РАЗДЕЛ 1. ВВЕДЕНИЕ В ВЕБ-ТЕХНОЛОГИИ АВТОМАТИЗАЦИИ ПРЕДПРИЯТИЙ И ПРОИЗВОДСТВ

ЛЕКЦИЯ 1. ПЛАТФОРМА .NET. ТЕХНОЛОГИЯ ASP.NET. ПЛАТФОРМА ASP.NET WEBFORMS

ASP.NET (Active Server Pages для .NET) — технология создания веб-приложений и веб-сервисов от компании Майкрософт. Она является составной частью платформы Microsoft .NET и развитием более старой технологии Microsoft ASP.

ASP.NET внешне во многом сохраняет схожесть с более старой технологией ASP, что позволяет разработчикам относительно легко перейти на ASP.NET. В то же время внутреннее устройство ASP.NET существенно отличается от ASP, поскольку она основана на платформе .NET и, следовательно, использует все новые возможности, предоставляемые этой платформой.

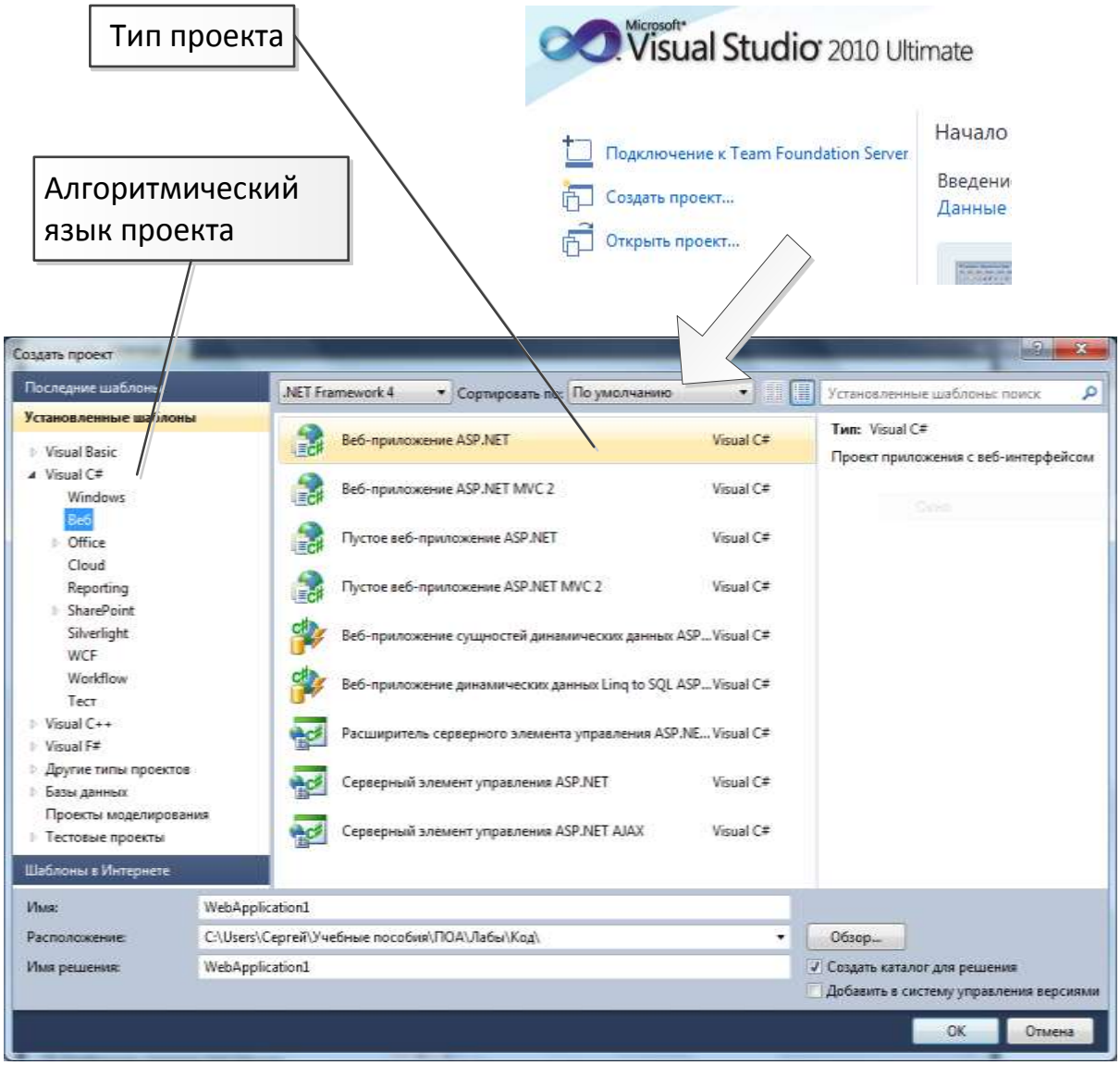


Рис. 2. Создание нового проекта в Visual Studio

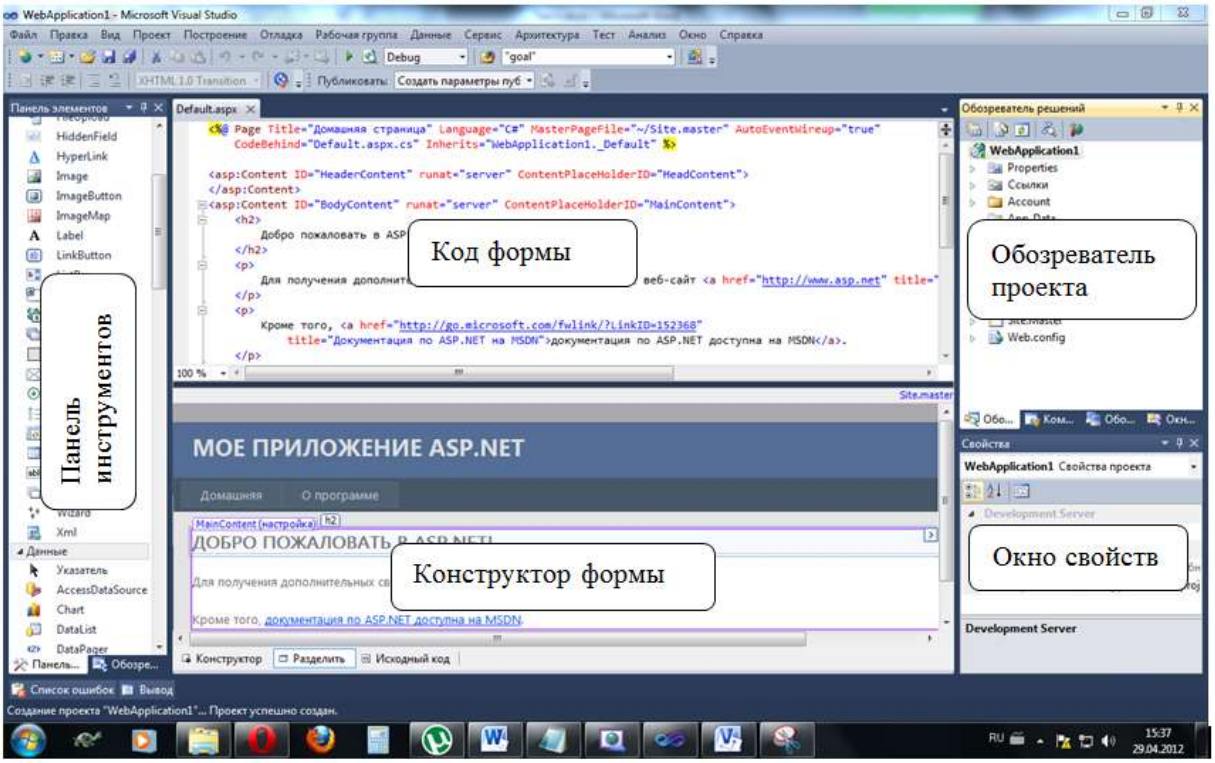
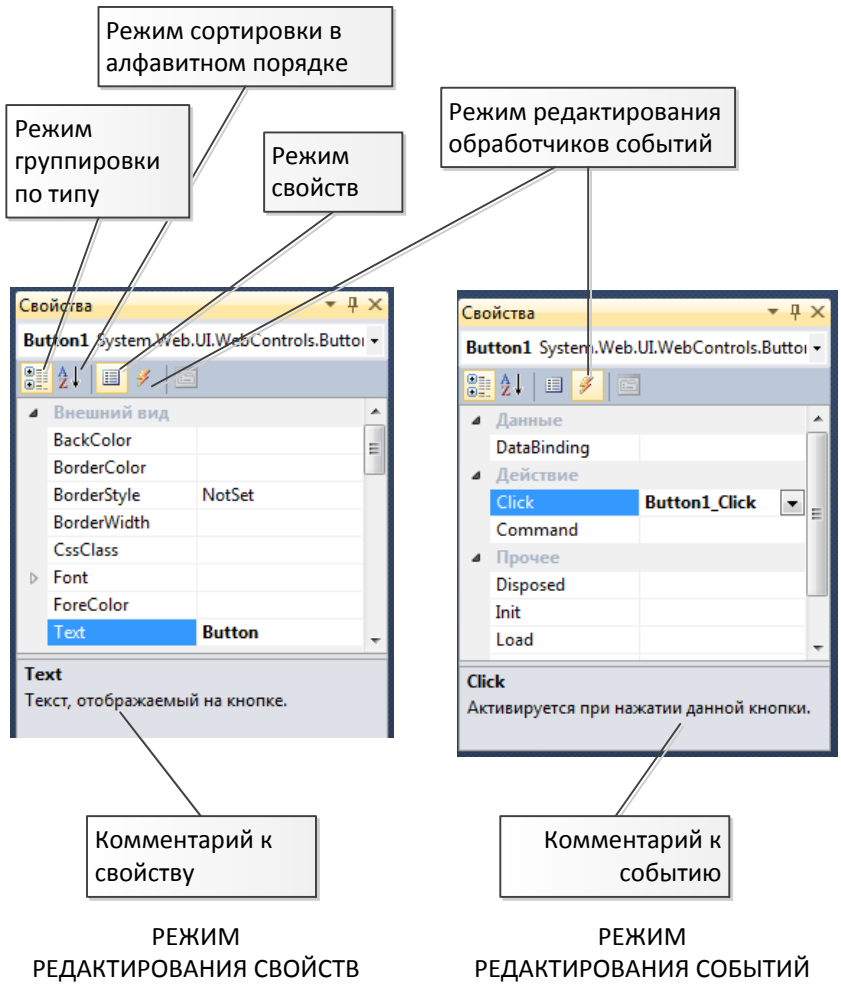


Рис. 3. Внешний вид созданного проекта в Visual Studio



РЕЖИМ РЕДАКТИРОВАНИЯ СВОЙСТВ

РЕЖИМ РЕДАКТИРОВАНИЯ СОБЫТИЙ

Рис. 4. Окно свойств элемента управления

Создание базы данных

После того, как создан проект приложения, необходимо создать базу данных, в которой будут храниться объекты объектной модели предметной области (точнее, значения атрибутов, определяющих состояние этих объектов). Последовательность создания базы данных следующая.

1. Создать новую базу данных
    a. Запустить Sql Server Management Studio
    b. Выполнить подключение к серверу БД (рис. 5)
    c. Выделить на дереве проектов «Базы данных», нажать правую кнопку мыши и выбрать «создать базу данных» (рис. 6)
    d. Ввести имя базы данных (латиницей) (рис. 6)
2. В проводнике проекта открыть вкладку «Обозреватель баз данных» и подключить вновь созданную базу данных (рис. 7)

Рис. 5. Подключение к серверу БД

Создание объектной (сущностной) модели

Объектная модель предметной области является основой современного программного приложения. В платформе Microsoft.NET существует

мощное средство создания и работы с объектными (сущностными) моделями – Entity Framework.

Процесс создания объектной модели следующий:

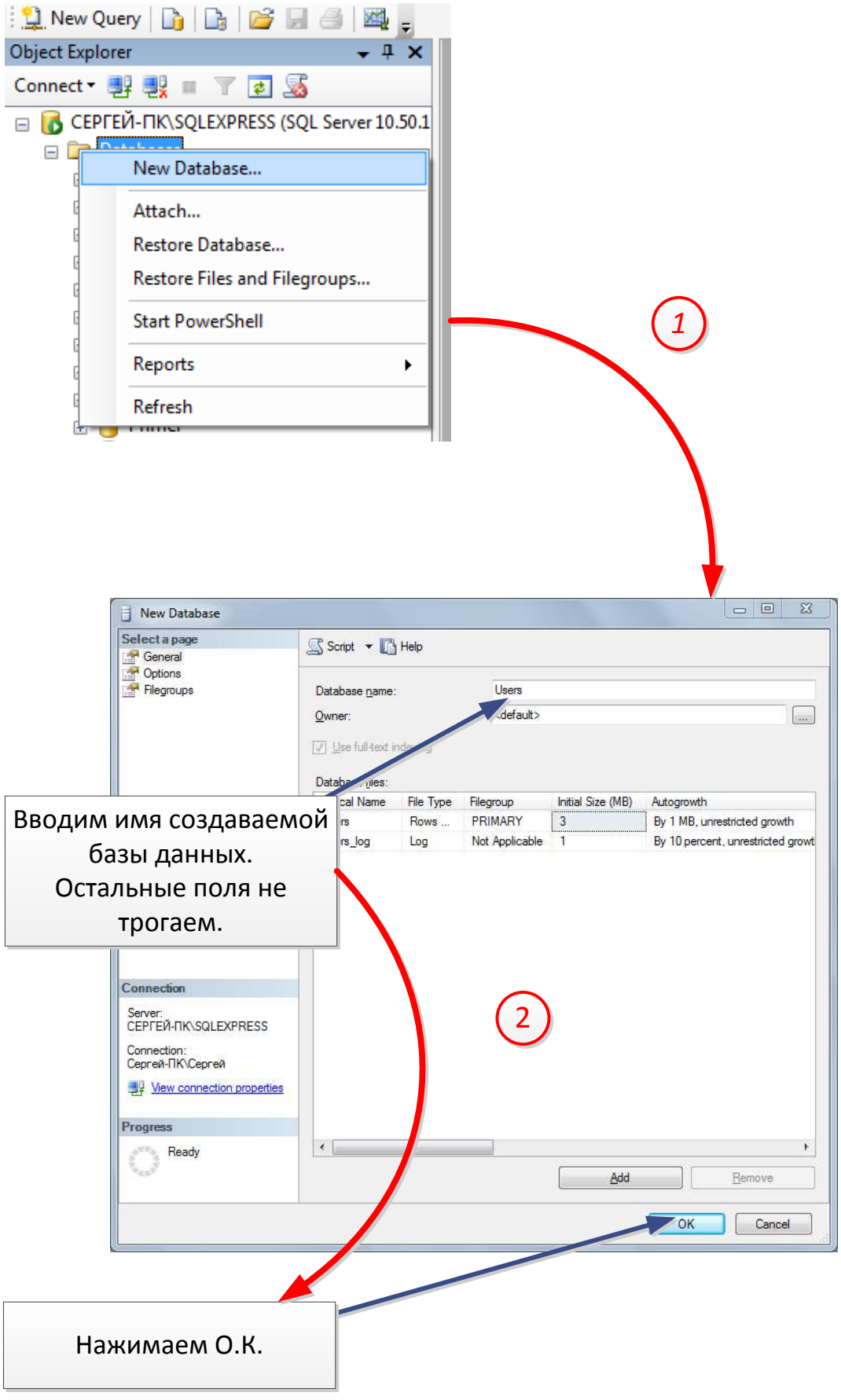


Рис. 6. Создание новой базы данных

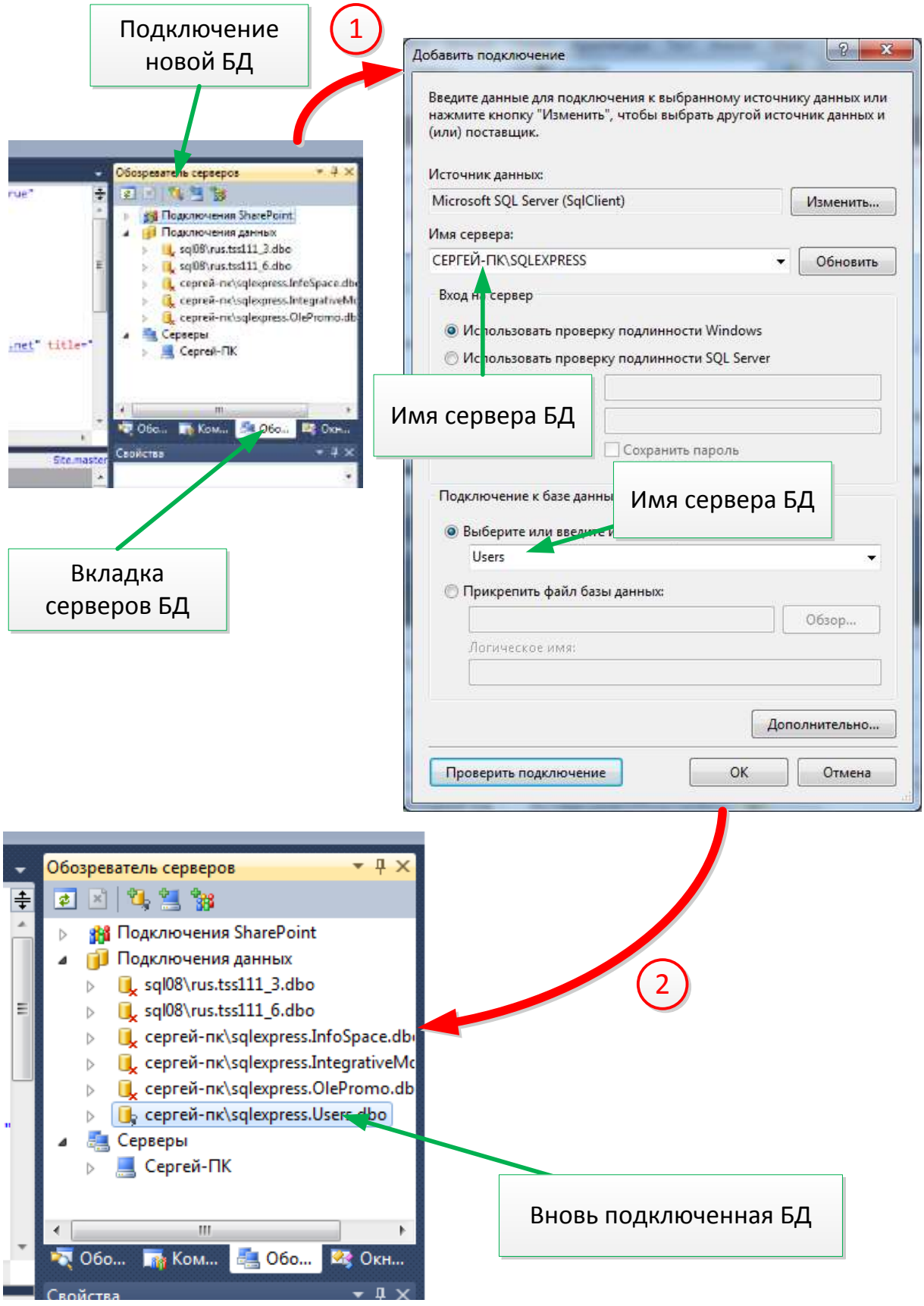


Рис. 7. Подключение базы данных к Visual Studio

1. В конструкторе моделей создать модель предметной области (Entity Model) (рис. 8)
    a. Создать новый сущностный класс (Сущность) (рис. 9)
    b. Выбрать имя сущности (рис. 9)
    c. Выбрать имя набора сущностей (рис. 9)
    d. В качестве типа идентификатора использовать Guid и не забывать устанавливать поле Store Generated Pattern в None (рис. 10)
    e. Создать другие сущностные классы
    f. Создать ассоциации и указать их кратность (рис. 11)
2. Сформировать SQL код базы данных (рис. 12)
    a. Установить курсор на пустое пространство объектной модели (рис. 12)
    b. Нажать правую кнопку мыши и выбрать пункт «Создать базу данных на основе модели» (рис. 12).
3. Выполнить сгенерированный SQL код.

