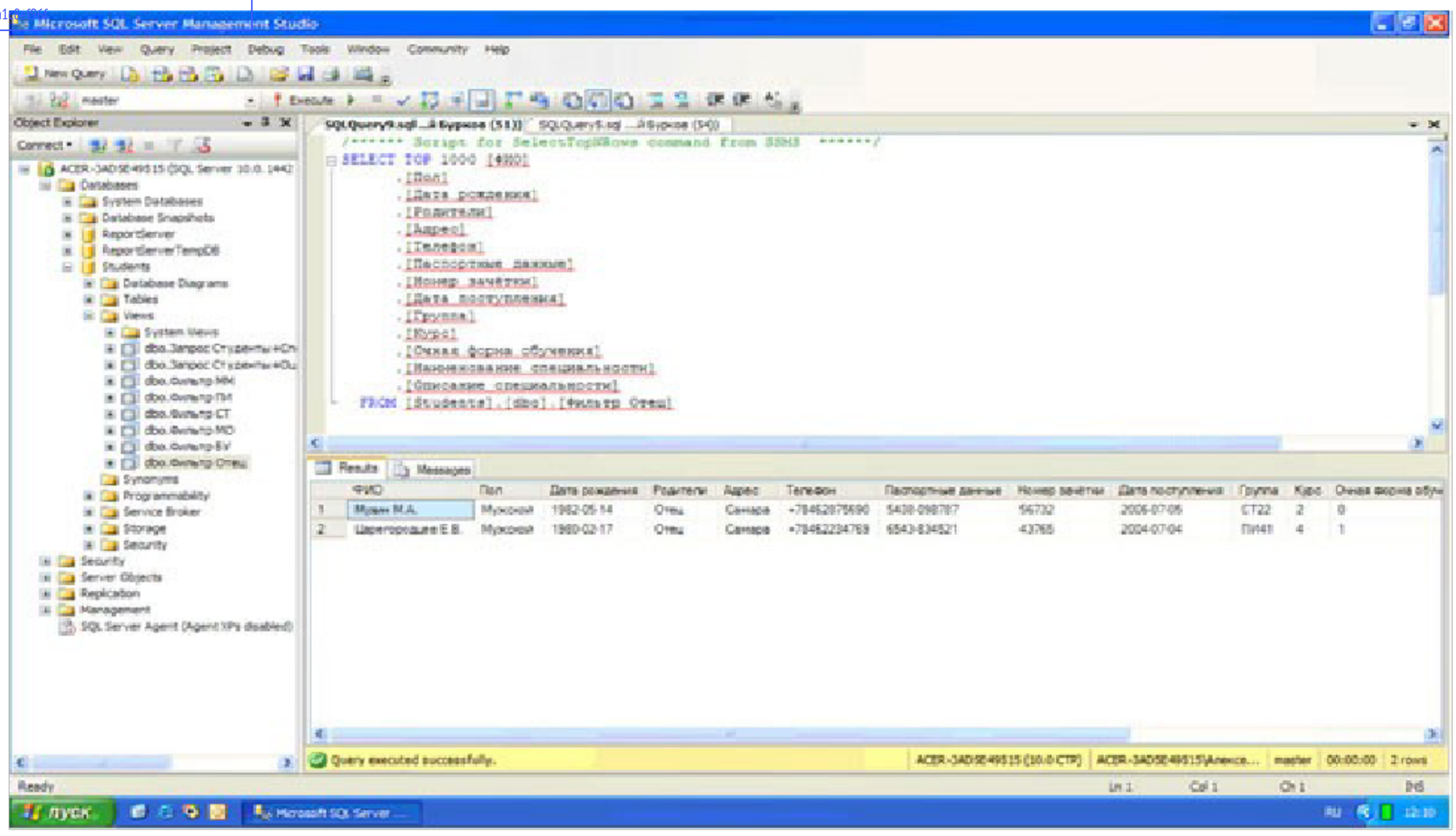
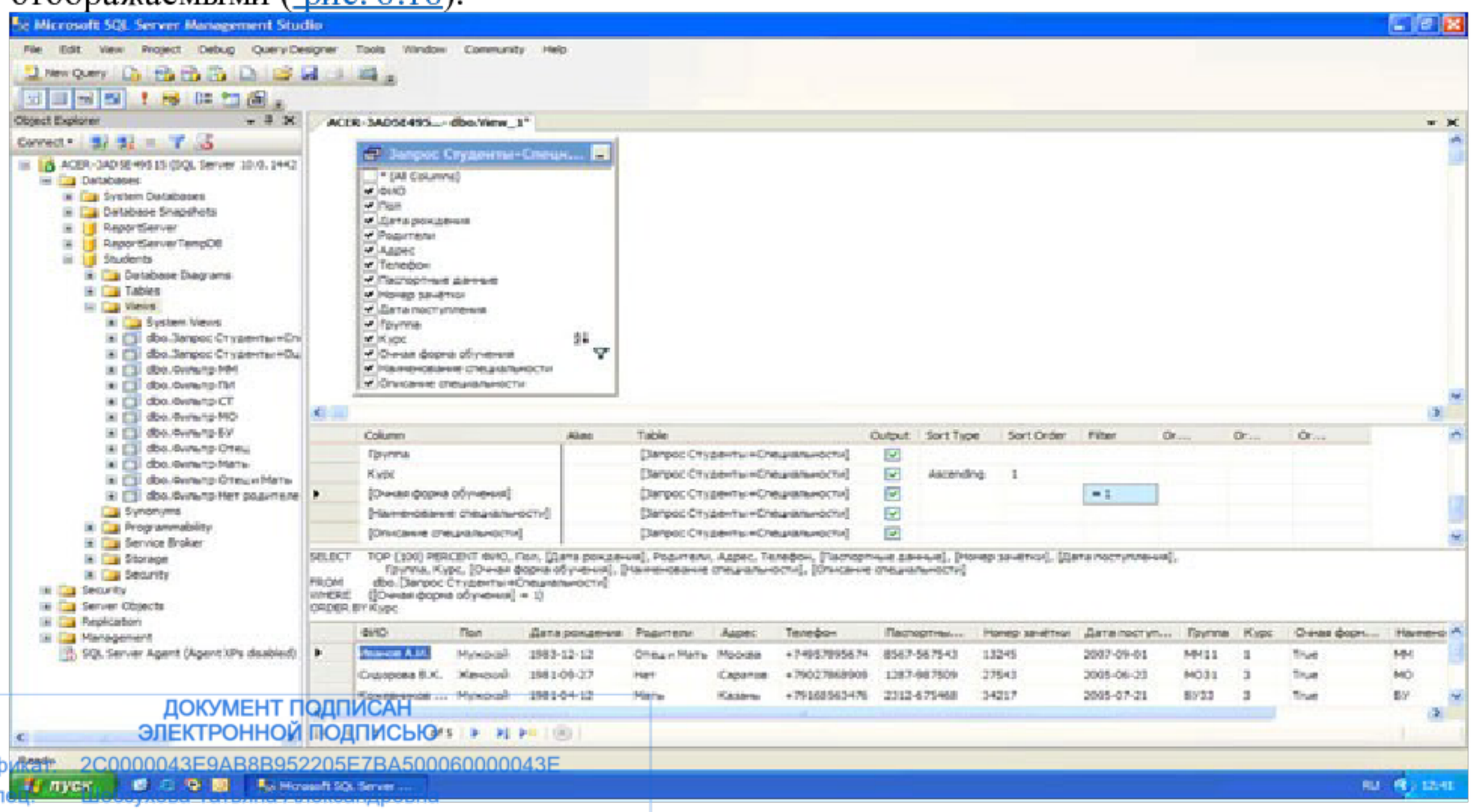


[увеличить изображение]

Рис. 8.17.

Создайте фильтры для отображения студентов с другими вариантами родителей. Данные фильтры создаются аналогично фильтру **"Фильтр Отец"** (смотри выше). Единственным отличием является условие отбора, накладываемое на поле **"Родители"**, оно должно быть не **"='Отец'"**, а **"='Мать'"**, **"='Отец, Мать'"** или **"='Нет'"**. При сохранении фильтров задаем их имена соответственно их условиям отбора, то есть **"Фильтр Мать"**, **"Фильтр Отец и Мать"** или **"Фильтр Нет родителей"**. Проверьте созданные фильтры на работоспособность.

Наконец создадим фильтры для отображения студентов очной и заочной формы обучения. Начнем с очной формы обучения. Создайте новый запрос и добавьте в него запрос **"Запрос Студенты+Специальности"**. Как и ранее сделайте все поля запроса отображаемыми (рис. 8.18).



увеличить изображение

Рис. 8.18.

В таблице отображаемых полей в столбце **"Filter"**, в строке для поля **"Очная форма обучения"** установите условие отбора равное **"=1"**

Замечание: Поле **"Очная форма обучения"** является логическим полем, оно может принимать значения либо **"True"** (Истина), либо **"False"** (Ложь). В качестве синонимов этих значений в "Microsoft SQL Server 2008" можно использовать 1 и 0 соответственно.

Установите сортировку по возрастанию, по полю курс, задав в строке для этого поля, в столбце **"Sort Type"**, значение **"Ascending"**.

Проверьте работу фильтра, выполнив его. После выполнения фильтра окно конструктора запросов должно выглядеть точно также как на рис. 8.18.

Закройте окно конструктора запросов. Сохраните фильтр под именем **"Фильтр очная форма обучения"** (рис. 8.19).

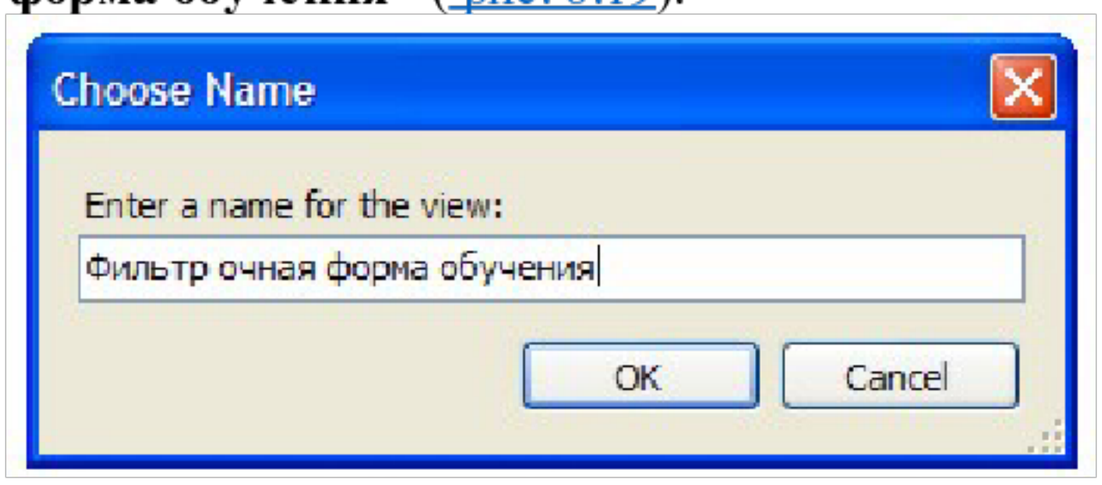
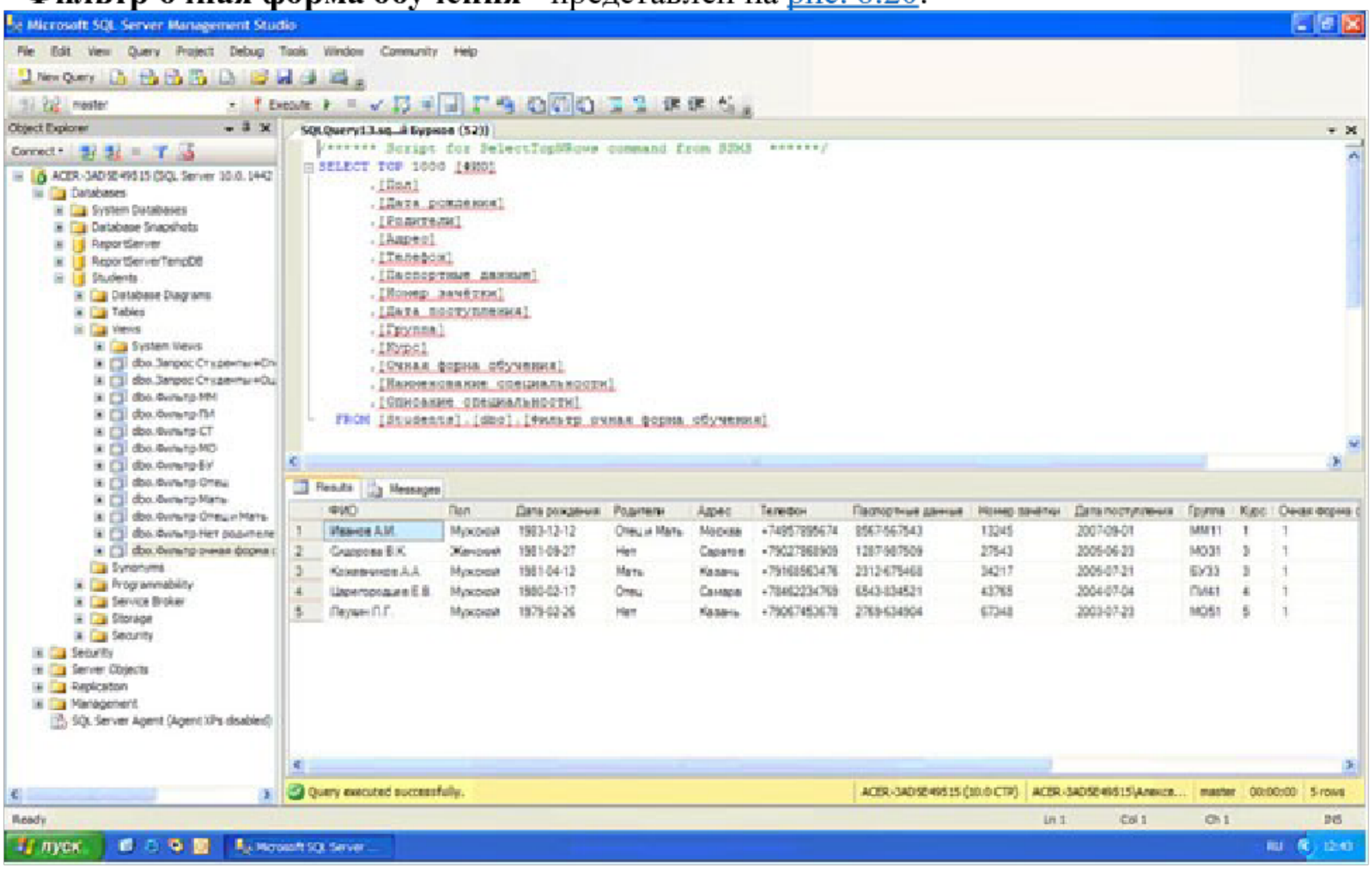


Рис. 8.19.

После появления фильтра **"Фильтр очная форма обучения"** в обозревателе объектов выполните фильтр вне окна конструктора запросов. Результат выполнения фильтра **"Фильтр очная форма обучения"** представлен на рис. 8.20.



увеличить изображение

Рис. 8.20.

