = dtTours.Columns.Add("Цена", typeof(Decimal));
        DataColumn dcInformation = dtTours.Columns.Add("Информация");
        conn.Open();
        dataAdapter.Fill(dsTours.Tables["Туры"]);
        conn.Close();
        dataGrid1.DataSource = dsTours.Tables["Туры"].DefaultView;
}
```

Запустим приложение (рис. 107).

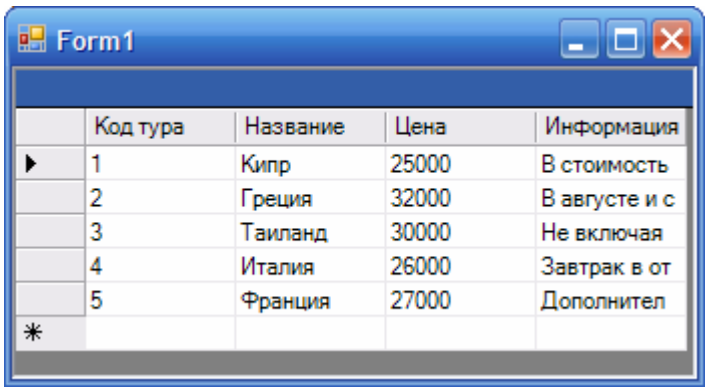


Рис. 107. Вывод содержимого таблицы «Туры»

Теперь добавим два объекта DataColumn, в которых будет вычисляться налог и скидка, после добавления объекта dcPrice:

```
...
DataColumn dcPrice = dtTours.Columns.Add("Цена", typeof(Decimal));
DataColumn dcPriceNDS = dtTours.Columns.Add("Цена с НДС", typeof(Decimal));
dcPriceNDS.Expression = "Цена*0.15+Цена";
DataColumn dcPricewithDiscount = dtTours.Columns.Add("Цена со скидкой",
                                                      typeof(Decimal));
dcPricewithDiscount.Expression = "Цена–Цена*0.10";
...
```

Свойство Expression созданного объекта DataColumn задает выражения для всех значений заданного поля (рис. 108).

Рис. 108. Программное формирование полей, основанных на значении

Свойство Expression поддерживает также агрегатные функции, объединение строк, ссылки на родительские и дочерние таблицы.

Отслеживание изменений в базе данных

Объекты DataSet и DataTable предоставляют перегруженный метод GetChanges, конструктор которого имеет следующий вид (рис. 109).

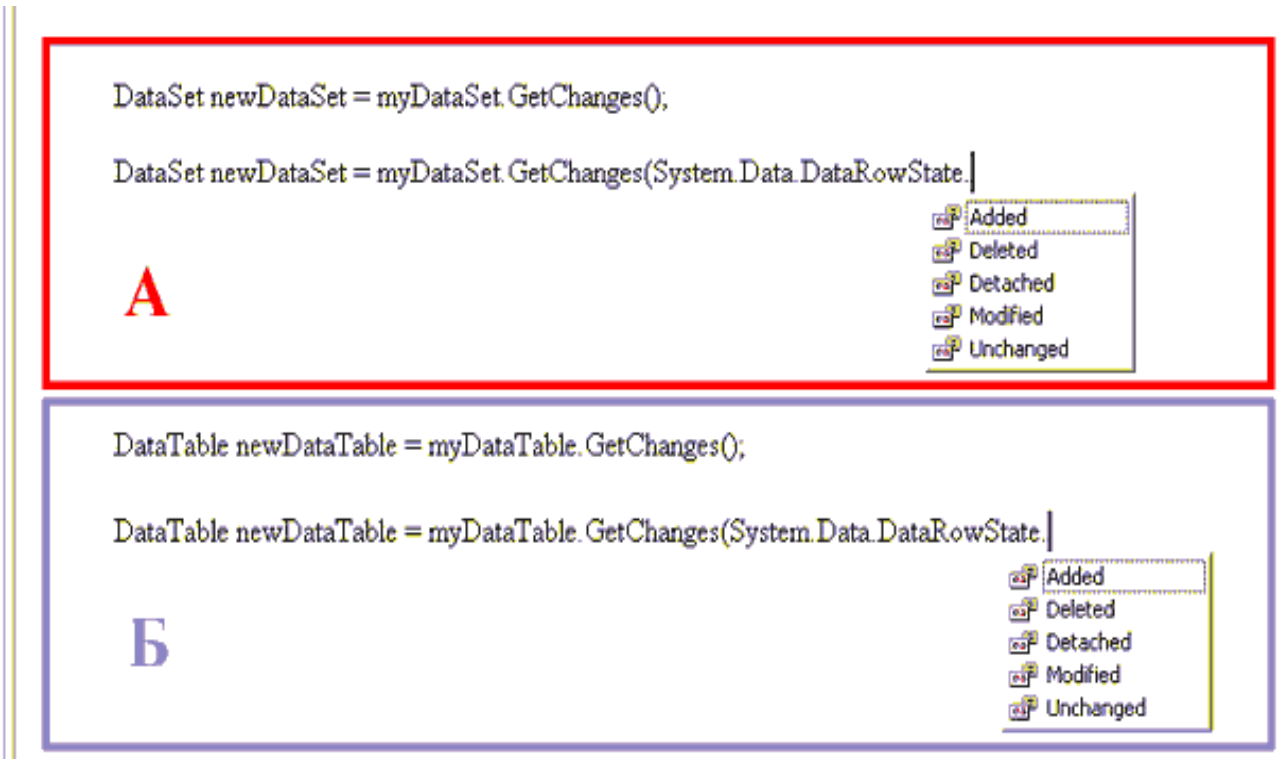


Рис. 109. Метод GetChanges

Метод GetChanges возвращает новый объект DataSet или DataTable со структурой исходного объекта, но содержащий только измененные записи из исходного объекта. На рис. 109 исходными объектами являются myDataSet (А) и myDataTable (Б), а новыми, созданными методом GetChanges – newDataSet (А) и newDataTable (Б). Применяя в качестве параметра значение перечисления DataRowState, можно получить записи с конкретным состоянием, например, только удаленные (Deleted) или добавленные (Added).

При передаче изменений в базу данных отправка объекта DataSet, полученного в результате вызова метода GetChanges, позволяет уменьшить объем трафика. В самом деле, отправка только внесенных изменений при прочих равных условиях займет меньше ресурсов, чем отправка полного объекта DataSet. Это становится особенно важным при работе с большими объемами данных или значительным числом подключений.

Метод GetChanges также применяется в качестве своеобразного сигнального флага, сообщающего, были ли затронуты данные. Реализовать такую проверку можно, например, при помощи следующего кода:

```
private void Form1_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{       DataSet ds = dataSet11.GetChanges();
        if(ds == null)  return;
        if(MessageBox.Show("Вы хотите сохранить изменения в базе данных?",
           "Завершение работы", MessageBoxButtons.YesNo,
           MessageBoxIcon.Question) == DialogResult.Yes)        DataAdapter1.Update(ds);
}
```

Обработка исключений

В процессе передачи изменений в базу данных могут возникать многочисленные исключения. Объекты DataSet, DataTable и DataRow имеют свойство HasErrors, позволяющее обрабатывать некоторые из них.

Для обработки исключений в процессе обновления таблиц БД можно использовать следующий код:

```
private void Form1_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
  try
  {
        sqlDataAdapter1.Update(dataSet11);
  }
  catch(Exception ex)
  {
        if(dataSet11.HasErrors)
        {
          foreach( DataTable myTable in dataSet11.Tables)
          {
            if(myTable.HasErrors)
            {
              foreach(DataRow myRow in myTable.Rows)
              {
                if(myRow.HasErrors)
                {
                  MessageBox.Show("Ошибка в записи #: " +
                                  myRow["Код туриста"], myRow.RowError);
                  foreach(DataColumn myColumn in myRow.GetColumnsInError())
                  {
                    MessageBox.Show(myColumn.ColumnName, " – в этом столбце ошибка");
                  }
                  myRow.ClearErrors();
                  myRow.RejectChanges();
                }
              }
            }
          }
        }
  }
}
```

Здесь происходит проход по каждому объекту, входящему в набор DataTable, DataRow или DataColumn. Метод ClearErrors удаляет все ошибки из объекта myRow, а метод RejectChanges производит откат всех изменений.

4.4.2. Объект DataRow

Содержимое объекта DataSet представляет собой набор записей, который представлен объектами DataRow. В запущенном приложении содержимое объекта DataSet доступно для изменений, например, если данные выводятся в элемент управления DataGridView, то, перемещаясь по отдельным клеткам, можно править значения как в обычной электронной таблице. При этом происходит изменение объекта DataRow, соответствующее заданной записи. Рассмотрим программное создание и изменение записей.

Программное создание и изменение записей таблицы данных

Создадим новое Windows-приложение. В конструкторе формы создаем экземпляр dtTours и поля, соответствующие таблице «Туры»:

```
public Form1()
{       InitializeComponent();
        DataTable dtTours = new DataTable();
        DataColumn dcIDtour = dtTours.Columns.Add("Код тура", typeof(Int32));
        dcIDtour.Unique = true;
        DataColumn dcName = dtTours.Columns.Add("Название");
        DataColumn dcPrice = dtTours.Columns.Add("Цена", typeof(Decimal));
        DataColumn dcInformation = dtTours.Columns.Add("Информация");
        DataView myDataView = new DataView(dtTours);
        dataGrid1.DataSource = myDataView;
}
```

Для того чтобы привязать созданные данные к элементу управления DataGridView, понадобилось создать экземпляр myDataView класса DataView. Каждый объект DataTable содержит объект DataView, причем этот объект, используемый по умолчанию, называется DataTable.DefaultView. Данный объект неоднократно использовался ранее, например, в предыдущем проекте для вывода данных:

```
dataGrid1.DataSource = dsTours.Tables["Туры"].DefaultView;
```

Один объект DataTable может иметь несколько объектов DataView – это удобно для вывода одних и тех же данных, отфильтрованных или отсортированных различным образом. Запускаем приложение. На экранной форме представлена готовая структура таблицы «Туры» (рис. 110).

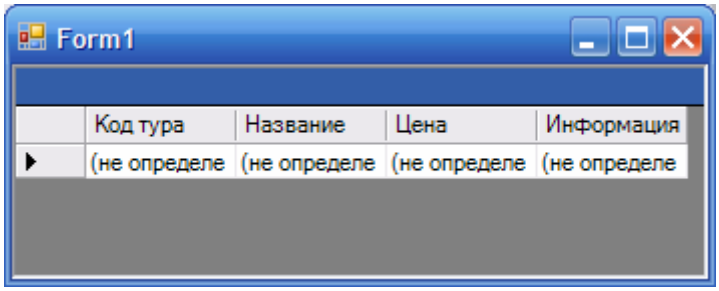


Рис. 110. Структура таблицы «Туры»

В данном проекте не будем подключаться к какой-либо базе данных – попробуем заполнить таблицу записями программно. Для добавлений одной новой записи перед созданием экземпляра myDataView вставляем следующий фрагмент кода:

```
DataRow myRow = dtTours.NewRow();
myRow["Код тура"] = 1;
myRow["Название"] = "Кипр";
myRow["Цена"] = 25000;
myRow["Информация"] =
                "В стоимость двух взрослых путевок входит цена одной детской (до 7лет)";
dtTours.Rows.Add(myRow);
```

Запускаем приложение (рис. 111). В таблице появилась первая запись.

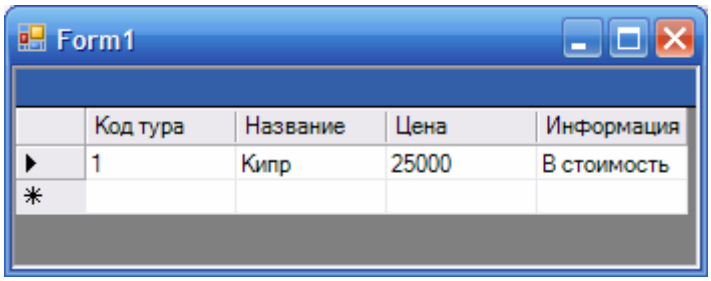


Рис. 111. Добавление записи в таблицу

Добавим еще одну запись:

```
DataRow myRow2 = dtTours.NewRow();
myRow2["Код тура"] = 2;
myRow2["Название"] = "Греция";
myRow2["Цена"] = 32000;
myRow2["Информация"] = "В августе и сентябре действуют специальные скидки";
dtTours.Rows.Add(myRow2);
```

Название, указываемое в квадратных скобках объектов myRow или myRow2, представляет собой имя столбца, которое мы определили в самом начале.

К столбцу можно обращаться и по индексу. Для демонстрации создадим следующий код:

```
DataRow myRow = dtTours.NewRow();
myRow[0] = 1;
myRow[1] = "Кипр";
myRow[2] = 25000;
myRow[3] = "В стоимость двух взрослых путевок входит цена одной детской (до 7лет)";
dtTours.Rows.Add(myRow);
DataRow myRow2 = dtTours.NewRow();
myRow2[0] = 2;
myRow2[1] = "Греция";
```

```
myRow2[2] = 32000;
myRow2[3] = "В августе и сентябре действуют специальные скидки";
dtTours.Rows.Add(myRow2);
```

Нумерация столбцов начинается с нуля.

Более удобный способ добавления записей – применение свойства ItemArray объекта DataRow, где можно задавать значения полей в виде массива:

```
DataRow myRow3 = dtTours.NewRow();
myRow3.ItemArray = new object[] { 3, "Таиланд", 30000, null };
 dtTours.Rows.Add(myRow3);
```

Здесь мы указали значение поля «Информация», равное null, – таким образом можно пропускать неизвестные поля (рис. 112).

Рис. 112. Вставка записи с одним значением null

Конечно, вставка записей вручную в объект DataSet, не связанный с хранилищем данных, имеет не очень большой смысл. Поэтому рассмотрим, как вставлять (и изменять) данные в уже загруженный кэш данных.

Изменим текущий проект. В коде проекта после отображения данных в элементе DataGrid:

```
dataGrid1.DataSource = dsTours.Tables["Туры"].DefaultView;
```

будем добавлять соответствующие строки.

Для изменения, например, пятой строки, мы указываем в свойстве Rows объекта dtTours ее индекс, равный числу 4, так как нумерация полей в строке начинается с нуля, затем вызываем метод BeginEdit для начала редактирования, устанавливаем группу свойств и в заключение принимаем изменения, вызывая метод EndEdit:

```
DataRow myRow=dtTours.Rows[4];
myRow.BeginEdit();
myRow["Код тура"] = 5;
myRow["Название"] = "Турция";
myRow["Цена"] = "27000";
myRow["Информация"] = "Осенние скидки с 15 октября";
myRow.EndEdit();
```

Тот же самый результат мы получим с помощью свойства ItemArray:

```
DataRow myRow=dtTours.Rows[4];
myRow.BeginEdit();
myRow.ItemArray = new object[]
                {5,"Турция", 27000, null, null,  "Осенние скидки с 15 октября"};
myRow.EndEdit();
```

Здесь мы установили для третьего и четвертого полей, которые являются вычисляемыми, значения null, подразумевая, что они останутся по умолчанию, а при запуске заполнятся своими значениями (рис. 113).

Код тура	Название	Цена	Цена с НДС	Цена со скид	Информация
1	Кипр	25000	28750,00	22500,00	В стоимость
2	Греция	32000	36800,00	28800,00	В августе и с
3	Таиланд	30000	34500,00	27000,00	Не включая
4	Италия	26000	29900,00	23400,00	Завтрак в от
5	Турция	27000	31050,00	24300,00	Осенние ски

Рис. 113. Пропущенные вычисляемые поля заполняются своими значениями

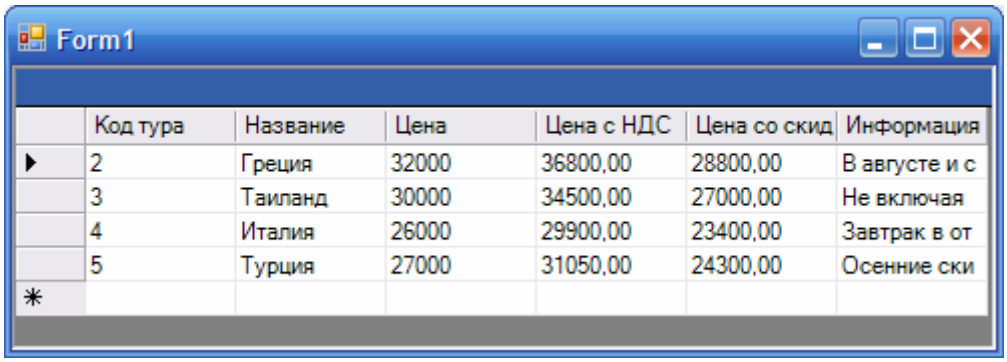
Для удаления заданной записи нужно создать объект DataRow, которому передается индекс строки, а затем вызвать метод Remove свойства Rows объекта DataTable:

```
DataRow myRow2 = dtTours.Rows[0];
dtTours.Rows.Remove(myRow2);
```

Этого достаточно для удаления строки, но для того, чтобы пометить заданную строку как удаленную, вызываем метод Delete:

```
myRow2.Delete();
```

В результате у нас удалится строка (рис. 114), причем объект DataTable пометит ее в качестве удаленной – это необходимо, чтобы избежать ошибок (например, в связанных записях).



Код тура	Название	Цена	Цена с НДС	Цена со скид	Информация
2	Греция	32000	36800,00	28800,00	В августе и с
3	Таиланд	30000	34500,00	27000,00	Не включая
4	Италия	26000	29900,00	23400,00	Завтрак в от
5	Турция	27000	31050,00	24300,00	Осенние ски

Рис. 114. Первая строка, имеющая индекс 0, была удалена

Свойство RowState

При работе с данными приходится постоянно вносить изменения в записи – добавлять, редактировать или удалять. Объект DataRow обладает свойством RowState, позволяющим отслеживать текущий статус строки.

Создадим новое приложение. В конструкторе формы мы создадим всего одно поле, затем одну запись, статус которой будем отслеживать:

```
public Form1()
{
  InitializeComponent();
  DataTable dtTours = new DataTable("Туры");
  DataColumn IDtour = new DataColumn("Код тура", typeof(Int32));
  dtTours.Columns.Add(IDtour);
  dataGrid1.DataSource = dtTours;
  DataRow myRow;

// Создаем новую, отсоединенную запись
  myRow = dtTours.NewRow();
  richTextBox1.Text += Convert.ToString("Новая запись: " + myRow.RowState);

//Добавляем запись в объект DataTable
  dtTours.Rows.Add(myRow);
  richTextBox1.Text += Convert.ToString("\nДобавление записи: " + myRow.RowState);
//Принимаем все изменения в объекте DataTable
  dtTours.AcceptChanges();
  richTextBox1.Text += Convert.ToString("\nМетод AcceptChanges: " + myRow.RowState);

//Редактируем запись
  myRow["Код тура"] = 1;
  richTextBox1.Text += Convert.ToString("\nРедактирование строки: " + myRow.RowState);

//Удаляем строку
  myRow.Delete();
  richTextBox1.Text += Convert.ToString("\nУдаление: " + myRow.RowState);

//Отменяем  все изменения в объекте DataTable
  dtTours.RejectChanges();
  richTextBox1.Text += Convert.ToString("\nМетод RejectChanges: " + myRow.RowState);
}
```

Запускаем приложение. В текстовое поле выводится статус записи myRow (рис. 115).

Значение Detached означает, что запись не относится к объекту DataTable. После добавления ее статус изменяется на Added – теперь она существует в объекте DataTable, но ее нет в базе данных. Конечно, здесь не рассматривается

взаимодействие с источником записей, но это же значение будет у записей, добавляемых в DataGrid после вывода данных из базы при наличии подключения. Вызывая метод AcceptChanges объекта DataTable, выполняется прием всех изменений, поэтому статус DataRow изменяется на Unchanged – теперь запись считается «своей», она не была изменена после вызова метода. После вызова метода Delete запись помечается удаленной – она еще не полностью удалена, в случае отмены изменений статус будет восстановлен. Действительно, вызов метода RejectChanges объекта DataTable восстанавливает запись до состояния Unchanged.

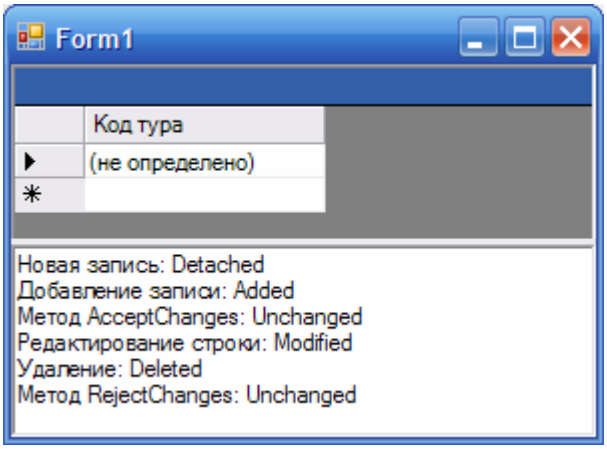


Рис. 115. Приложение RowState

Свойство RowState всегда возвращает отдельное, доступное только для чтения значение. Это свойство используется для поиска записей, соответствующих заданному статусу, а также при передаче изменений в базу данных.

Свойство RowVersion

Свойство RowVersion предназначено для извлечения значения записи (объекта DataRow), зависящего от совершенных изменений.

Возможны следующие версии записи:

- Current – текущее значение
- Default – значение по умолчанию
- Original – первоначальное значение
- Proposed – предполагаемое значение

Создадим новое приложение. В классе формы объявим объекты DataRow и DataTable:

```
DataRow  myRow;
DataTable  dtTours;
```

В конструкторе формы создаем запись, определяем источник данных для элемента DataGridView, а также отключаем его доступность:

```
public Form1()
{
        InitializeComponent();
        dtTours = new DataTable("Туры");
        DataColumn IDtour = new DataColumn("Название", typeof(string));
        dtTours.Columns.Add(IDtour);
        myRow = dtTours.NewRow();
        dtTours.Rows.Add(myRow);
        myRow["Название"] = "Таиланд";
        dataGrid1.DataSource = dtTours;
        dataGrid1.Enabled = false;
}
```

Создадим метод TestRowVersion, в котором будет проверяться свойство RowVersion записи:

```
private void TestRowVersion()
{
  if(myRow.HasVersion(DataRowVersion.Original))
        rtbReport.Text += String.Format("Значение original: {0}\n",
                                myRow["Название", DataRowVersion.Original]);
  if(myRow.HasVersion(DataRowVersion.Current))
        rtbReport.Text += String.Format("Значение current: {0}\n",
                                myRow["Название", DataRowVersion.Current]);
  if(myRow.HasVersion(DataRowVersion.Default))
        rtbReport.Text += String.Format("Значение default: {0}\n",
                                myRow["Название", DataRowVersion.Default]);
  if(myRow.HasVersion(DataRowVersion.Proposed))
        rtbReport.Text += String.Format("Значение proposed: {0}\n",
                                myRow["Название", DataRowVersion.Proposed]);
}
```

Метод HasVersion позволяет определить, поддерживает ли объект myRow версию данных, указываемую в скобках. В случае подтверждения будет выполняться код оператора – выводится в элемент rtbReport соответствующее сообщение.

В обработчике кнопки btnBeginEdit (Редактировать) вызываем метод BeginEdit, устанавливаем новое значение записи:

```
private void btnBeginEdit_Click(object sender, System.EventArgs e)
{
        myRow.BeginEdit();
        myRow["Название"] = "Франция";
        rtbReport.Text += "BeginEdit\n";
        TestRowVersion();
}
```

В обработчике кнопки btnEndEdit завершаем редактирование записи:

```
private void btnEndEdit_Click(object sender, System.EventArgs e)
{
	myRow.EndEdit();
	rtbReport.Text += "EndEdit\n";
	TestRowVersion();
}
```

В обработчике кнопки btnCancelEdit отказываемся от внесенных изменений:

```
private void btnCancelEdit_Click(object sender, System.EventArgs e)
{
	myRow.CancelEdit();
	rtbReport.Text += "CancelEdit\n";
	TestRowVersion();
}
```

В обработчике кнопки btnDelete удаляем объект myRow:

```
private void btnDelete_Click(object sender, System.EventArgs e)
{
	myRow.Delete();
	rtbReport.Text += "Запись удалена\n";
	TestRowVersion();
}
```

В обработчике кнопки «Очистить отчет» просто удаляем содержимое текстового поля:

```
private void btnClear_Click(object sender, EventArgs e)
{
	this.rtbReport.Text = "";
}
```

Запускаем приложение.

После нажатия кнопки «Begin Edit» мы начинаем редактирование записи, вводится новое значение – «Франция». Оно становится значением по умолчанию (Default) и предполагаемым Proposed, значение «Таиланд» является текущим (Current) (рис. 116).

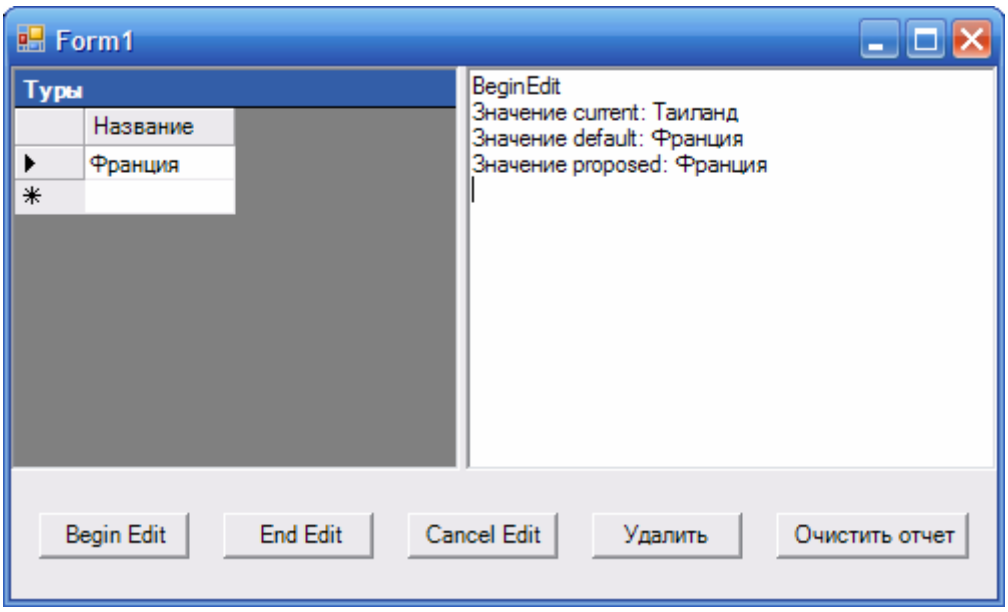


Рис. 116. Демонстрация работы. Шаг 1 – Редактирование

Отменяем редактирование, нажимая кнопку «Cancel Edit». При этом значение «Таиланд» становится текущим (Current) и по умолчанию (Default) (рис. 117).

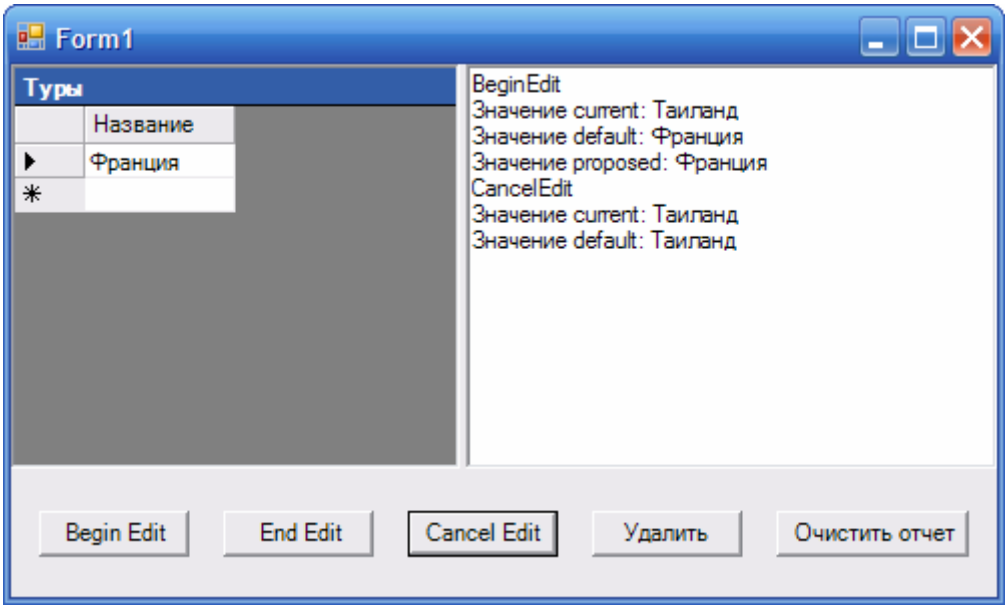


Рис. 117. Демонстрация работы. Шаг 2 – Отмена редактирования

Снова начинаем редактирование – картина повторяется (рис. 118).

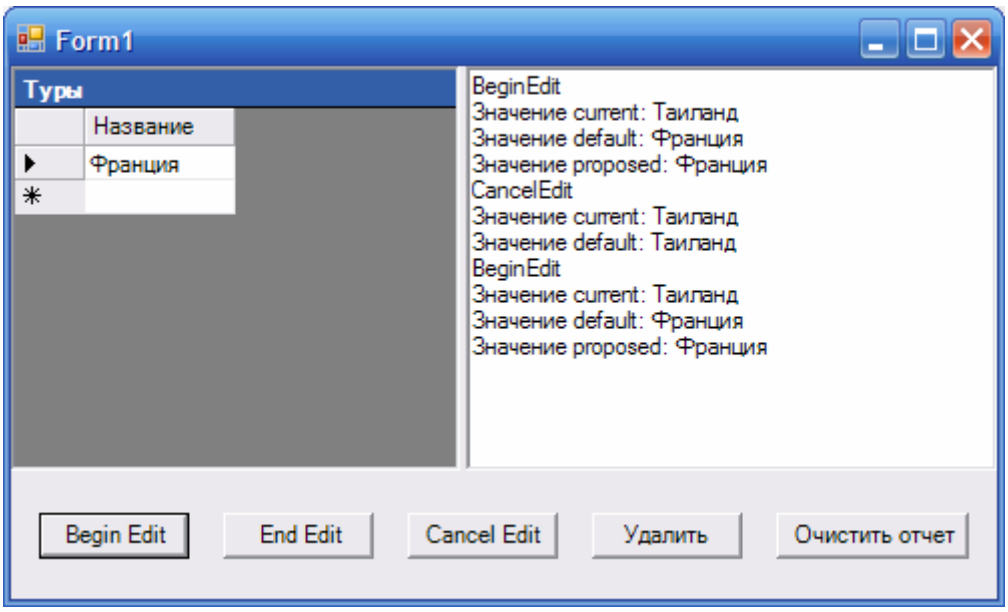


Рис. 118. Демонстрация работы. Шаг 3 – Повторное редактирование

На этот раз завершаем редактирование, нажимая кнопку «End Edit» – новое значение «Франция» становится текущим (Current) и по умолчанию (Default) (рис. 119).

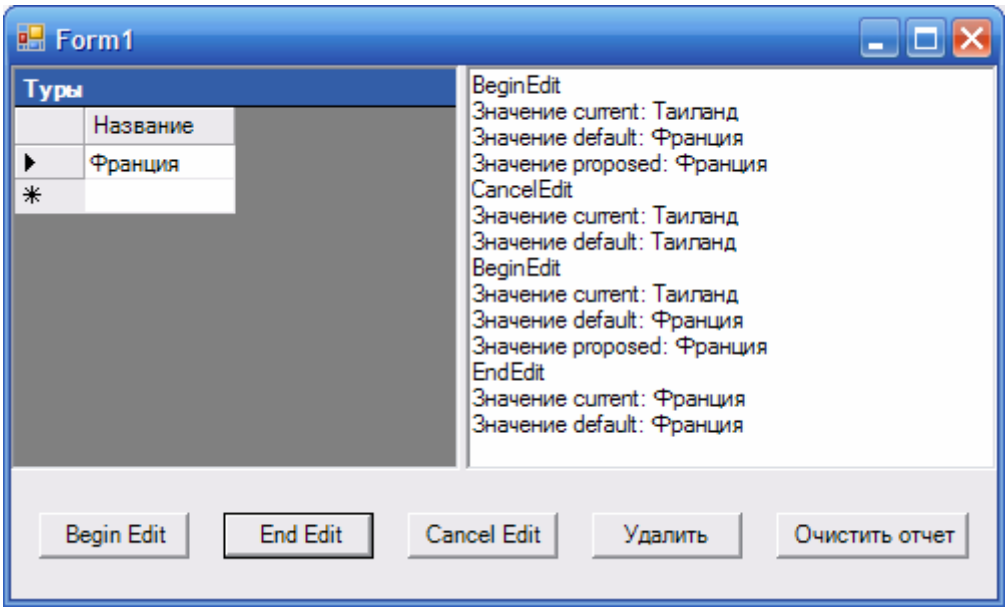


Рис. 119. Демонстрация работы. Шаг 4 – Завершение редактирования

Теперь нажимаем кнопку «Удалить» – при этом удаляется сам объект myRow и дальнейшее изменение его значений оказывается невозможным (рис. 120).

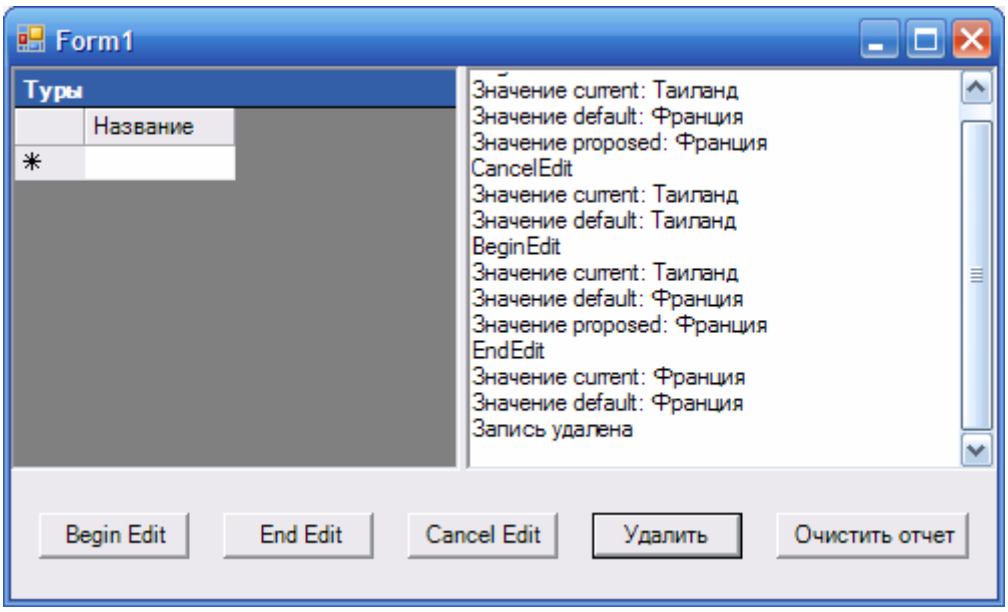


Рис. 120. Демонстрация работы. Шаг 5 – Удаление

События объекта DataTable

Объект DataTable содержит ряд событий, которые могут применяться для слежения за происходящими изменениями. Наиболее часто используются следующие события:

ColumnChanged – наступает после изменения содержимого поля таблицы;

ColumnChanging – происходит в течение редактирования содержимого поля таблицы;

RowChanged – наступает после изменения объекта DataRow (записи);

RowChanging – происходит в течение редактирования объекта DataRow;

RowDeleted – наступает после удаления объекта DataRow;

RowDeleting – происходит при удалении объекта DataRow.

В конструкторе формы предыдущего приложения добавим обработку четырех событий объекта DataTable:

```
public Form1()
{
  ...
dtTours.RowChanging += new DataRowChangeEventHandler(dtTours_RowChanging);
dtTours.RowChanged += new DataRowChangeEventHandler(dtTours_RowChanged);
dtTours.RowDeleting += new DataRowChangeEventHandler(dtTours_RowDeleting);
dtTours.RowDeleted += new DataRowChangeEventHandler(dtTours_RowDeleted);
  ...
}
```

В соответствующих методах просто выводим сообщение в текстовое поле:

```
private void dtTours_RowChanging(object sender, DataRowChangeEventArgs e)
{
  rtbReport.Text += String.Format("Событие – изменение записи\n", e.Row["Название"]);
}

private void dtTours_RowChanged(object sender, DataRowChangeEventArgs e)
{
  rtbReport.Text += "\nСобытие – запись изменена\n";
}

private void dtTours_RowDeleting(object sender, DataRowChangeEventArgs e)
{
  rtbReport.Text += String.Format("Событие – удаление записи\n", e.Row["Название"]);
}

private void dtTours_RowDeleted(object sender, DataRowChangeEventArgs e)
{
  rtbReport.Text += "\nСобытие – запись удалена\n";
}
```

Запускаем приложение.

Нажимаем кнопку «Begin Edit», затем «End Edit» – происходят события RowChanging и RowChanged. Удаляем запись – происходят события RowDeleting и RowDeleted (рис. 121).

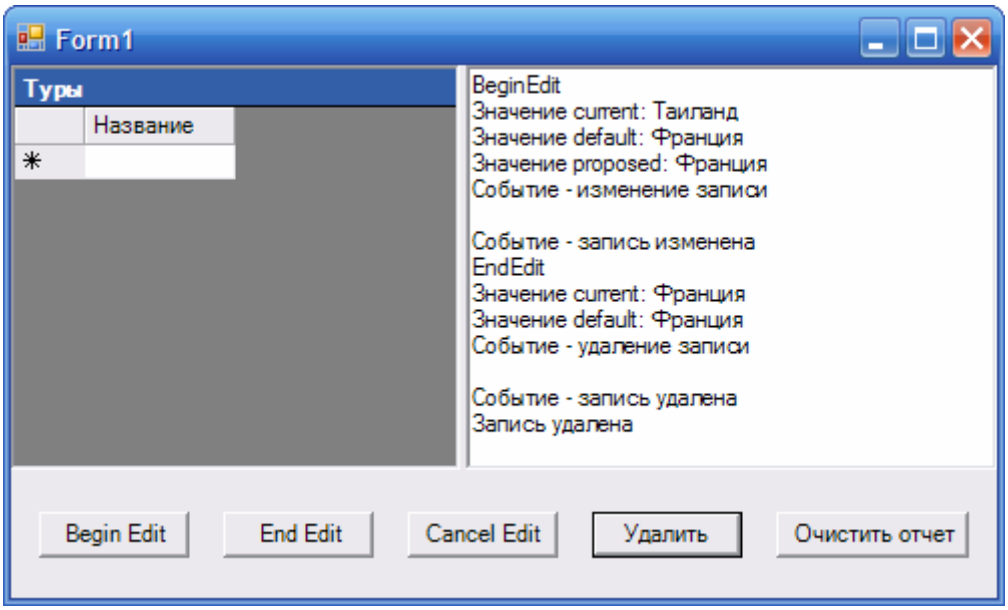


Рис. 121. Демонстрация событий таблиц

В обработчиках событий можно добавить соответствующие действия, например, подтверждение изменения (RowChanging) или удаления (RowDeleting).

4.4.3. Объект DataGridView

Вывод двух связанных таблиц данных в один элемент DataGridView

База данных Microsoft Access BDTur_firm.mdb содержит таблицу «Туристы», которая связана с другими таблицами. Было бы удобно выводить эту таблицу в элемент DataGridView вместе с другими таблицами, а также выводить связанные записи этой таблицы. Создадим новое Windows-приложение.

Разместим на форме элемент управления DataGridView, свойству Dock которого устанавливаем значение Fill. Переходим в код формы и подключаем пространство имен:

```
using System.Data.OleDb;
```

В конструкторе формы создаем соединение, объект OleDbCommand, определяем для него соединение и строку CommandText:

```
OleDbConnection conn = new OleDbConnection(connectionString);
OleDbCommand myCommand = new OleDbCommand();
myCommand.Connection = conn;
myCommand.CommandText = commandText;
```

Подключаемся к файлу базы данных BDTur_firm.mdb, указываем соответствующие параметры строк connectionString и commandText:

```
string commandText = "SELECT [Код туриста], Фамилия, Имя, Отчество  FROM Туристы";
string connectionString = @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source="+
                                @"D:\ВМИ\For ADO\BDTur_firm.mdb";
string commandText2 =
        "SELECT  [Код туриста], [Серия паспорта], Город, Страна, Телефон, Индекс " +
        "FROM  [Информация о туристах]";
```

Создаем объект DataAdapter и в его свойстве SelectCommand устанавливаем значение myCommand, открываем соединение:

```
OleDbDataAdapter dataAdapter = new OleDbDataAdapter();
dataAdapter.SelectCommand = myCommand;
conn.Open();
```

Создаем объект DataSet:

```
DataSet ds = new DataSet();
```

В объекте DataSet здесь будут храниться две таблицы – главная и связанная с ней дочерняя. Поэтому воспользуемся свойством TableMappings объекта DataAdapter для занесения в него первой таблицы «Туристы»:

```
dataAdapter.TableMappings.Add("Table", "Туристы");
dataAdapter.Fill(ds);
```

Теперь нам следует добавить объекты OleDbDataAdapter и OleDbCommand для таблицы «Информация о туристах»:

```
OleDbCommand myCommand2 = new OleDbCommand();
myCommand2.Connection = conn;
myCommand2.CommandText = commandText2;
```

```
OleDbDataAdapter dataAdapter2 = new OleDbDataAdapter();
```

Обратим внимание на то, что dataAdapter2 использует то же самое подключение conn, что и dataAdapter.

Строку commandText2 определим следующим образом:

```
string commandText2 =
        "SELECT  [Код туриста], [Серия паспорта], Город, Страна, Телефон, Индекс " +
        "FROM   [Информация о туристах]";
```

Теперь свяжем второй объект OleDbDataAdapter с только что созданной второй командой и отобразим «Информацию о туристах» на его таблицу. Затем можно заполнить объект DataSet данными из второй таблицы:

```
dataAdapter2.SelectCommand = myCommand2;
dataAdapter2.TableMappings.Add("Table", "Информация о туристах");
dataAdapter2.Fill(ds);
```

В итоге получился объект DataSet с двумя таблицами. Теперь можно выводить одну из этих таблиц на форму, или две сразу. Но связь между таблицами еще не создана. Для конфигурирования отношения по полю «Код туриста» создадим два объекта DataColumn:

```
DataColumn dcTouristsID = ds.Tables["Туристы"].Columns["Код туриста"];
DataColumn dcInfoTouristsID =
                        ds.Tables["Информация о туристах"].Columns["Код туриста"];
```

Далее создаем объект DataRelation, в его конструкторе передаем название отношения между таблицами и два объекта DataColumn:

```
DataRelation dataRelation = new
            DataRelation("Дополнительная информация", dcTouristsID, dcInfoTouristsID);
```

Добавляем созданный объект отношения к объекту DataSet:

```
ds.Relations.Add(dataRelation);
```

Создаем объект DataViewManager, отвечающий за отображение DataSet в объекте DataGrid:

```
DataViewManager dsview = ds.DefaultViewManager;
```

Присваиваем свойству DataSource объекта DataGridView созданный объект DataViewManager:

```
dataGrid1.DataSource = dsview;
```

Последнее, что осталось сделать, – сообщить объекту DataGrid, какую таблицу считать главной (родительской) и, соответственно, отображать на форме:

```
dataGrid1.DataMember = "Туристы";
```

Закрываем соединение:

```
conn.Close();
```

Полностью созданный код конструктора Form1 выглядит следующим образом:

```
public Form1()
{
        InitializeComponent();
```

```
        OleDbConnection conn = new OleDbConnection(connectionString);
        OleDbCommand myCommand = new OleDbCommand();
        myCommand.Connection = conn;
        myCommand.CommandText = commandText;
        OleDbDataAdapter dataAdapter = new OleDbDataAdapter();
        dataAdapter.SelectCommand = myCommand;
        conn.Open();

        DataSet ds = new DataSet();
        dataAdapter.TableMappings.Add("Table", "Туристы");
        dataAdapter.Fill(ds);

        OleDbCommand myCommand2 = new OleDbCommand();
        myCommand2.Connection = conn;
        myCommand2.CommandText = commandText2;
        OleDbDataAdapter dataAdapter2 = new OleDbDataAdapter();
        dataAdapter2.SelectCommand = myCommand2;
        dataAdapter2.TableMappings.Add("Table", "Информация о туристах");
        dataAdapter2.Fill(ds);
        DataColumn dcTouristsID = ds.Tables["Туристы"].Columns["Код туриста"];
        DataColumn dcInfoTouristsID =
                    ds.Tables["Информация о туристах"].Columns["Код туриста"];
        DataRelation dataRelation = new
            DataRelation("Дополнительная информация", dcTouristsID, dcInfoTouristsID);
        ds.Relations.Add(dataRelation);
        DataViewManager dsview = ds.DefaultViewManager;
        dataGrid1.DataSource = dsview;
        dataGrid1.DataMember = "Туристы";
        conn.Close();
}
```

Запускаем приложение. У каждой строки таблицы появился знак «+», говорящий о том, что у данной записи имеются дочерние записи (рис. 122).

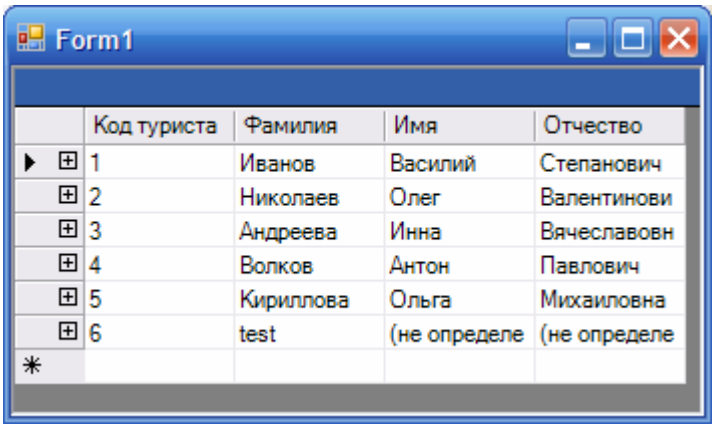


Рис. 122. Демонстрация наличия дочерних записей

Для перехода на дочернюю запись нажимаем на «+» и нажимаем на ссылку «Дополнительная информация». Окно приложения приобретает следующий вид (рис. 123).

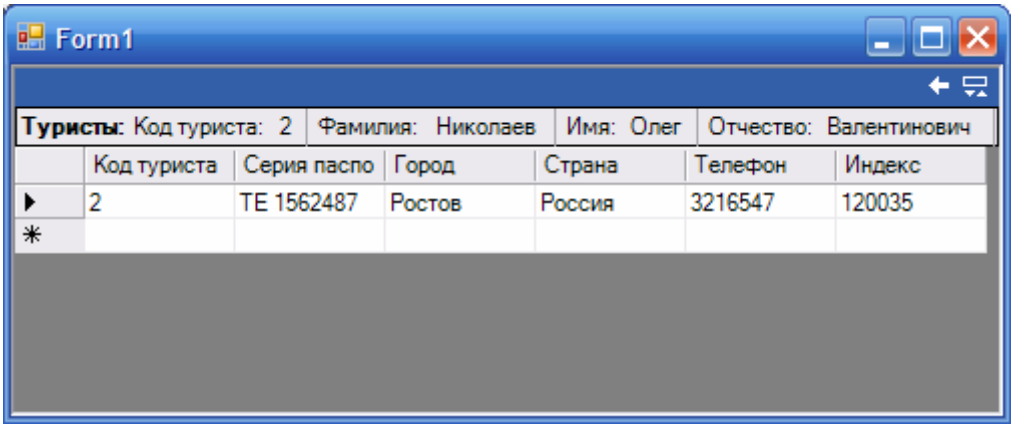


Рис. 123. Демонстрация дочерних записей

Для возвращения на родительскую запись нажимаем на кнопку со стрелкой. Обратите внимание, что здесь не накладываются ограничения – это сделано для упрощения кода.

Закомментируем фрагмент кода, отвечающий за создание связи. В этом случае в приложении будут отдельные ссылки на две таблицы – «Туристы» и «Информация о туристах» (рис. 124).

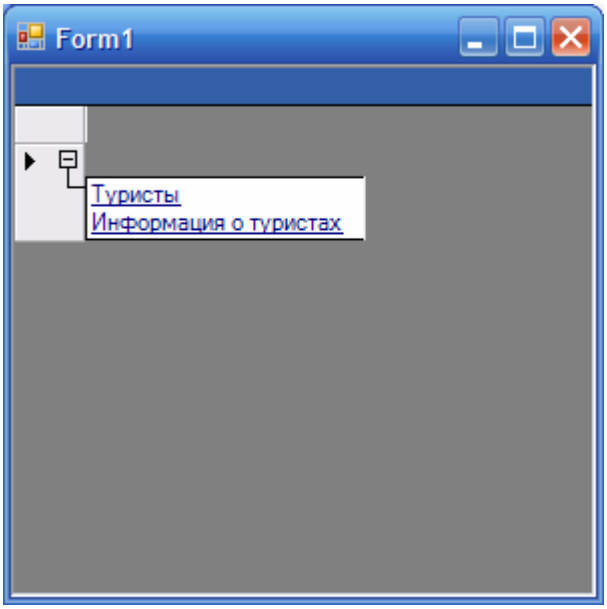


Рис. 124. Переход на две таблицы – «Туристы» и «Информация о туристах»

Вывод связанных таблиц данных в два элемента DataGridView

Наиболее часто встречаемая задача при разработке приложений, связанных с базами данных, – это одновременный вывод двух таблиц на форму, причем при перемещении по записям главной таблицы в дочерней автоматически отображаются связанные записи.

Добавим на форму еще один компонент DataGridView, в котором будут отображаться связанные записи.

В конструктор формы добавим следующий код:

```
dataGridView1.DataSource = dsview;
dataGridView1.DataMember = "Туристы.Дополнительная информация";
```

Окно запущенного приложения показано на рисунке 125.

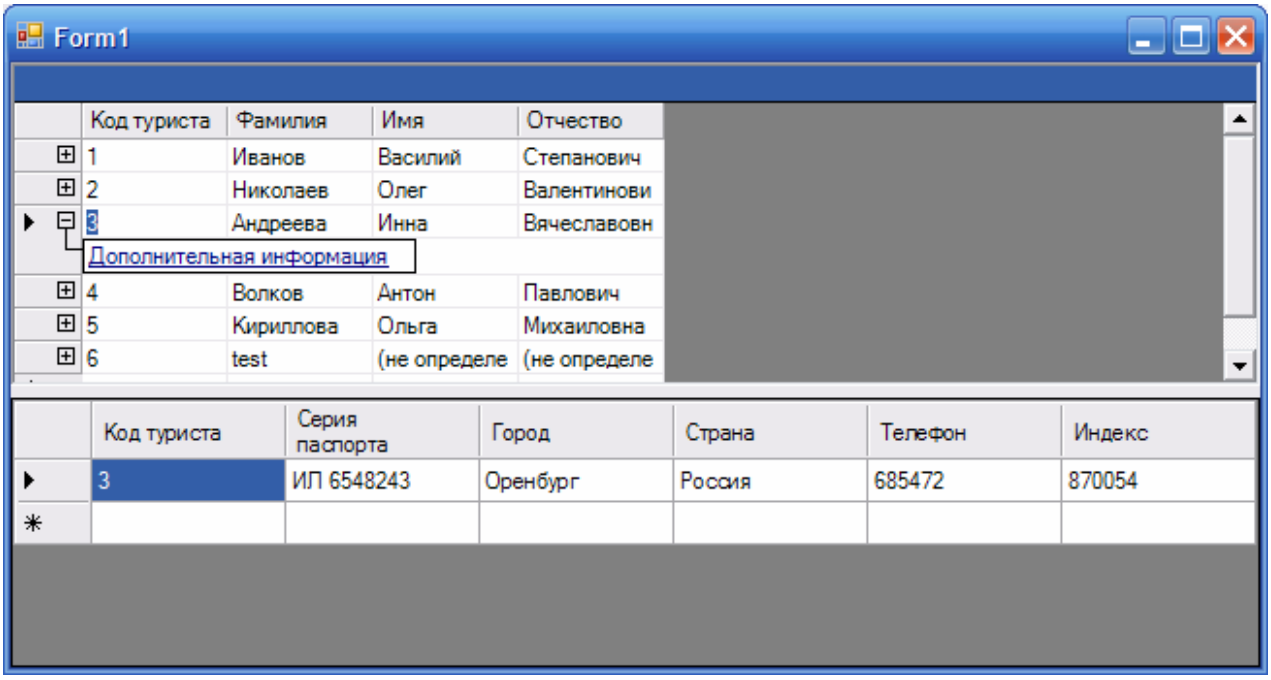


Рис. 125. Отображение связанных данных

Ключевым моментом здесь является определение связи «Дополнительная информация»:

```
dataGridView1.DataMember = "Туристы.Дополнительная информация";
```

Эта же связь доступна и в окне родительской таблицы – при нажатии на гиперссылку возвращается связанная запись.

4.4.4. Объект DataView

В объект DataSet можно загрузить большое количество данных и затем, отсоединившись от источника, использовать их по частям. Объект DataView предназначен для работы с упорядоченной определенным образом частью данных, загруженных в DataSet. Подобно всем объектам ADO .NET, с ним можно работать как при помощи визуальных средств среды, так и программно.

Фильтрация данных

Создадим новое Windows-приложение. На форму добавим соединение с SQL-базой данных и два компонента DataGridView. Первый компонент настроим на отображение таблицы «Туристы». Для второго компонента сформируем следующий код в обработчике события Form_Load:

```
DataView myDataView = new DataView(bDTur_firmSQL2DataSet.Туристы);
myDataView.RowStateFilter = DataViewRowState.Deleted;
dataGridView2.DataSource = myDataView;
```

Запустив приложение, удалим последнюю строку таблицы «Туристы». Она удаляется из первого компонента DataGridView и появляется во втором (рис. 126).

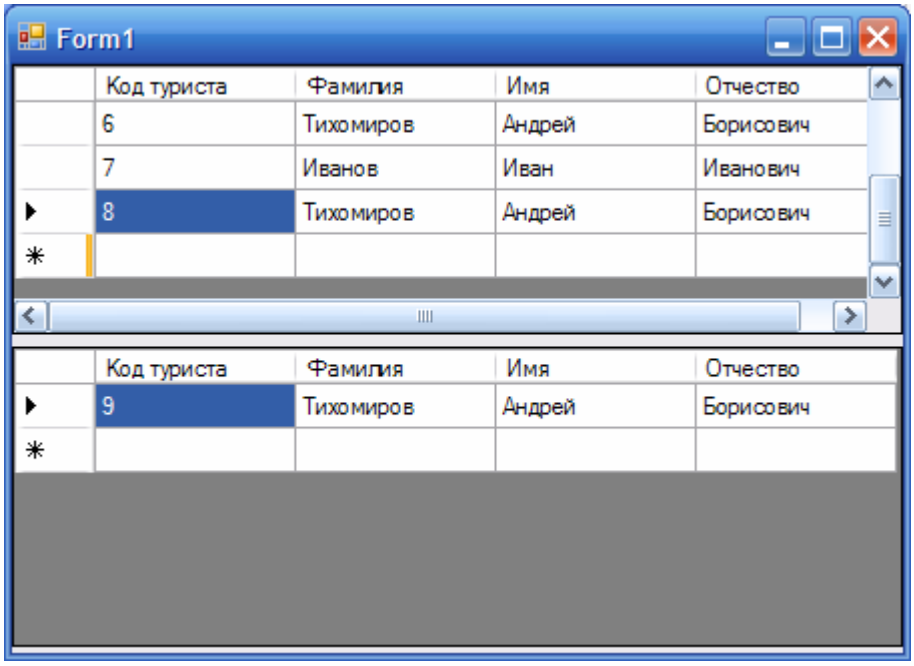


Рис. 126. Демонстрация удаленных записей

Для одного объекта DataView в пределах одной версии объекта DataSet (исходной или текущей) возможно объединение фильтров – например, для отображения новых и удаленных записей.

Для вывода новых строк (DataViewRowState.Added) и измененных (DataViewRowState.ModifiedCurrent) фильтр будет выглядеть так:

```
myDataView.RowStateFilter =
 ((DataViewRowState)((DataViewRowState.Added | DataViewRowState.ModifiedCurrent)));
```

Возможные значения свойства RowStateFilter приведены в таблице 17.

Свойство RowFilter предназначено для вывода записей, содержащих определенное значение заданного поля.

Таблица 17

Значения свойства RowStateFilter объекта DataView

Свойство	**Описание**
Unchanged	Записи без изменений
New	Новые записи
Deleted	Удаленные записи
Current Modified	Измененные записи с их текущими значениями
Original Modified	Измененные записи с их первоначальными значениями

В качестве примера установим фильтрацию по фамилии «Иванов»:

```
DataView myDataView = new DataView(bDTur_firmSQL2DataSet.Туристы);
myDataView.RowFilter = "Фамилия = 'Иванов'";
dataGridView2.DataSource = myDataView;
```

После запуска приложения во второй таблице выводятся все записи с фамилиями туристов «Иванов» (рис. 127).

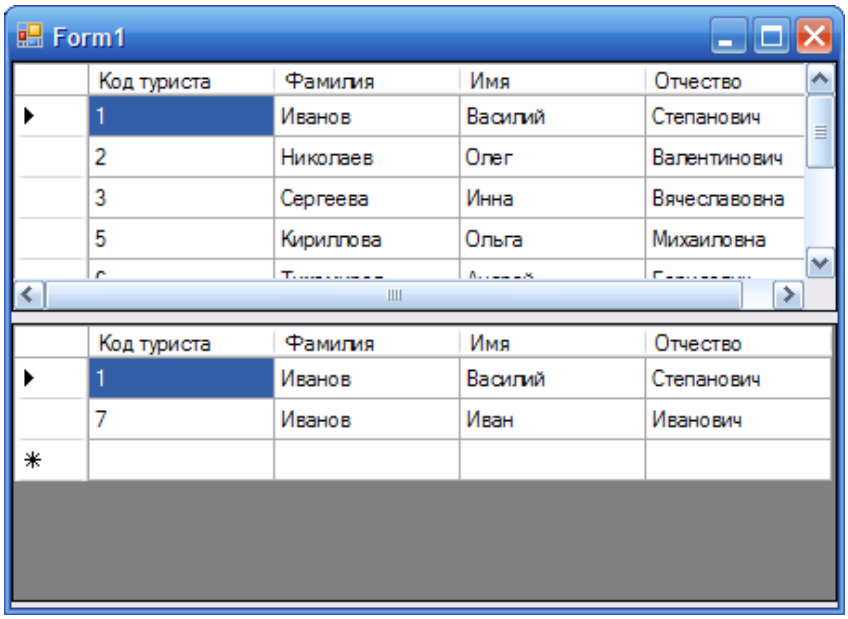


Рис. 127. Фильтрация записей по содержимому поля

Сортировка данных

Свойство Sort предназначено для вывода записей в порядке возрастания (ascending, ASC) или убывания (descending, DESC) по значениям заданного поля. В принципе, элемент DataGridView сам по себе поддерживает сортировку – достаточно просто щелкнуть по заголовку поля. Однако это требует действий от пользователя, тогда как объект DataView может предоставлять данные уже в готовом виде.

Удалим значение свойства RowFilter, в поле свойства Sort введем «Фамилия ASC» (результат на рис. 128):

```
DataView myDataView = new DataView(bDTur_firmSQL2DataSet.Туристы);
```

```
myDataView.Sort = "Фамилия, Имя ASC";
dataGridView2.DataSource = myDataView;
```

Рис. 128. Сортировка данных по полю «Фамилия»

Поиск данных

Технология ADO .NET предоставляет значительные возможности для поиска данных – такие объекты, как DataSet, DataTable, содержат специализированные методы для быстрого решения этой задачи. Свойство RowFilter объекта DataView может применяться для создания простого и эффективного поиска.

Запускаем Visual Studio 2008 и создаем новый проект типа Windows Forms Control Library (рис. 129).

Внешний вид формы в режиме дизайна представлен на рисунке 130.

Перейдем к коду. В конструкторе формы отключаем доступность текстового поля и привязываем обработчик события CheckedChanged для элемента CheckBox:

```
public FindCheckBox()
{
  InitializeComponent();
  txtColumnValue.Enabled = false;
  chbForSearching.CheckedChanged += new
                EventHandler(chbForSearching_CheckedChanged);
}
```

В классе формы создаем обработчик события CheckedChanged для CheckBox и определяем действия для значений текстового поля, надписи и элемента CheckBox.

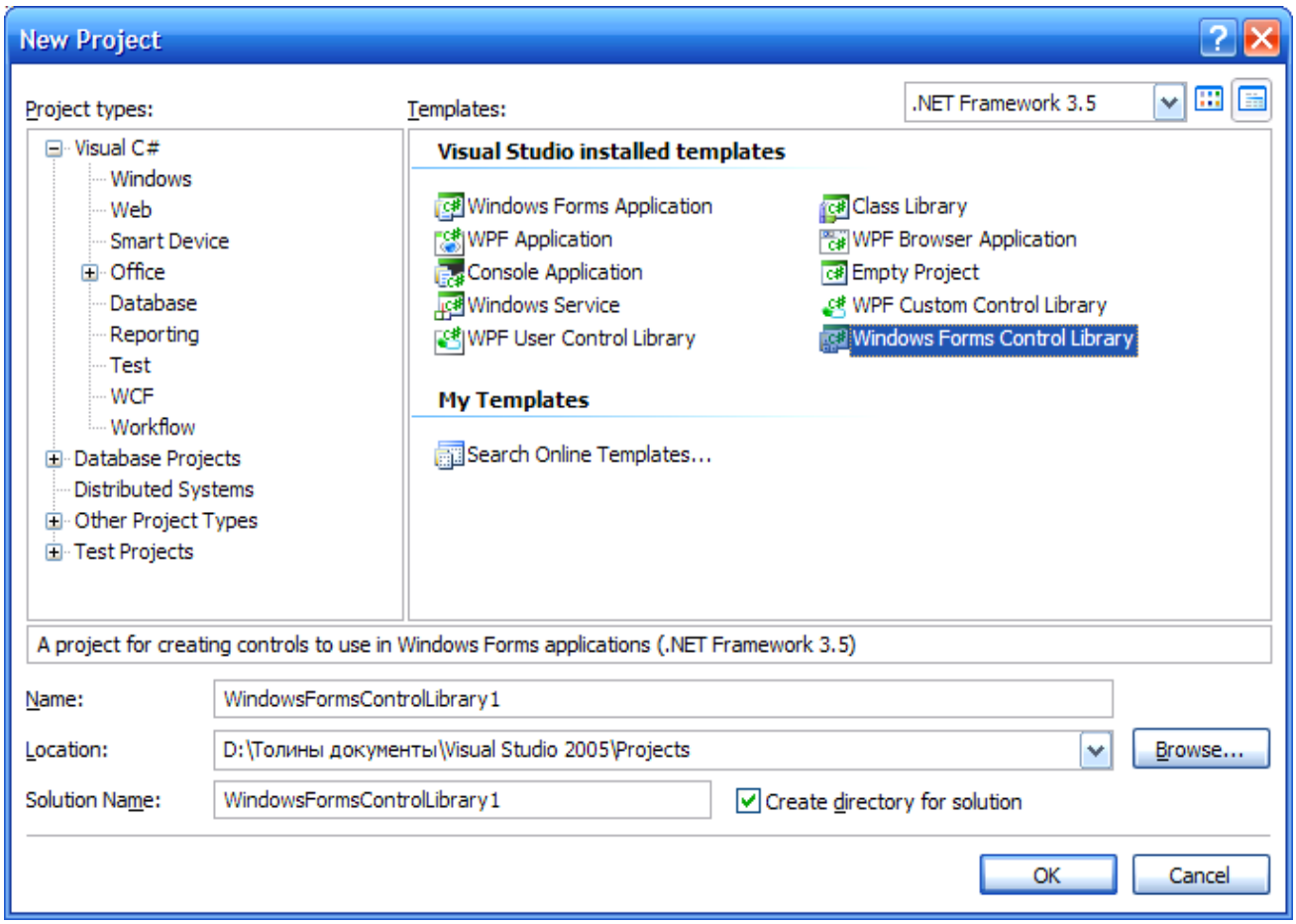


Рис. 129. Выбор шаблона Windows Forms Control Library

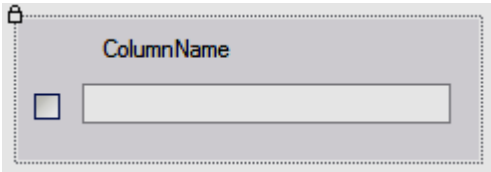


Рис. 130. Форма компонента в режиме дизайна

```
private void chbForSearching_CheckedChanged(object sender, EventArgs e)
{
        txtColumnValue.Enabled = chbForSearching.Checked;
}

 [Category("Appearance"),
        Description("Название поля, по которому будет осуществляться поиск")]
  public string ColumnName
  {
        get {return lblColumnName.Text;}
        set {lblColumnName.Text = value;}
  }
```

```
[Category("Appearance"), Description("Ключевое слово для поиска")]
  public string ColumnValue
 {
        get {return txtColumnValue.Text;}
        set {txtColumnValue.Text = value;}
  }

[Category("Appearance"), Description("Включение поиска по заданному полю")]
  public bool SearchEnabled
 {
        get {return chbForSearching.Checked;}
        set {chbForSearching.Checked = value;}
  }
}
```

Свойства ColumnName, ColumnValue и SearchEnabled будут свойствами композитного элемента управления, которые будут отображаться в его окне Properties. В квадратных скобках указан атрибут для помещения свойства в заданную группу и описание, выводимое на информационную панель.

Откомпилируем приложение и закроем его.

Создадим новое Windows-приложение, которое будет использовать созданный компонент. Расположим на форме следующие элементы управления: Panel, свойству Dock которого установим значение Left, Splitter и DataGridView (свойству Dock последнего устанавливаем значение Fill). Добавим созданный компонент в окно ToolBox и перетаскиваем на панель формы из окна Toolbox четыре копии элемента FindCheckBox. Вид формы приложения в режиме дизайна приведен на рисунке 131.

Обратите внимание, что в окне Properties, при групповом расположении, свойство ColumnName находится в группе Appearance. На информационную панель выводится описание этого свойства на русском языке. Именно эти параметры были указаны при создании композитного элемента.

Добавим обработчик кнопки «Поиск»:

```
private void btnSearch_Click(object sender, System.EventArgs e)
{
        try
        {
                FindCustomers();
        }
        catch(Exception ex)
        {
                MessageBox.Show(ex.Message);
        }
}
```

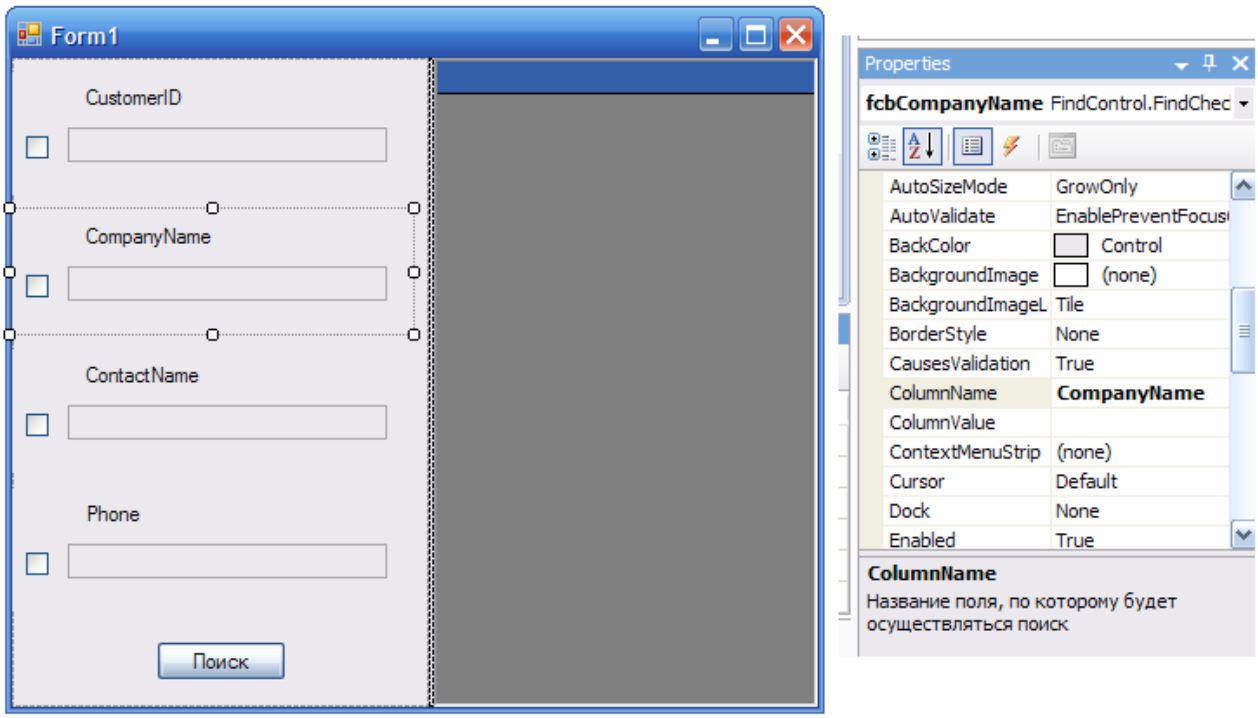


Рис. 131. Расположение элементов на форме

В классе формы создаем метод FindCustomers, в котором будет осуществляться обработка и который вызывается в обработчике кнопки «Поиск»:

```
private void FindCustomers()
{
//Создаем экземпляр filteringFields класса ArrayList
  ArrayList filteringFields = new ArrayList();
//Если элемент fcbCustomerID доступен для поиска
  if (fcbCustomerID.SearchEnabled)
//Добавляем в массив filteringFields значение текстового поля ColumnValue
        filteringFields.Add("CustomerID LIKE \'" + fcbCustomerID.ColumnValue + "%\'");

  if (fcbCompanyName.SearchEnabled)
        filteringFields.Add("CompanyName LIKE \'" + fcbCompanyName.ColumnValue + "%\'");

  if (fcbContactName.SearchEnabled)
        filteringFields.Add("ContactName LIKE \'" + fcbContactName.ColumnValue + "%\'");

  if (fcbPhone.SearchEnabled)
        filteringFields.Add("Phone LIKE \'" + fcbPhone.ColumnValue + "%\'");

  string filter = "";

//Комбинируем введенные в текстовые поля значения.
//Для объединения используем  логический оператор "ИЛИ"
  if (filteringFields.Count == 1)
        filter = filteringFields[0].ToString();
```

```
  else if (filteringFields.Count > 1)
        {
          for(int i = 0; i < filteringFields.Count – 1; i++)
                filter += filteringFields[i].ToString() + " OR ";
//Для объединения полей в запросе используем  логический оператор "И"
//                                        for(int i = 0; i < filteringFields.Count – 1; i++)
//                                        filter += filteringFields[i].ToString() + " AND ";

                filter += filteringFields[filteringFields.Count – 1].ToString();
        }
//Создаем экземпляр dvSearch класса DataView
  DataView dvSearch = new DataView(dsCustomers1.Customers);
//Передаем свойству RowFilter объекта DataView скомбинированное значение filter
  dvSearch.RowFilter = filter;
  dataGrid1.DataSource = dvSearch;
}
```

Как видно из приведенного кода, для объединения условий, накладываемых на поля, используется логический оператор «ИЛИ». Путем некоторого усложнения кода и созданного компонента для поиска можно создать возможность одновременного использования операторов «И» и «ИЛИ»

Запустим приложение.

Первый поисковый запрос будет запрашивать всего одно поле (рис. 132).

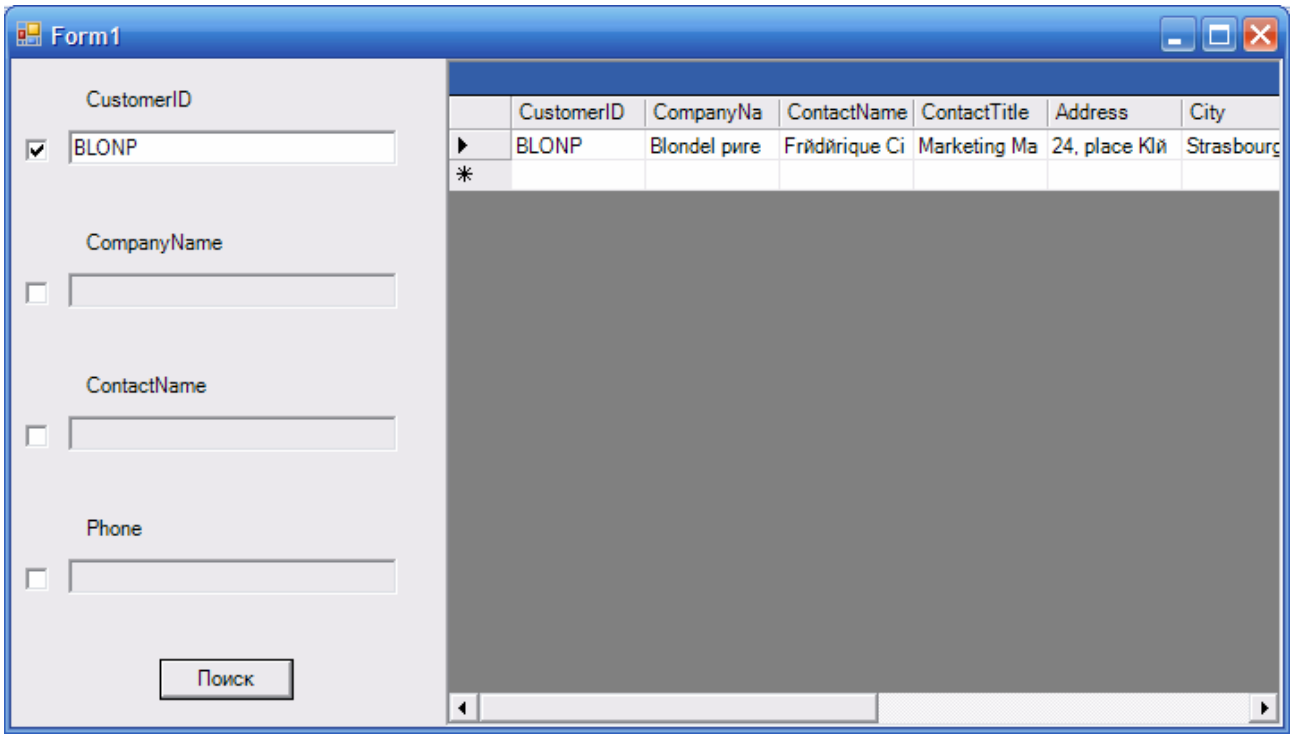


Рис. 132. Простой поисковый запрос

Усложним запрос: зададим условие на два поля (рис. 133).

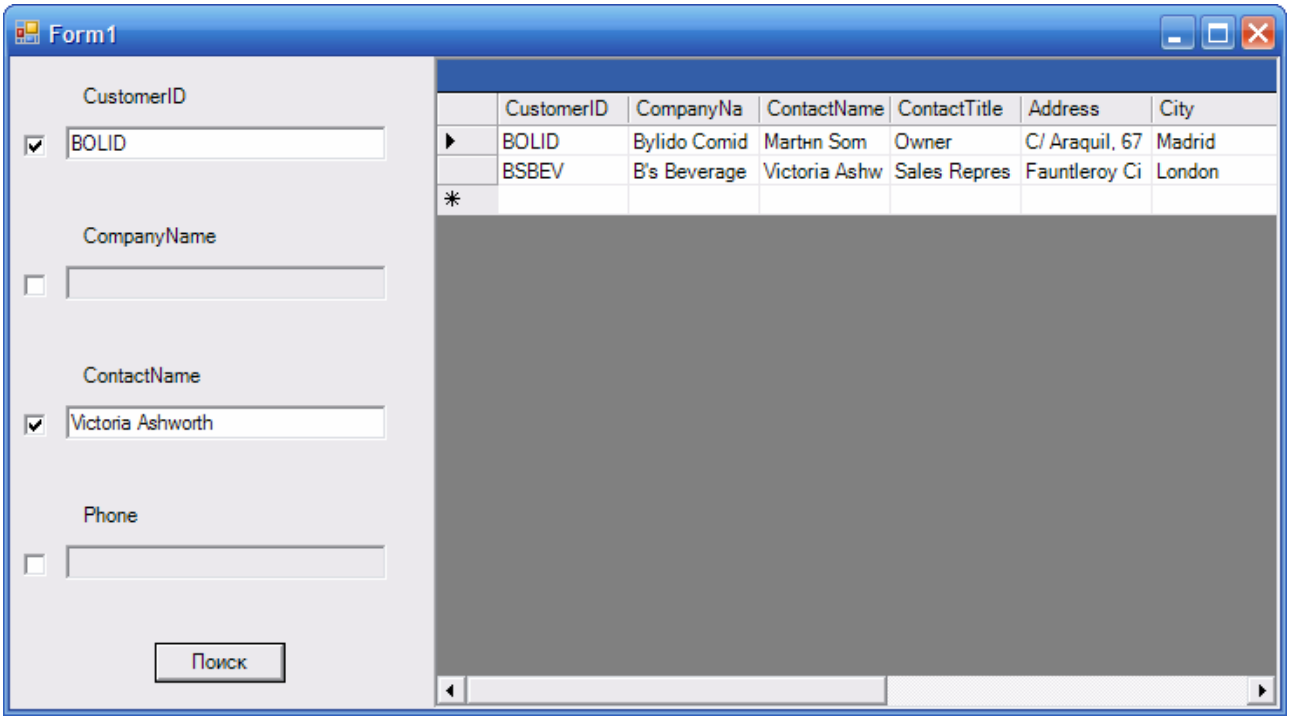


Рис. 133. Более сложный поисковый запрос

4.4.5. Вспомогательные классы

Класс HashTable

Hashtable – это структура данных, предназначенная для осуществления быстрого поиска. Это достигается за счет связывания ключа с каждым объектом, который сохраняется в таблице. Hashtable – это объект, в котором хранятся пары значений: так называемый ключ и само значение. Элементы каждой коллекции – и ключей (Keys), и значений (Values) – являются типом object, а это значит, что в качестве индекса элемента в привычном понимании теперь выступает не int, а именно object!

Создадим новое консольное приложение. Листинг этого приложения:

```
using System;
using System.Collections;

namespace HashtableExample
{   class Statistics
  {       public Hashtable AbonentList;
          public Statistics()
          {
                  AbonentList = new Hashtable();
          }
  }
```

```
 class Abonent
{
        public string Name;
        public int Phone;
        public Abonent(string n, int p)
        {
                Name = n; Phone = p;
        }
 }

 class Class1
{       [STAThread]
        static void Main(string[] args)
        {       Abonent a1 = new Abonent("Иванов", 1234567);
                Abonent a2 = new Abonent("Николаев", 3216547);
                Abonent a3 = new Abonent("Андреева", 685472);
                Abonent a4 = new Abonent("Волков", 1234500);
                Abonent a5 = new Abonent("Кириллова", 3245637);
                Statistics myStatistics = new Statistics();
                myStatistics.AbonentList.Add(a1.Phone, a1.Name);
                myStatistics.AbonentList.Add(a2.Phone, a2.Name);
                myStatistics.AbonentList.Add(a3.Phone, a3.Name);
                myStatistics.AbonentList.Add(a4.Phone, a4.Name);
                myStatistics.AbonentList.Add(a5.Phone, a5.Name);

                Console.WriteLine(myStatistics.AbonentList[685472]);
        }
 }
}
```

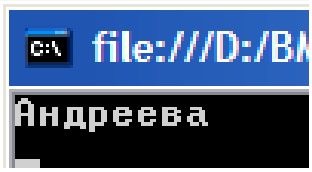


Рис. 134. Работа с классом HashTable

В методе Main создаются пять объектов класса Abonent, которые затем добавляются в Hashtable AbonentList (myStatistics.AbonentList) в коллекцию Values. Ключами для этих элементов будут служить значения их полей Phone. Обратите внимание, что метод Add() класса Hashtable требует два параметра: значение первого аргумента будет выступать в роли ключа для элемента, которым является значение второго аргумента.

Результатом выполнения программы будет вывод фамилии абонента, с заданным номером телефона (ключом) (рис. 134).

Класс ArrayList

Класс ArrayList, подобно классу Hashtable, определенный в пространстве имен System.Collections, представляет собой один из чрезвычайно простых и

удобных способов работы с наборами элементов. Объекты этого класса не имеют фиксированного размера и при необходимости могут менять его. Объект ArrayList при своем создании резервирует место в памяти для 16 элементов – указателей на тип object. При добавлении семнадцатого элемента размерность ArrayList увеличивается до 32 элементов. Обращение к объектам осуществляется аналогично обращению к элементам массива.

Создадим новое консольное приложение. Ниже приводится его полный его листинг.