Для этого, в случае использования MS Visual Studio Professional 2010 или выше, необходимо выбрать файл EModel.edmx.sql в дереве проекта, на поле кода нажать правую кнопку мыши, и выбрать пункт «Выполнить SQL». Далее, в процессе диалога выбрать требуемую базу данных. В случае использования Web Developer для создания базы данных необходимо воспользоваться Sql Server Management Studio.

Ниже приведем сгенерированный SQL-код.

```
-- --------------------------------------------------
-- Entity Designer DDL Script for SQL Server 2005, 2008,
and Azure
-- --------------------------------------------------
-- Date Created: 09/23/2012 15:39:28
-- Generated from EDMX file: C:\Users\Сергей\Учебные посо-
бия\ПОА\Курсовая\Users\Users\EModel.edmx
-- --------------------------------------------------

SET QUOTED_IDENTIFIER OFF;
GO
USE [Users];
GO
IF SCHEMA_ID(N'dbo') IS NULL EXECUTE(N'CREATE SCHEMA
[dbo]');
GO
```

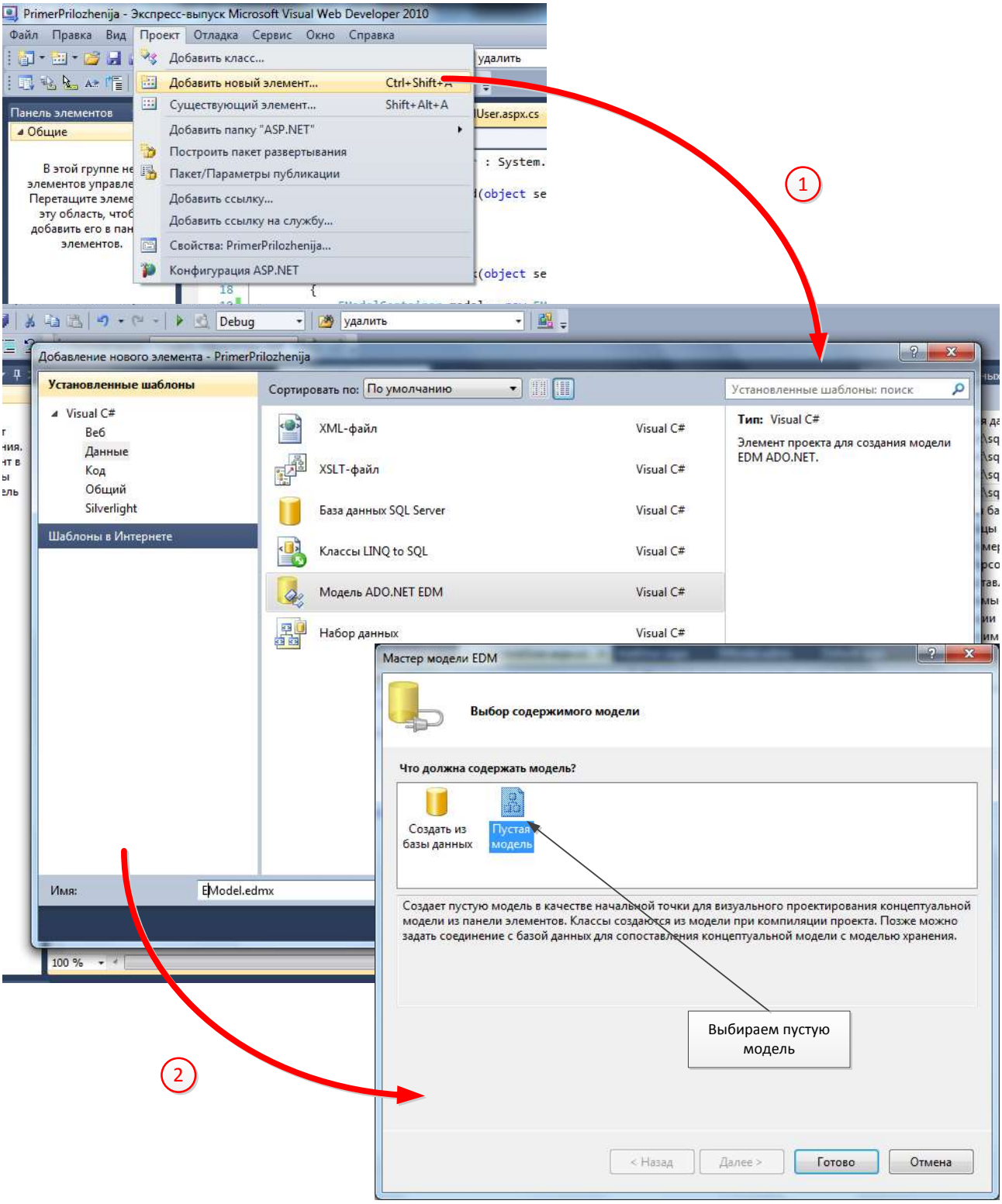


Рис. 8. Создание объектной модели

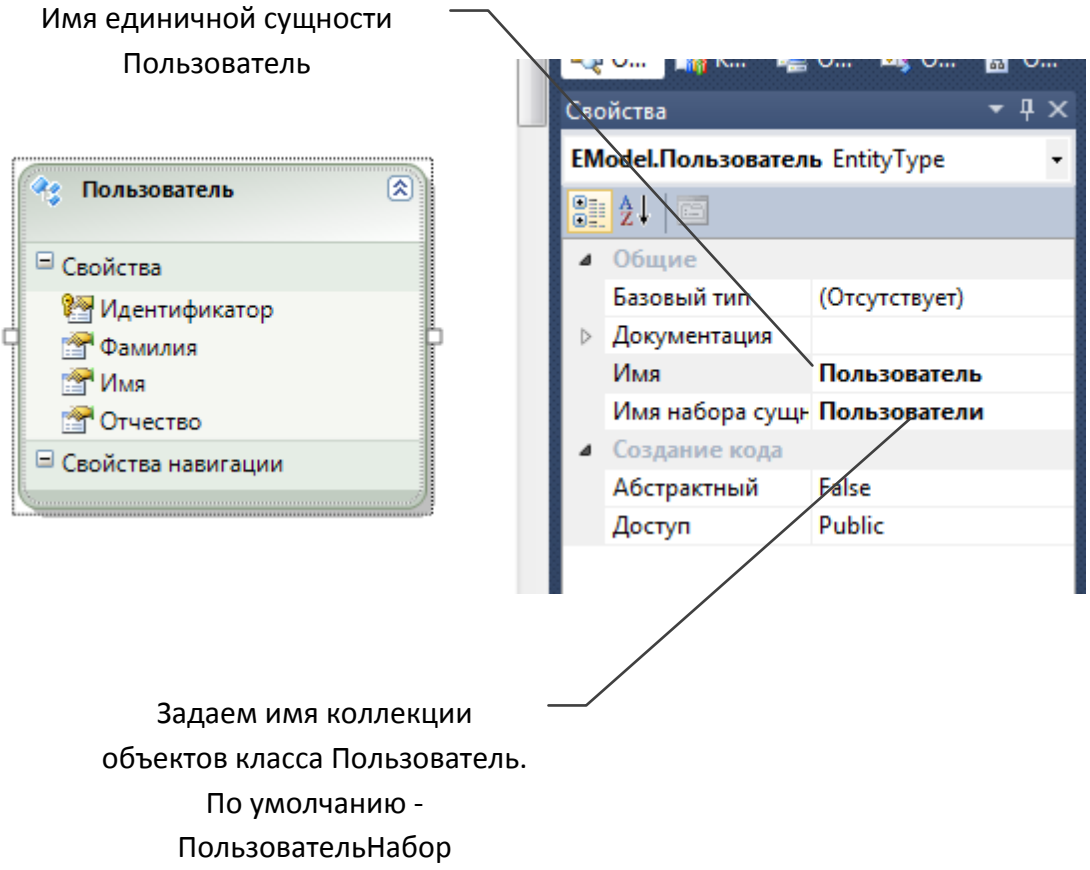


Рис. 9. Изменение имени коллекции объектов класса Пользователь

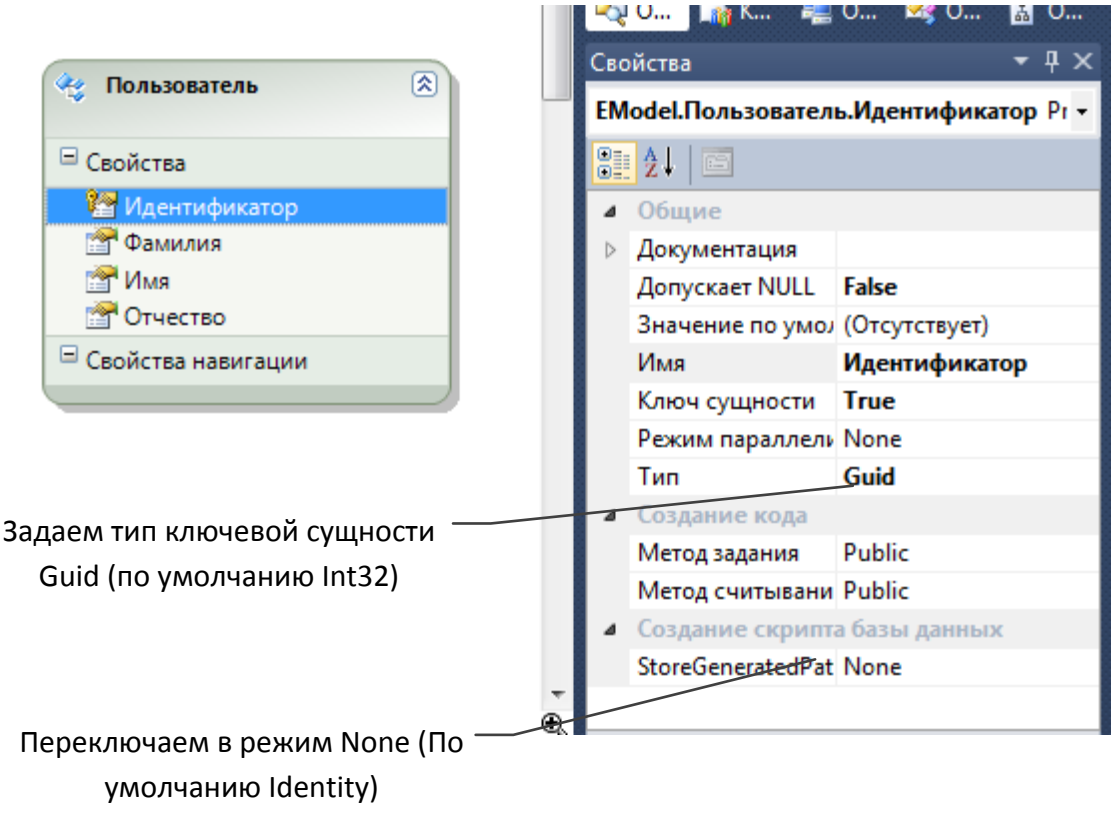


Рис. 10. Изменение типа ключевого свойства

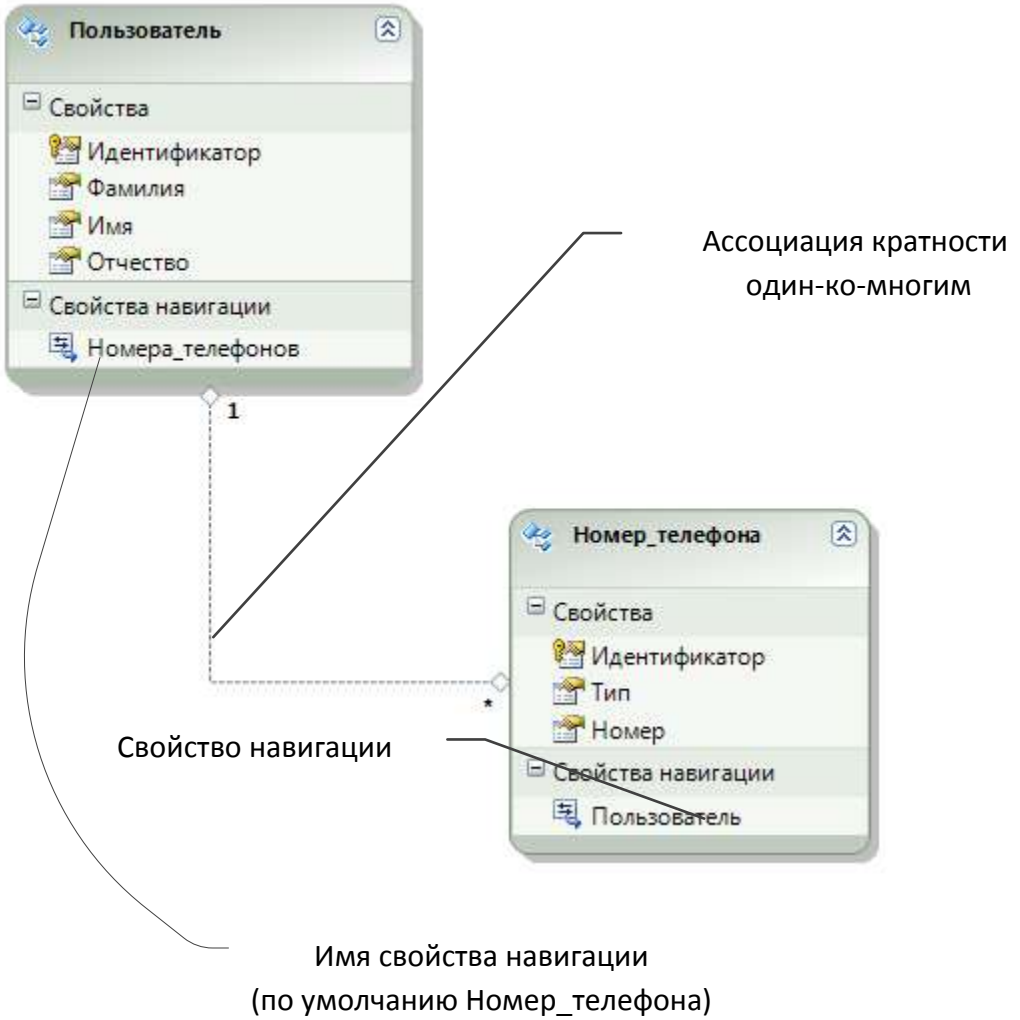