Самостоятельно создайте фильтр для отображения студентов заочной формы обучения. Данный фильтр создается точно также как и фильтр **"Фильтр очная форма обучения"**. Единственным отличием является условие отбора, накладываемое на поле **"Очная форма**

обучения", оно должно быть не "=1", а "=0". При сохранении фильтра задайте его имя как "Фильтр заочная форма обучения". Проверьте созданный фильтр на работоспособность.
В итоге, после создания всех запросов и фильтров окно обозревателя объектов должно выглядеть следующим образом (рис. 8.21):

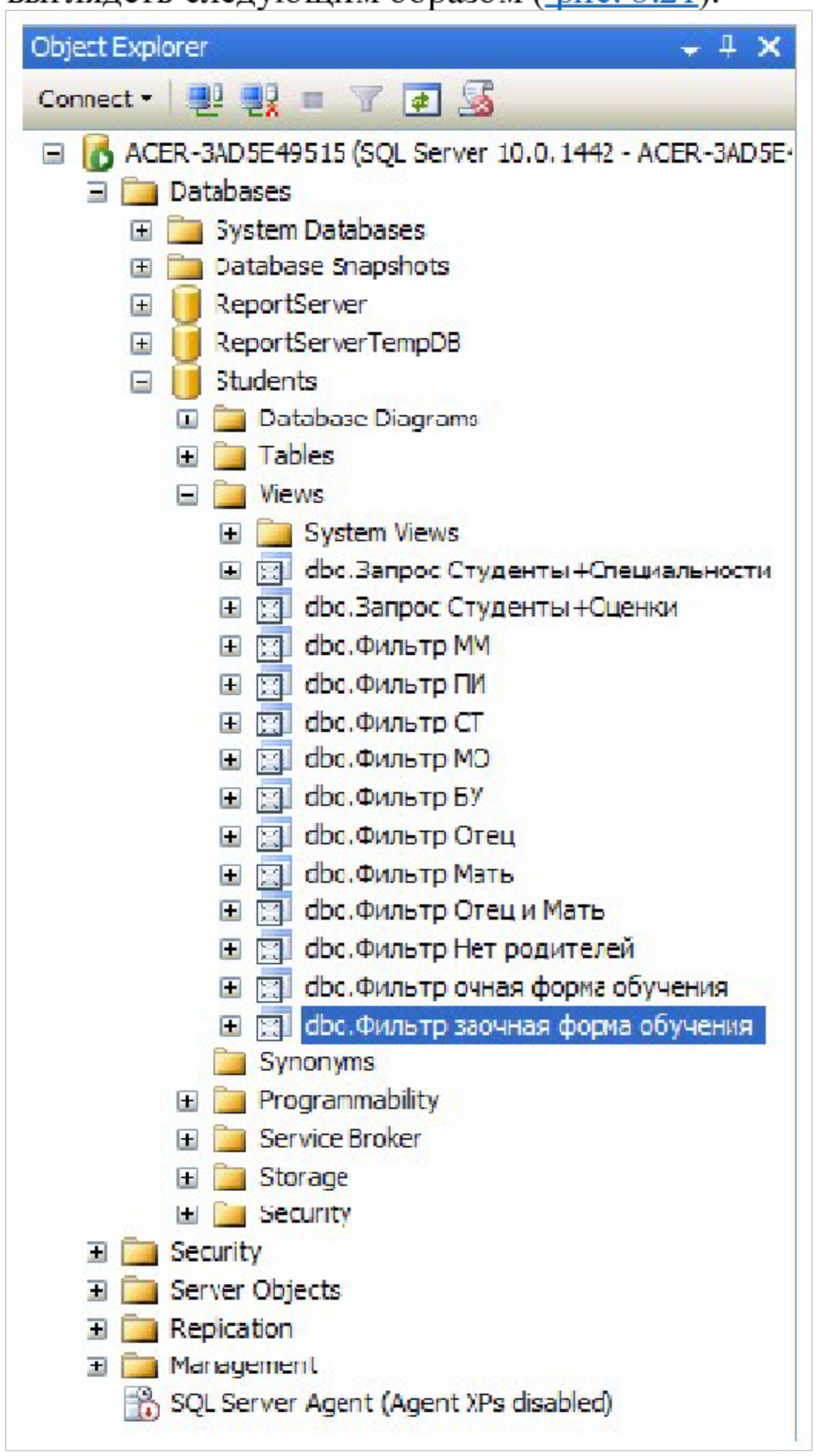


Рис. 8.21.

Контрольные вопросы

1. Как создать новый запрос?
2. Как создать новый фильтр?
3. Как использовать фильтр?

Лабораторная работа № 7

«Создание динамических запросов при помощи хранимых процедур в Microsoft SQL Server 2012»

Цель работы:

- научиться работать с хранимыми процедурами.

Перейдем к созданию хранимых процедур. Для работы с хранимыми процедурами в обозревателе объектов необходимо выделить папку **"Programmability/Stored Procedures"** базы данных **"Students"** (рис. 10.1).

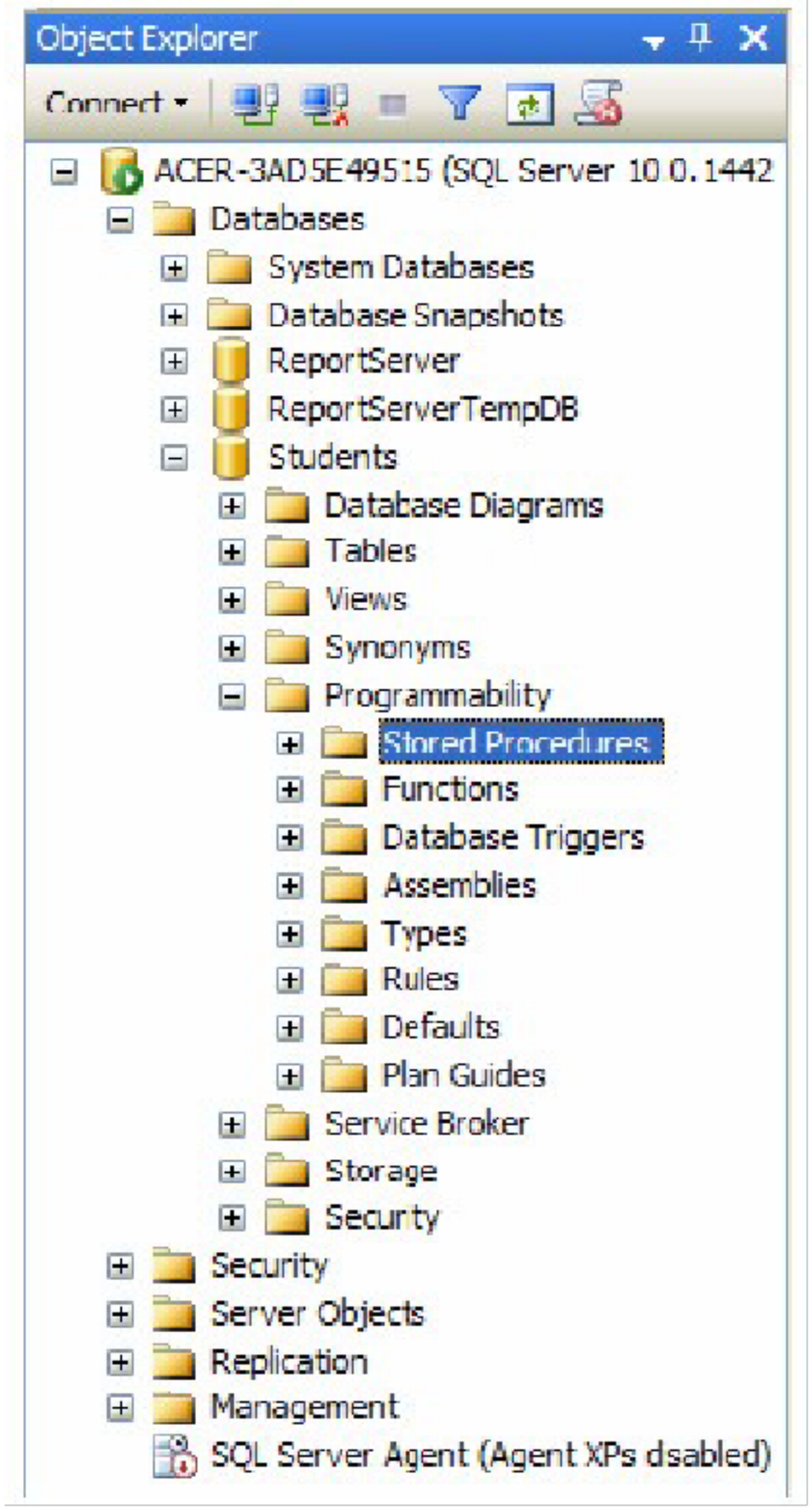
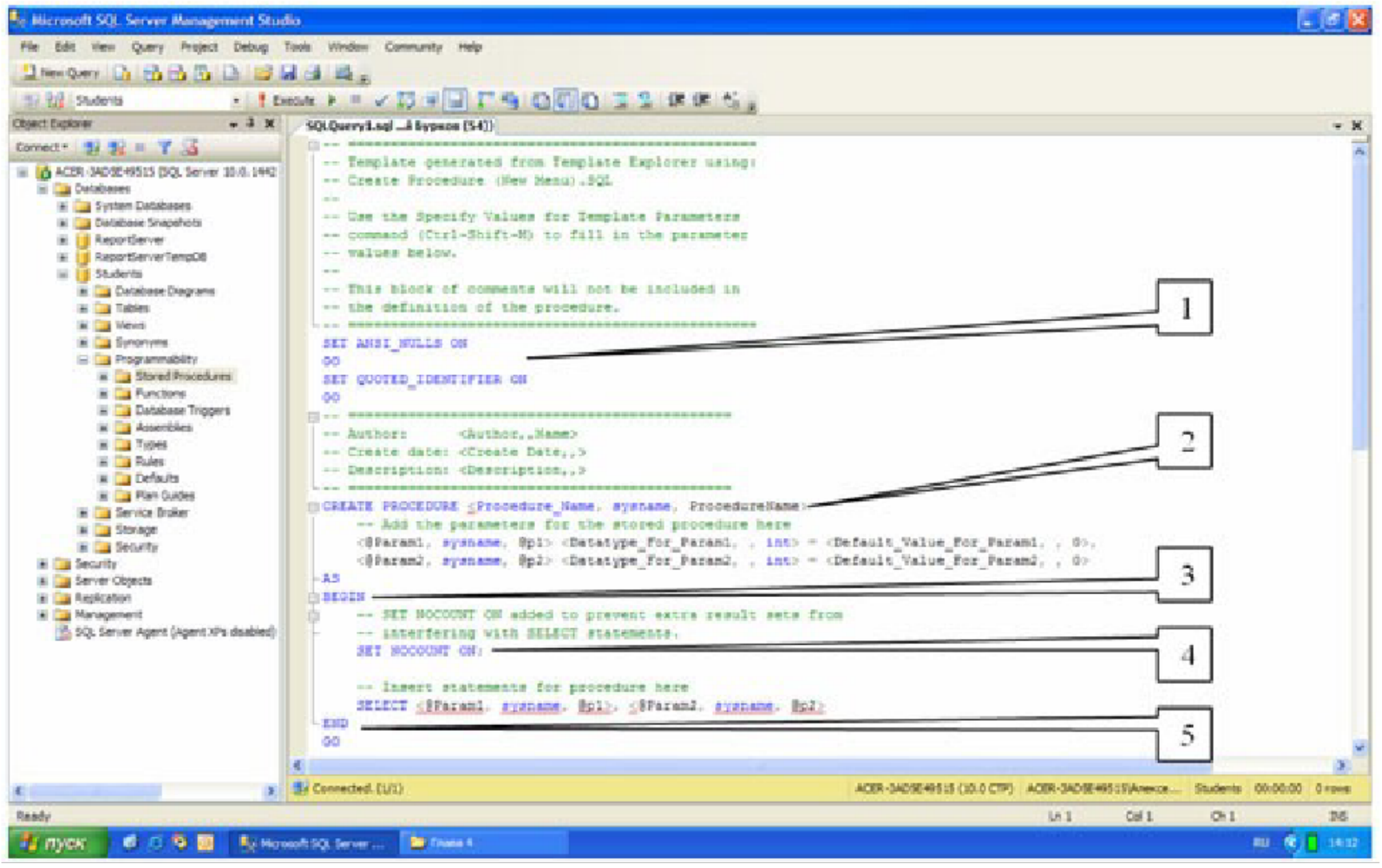


Рис. 10.1.

Создадим процедуру, вычисляющую среднее трех чисел. Для создания новой хранимой процедуры щелкните **ПКМ** по папке **"Stored Procedures"** (рис. 10.1) и в появившемся меню выберите пункт **"New Stored Procedure"**. Появится окно кода новой хранимой процедуры (рис. 10.2).



увеличить изображение

Рис. 10.2.

Хранимая процедура имеет следующую структуру (рис. 10.2):

1. Область настройки параметров синтаксиса процедуры. Позволяет настраивать некоторые синтаксические правила, используемые при наборе кода процедуры. В нашем случае это:
   - SET ANSI_NULLS ON - включает использование значений NULL (Пусто) в кодировке ANSI,
   - SET QUOTED_IDENTIFIER ON - включает возможность использования двойных кавычек для определения идентификаторов;
2. Область определения имени процедуры (**Procedure_Name**) и параметров передаваемых в процедуру (**@Param1, @Param2**). Определение параметров имеет следующий синтаксис:
   @<Имя параметра> <Тип данных> = <Значение по умолчанию>
   Параметры разделяются между собой запятыми;
3. Начало тела процедуры, обозначается служебным словом "BEGIN" ;
4. Тело процедуры, содержит команды языка программирования запросов T-SQL;
5. Конец тела процедуры, обозначается служебным словом "END".

Замечание: В коде зеленым цветом выделяются комментарии. Они не обрабатываются сервером и выполняют функцию пояснений к коду. Строки комментариев начинаются с подстроки **"--"**. Далее в коде, мы не будем отображать комментарии, они будут свернуты. Слева от раздела с комментариями будет стоять знак **"+"**, щелкнув по которому можно развернуть комментарий.

Наберем код процедуры вычисляющей среднее трех чисел, как это показано на рис. 10.3.

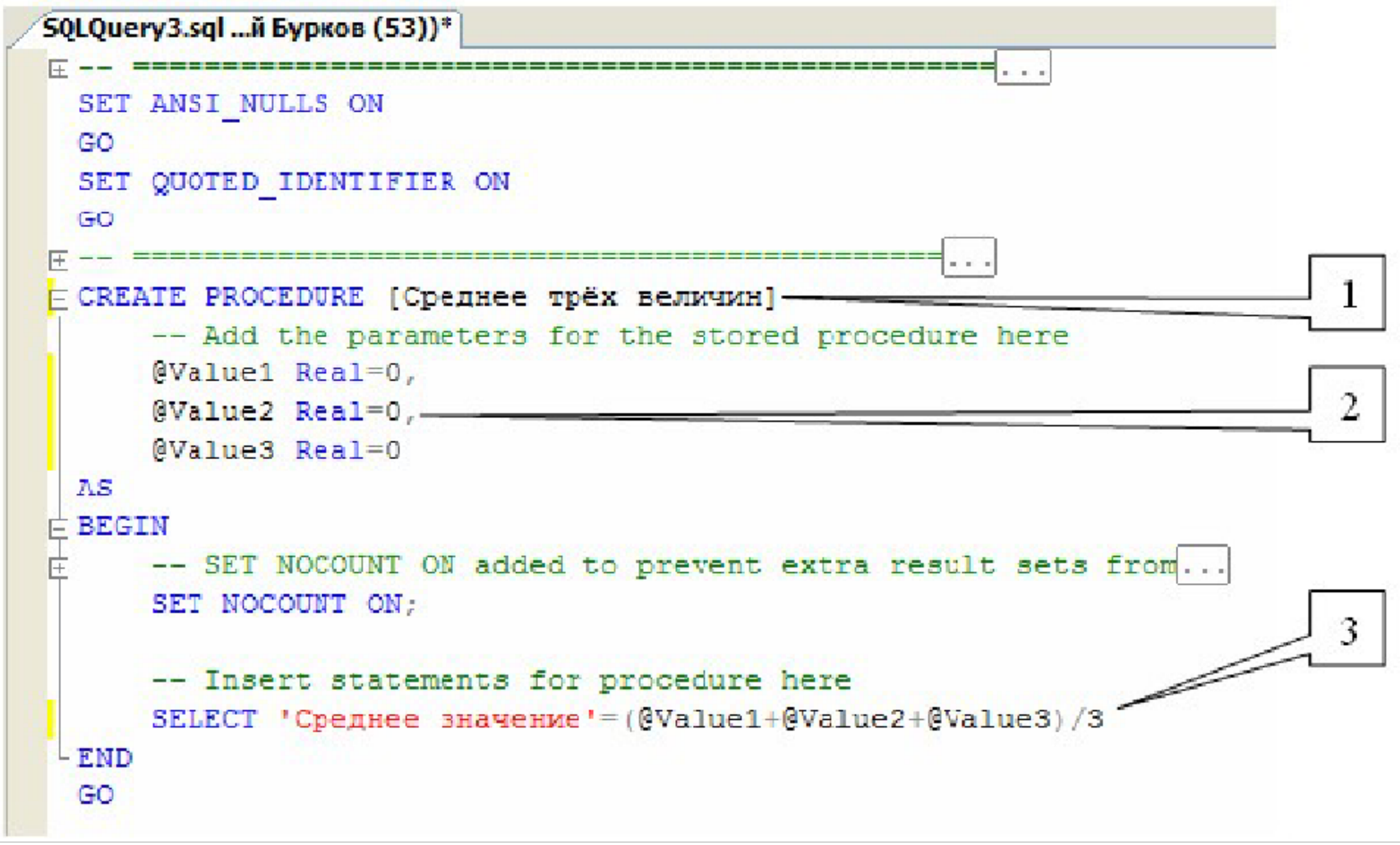


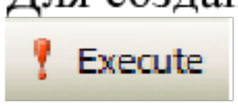
Рис. 10.3.