```
using System;
using System.Collections;

namespace ClassArrayList
{
  class Class1
  {
        [STAThread]
        static void Main(string[] args)
        {
          ArrayList ar = new ArrayList();
          ar.Add("A");
          ar.Add("AB");
          ar.Add("ABC");
          ar.Add("ABCD");
          ar.Add("ABCDE");
          ar.Add("ABCDEF");
          ar.Add("ABCDEFG");
          ar.Add("ABCDEFGH");
          ar.Add("");
          ar.Add("");
          Console.WriteLine("Вывод элементов массива:\n");
          foreach (object element in ar)
         {
                Console.WriteLine(element);
          }
          ar.Remove("ABCD");
          Console.WriteLine("Удаление элемента:\n");
          foreach (object element in ar)
         {
                Console.WriteLine(element);
          }
          ar.Insert(6, "XYZ");
          Console.WriteLine("Вставка элемента \nна заданную позицию:\n");
          foreach (object element in ar)
         {
```

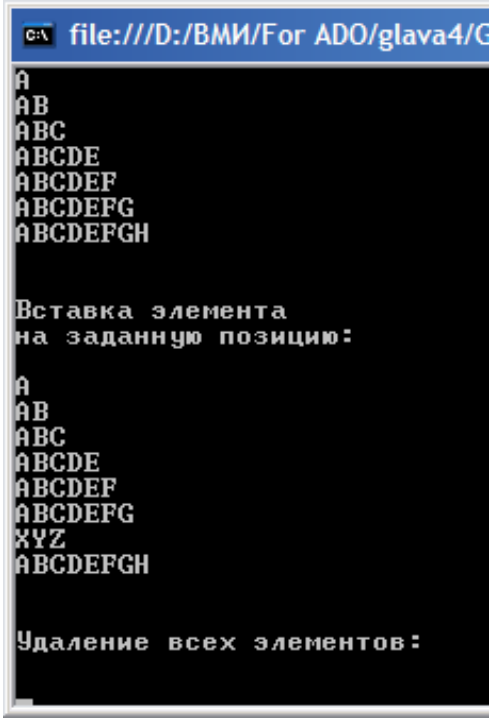


Рис. 135. Работа с классом ArrayList