Рис. 11. Диаграмма объектной модели и изменение свойств навигации

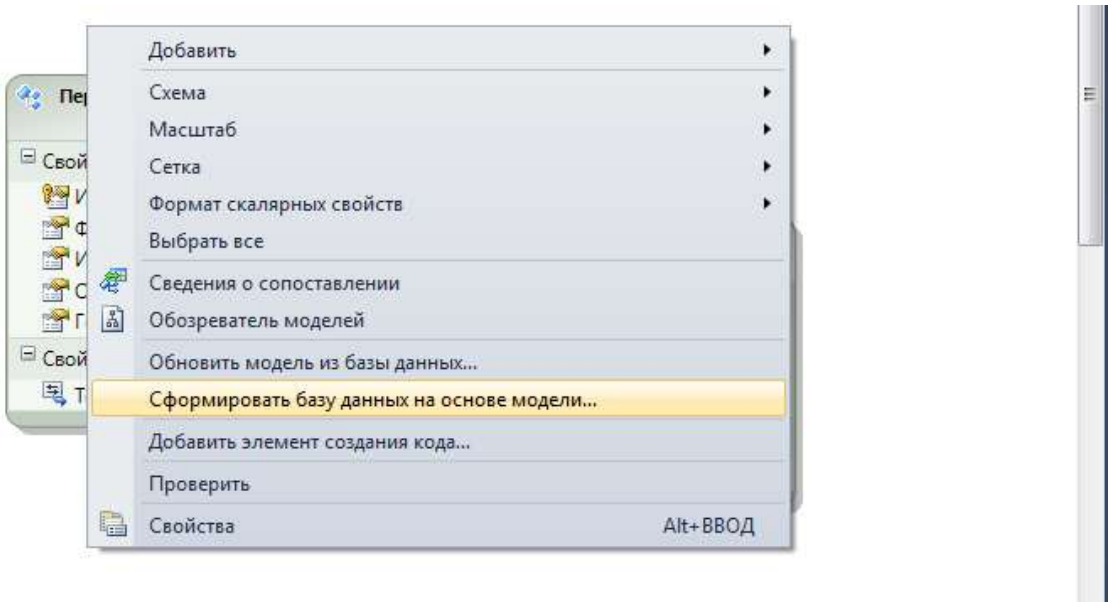


Рис. 12. Генерация SQL кода структуры базы данных

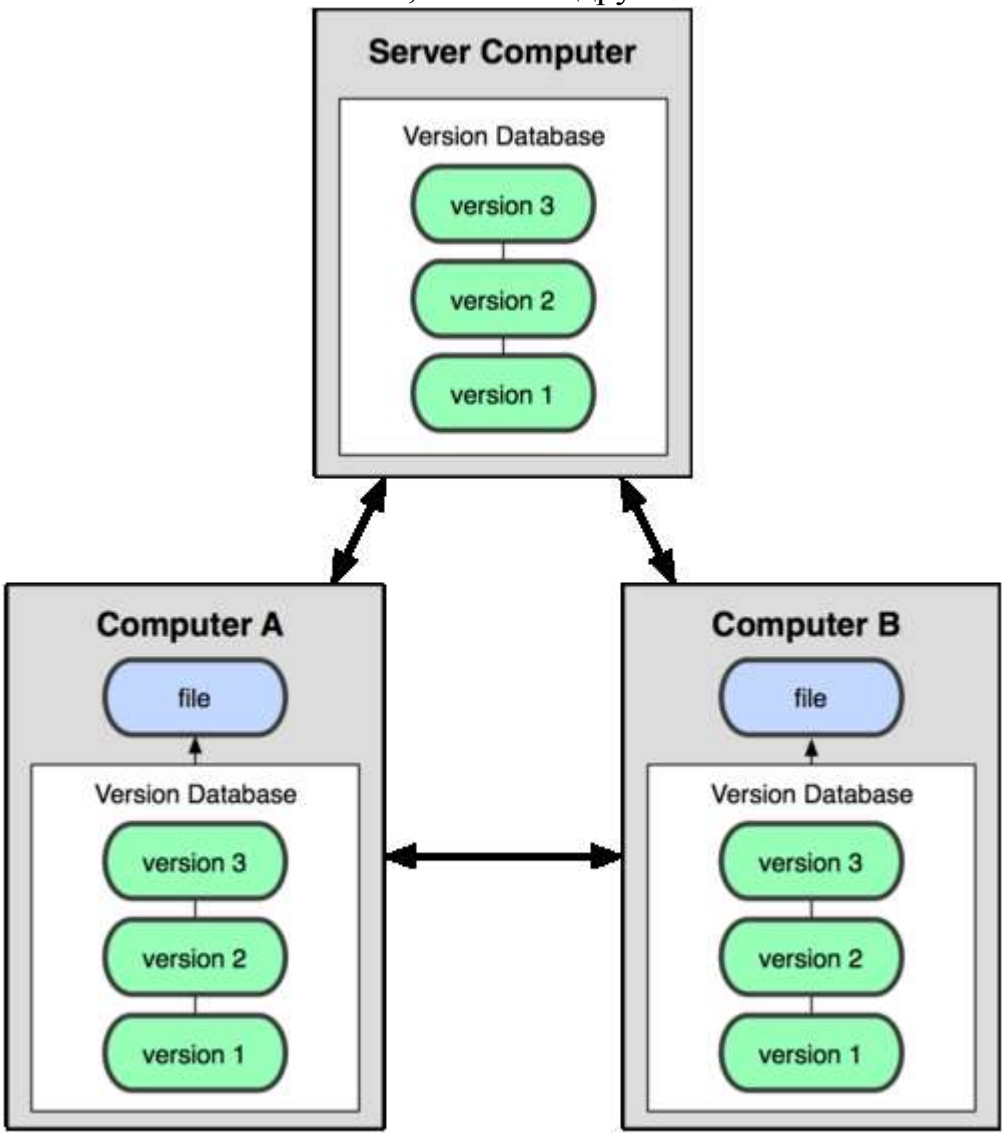
```
-- Dropping existing FOREIGN KEY constraints
-- --------------------------------------------------

-- --------------------------------------------------
-- Dropping existing tables-- ----------------------------
--------------------

-- --------------------------------------------------
-- Creating all tables
-- --------------------------------------------------

-- Creating table 'Пользователи'
CREATE TABLE [dbo].[Пользователи] (
    [Идентификатор] uniqueidentifier  NOT NULL,
    [Фамилия] nvarchar(max)  NOT NULL,
    [Имя] nvarchar(max)  NOT NULL,
    [Отчество] nvarchar(max)  NOT NULL
);
GO

-- Creating table 'Номера_телефонов'
CREATE TABLE [dbo].[Номера_телефонов] (
    [Идентификатор] uniqueidentifier  NOT NULL,
    [Тип] nvarchar(max)  NOT NULL,
    [Номер] nvarchar(max)  NOT NULL,
    [Пользователь_Идентификатор] uniqueidentifier  NOT NULL
```

```
);
GO

-- ------------------------------------------------
-- Creating all PRIMARY KEY constraints