Рассмотрим код данной процедуры более подробно (рис. 10.3):

1. CREATE PROCEDURE [Среднее трех величин] - определяет имя создаваемой процедуры как "Среднее трех величин";
2. @Value1 Real = 0, @Value2 Real = 0, @Value3 Real = 0 - определяют три параметра процедуры Value1, Value2 и Value3. Данным параметрам можно присвоить дробные числа (Тип данных Real), значения по умолчанию равны 0;
3. SELECT 'Среднее значение'=(@Value1+@Value2+@Value3)/3 - вычисляет среднее и выводит результат с подписью "Среднее значение".

Остальные фрагменты кода рассмотрены выше (рис. 10.2).

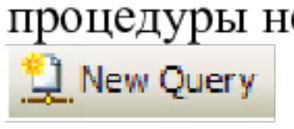
Для создания процедуры, выполним вышеописанный код, нажав кнопку



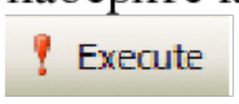
(Выполнить) на панели инструментов. В нижней части окна с кодом появиться сообщение **"Command(s) completed successfully."**. Закройте окно с кодом, щелкнув мышью по кнопке закрытия

расположенной в верхнем правом углу окна с кодом процедуры.

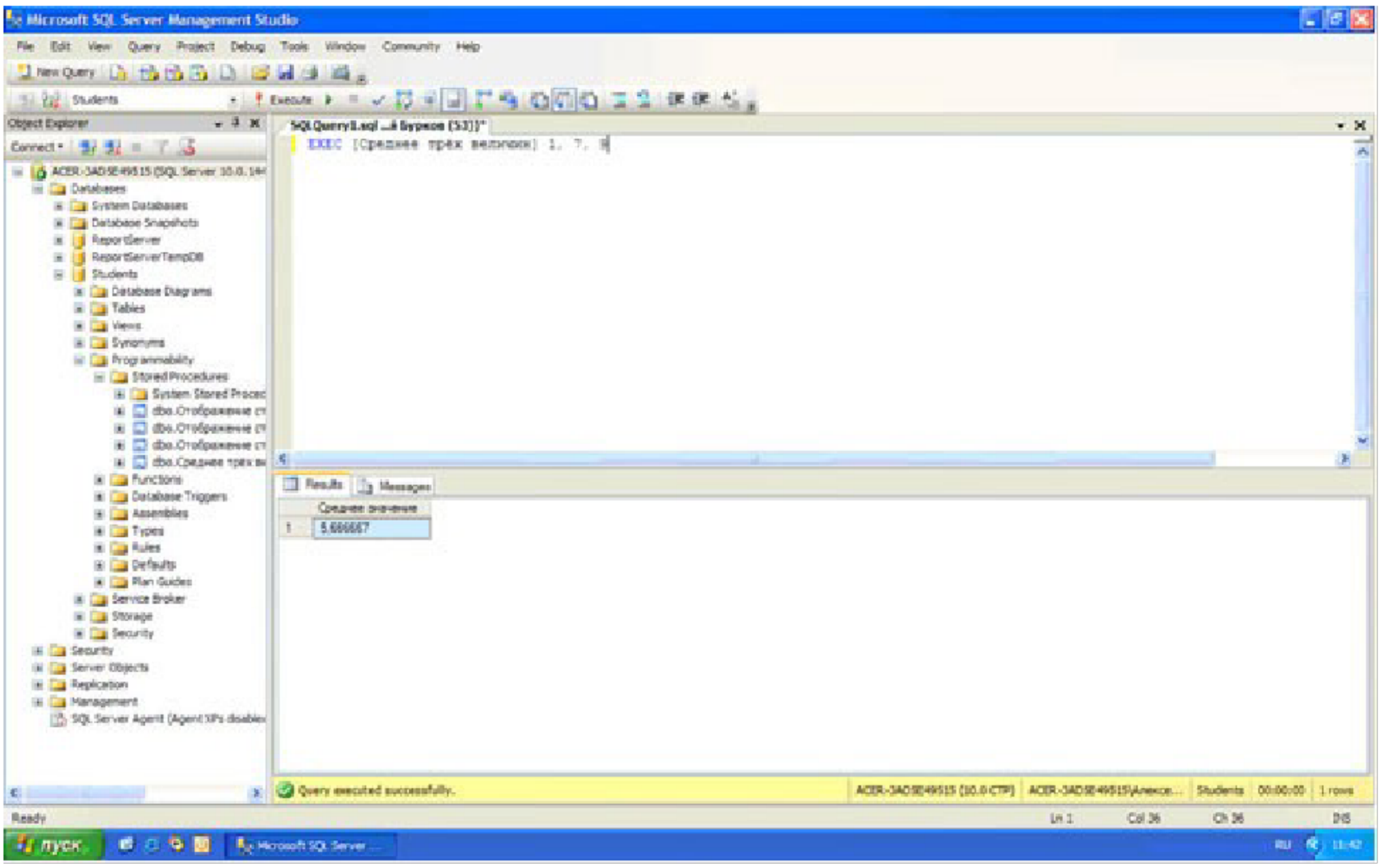
Проверим работоспособность созданной хранимой процедуры. Для запуска хранимой процедуры необходимо создать новый пустой запрос, нажав на кнопку



(Новый запрос) на панели инструментов. В появившемся окне с пустым запросом наберите команду EXEC [Среднее трех величин] 1, 7, 9 и нажмите кнопку



на панели инструментов (рис. 10.4).

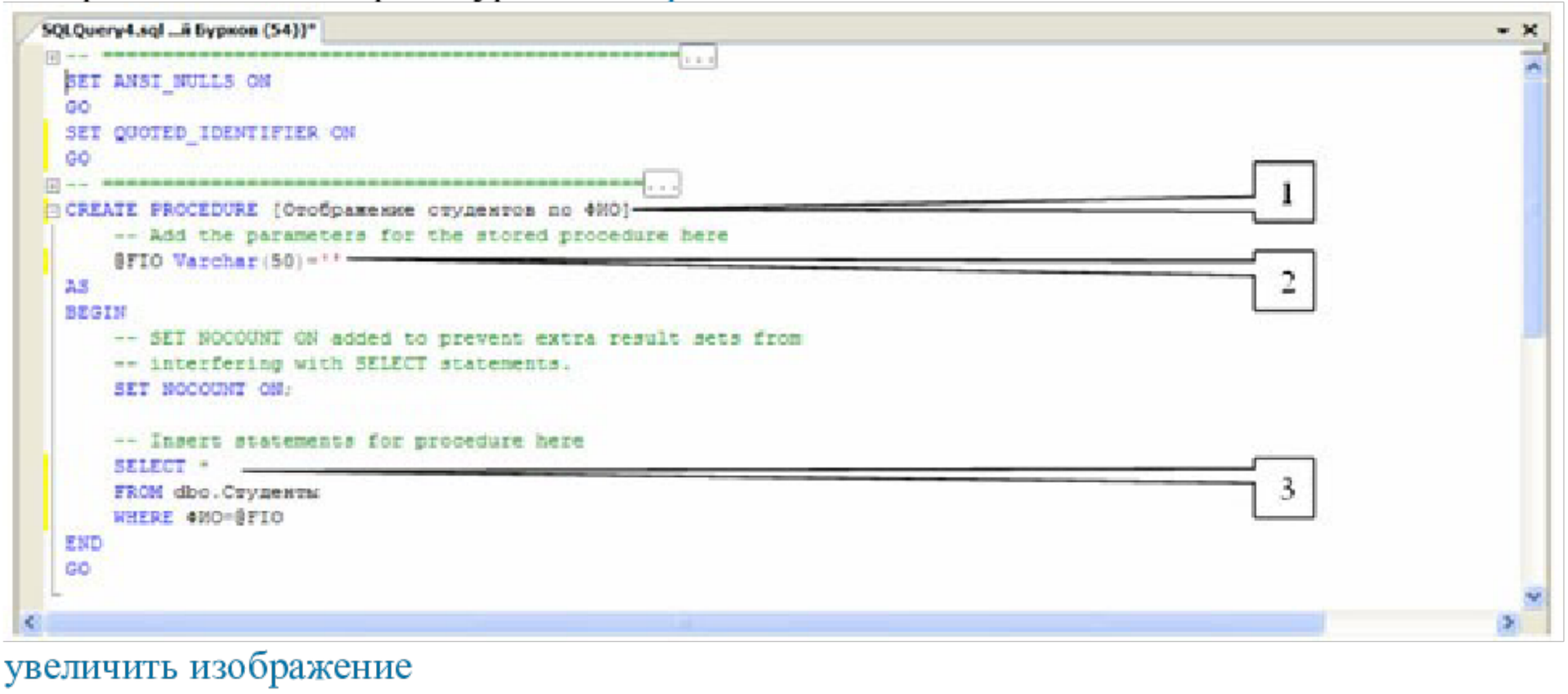


увеличить изображение

Рис. 10.4.

В нижней части окна с кодом появится результат выполнения новой хранимой процедуры: **Среднее значение 5,66667** (рис. 10.4).

Теперь создадим хранимую процедуру для отбора студентов из таблицы студенты по их "ФИО". Для этого создайте новую хранимую процедуру, как это описано выше, и наберите код новой процедуры как на рис. 10.5.



увеличить изображение

Рис. 10.5.

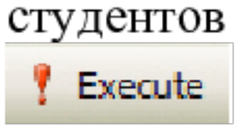
Рассмотрим код процедуры **"Отображение студентов по ФИО"** более подробно (рис. 10.5):

1. CREATE PROCEDURE [Отображение студентов по ФИО] - определяет имя создаваемой процедуры как "Отображение студентов по ФИО";
2. @FIO Varchar(50)='' - определяют единственный параметр процедуры **FIO**. Параметру можно присвоить текстовые строки переменной длины, длинной до 50 символов (Тип данных Varchar(50)), значения по умолчанию равны пустой строке;

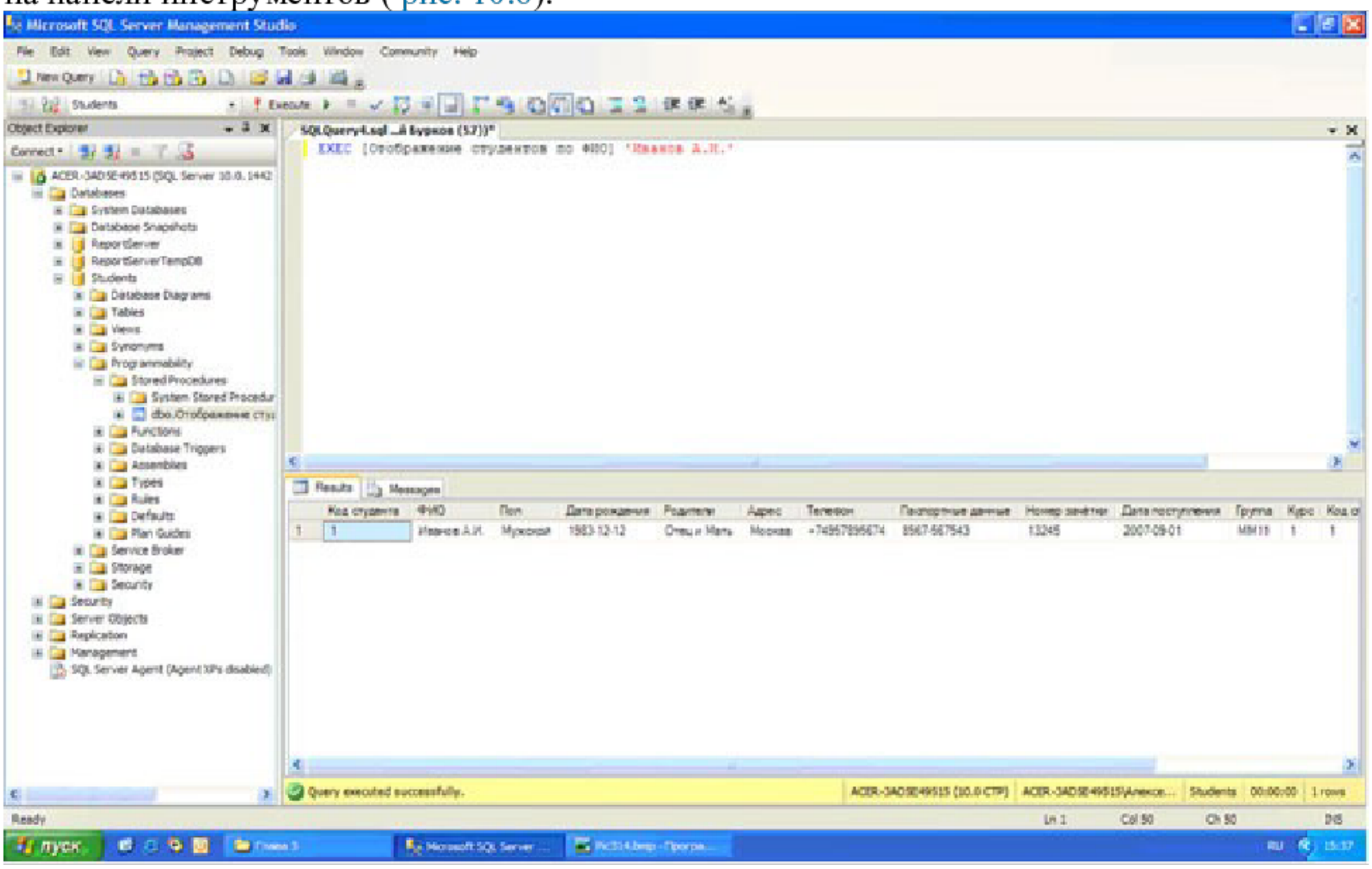
3. SELECT * FROM dbo.Студенты WHERE ФИО=@FIO - отобразить все поля (*) из таблицы студенты (dbo.Студенты), где значение поля ФИО равно значению параметра **FIO (ФИО=@FIO)**.

Выполним вышеописанный код и закроем окно с кодом, как описано выше.

Проверим работоспособность созданной хранимой процедуры. Создайте новый пустой запрос. В появившемся окне с пустым запросом наберите команду EXEC [Отображение студентов по ФИО] 'Иванов А.И.' и нажмите кнопку



на панели инструментов (рис. 10.6).