```
  Console.WriteLine(element);
  }
  ar.Clear();
  Console.WriteLine("Удаление всех элемен-
тов:\n");
  foreach (object element in ar)
    {
        Console.WriteLine(element);
     }
 } // end Main()
} // end class
} // end namespace
```

Результат работы приложения приведен на рисунке 135.

Из рисунка 135 видно, что элементами ArrayList могут быть любые значения, поскольку он содержит в себе указатели на тип object. Для возвращения значений из массива необходимо осуществлять явное преобразование типов.

4.5. Контрольные вопросы и задания к разделу 4

1. Соберите приложение примера, описанного в п. 4.1.2.
2. Повторите приложение-пример из п. 4.1.3.
3. Создайте приложение, в котором для соединения с базой данных используйте строку с несуществующим именем файла БД, выведите на экран в обработчике ошибок подключения к БД номер ошибки, ее тип и причину.
4. Поэкспериментируйте на базе любого уже созданного приложения БД с пулом соединений. Что изменится в работе приложения при отказе приложения использовать пул соединений? Как влияют на его работу другие параметры управления пулом соединений (табл. 12)?
5. Постройте UML-диаграмму взаимодействия обектов или диаграмму последовательности, описывающую реализацию метода btnFill_Click (п. 4.1.3).
6. Исследуйте с помощью документции возможности использования события InfoMessage (табл. 9).
7. Создайте хранимые процедуры, описанные в примерах (табл. 13), и проверьте их работу.
8. Создайте приложение, вызывающее любую из хранимых процедур предыдущего (7-го) задания.
9. Создайте хранимые процедуры для запросов, разработанных вами по заданиям к разделу 2 (п. 2.6).

10. Создайте приложение, вызывающее любую из хранимых процедур предыдущего (9-го) задания.
11. Создайте хранимые процедуры, описанные в примерах (табл. 14), при помощи встроенного шаблона программы SQL Management Studio и проверьте их работу. Создайте приложение, вызывающее любую из этих хранимых процедур.
12. Создайте хранимые процедуры, описанные в примерах (табл. 15), с помощью средств среды Visual Studio 2008 и проверьте их работу. Создайте приложение, вызывающее любую из этих хранимых процедур.
13. Создайте элементарные приложения для вывода содержимого таблицы «Туристы» такие же, как показано на рисунках 75 и 76.
14. Испытайте оба способа создания и инициализации объекта Command (п.4.3.2) в приложениях задания 13. Соберите также оба варианта приложений: для работы с БД Microsoft Access и БД Microsoft SQL Server.
15. Самостоятельно соберите приложение, описанное в п. 4.3.2, для демонстрации использования свойств CommandType и CommandText объекта Command.
16. Каковы назначение и область применения методов ExecuteNonQuery, ExecuteScalar, ExecuteReader объекта Command?
17. Каковы назначение и способы применения параметризованных запросов к базе данных?
18. Расширьте функциональные возможности примера Windows-приложения, описанного в п. 4.3.3 (рис. 87 – 90), демонстрацией возможностей метода ExecuteScalar объекта Command добавлением вывода полей изменяемой записи БД и/или сообщения о соответствующем изменении БД в область главного окна приложения.
19. Каковы назначение и способы применения транзакций в БД?
20. Выделите в построенных вами по заданиям данного пособия запросах к БД возможные нарушения целостности данных и опишите данные запросы в виде танзакций. Исследуйте эти запросы с помощью консольных приложений, как это сделано в примере приложения EasyTransaction (рис. 101, 102).
21. Какие столбцы (имя и тип данных) добавлены в таблицу «Questions» и в таблицу «Variants» (см. «Программное создание объектов DataTable и DataColumn» в п 4.4.1)?
22. Какие ограничения БД можно задавать с помощью свойств объекта DataTable?
23. Опираясь на материал п. 4.4.1, разработайте приложение, создающее набор данных dsTests (объект DataSet), состоящий из двух связанных по полю questID таблиц (объектов DataTable) «Вопросы» (Questions) и «Варианты ответов» (Variants), при этом связь между таблицами должна иметь ограничения вида ForeignKeyConstraint.

24. В приложении, созданном по заданию 23, выведите обе связанные таблицы на главной форме приложения для ввода данных. Используйте метод GetChanges объекта DataSet для проверки наличия изменений для сохранения в БД.
25. Включите в приложение, созданное по заданию 24, программный ввод нескольких записей в таблицы БД. Используйте прямое присваивание новых значений с доступом к полям записи: а) по имени, б) по индексу, в) с помощью свойства ItemArray объекта DataRow, г) с помощью методов объекта DataRow BeginEdit и EndEdit.
26. Каково назначение свойств RowState, RowVersion объекта DataRow?
27. Какие события объекта DataTable позволяют отслеживать происходящие изменения в БД?
28. Добавьте в приложение, иллюстрирующее работу со свойством RowVersion (см. «Свойство RowVersion» в п. 4.4.2), обработку событий ColumnChanged и ColumnChanging, выводящих также сообщение в текстовое поле о себе (по аналогии с примером обработки событий RowChanged, RowChanging, RowDeleted, RowDeleting).
29. В приложении, созданном по заданию 24, выведите обе связанные таблицы в один элемент DataGridView, в два элемента DataGridView.
30. Каким образом можно осуществить вывод отфильтрованных данных? Как задаются условия фильтрации?
31. Каким образом можно осуществить вывод отсортированных данных? Как задаются условия сортировки?
32. Перепишите приложение, демонстрирующее возможности поиска в БД (см. «Поиск данных» в п. 4.4.4, рис. 132-133), для БД турфирмы.
33. Для чего нужны классы Hashtable, ArrayList? И как их использовать?

ЗАКЛЮЧЕНИЕ

В современном мире в основе любой информационной системы лежит база данных, а точнее СУБД. И выбор той или иной СУБД существенно влияет на функциональные возможности информационной системы и проектные решения. Предложение на рынке СУБД огромно, и перед разработчиком встает сложный выбор, какую СУБД использовать. Ситуация усугубляется при необходимости обеспечить поддержку различных источников данных, причем каждый из таких источников данных может хранить и обрабатывать данные по-своему. Кроме того, в различных языках программирования различна поддержка работы с той или иной СУБД. То есть, еще возникает проблема несоответствия обработки информации большинством СУБД и способам обработки информации различными языками программирования.

Решение выдвинутых проблем предлагается в рассмотренной в данном пособии технологии ADO .NET, разработанной компанией Microsoft, и включенной в их новую платформу .NET Framework. ADO .NET, как часть Microsoft .NET Framework, представляет собой набор средств и слоев, позволяющих приложению легко управлять и взаимодействовать со своим файловым или серверным хранилищем данных (рис 136).

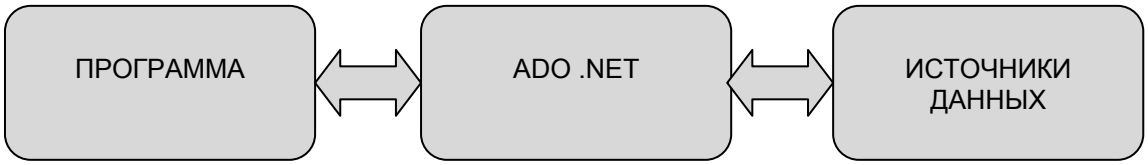


Рис. 136. Место ADO.NET в организации доступа к данным

ADO.NET наследует предыдущую технологию доступа к данным, разработанную Microsoft, которая называется классической ADO, или просто ADO. Хотя ADO .NET и ADO – это полностью различные архитектуры доступа к данным.

ADO.NET отличается от предыдущих технологий доступа к данным тем, что она позволяет взаимодействовать с базой данных автономно, с помощью отделенного от базы кеша данных. Автономный доступ к данным необходим, когда невозможно удерживать открытое физическое подключение к базе данных каждого отдельного пользователя или объекта.

Как и любая другая технология, ADO .NET состоит из нескольких важных компонентов. Все классы .NET группируются в пространства имен. Все функции, относящиеся к ADO .NET, находятся в пространстве имен System.Data. Кроме того, как и любые другие компоненты .NET, ADO .NET работает не изолировано и может взаимодействовать с различными другими компонентами .NET.

Архитектуру ADO .NET можно разделить на две фундаментальные части: подключаемую и автономную. Все классы в ADO .NET можно поделить по этому критерию. Единственное исключение составляет класс DataAdapter, который является посредником между подключенной и автономной частями ADO .NET.

Подключаемая часть ADO .NET представляет собой набор объектов подключений. Объекты подключений разделяются в ADO .NET по конкретным реализациям для различных СУБД. То есть для подключения к базе данных SQL SERVER имеется специальных класс SqlConnection. Эти отдельные реализации для конкретных СУБД называются *поставщиками данных* .NET.

В подключаемой части ADO.NET имеются следующие основные классы:

Connection. Этот класс, позволяющий устанавливать подключение к источнику данных (OleDbConnection, SqlConnection, OracleConnection).

Transaction. Объект транзакций (OleDbTransaction, SqlTransaction, OracleTransaction). В ADO .NET имеется пространство имен System.Transaction.

DataAdapter. Это своеобразный шлюз между автономными и подключенными аспектами ADO .NET. Он устанавливает подключение, и если подключение уже установлено, содержит достаточно информации, чтобы воспринимать данные автономных объектов и взаимодействовать с базой данных. (DataAdapter – SqlDataAdapter, OracleDataAdapter)

Command. Это класс, представляющий исполняемую команду в базовом источнике данных.

Parameter. Объект параметр команды.

DataReader. Это эквивалент конвейерного курсора с возможностью только чтения данных в прямом направлении.

Чтобы открыть подключение, необходимо указать, какая информация необходима, например, имя сервера, идентификатор пользователя, пароль и т. д. Поскольку каждому целевому источнику подключения может понадобиться особый набор информации, позволяющий ADO .NET подключится к источнику данных, выбран гибкий механизм указания всех параметров через строку подключения.

Строка подключения содержит элементы с минимальной информацией, необходимой для установления подключений, в виде последовательности пар «ключ – значение». Различные пары ключей-значений в строке подключений могут определять некоторые конфигурируемые параметры, определяющие поведение подключения. Сам объект подключения источника данных наследуется от класса DbConnection и получает уже готовую логику, реализованную в базовых классах.

Приложение должно разделять дорогостоящий ресурс – открытое подключение – и совместно использовать его с другими пользователями. Для этих целей введен пул подключений. По умолчанию пул подключений включен. При запросе ADO .NET неявно проверяет, имеется ли доступное неиспользуемое физическое подключение к базе данных. Если такое подключение имеется, то оно и используется. Для принятия решения, имеется ли такое физическое подключение или нет, ADO .NET учитывает загрузку приложения, и если поступает слишком много одновременных запросов, ADO .NET может удерживать одновременно открытыми несколько физических подключений, то есть увеличивать при необходимости количество подключений.

Вторым наиболее ресурсоемким объектом в ADO.NET являются транзакции, отвечающие за корректность изменений в БД. Транзакции – это набор операций, которые для обеспечения целостности и корректного поведения системы должным быть выполнены успешно или неудачно только все вместе. Обычно транзакции следуют определенным правилам, известным как свойства ACID, это: неделимость (Atomic), согласованность (Consistent), изолированность (Isolated) и долговечность (Durable). Для гибкого управления поведением тран-

закций используются уровни изоляции, которые описаны в перечислении IsolationLevel, это: Chaos, ReadUncommited, ReadCommited, RepeatableRead, Snapshot, Serializable.

В ADO .NET реализован мощный механизм поддержки транзакций БД. Сама технология ADO .NET поддерживает транзакции одиночной БД, которые отслеживаются на основании подключений. Но она может задействовать пространство имен System.Transactions для выполнения транзакций с несколькими БД или транзакций с несколькими диспетчерами ресурсов.

В ADO.NET класс подключений используется для начала транзакции.

Все управляемые в .NET поставщики, доступные в .NET Framework OleDb, SqlClient, OracleClient, ODBC имеют свои собственные реализации класса транзакций. Все эти классы реализуют интерфейс IDbTransaction из пространства имен System.Data.

Основное преимущество транзакций – производительность. При одиночных или коротких операциях транзакции выполняются медленнее, но для больших наборов данных они быстрее.

Одни лишь подключенные приложения не удовлетворяют всем требованиям, предъявляемым к современным распределенным приложениям. В автономных приложениях, созданных с помощью ADO .NET, используют иной подход. Для обеспечения автономности используются объекты DataAdapter. Они осуществляют выполнение запросов, используя для этого объекты подключения. А результаты выполнения, то есть данные, передает автономным объектам. Благодаря такому принципу автономные объекты не знают о существовании объектов подключения, так как напрямую не работают с ними. Таким образом, реализация объекта, хранящего данные, не зависит от конкретного поставщика данных, а именно от СУБД. Поскольку конкретная реализация адаптера данных зависит от соответствующего источника данных, конкретные адаптеры данных реализованы в составе конкретных поставщиков.

Автономные приложения обычно подключаются к базе как можно позже и отключаются как можно раньше. Важным элементом в такой схеме подключения и предоставления автономного доступа к данным является контейнер для табличных данных, который не знает о СУБД. Такой незнающий о СУБД автономный контейнер для табличных данных представлен в библиотеках ADO .NET классом DataSet или DataTable.

При работе в автономном режиме ADO.NET ведет пул реальных физических подключений для различных запросов, за счет которого достигается максимальная эффективность использования ресурсов подключения.

Напомним несколько основных классов автономной модели ADO .NET:

DataSet. Класс DataSet является ядром автономного режима доступа к данным в ADO.NET. Лучше всего рассматривать его с позиции, как будто в нем есть своя маленькая СУБД, полностью находящаяся в памяти.

DataTable. Больше всего этот класс похож на таблицу БД. Он состоит из объектов DataColumn, DataRow, представляющих из себя строки и столбцы.

DataView. Это объект представлений базы данных.

DataRelation. Этот класс позволяет задавать отношения между различными таблицами, с помощью которых можно проверять соответствие данных из различных таблиц.

Технология ADO.NET в полной мере способна предоставить механизм для доступа к любому источнику данных, тем самым давая разработчику мощный механизм взаимодействия с базами данных, способный в полной мере реализовать все потребности, возникающие при проектировании ИС.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ACID, 164
ADO .NET, 5, 163
  архитектура, 163
DCL, 93
  deny, 93
  grant, 93
  revoke, 93
DDL, 38, 92
  alter, 92
  as, 70
  create, 92
  create procedure, 70
  create table, 38
  drop, 80, 92
DML, 39
  delete, 39, 89
  insert, 39, 89
  update, 39, 89
ER-диаграмма, 10
SQL, 32
  BEGIN TRAN, 117
  between, 36
  COMMIT TRAN, 117
  DECLARE, 117
  from, 33
  group by, 36
  in…….., 36
  inner join, 73
  left join, 73
  like, 37
  null, 37
  order by, 35
  percent, 35
  ROLLBACK TRAN, 117
  select, 33
  where, 33, 36
SQL-функция, агрегирующая
  avg, 37, 94
  count, 36, 93
  max, 37, 94
  min, 37, 94
  sum, 37
Windows-приложение, 42, 56, 60, 61, 88, 98, 108, 112, 113, 129, 133, 137, 138, 145, 150, 154
атрибут, 7, 11
база данных (БД), 7
база данных, реляционная (РБД), 7
база данных, структура, 8
зависимость
  многозначная, 13
  функциональная, 13
  частичная, 13
запись, 7
запрос, 32
  параметризированный, 97
значение
  Bottom, 104
  Cascade, 127
  Detached, 137
  Details, 104
  Fill, 88
  input, 113
  InputOutput, 113
  None, 128
  output, 113
  ReturnValue, 113
  SetDefault, 128
  SetNull, 128
  StoredProcedure, 86
  TableDirect, 86
  Text, 86
исключение, 132
источник данных, 44
  Database, 44
  Object, 44
  Service, 44
  связь (Binding Source), 49
класс
  ArrayList, 158
  Hashtable, 157
  OleDbCommand, 87
  OleDbConnection, 87
  OleDbDataAdapter, 87
  OleDbException, 65
  Parameters, 113
  SqlCommand, 88
  SqlConnection, 88
  SqlDataAdapter, 88
  SqlError, 64
  SqlException, 64
  TableAdapter, 53
ключ, 17
  внешний, 12
  вторичный, 38
  главный, 38

первичный, 7, 12, 127
составной, 12
компонент
Binding Source, 42
BindingNavigator, 50
CheckBox, 152
DataGridView, 47, 56, 129
DataSet, 47
ListBox, 104, 113
ListView, 104
Panel, 50
Splitter, 104, 113
TableAdapter, 48
TextBox, 50
массив значений полей, 135
метод
AcceptChanges, 138
Add, 99, 125, 158
BeginEdit, 135
ClearErrors, 132
Close, 61
CreateCommand, 87
Delete, 136
Dispose, 61
EndEdit, 135
ExecuteNonQuery, 89, 91
ExecuteReader, 95, 104
ExecuteScalar, 93, 104
Fill, 54
Get<ТипДанных>, 105
GetChanges, 130
HasVersion, 139
InitializeComponent, 56, 86
Read, 96
RejectChanges, 132, 138
Remove, 136
Update, 53
модель
трехуровневая, 65
модель БД
логическая, 10
физическая, 15
нормализация РБД, 12
нормальная форма
(1НФ) первая, 13
(2НФ) вторая, 13
(3НФ) третья, 13
(4НФ) четвертая, 13
объект
Command, 86
создание, инициализация, 87
Connection, 60, 87, 112
Constraint, 123
DataAdapter, 54
DataColumn, 53, 123, 129, 130
DataReader, 95, 96, 105
DataRelation, 53, 123, 126
DataRelationCollection, 53
DataRow, 123, 133, 135, 137
DataSet, 53, 123
DataTable, 53, 106, 123, 133
DataTable.DefaultView, 133
DataTableCollection, 53
DataView, 133, 150
DataViewManager, 146
DBCommand, 54
DBConnection, 54
ForeighKeyConstraint, 123
Rows, 53
UniqueConstraint, 123
объект DataRelation, 146
отношение, 12
ошибки SQL Server, 64
уровень, 64
параметр
CommandText, 56, 85
ConnectionString, 56, 59
поле, 7
вычисляемое, 129
правило
AcceptRejectRule, 127
DeleteRule, 127
UpdateRule, 127
приложение, консольное, 89, 93, 95, 119, 157, 159
пространство имен, 56
System.Collections, 158
System.Data, 56
System.Data.OleDb, 56
System.Data.SqlClient, 56
процедура, хранимая, 67, 113
без параметров, 71
выполнение, 83, 108
с параметрами
входными, 75
входными и выходными, 78
шаблон, 80, 83
псевдоним, 78
пул соединений, 66
параметры, 67
свойства, группа
DataBindings, 51

свойство
Advanced, 51
AllowDBNull, 127
AutoIncrement, 125
AutoSize, 61
BindingSource, 50
Columns, 104, 125
CommandType, 86
Connection, 86
Constraint, 127
DataSource, 43
DataType, 53
Direction, 113
Dock, 56, 88, 104
Errors, 65
Expression, 130
ForeignKeyConstraint, 127
GridLines, 104
HasErrors, 132
ItemArray, 135
Message, 65
NativeError, 65
PrimaryKey, 126
Relations, 53
RowFilter, 150, 152
Rows, 135
RowState, 137
RowStateFilter, 150
RowVersion, 138, 139
SelectCommand, 145
Sort, 151
SQL State, 65
TableMappings, 145
Tables, 53
Unique, 125, 127
UniqueConstraint, 127
View, 104
связь, 19
«многие-ко-многим», 20
«один-к-одному», 20
«один-ко-многим», 20
словарь данных, 9
событие
CheckedChanged, 152
ColumnChanged, 143
ColumnChanging, 143
Disposed, 61
InfoMessage, 61
RowChanged, 143
RowChanging, 143
RowDeleted, 143
RowDeleting, 143
StateChange, 61
строка соединения
параметры, 59
схема данных, 12
таблица, 15
дочерняя, 20
тип данных, 16, 24
согласованный, 19
транзакция, 115
выполнение, 120
проблемы выполнения, 121
Dirty reads, 121
Non-repeatable reads, 121
Phantom reads, 121
уровень изоляции, 121
Chaos, 123
Read committed, 121
Read uncommitted, 121
Repeatable read, 122
Serializable, 122
Unspecified, 123
установка, 122, 123
управление данными, 8
функциональная спецификация БД, 10
функция, агрегирующая, 36

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. ADO .NET – Википедия [Электронный ресурс]. – Режим доступа: http://ru.wikipedia.org/wiki/ADO.NET/, свободный. – Загл. с экрана.
2. ADO. NET [Электронный ресурс] / П. В. Ветров, Тюменский государственный университет. Институт математики и компьютерных наук. – Режим доступа: http://www.codenet.ru/db/other/ado-dot-net/, свободный. – Загл. с экрана.
3. ADO.NET: Обзор технологии [Электронный ресурс]. – Режим доступа: http://www.cyberguru.ru/dotnet/ado-net/adonet-overview.html, свободный. – Загл. с экрана.
4. C# 2005 для профессионалов / К. Нейгел, Б. Ивьен, Д. Глинн, К. Уотсон, М. Скиннер, А. Джонс. – Москва; Санкт-Петербург; Киев: «Диалектика», 2007.
5. Воройский, Ф. С. Информатика. Новый систематизированный толковый словарь-справочник / Ф. С. Воройский. – М. : ФИЗМАТЛИТ, 2003.
6. Кариев, Ч. А. Разработка Windows-приложений на основе Visual C# / Ч. А. Кариев. – М. : БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий – ИНТУИТ.ру, 2007.
7. Кариев, Ч. А. Технология Microsoft ADO. NET / Ч. А. Кариев. – М. : БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий – ИНТУИТ.ру, 2007.
8. Классы, интерфейсы и делегаты в. C# 2005 : учебное пособие / сост. О. Н. Евсеева, А. Б. Шамшев. – Ульяновск : УлГТУ, 2008.
9. Лабор, В. В. Си Шарп создание приложений для Windows / В. В. Лабор. – Минск, Харвест, 2003.
10. Марченко, А. Л. Основы программирования на C# 2.0 / А. Л. Марченко. – М. : БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий – ИНТУИТ.ру, 2007.
11. Основы языка C# 2005 : учебное пособие / сост. О. Н. Евсеева, А. Б. Шамшев. – Ульяновск: УлГТУ, 2008.
12. Петцольд, Ч. Программирование для Microsoft Windows на C# / Ч. Петцольд. В 2 т. : пер. с англ. – М. : Издательско-торговый дом «Русская Редакция», 2002.

Учебное издание

РАБОТА С БАЗАМИ ДАННЫХ НА ЯЗЫКЕ C#. ТЕХНОЛОГИЯ ADO .NET
Учебное пособие

Составители ЕВСЕЕВА Ольга Николаевна
ШАМШЕВ Анатолий Борисович

Редактор Н. А. Евдокимова

ЛР № 020640 от 22.10.97.
Подписано в печать 29.10.2009. Формат 60×84/16.
Усл. печ. л. 10,23. Тираж 100 экз. Заказ 1200.

Ульяновский государственный технический университет, 432027, г. Ульяновск, Сев. Венец, 32.
Типография УлГТУ, 432027, г. Ульяновск, Сев. Венец, 32.