-- ------------------------------------------------

-- Creating primary key on [Идентификатор] in table 'Поль-
зователи'
ALTER TABLE [dbo].[Пользователи]
ADD CONSTRAINT [PK_Пользователи]
    PRIMARY KEY CLUSTERED ([Идентификатор] ASC);
GO

-- Creating primary key on [Идентификатор] in table
'Номера_телефонов'
ALTER TABLE [dbo].[Номера_телефонов]
ADD CONSTRAINT [PK_Номера_телефонов]
    PRIMARY KEY CLUSTERED ([Идентификатор] ASC);
GO

-- ------------------------------------------------
-- Creating all FOREIGN KEY constraints
-- ------------------------------------------------

-- Creating foreign key on [Пользователь_Идентификатор] in
table 'Номера_телефонов'
ALTER TABLE [dbo].[Номера_телефонов]
ADD CONSTRAINT [FK_ПользовательНомер_телефона]
    FOREIGN KEY ([Пользователь_Идентификатор])
    REFERENCES [dbo].[Пользователи]
        ([Идентификатор])
    ON DELETE NO ACTION ON UPDATE NO ACTION;

-- Creating non-clustered index for FOREIGN KEY
'FK_ПользовательНомер_телефона'
CREATE INDEX [IX_FK_ПользовательНомер_телефона]
ON [dbo].[Номера_телефонов]
    ([Пользователь_Идентификатор]);
GO

-- ------------------------------------------------
```

```
-- Script has ended
-- ------------------------------------------------
```