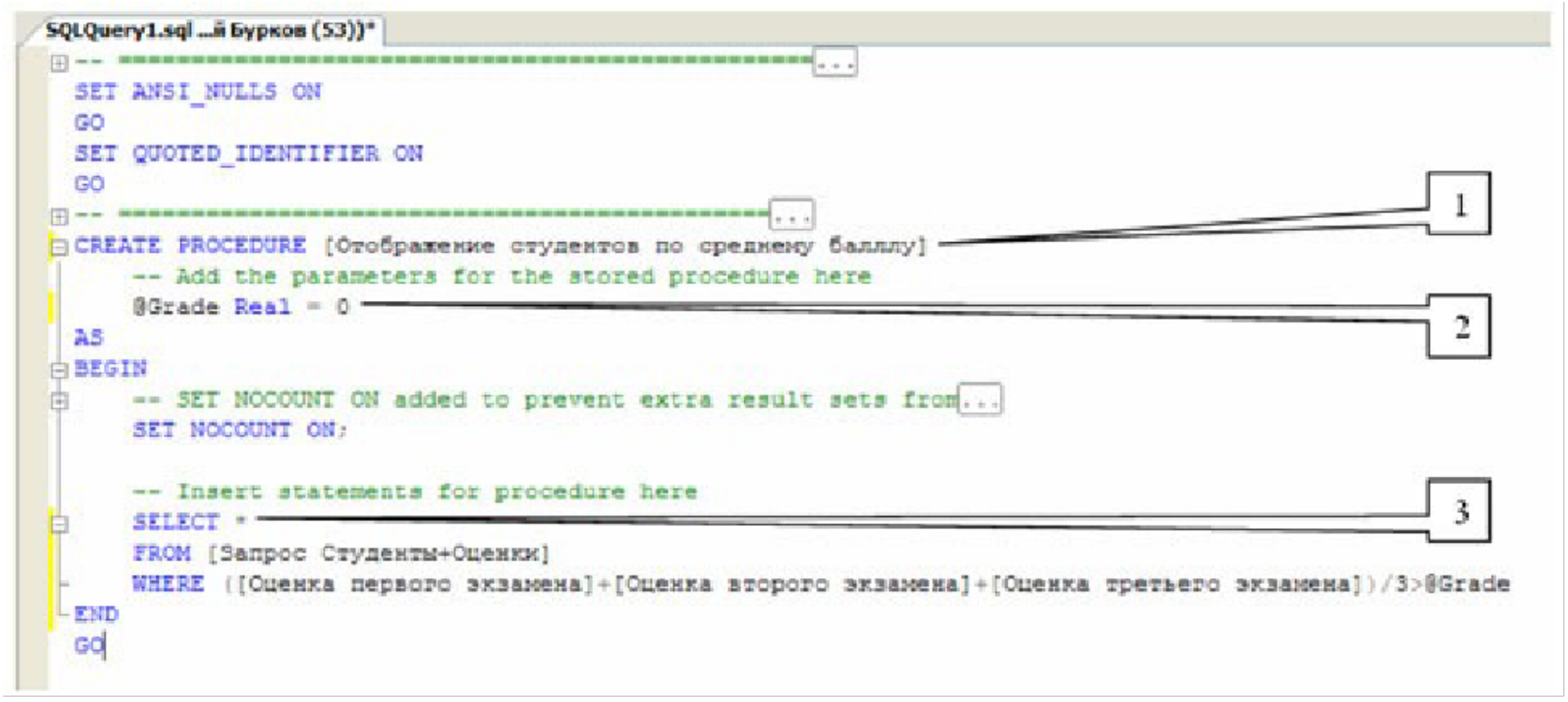


увеличить изображение

Рис. 10.6.

В нижней части окна с кодом появиться результат выполнения хранимой процедуры **"Отображение студентов по ФИО"** (рис. 10.6).

Теперь перейдем к более сложной задаче - отобразить студентов, у которых средний балл выше заданного. Создайте новую хранимую процедуру и наберите код новой процедуры как на рис. 10.7.



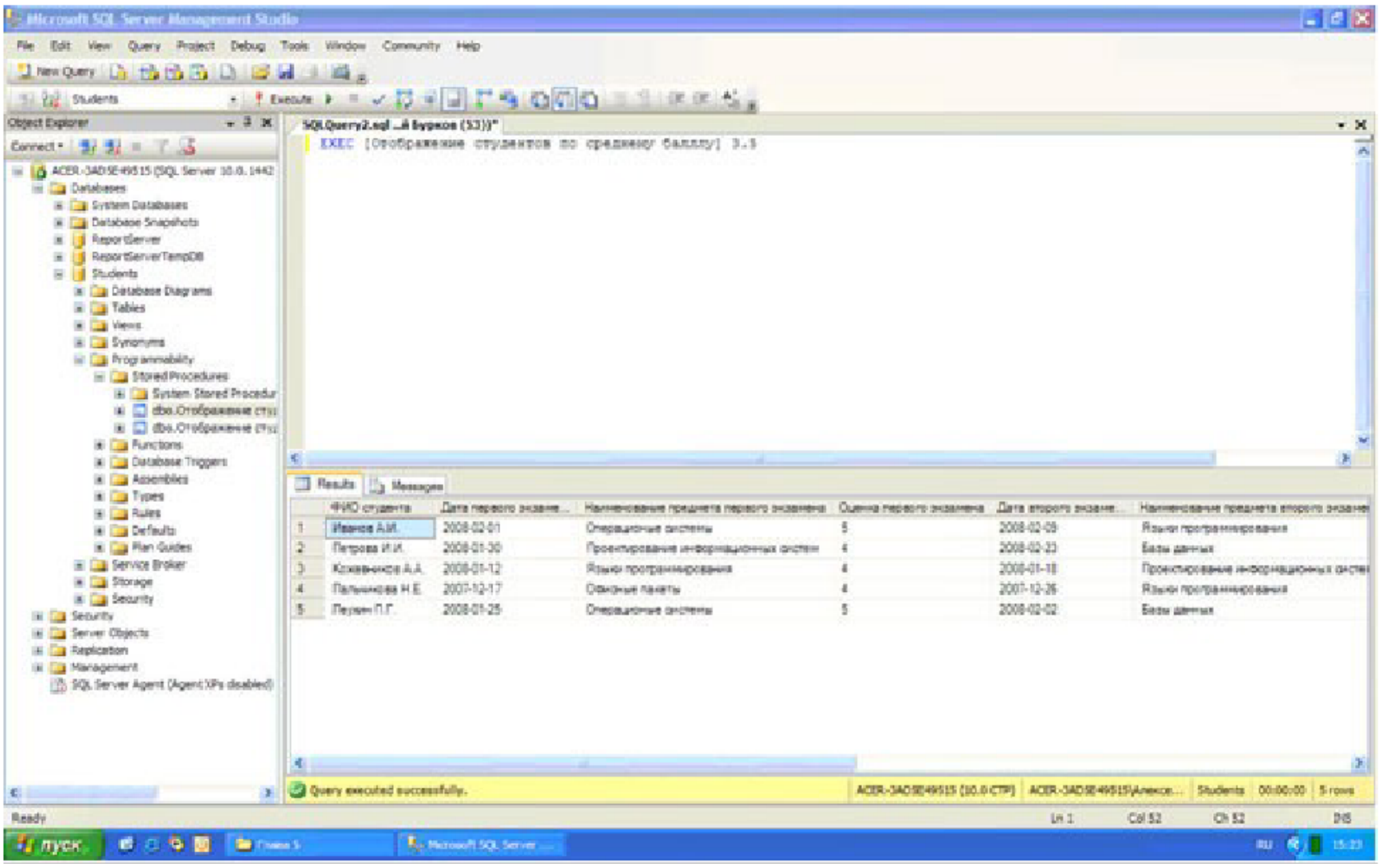
увеличить изображение

Рис. 10.7.

Рассмотрим код процедуры **"Отображение студентов по среднему баллу"** более подробно (рис. 10.7):

1. CREATE PROCEDURE [Отображение студентов по среднему баллу] - определяет имя создаваемой процедуры как "Отображение студентов по среднему баллу";
2. @Grade Real=0 - определяют параметр процедуры **Grade**. Параметру можно присвоить дробные числа (Тип данных Real), значения по умолчанию равны 0;
3. SELECT * FROM [Запрос Студенты+Оценки] WHERE ([Оценка первого экзамена]+[Оценка второго экзамена]+[Оценка третьего экзамена])/3>@Grade - отобразить все поля (*) из запроса **"Запрос Студенты+Оценки"** (Запрос Студенты+Оценки), где средний балл больше чем значение параметра **Grade (([Оценка первого экзамена]+[Оценка второго экзамена]+[Оценка третьего экзамена])/3>@Grade)**.

Выполним вышеописанный код и закроем окно с кодом, как описано выше. Проверим, как работает запрос, описанный выше. Для этого, создайте новый запрос и в нем наберите команду EXEC [Отображение студентов по среднему баллу] 3.5 и выполните ее (Смотри выше) (рис. 10.8).



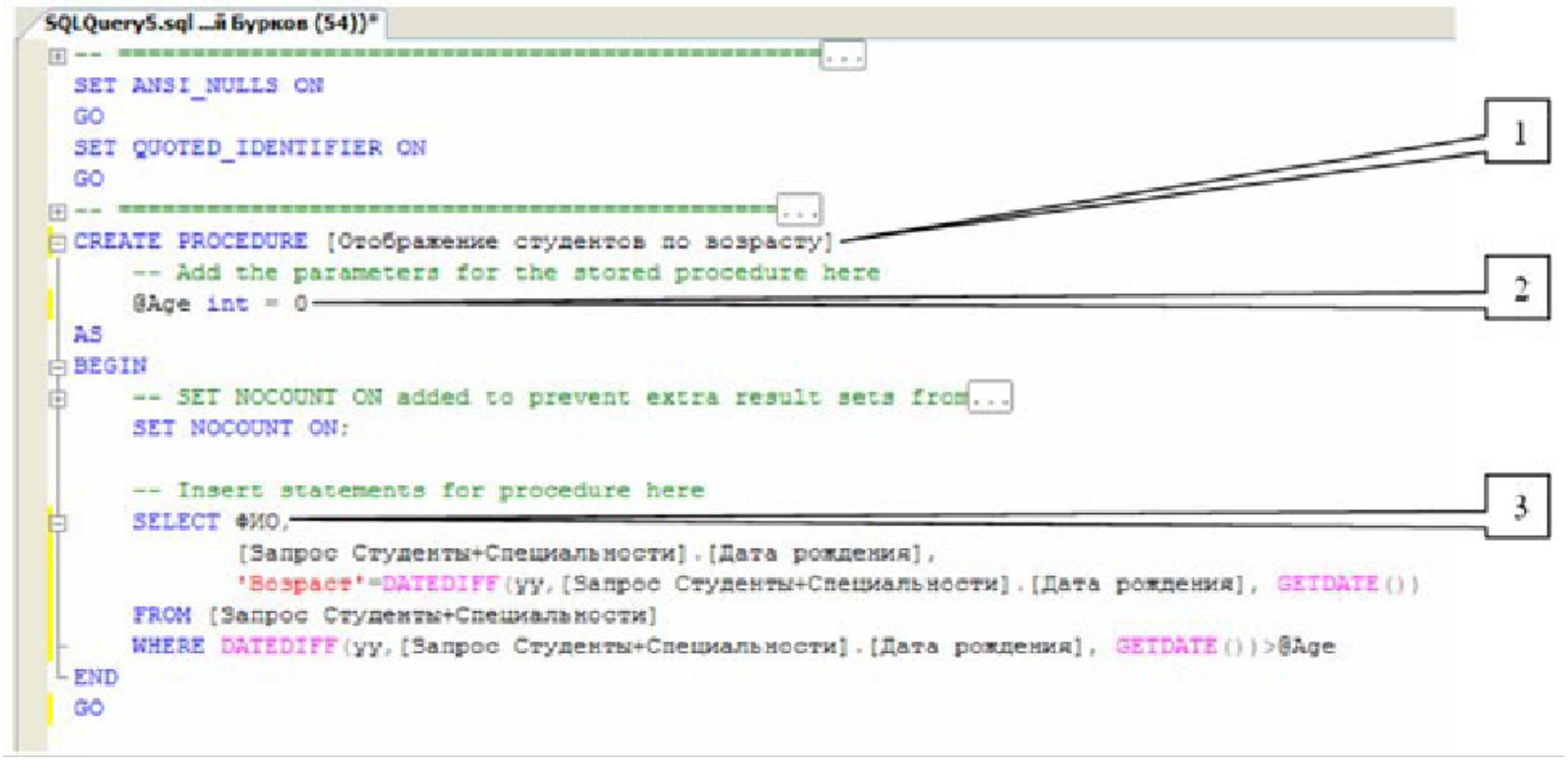
увеличить изображение

Рис. 10.8.

В нижней части окна с кодом появится результат выполнения хранимой процедуры **"Отображение студентов по среднему баллу"** (рис. 10.8).

В заключение решим более сложную задачу - отображение студентов старше заданного возраста. При чем возраст будет автоматически вычисляться в зависимости от даты рождения.

Создадим новую хранимую процедуру и наберем код новой процедуры как представлено на рис. 10.9.



увеличить изображение

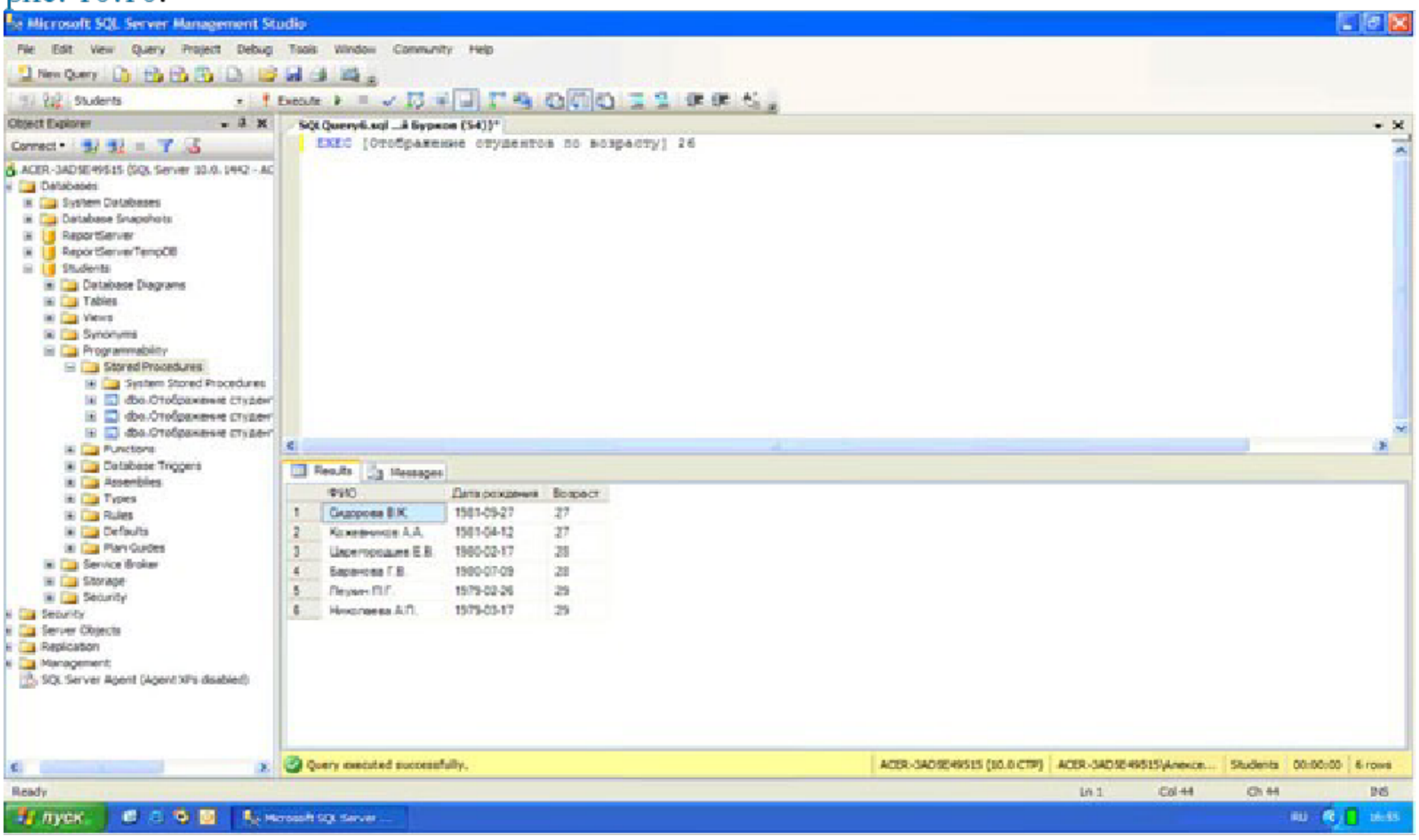
Рис. 10.9.