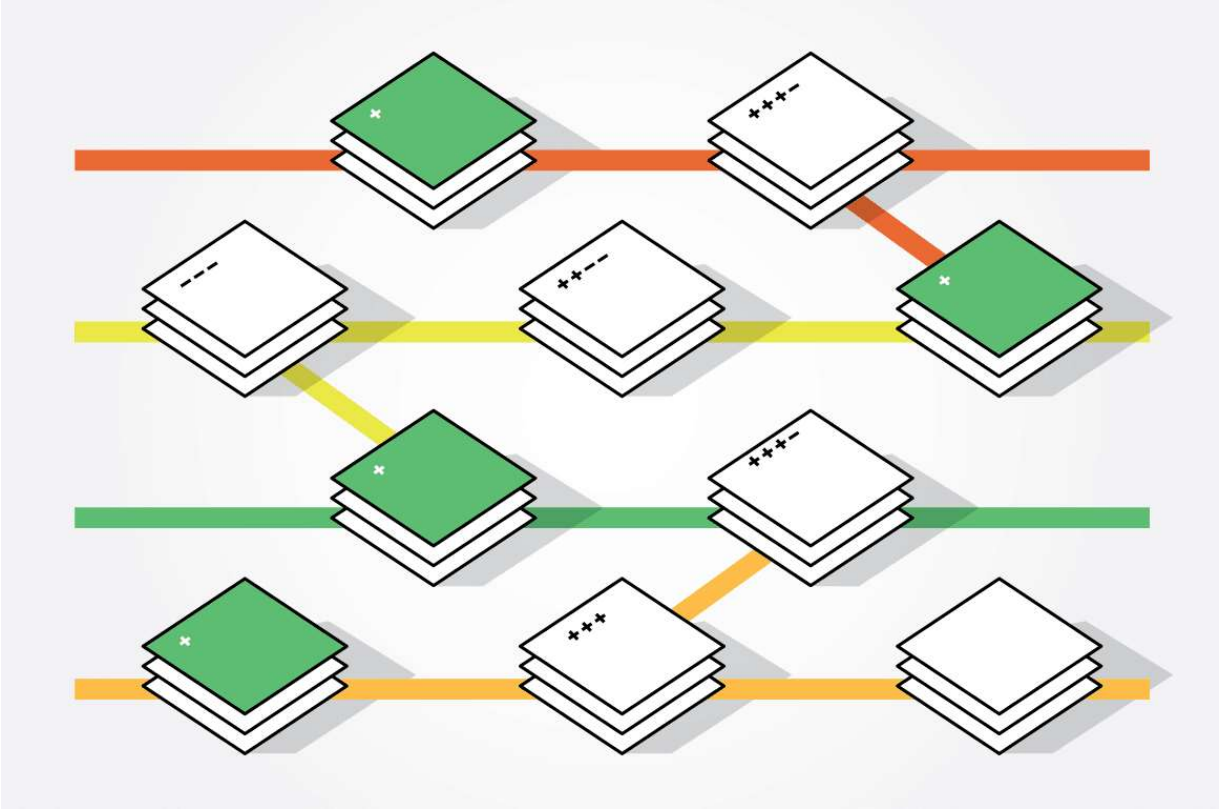
ЛЕКЦИЯ 2. СИСТЕМЫ УПРАВЛЕНИЯ ВЕРСИЯМИ

Система управления версиями (от англ. Version Control System, VCS или Revision Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.



Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях,

в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. В частности, системы управления версиями применяются в САПР, обычно в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления (Software Configuration Management Tools).



Ситуация, в которой электронный документ за время своего существования претерпевает ряд изменений, достаточно типична. При этом часто бывает важно иметь не только последнюю версию, но и несколько предыдущих. В простейшем случае можно просто хранить несколько вариантов документа, нумеруя их соответствующим образом. Такой способ неэффективен (приходится хранить несколько практически идентичных копий), требует повышенного внимания и дисциплины и часто ведёт к ошибкам, поэтому были разработаны средства для автоматизации этой работы.