Рассмотрим код создаваемой процедуры **"Отображение студентов по возрасту"** более подробно (рис. 10.9):

1. CREATE PROCEDURE [Отображение студентов по возрасту] - определяет имя создаваемой процедуры как "Отображение студентов по возрасту";
2. @Age int=0 - определяют параметр процедуры **Age**. Параметру можно присвоить целые числа (Тип данных int), значения по умолчанию равны 0;
3. ФИО, [Запрос Студенты+Специальности].[Дата рождения], 'Возраст'=DATEDIFF(yy, [Запрос Студенты+Специальности].[Дата рождения], GETDATE()) - отображает из запроса **"Запроса Студенты+Специальности"** (FROM [Запрос Студенты+Специальности]) поля **"ФИО"** (ФИО) и **"Дата рождения"** ([Запрос Студенты+Специальности].[Дата рождения]), а также отображает возраст студента ('Возраст') в годах (yy), вычисленный исходя из его даты рождения и текущей даты (DATEDIFF(yy,[Запрос Студенты+Специальности].[Дата рождения], GETDATE())). Более того, выводятся студенты возраст которых больше определенного в параметре **"Age"** (DATEDIFF(yy,[Запрос Студенты+Специальности].[Дата рождения], GETDATE())>@Age).

Замечание: Встроенная функция **DATEDIFF** вычисляющая количество периодов между двумя датами, имеет следующий синтаксис: DATEDIFF(<период>,<начальная дата>, <конечная дата>)

Выполним код запроса **"Отображение студентов по возрасту"**, а затем закроем окно с кодом, как описано выше. Проверим, как работает запрос. Для этого, создадим новый запрос и в нем наберем команду EXEC [Отображение студентов по возрасту] 26 и выполните ее. Должен появиться результат аналогичный результату, представленному на рис. 10.10.



увеличить изображение

Рис. 10.10.

На этом мы заканчиваем описание хранимых процедур и переходим к рассмотрению пользовательских функций. В итоге, обозреватель объектов должен иметь вид как на рис. 10.11.

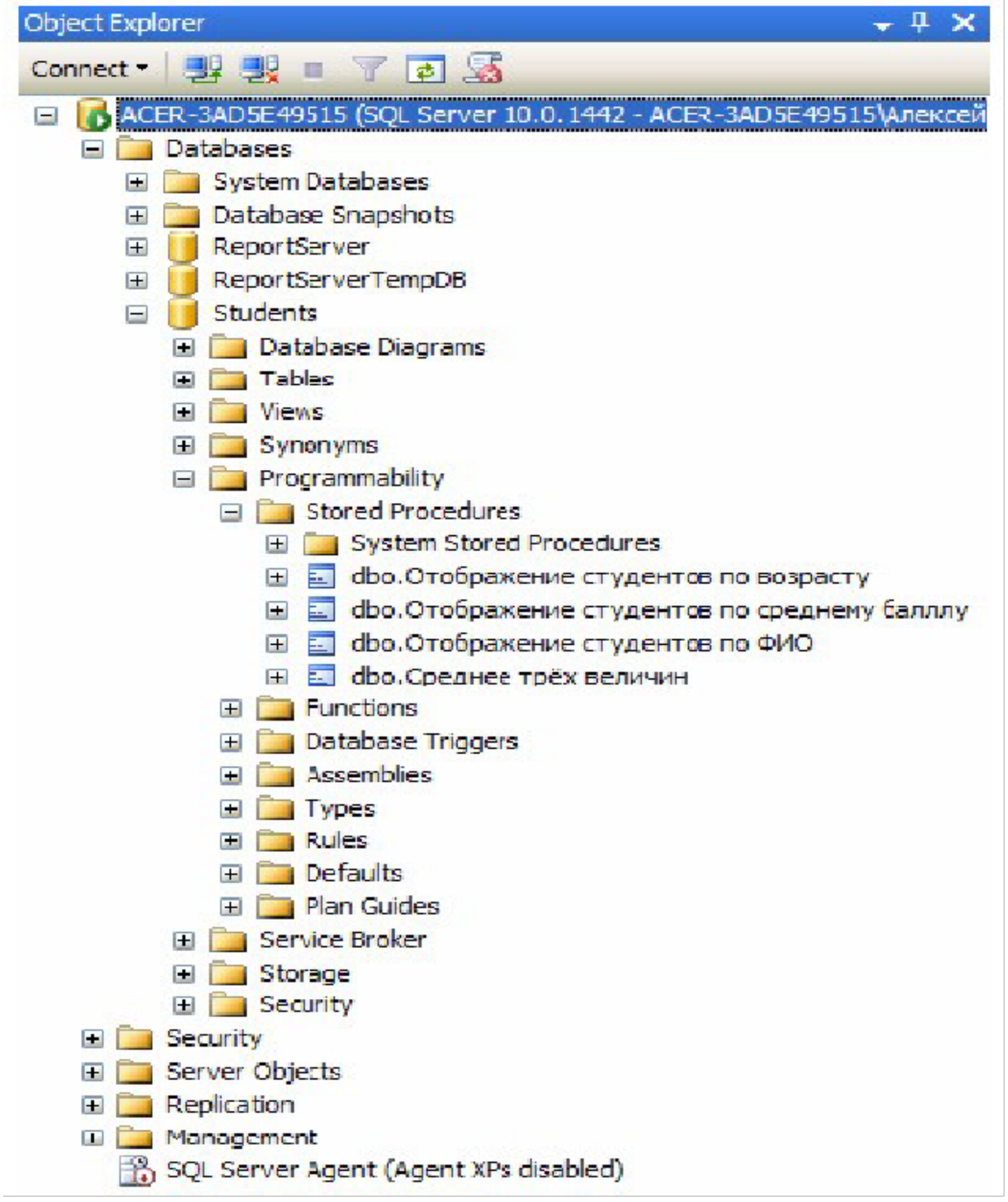


Рис. 10.11.

Контрольные вопросы

1. Что такое хранимая процедура?
2. Какова структура хранимой процедуры?

Лабораторная работа № 8
«Создание функций пользователя в Microsoft SQL Server 2012.»

Цель работы:

Научиться работать с пользовательскими функциями.

Теперь рассмотрим создание и применение пользовательских функций. В БД "Microsoft SQL Server 2008" все пользовательские функции находятся в папке **"Functions"** расположенной в папке **"Programmability"** в обозревателе объектов (рис. 12.1).

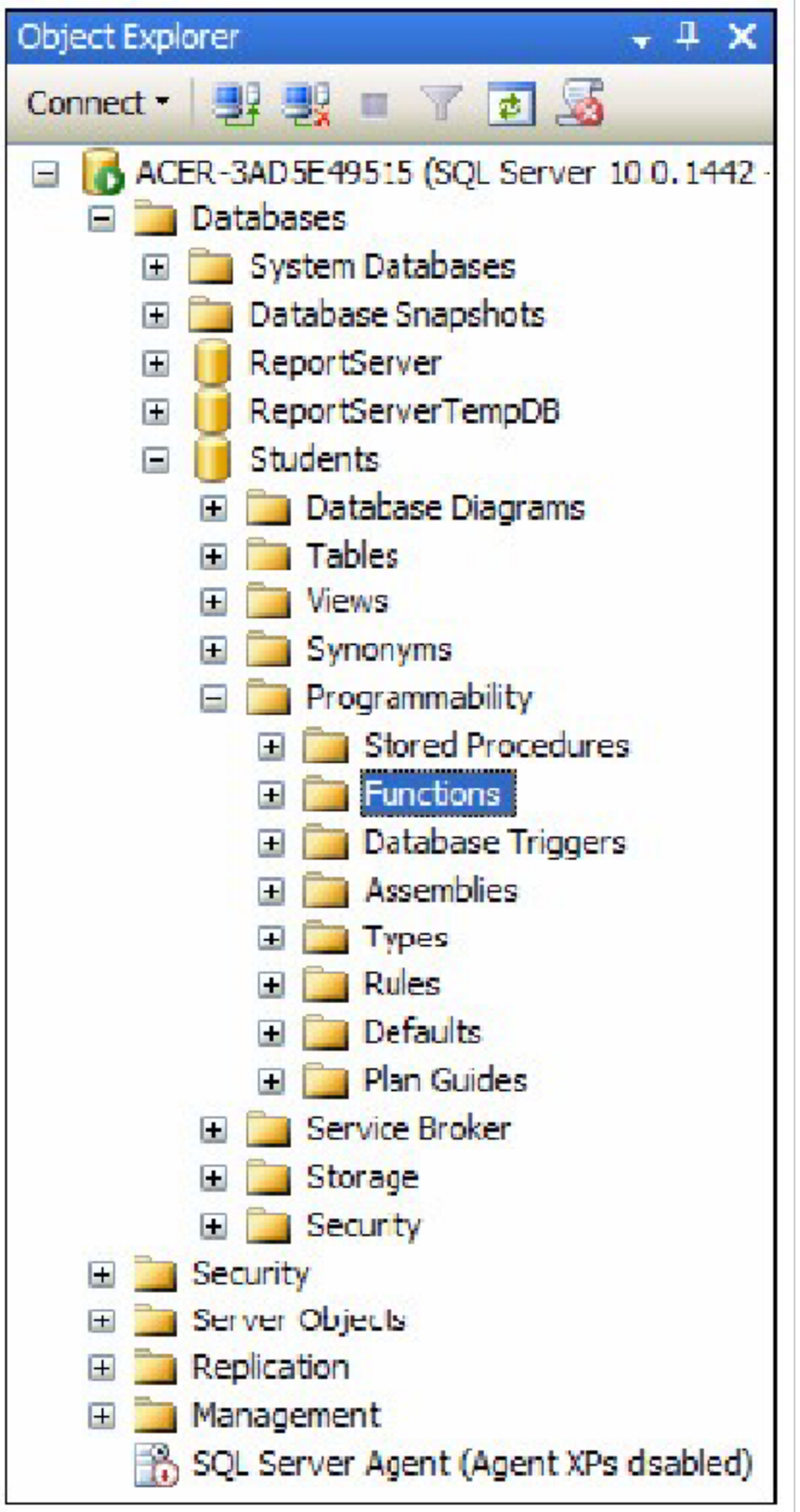
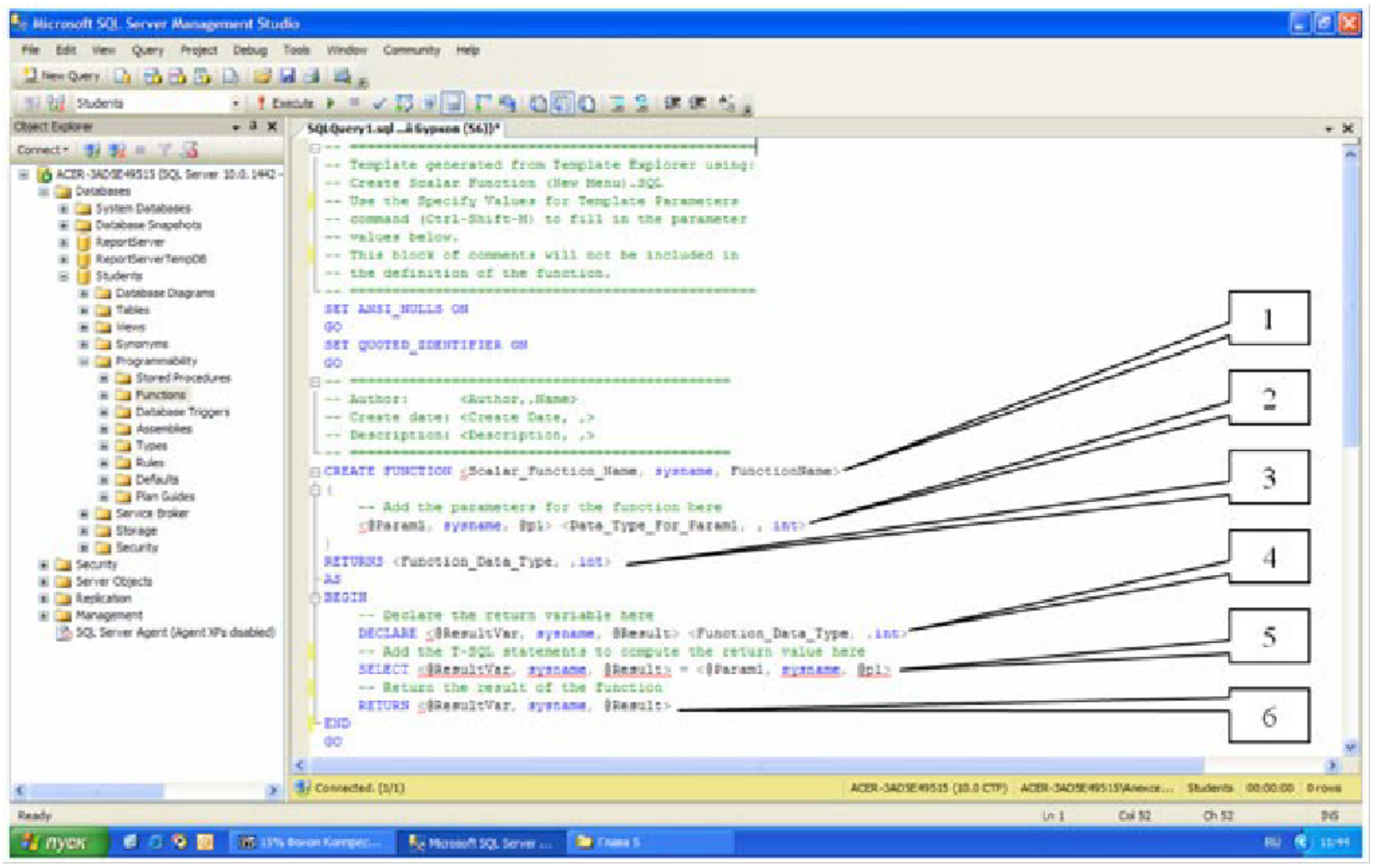


Рис. 12.1.

Начнем с создания скалярных пользовательских функций. Для создания новой скалярной пользовательской функции в обозревателе объектов, в БД **"Students"**, в папке **"Programmability"**, щелкните **ПКМ** по папке **"Functions"** и в появившемся меню выберите пункт **"New/Scalar-valued Function"**. Появится окно новой скалярной пользовательской функции (рис. 12.2)



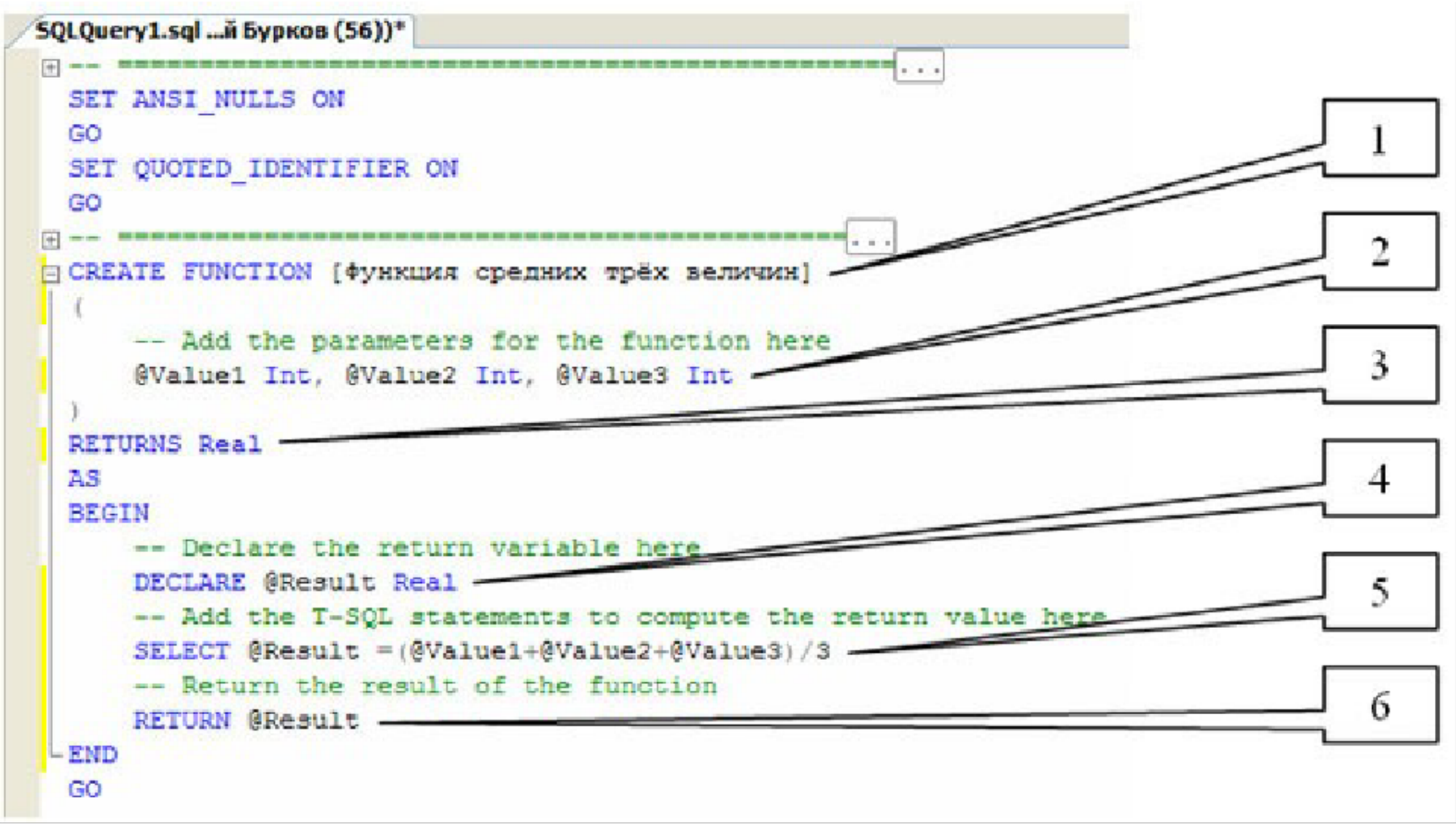
увеличить изображение

Рис. 12.2.

Синтаксис скалярной пользовательской функции похож на синтаксис хранимой процедуры (см. "Интерфейс информационных систем. Создание интерфейса пользователя"). Однако имеется ряд существенных отличий (рис. 12.2):

1. Область определения имени функции (Inline_Function_Name);
2. Параметры, передаваемые в процедуру **(@Param1)**. Определение параметров аналогично определению параметров в хранимой процедуре (см. "Таблицы. Типы данных и свойства полей. Создание и заполнение таблиц");
3. Тип данных значения возвращаемого процедурой;
4. Область объявления переменных, используемых внутри функции. Объявление переменных имеет следующий синтаксис:
   DECLARE @<Имя переменной> <Тип данных>
5. Тело самой пользовательской функции, содержит команды языка программирования запросов T-SQL;
6. Команда RETURN возвращающая результат выполнения функции. Имеет следующий синтаксис:
   RETURN @<Имя переменной с результатом>
   Переменная должна быть того же типа данных, который был указан в пункте 3.

Создадим скалярную пользовательскую функцию, вычисляющую среднее трех величин. В окне новой пользовательской функции наберите код представленный на рис. 12.3.



увеличить изображение

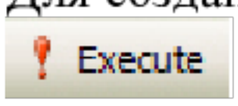
Рис. 12.3.

Рассмотрим более подробно код данной скалярной пользовательской функции (рис. 12.3):

1. CREATE FUNCTION [Функция средних трех величин] - определяет имя создаваемой функции как "Функция средних трех величин";
2. @Value1 Real, @Value2, @Value3 - определяют три параметра процедуры **Value1**, **Value2** и **Value3**. Данным параметрам можно присвоить целые числа (Тип данных Int);
3. RETURNS Real - показывает, что функция возвращает дробные числа (Тип данных Real);
4. DECLARE @Result Real - объявляется переменная @Result для хранения результата работы функции, то есть дробного числа (Тип данных Real);
5. SELECT @Result=(@Value1+@Value2+@Value3)/3 - вычисляет среднее и помещает результат в переменную @Result ;
6. RETURN @Result - возвращает значение переменной @Result.

Остальные фрагменты кода рассмотрены выше (рис. 12.2).

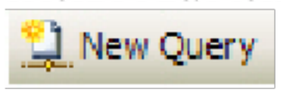
Для создания функции, выполним вышеописанный код, нажав кнопку



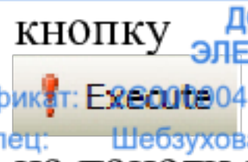
(Выполнить) на панели инструментов. В нижней части окна с кодом появиться сообщение **"Command(s) completed successfully."**. Закройте окно с кодом, щелкнув мышью по кнопке закрытия

расположенной в верхнем правом углу окна с кодом функции.

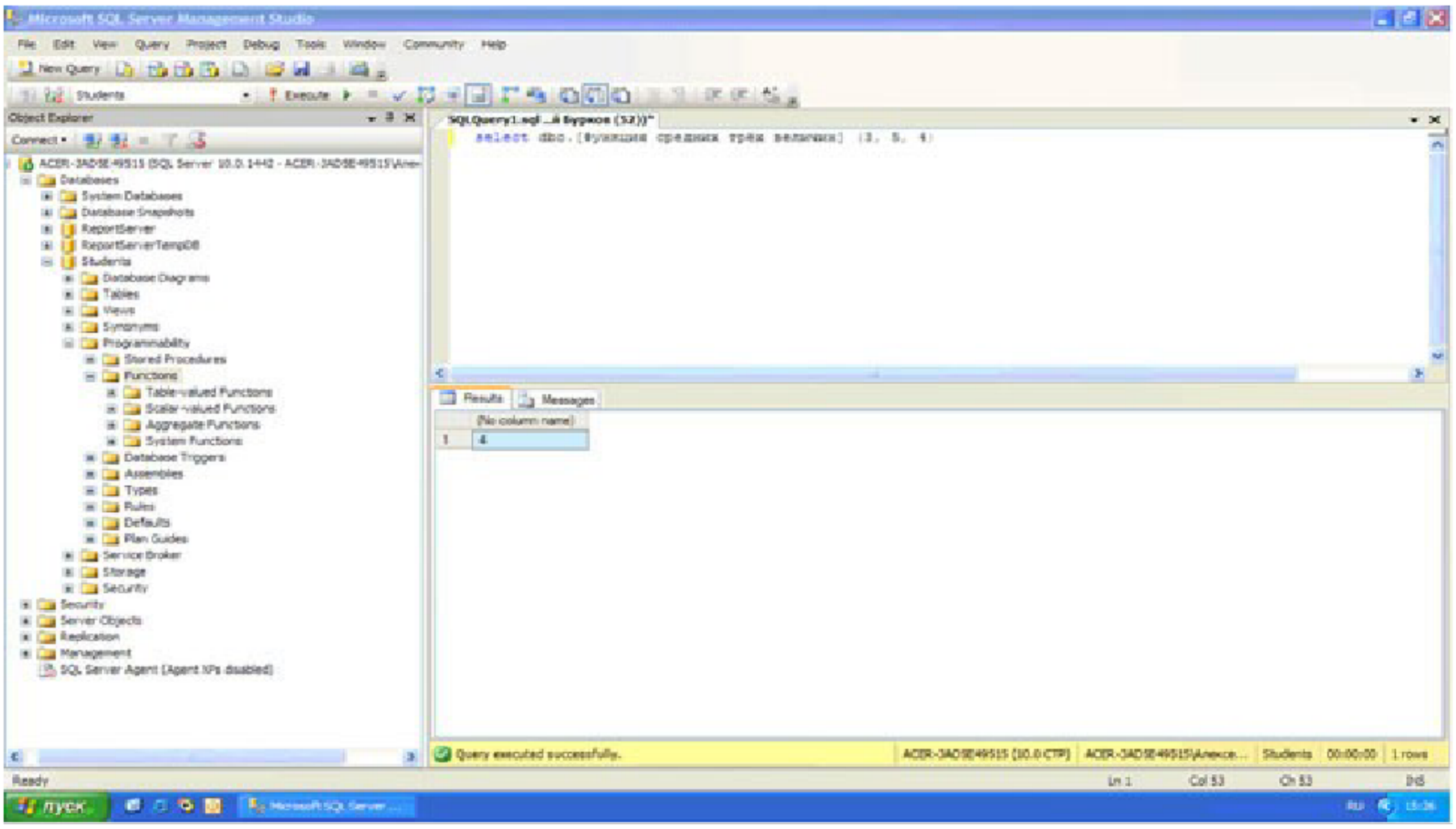
Проверим работу созданной скалярной пользовательской функции. Для запуска пользовательской функции необходимо создать новый пустой запрос, нажав на кнопку



(Новый запрос) на панели инструментов. В появившемся окне с пустым запросом наберите команду SELECT dbo.[Функция средних трех величин] (3, 5, 4) и нажмите кнопку



на панели инструментов (рис. 12.4).



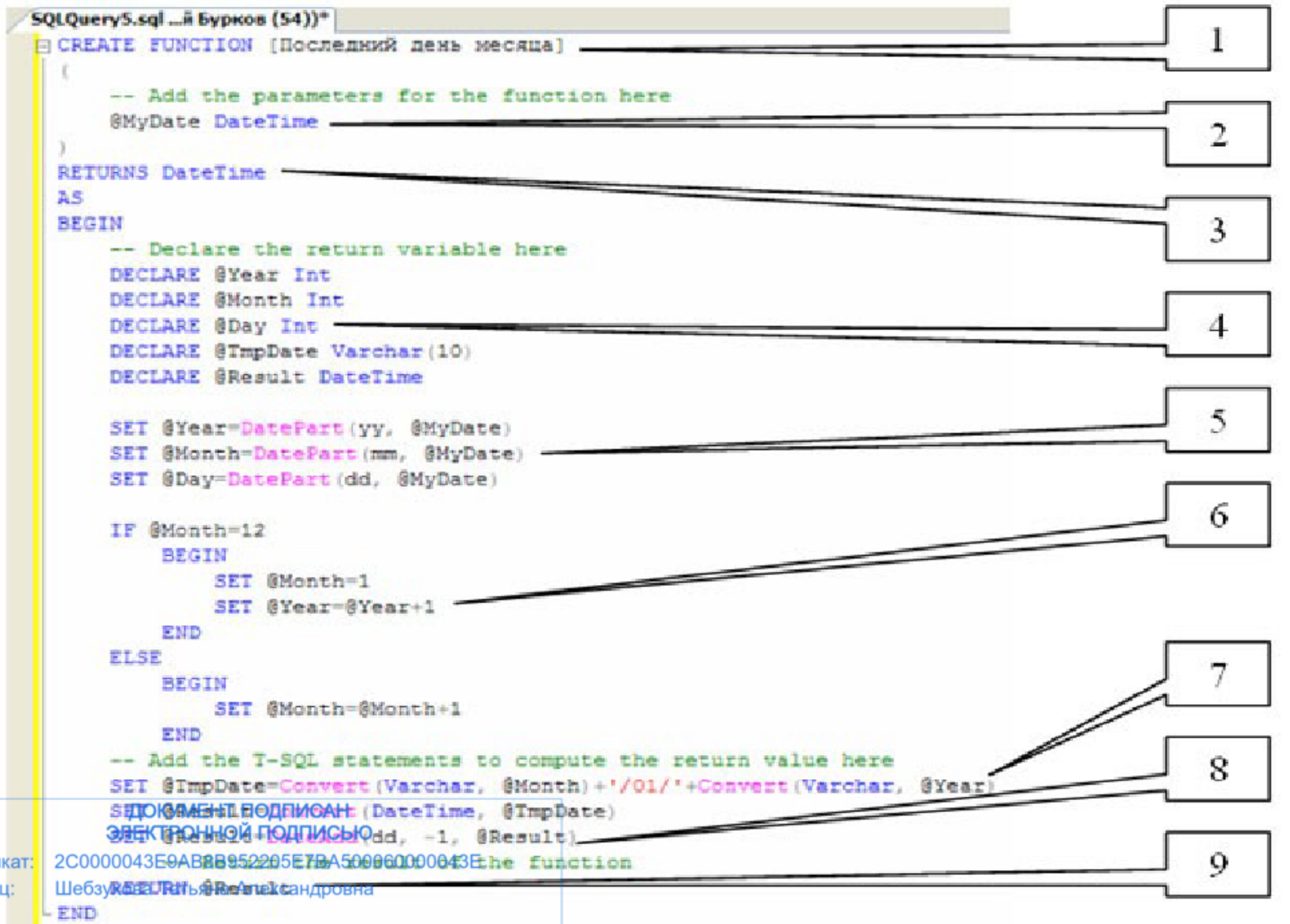
увеличить изображение

Рис. 12.4.

В нижней части окна с кодом появится результат выполнения новой скалярной пользовательской функции: **4** (рис. 12.4).

Теперь создадим более сложную скалярную пользовательскую функцию, предназначенную для определения последнего дня месяца введенной даты.

Создайте новую скалярную пользовательскую функцию, так как об этом сказано выше. В окне новой пользовательской функции наберите следующий код (рис. 12.5):



увеличить изображение

Рис. 12.5.

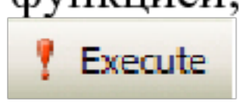
Перейдем к рассмотрению вышеприведенного кода (рис. 12.5). Код состоит из следующих групп команд:

1. CREATE FUNCTION [Последний день месяца] - определяет имя создаваемой функции как "Последний день месяца";
2. @MyDate - определяют параметр процедуры **MyDate**. Параметру можно присвоить значения дат или времени (Тип данных DateTime);
3. RETURNS DateTime - показывает, что функция возвращает дату или время (Тип данных DateTime);
4. DECLARE @Year Int, DECLARE @Month Int, DECLARE @Day Int - объявляются переменные @Year, @Month и @Day для хранения целочисленных значений года, месяца и дня введенной даты (Тип данных Int).
   DECLARE @TmpDate VarChar(10) объявляет переменную "TmpDate" для хранения промежуточного значения даты в строке длинной до 10 символов (Тип данных VarChar(10)).
   DECLARE @Result DateTime объявляет переменную "Result" для хранения результата - даты последнего дня месяца (Тип данных DateTime).
5. SET @Year=DatePart(yy, @MyDate), SET @Month=DatePart(mm, @MyDate), SET @Day=DatePart(dd, @MyDate) - определяются части введенной даты и помещаются в переменныне @Year, @Month и @Day. Для определения частей даты используется функция **DatePart**, имеющая следующий синтаксис: DatePart(<часть даты>, <дата>). Здесь **"часть даты"** - это закодированная специальными символами определяемая часть даты (yy - год, mm - месяц, dd - день), **"дата"** - это дата, части которой определяем.
6. IF @Month=12
7. BEGIN
8. SET @Month=1
9. SET @Year=@Year+1
10. END
11. ELSE
12. BEGIN
13. SET @Month=@Month+1
14. END

    Вышепреведенный фрагмент кода выполняет следующие действия: Если номер месяца равен 12 то установить номер месяца (@Month) равным 1 и увеличить год (@Year) на 1, иначе увеличить месяц на 1.
15. SET @TmpDate=Convert(Varchar, @Month)+'/01/'+Convert(Varchar, @Year), SET @Result=Convert(DateTime, @TmpDate) - переводит числовые значения даты в дату в строковом формате и записывает ее в переменную @TmpDate, затем переводит дату в строковом формате в тип данных даты и времени и помещает ее в переменную @Result. Для конвертации используется функция **Convert**, имеющая следующий синтаксис:
    Convert(<тип данных>, <значение>), здесь "тип данных" это тип данных в который переводится "значение".
16. SET @Result=DateAdd(dd, -1, @Result) - из даты, хранимой в перменной @Result вычитается 1 день, для этого используется функция **DateAdd**, имеющая следующий синтаксис:
    DateAdd(<часть даты>, <количество периодов>, <дата>) - здесь "часть даты" - это закодированная специальными символами определяемая часть даты (см. функцию **DatePart**), "количество периодов" - это количество частей даты прибавляемой к введенной дате (параметр **"дата"**).