Традиционные системы управления версиями используют централизованную модель, когда имеется единое хранилище документов, управляемое специальным сервером, который и выполняет бóльшую часть функций по управлению версиями. Пользователь, работающий с документами, должен сначала получить нужную ему версию документа из хранилища; обычно создаётся локальная копия документа, так называемая «рабочая копия». Может быть получена последняя версия или любая из предыдущих, которая может быть выбрана по номеру версии или дате создания, иногда и по другим признакам. После того, как в документ внесены нужные изменения, новая версия помещается в хранилище. В отличие от простого сохранения файла, предыдущая версия не стирается, а тоже остаётся в хранилище и

может быть оттуда получена в любое время. Сервер может использовать т. н. дельта-компрессию — такой способ хранения документов, при котором сохраняются только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных.

Поскольку обычно наиболее востребованной является последняя версия файла, система может при сохранении новой версии сохранять её целиком, заменяя в хранилище последнюю ранее сохранённую версию на разницу между этой и последней версией. Некоторые системы (например, ClearCase) поддерживают сохранение версий обоих видов: большинство версий сохраняется в виде дельт, но периодически (по специальной команде администратора) выполняется сохранение версий всех файлов в полном виде; такой подход обеспечивает максимально полное восстановление истории в случае повреждения репозитория.

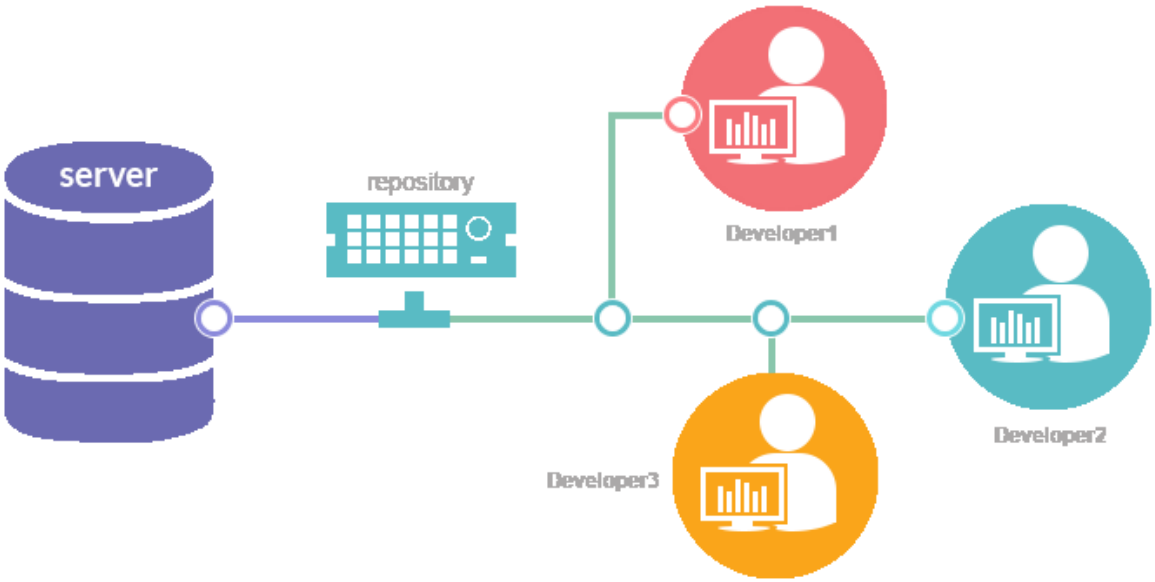
Иногда создание новой версии выполняется незаметно для пользователя (прозрачно), либо прикладной программой, имеющей встроенную поддержку такой функции, либо за счёт использования специальной файловой системы. В этом случае пользователь просто работает с файлом, как обычно, и при сохранении файла автоматически создаётся новая версия.

Часто бывает, что над одним проектом одновременно работают несколько человек. Если два человека изменяют один и тот же файл, то один из них может случайно отменить изменения, сделанные другим. Системы управления версиями отслеживают такие конфликты и предлагают средства их решения. Большинство систем может автоматически объединить (слить) изменения, сделанные разными разработчиками. Однако такое автоматическое объединение изменений, обычно, возможно только для текстовых файлов и при условии, что изменялись разные (непересекающиеся) части этого файла. Такое ограничение связано с тем, что большинство систем управления версиями ориентированы на поддержку процесса разработки программного обеспечения, а исходные коды программ хранятся в текстовых файлах. Если автоматическое объединение выполнить не удалось, система может предложить решить проблему вручную.

Часто выполнить слияние невозможно ни в автоматическом, ни в ручном режиме, например, если формат файла неизвестен или слишком сложен. Некоторые системы управления версиями дают возможность заблокировать файл в хранилище. Блокировка не позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла (например, средствами файловой системы) и обеспечивает, таким образом, исключительный доступ только тому пользователю, который работает с документом.

Многие системы управления версиями предоставляют ряд других возможностей:

Позволяют создавать разные варианты одного документа, т. н. ветки, с общей историей изменений до точки ветвления и с разными — после неё.



Дают возможность узнать, кто и когда добавил или изменил конкретный набор строк в файле.

Ведут журнал изменений, в который пользователи могут записывать пояснения о том, что и почему они изменили в данной версии.

Контролируют права доступа пользователей, разрешая или запрещая чтение или изменение данных, в зависимости от того, кто запрашивает это действие.

РАЗДЕЛ 2. АРХИТЕКТУРА КОРПОРАТИВНОГО ВЕБ-ПРИЛОЖЕНИЯ

ЛЕКЦИЯ 3. ТЕХНОЛОГИЯ DDD

Современные программные приложения, будь то настольные или веб-приложения, имеют многослойную архитектуру. Типичная архитектура современного программного приложения приведена на рис. 20. Следует заметить, что в настоящий момент все больше стирается грань между «настольными» приложениями, которые должны исполняться на компьютере (ноутбуке, планшетнике) пользователя, и веб-приложениями.

Сердцем современного программного приложения является объектная модель. Именно она является моделью предметной области, именно в ней содержится логика предметной области (бизнес-логика), которую программист должен выявить, общаясь со специалистами той предметной области, для которой пишется программа. Поэтому создавая модель предметной области, программист с одной стороны выступает в роли системного аналитика, а с другой – проектировщика программного обеспечения, который переводит созданную модель на язык объектно-ориентированного программирования.

В современных программных приложениях информация, представленная объектами объектной (сущностной) модели, обычно сохраняется в реляционной базе данных, которая, как видно из названия, использует реляционная модель, отличную от объектной модели. Поэтому непосредственное сохранение объектов в реляционной базе данных невозможно. Для того, чтобы можно было сохранить объекты (а точнее значения атрибутов объектов, которые, как известно, определяют состояние объектов) в реляционной базе данных, между объектной моделью и базой данных создается промежуточный слой – объектно-реляционных преобразователь или ORM (Object Relation Mapper).