17. RETURN @Result - возвращает значение, хранимое в переменной @Result.

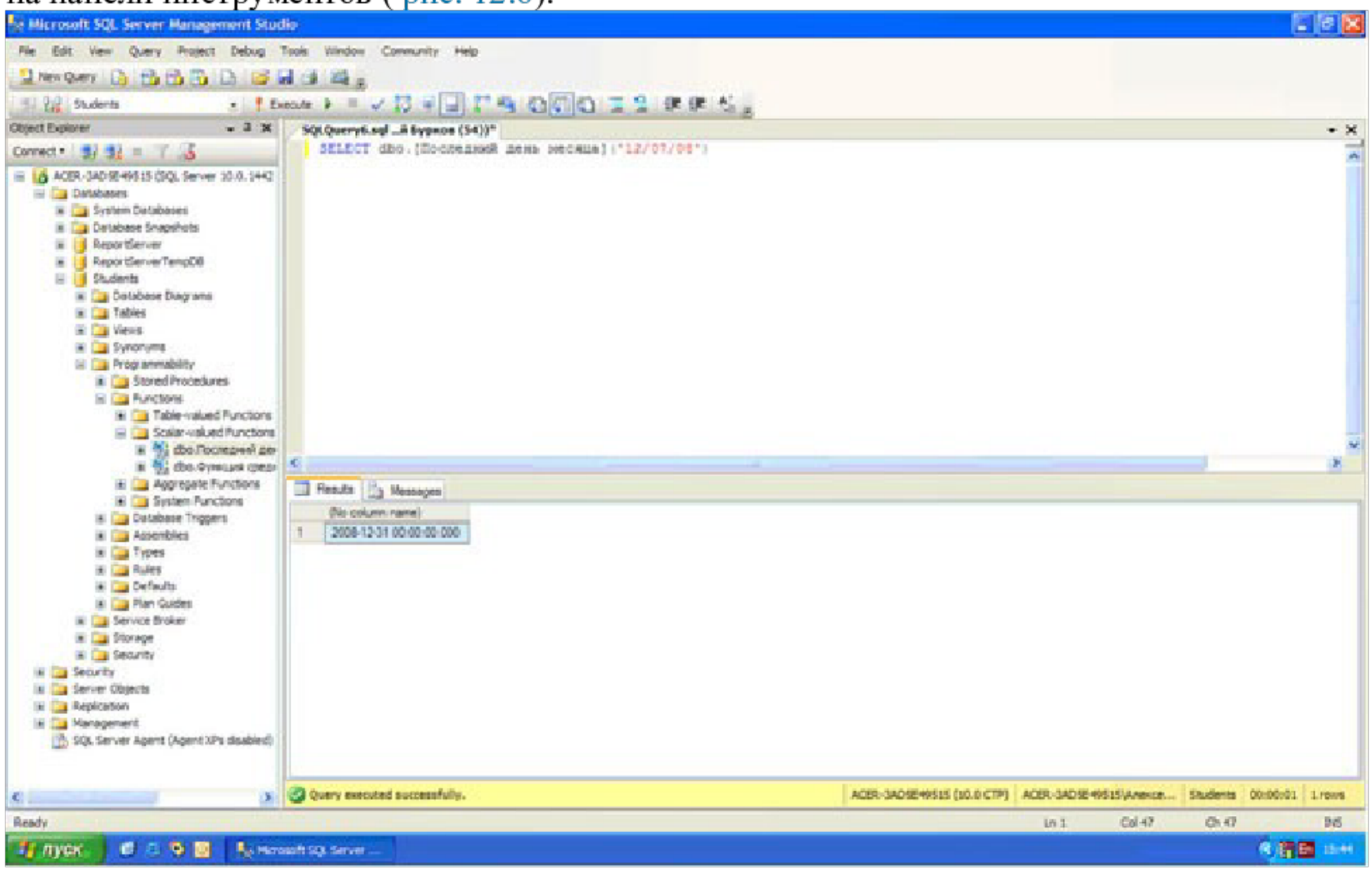
Для создания функции, выполним вышеописанный код, как и в случае с предыдущей функцией, нажав кнопку



После появления сообщения **"Command(s) completed successfully."** закройте окно с кодом.

Проверим работу функции **"Последний день месяца"** выполнив ее. Создайте новый пустой запрос, затем в окне с пустым запросом наберите команду SELECT dbo. [Последний день месяца] ('12/07/08') и нажмите кнопку

на панели инструментов (рис. 12.6).

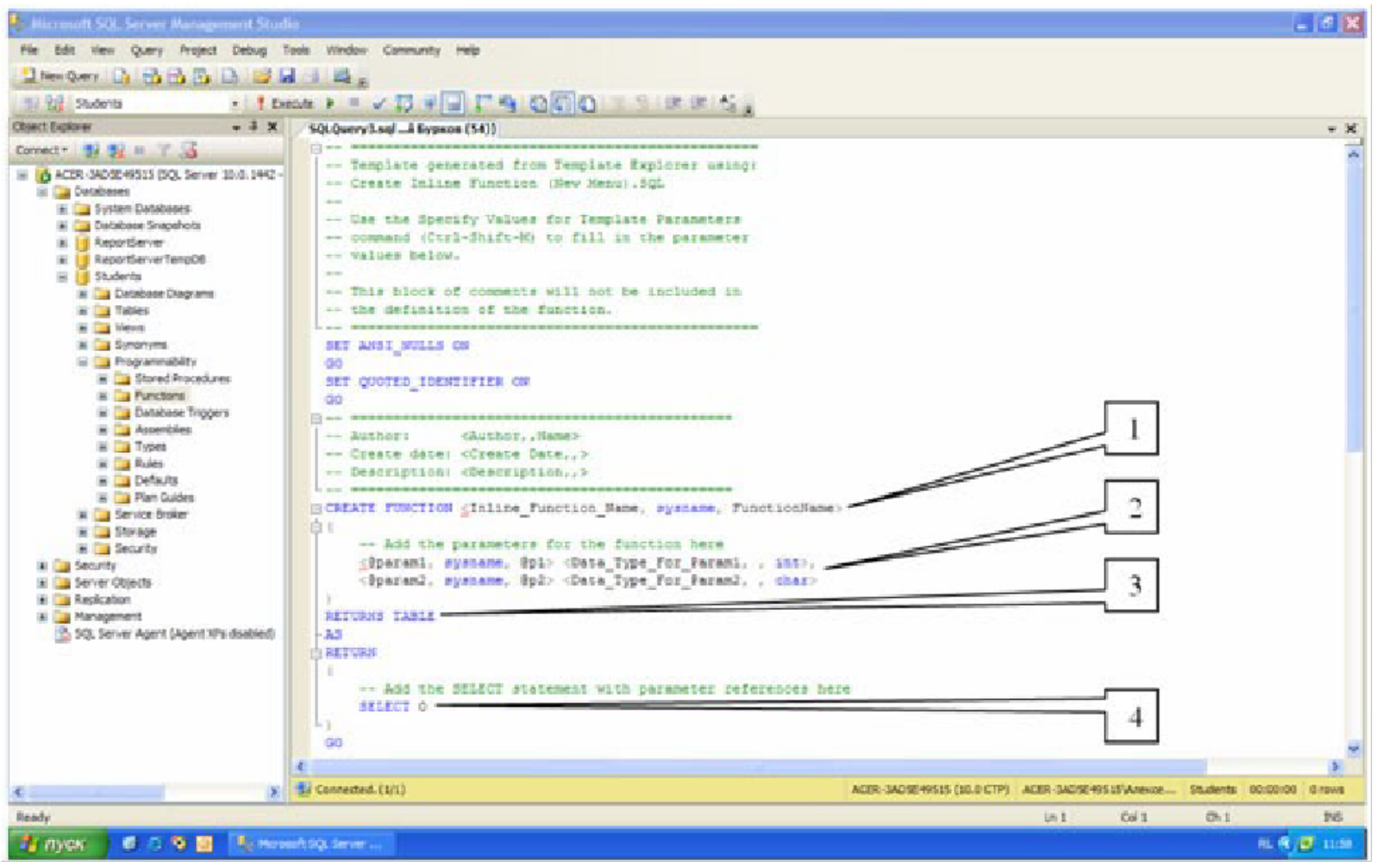


увеличить изображение

Рис. 12.6.

Появится результат выполнения новой скалярной пользовательской функции: **2008-12-31** (рис. 12.6).

Теперь перейдем к созданию табличных пользовательских функций. Для создания табличной пользовательской функции в обозревателе объектов, в БД **"Students"**, в папке **"Programmability"**, щелкните **ПКМ** по папке **"Functions"** и в появившемся меню выберите пункт **"New/Table-valued Function"**. Появится окно новой табличной пользовательской функции (рис. 12.7)



увеличить изображение

Рис. 12.7.

Рассмотрим структуру кода табличной пользовательской функции. Табличная пользовательская функция состоит из следующих разделов:

1. Область определения имени функции (Inline_Function_Name);
2. Параметры, передаваемые в процедуру (**@Param1**, **@Param2**);
3. RETURNS TABLE показывает что функция является табличной, то есть возвращает таблицу;
4. Тело самой пользовательской функции, состоит из команды SELECT языка программирования запросов T-SQL.

Остальные разделы табличной пользовательской функции аналогичны таким же разделам хранимых процедур и скалярных пользовательских функций.

В заключение рассмотрим создание табличной пользовательской функции **"Функция отбора по возрасту"**, вычисляющих текущий возраст студентов в зависимости от их даты рождения. В окне новой пользовательской функции (рис. 12.7) наберите следующий код (рис. 12.8):

```
SQLQuery11.sql... Бурков (61))*
-- ================================================...
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- ================================================...
CREATE FUNCTION [Функция отбора по возрасту]
(
    -- Add the parameters for the function here
)
RETURNS TABLE
AS
RETURN
(
    -- Add the SELECT statement with parameter references here
    SELECT ФИО,[Дата рождения], Возраст = DateDiff (yy,[дата рождения], GetDate())
    FROM Студенты

)
GO
```

увеличить изображение

Рис. 12.8.

Из кода представленного на рис. 12.8 видно, что данная табличная функция не имеет параметров и реализуется командой

SELECT ФИО, [Дата рождения], Возраст = DateDiff(yy, [Дата рождения], GetDate())
FROM Студенты

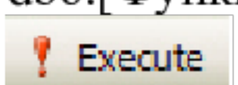
Из вышепредставленной команды видно, что из таблицы **"Студенты"** отображаются поля **"ФИО"** и **"Дата рождения"**, а также вычислимое поле **"Возраст"**. Поле **"Возраст"** вычисляется при помощи встроенной функции **DateDiff** вычисляющей различие между датами в определенных единицах измерения (частях даты) и имеющей следующий синтаксис:

DateDiff(<часть даты>, <начальная дата>, <конечная дата>).

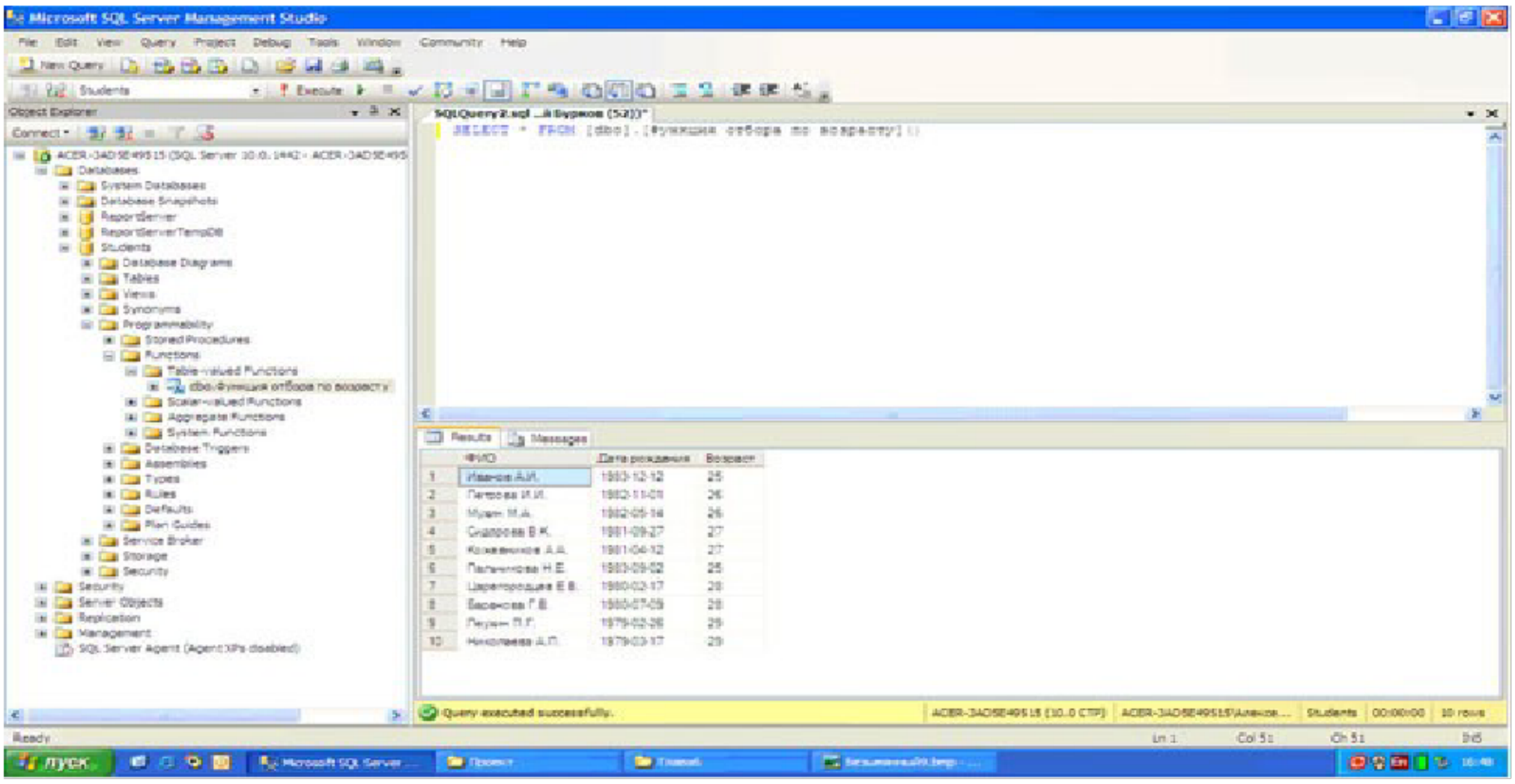
Здесь "часть даты" - это закодированные специальными символами единицы измерения (часть даты) (yy - год, mm - месяц, dd - день), "начальная дата" - дата начала периода и "конечная дата" - дата конца периода. В нашем случае в качестве начальной даты берем дату рождения студента, а в качестве конечной даты берем текущую дату (функция **GetDate()**).

Для создания функции, выполним вышеописанный код, как и в случае с предыдущей функцией. После появления сообщения **"Command(s) completed successfully."** закройте окно с кодом.

Проверим работоспособность новой табличной пользовательской функции. Создайте новый пустой запрос, затем в окне с пустым запросом наберите команду SELECT * FROM dbo.[Функция отбора по возрасту]() и нажмите кнопку



на панели инструментов (рис. 12.9).



увеличить изображение

Рис. 12.9.

В нижней части окна появиться таблица с фамилиями, датами рождения и возрастом студентов на данный момент времени (рис. 12.9).

Замечание: Обратите внимание на тот факт, что мы работаем с табличной функцией как с обыкновенной таблицей.

На этом мы заканчиваем рассмотрение пользовательских функций и переходим к рассмотрению целостности данных, диаграмм и триггеров. По окончании выполнения главы 6 обозреватель объектов будет иметь следующий вид (рис. 12.10):

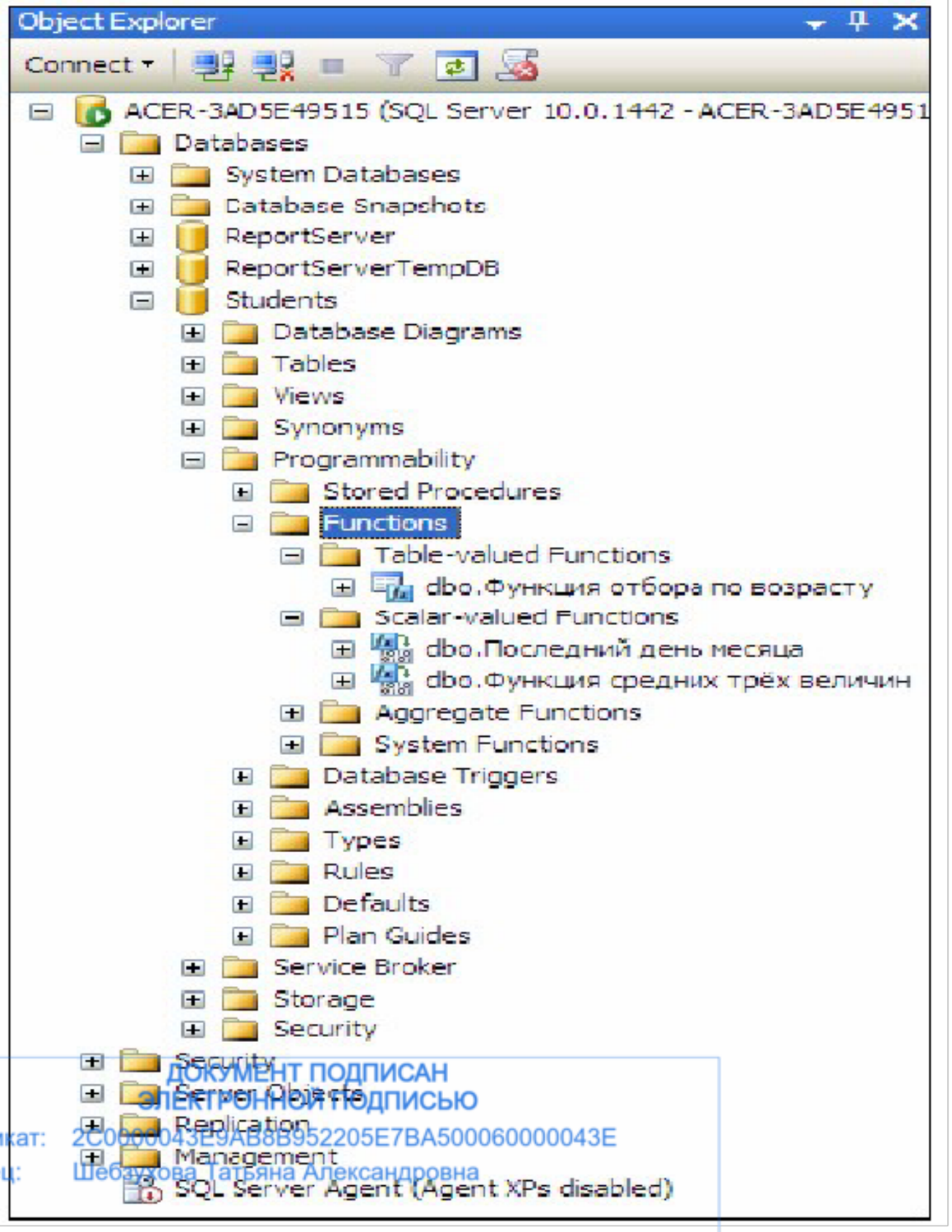


Рис. 12.10.

Контрольные вопросы

1 Назначение пользовательских функций.

2. Синтаксис скалярной пользовательской функции.

3. Табличные пользовательские функции.

Лабораторная работа № 9

«Обеспечение целостности данных в Microsoft SQL Server 2012. Создание диаграмм и триггеров.»

Цель работы:

Научиться создавать диаграммы и триггеры.

Перейдем теперь к созданию диаграмм. В БД "Microsoft SQL Server 2008" все диаграммы находятся в папке **"Database Diagrams"** обозревателя объектов (рис. 14.1).

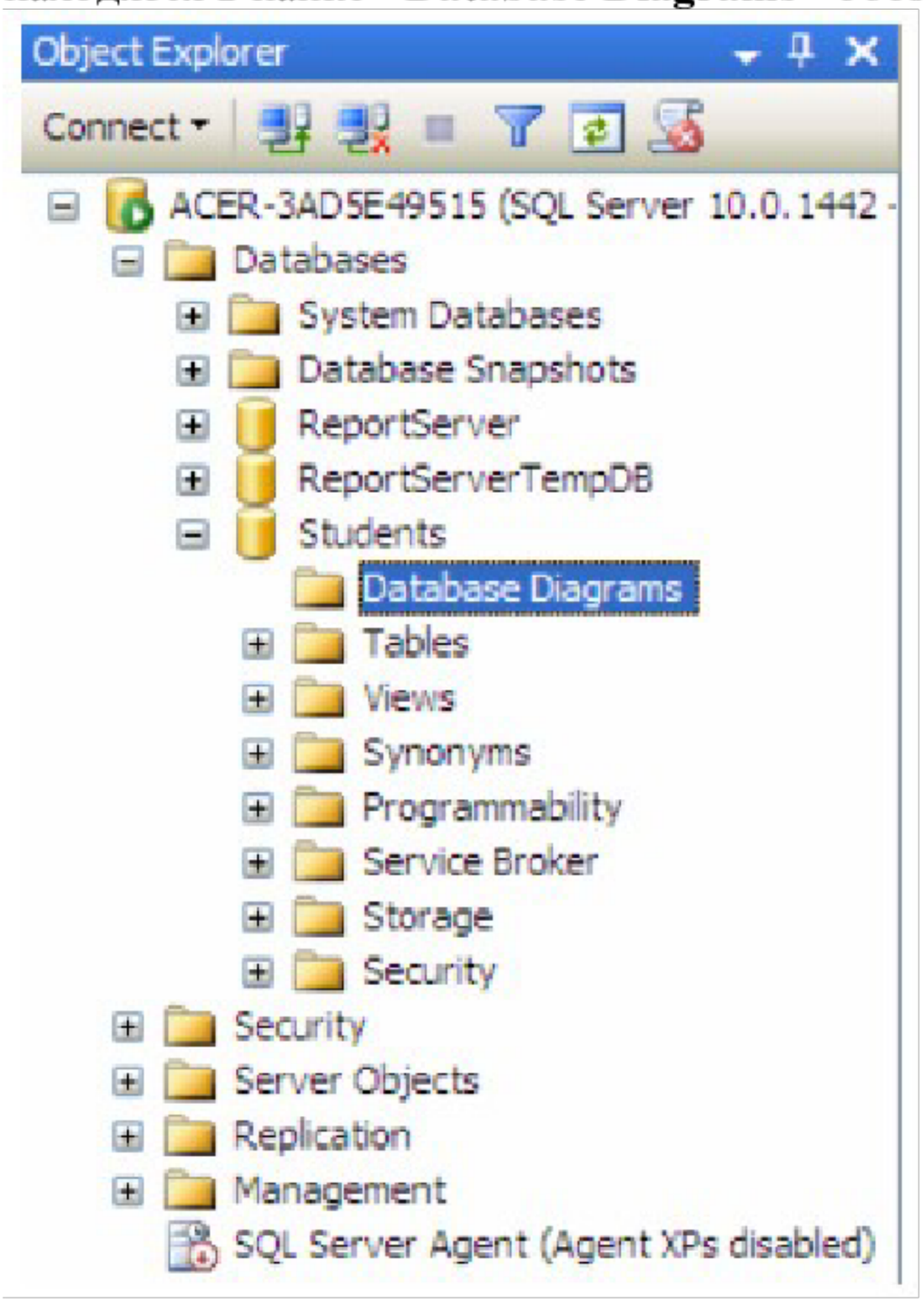


Рис. 14.1.

Создадим диаграмму, обеспечивающую целостность данных нашей БД **"Students"**. Для создания новой диаграммы в БД **"Students"** щелкните **ПКМ** по папке **"Database Diagrams"** и в появившемся меню выберем пункт **"New Database Diagram"**. Сначала появится окно с вопросом о добавлении нового объекта **"Диаграмма"**. В этом окне нужно нажать кнопку **"Yes"**. Затем появится окно **"Add Table"** предназначенное для добавления таблиц в новую диаграмму (рис. 14.2).

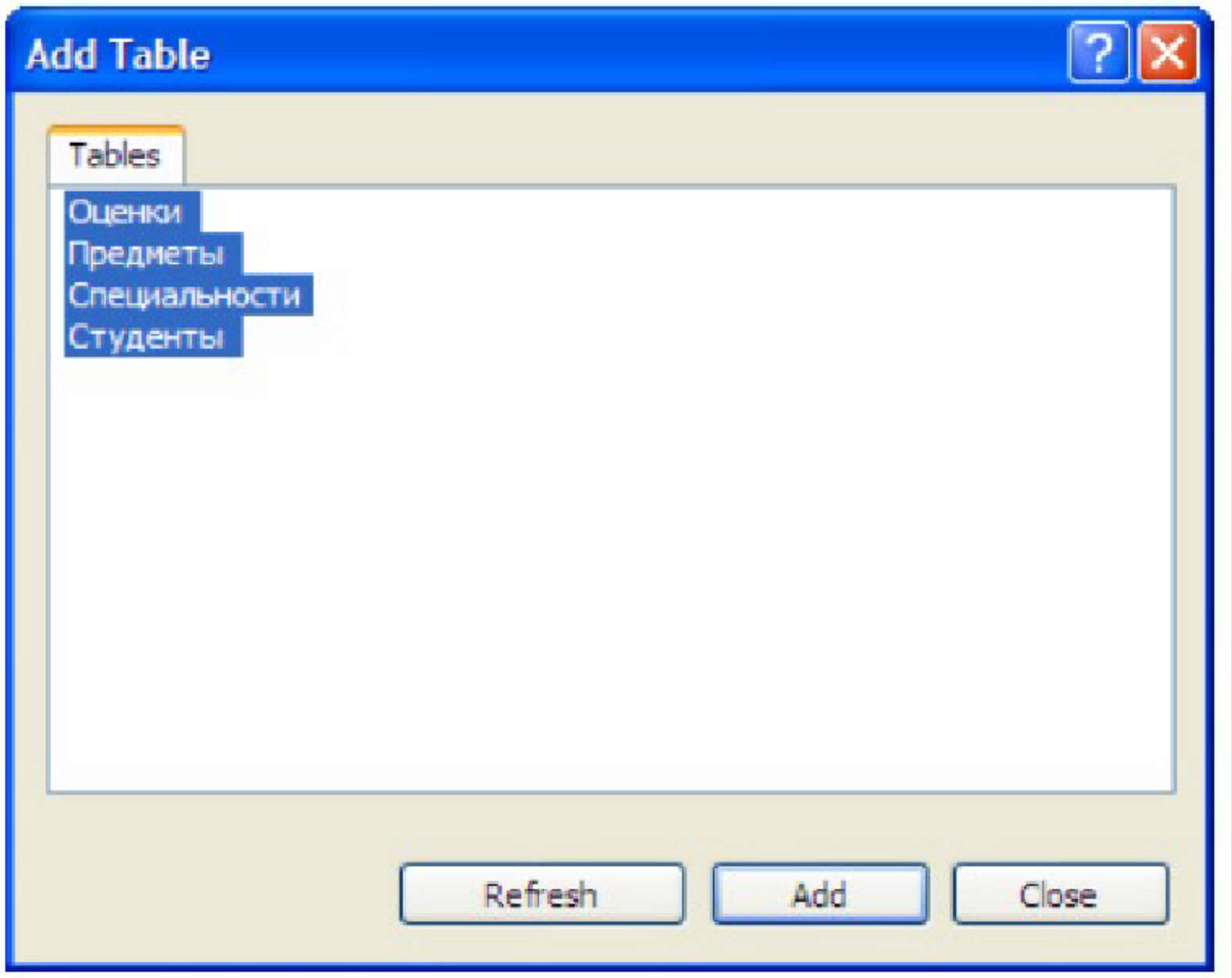


Рис. 14.2.

В окне добавления таблиц выделите все таблицы нашей БД и нажмите кнопку **"Add"** (рис. 14.2). Закройте окно **"Add Table"** нажатием на кнопку **"Close"**.

Появится окно диаграммы, где будут отображены отобранные таблицы. Теперь необходимо определить связи между таблицами. Перетащите поле **"Код специальности"** из таблицы **"Специальности"** на такое же поле в таблице **"Студенты"**. Появится окно создания связи между таблицами **"Tables and Columns"** (рис. 14.3).

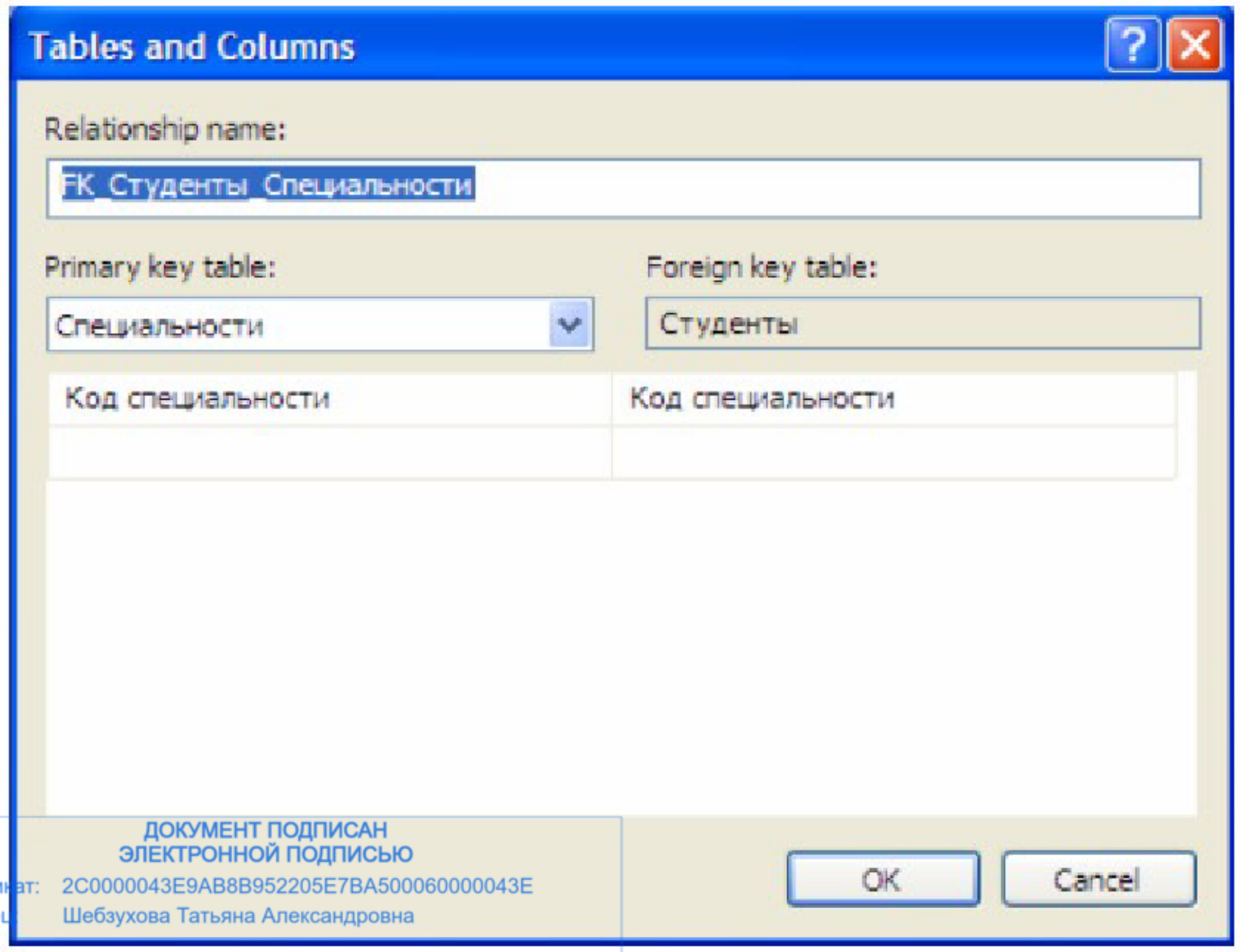
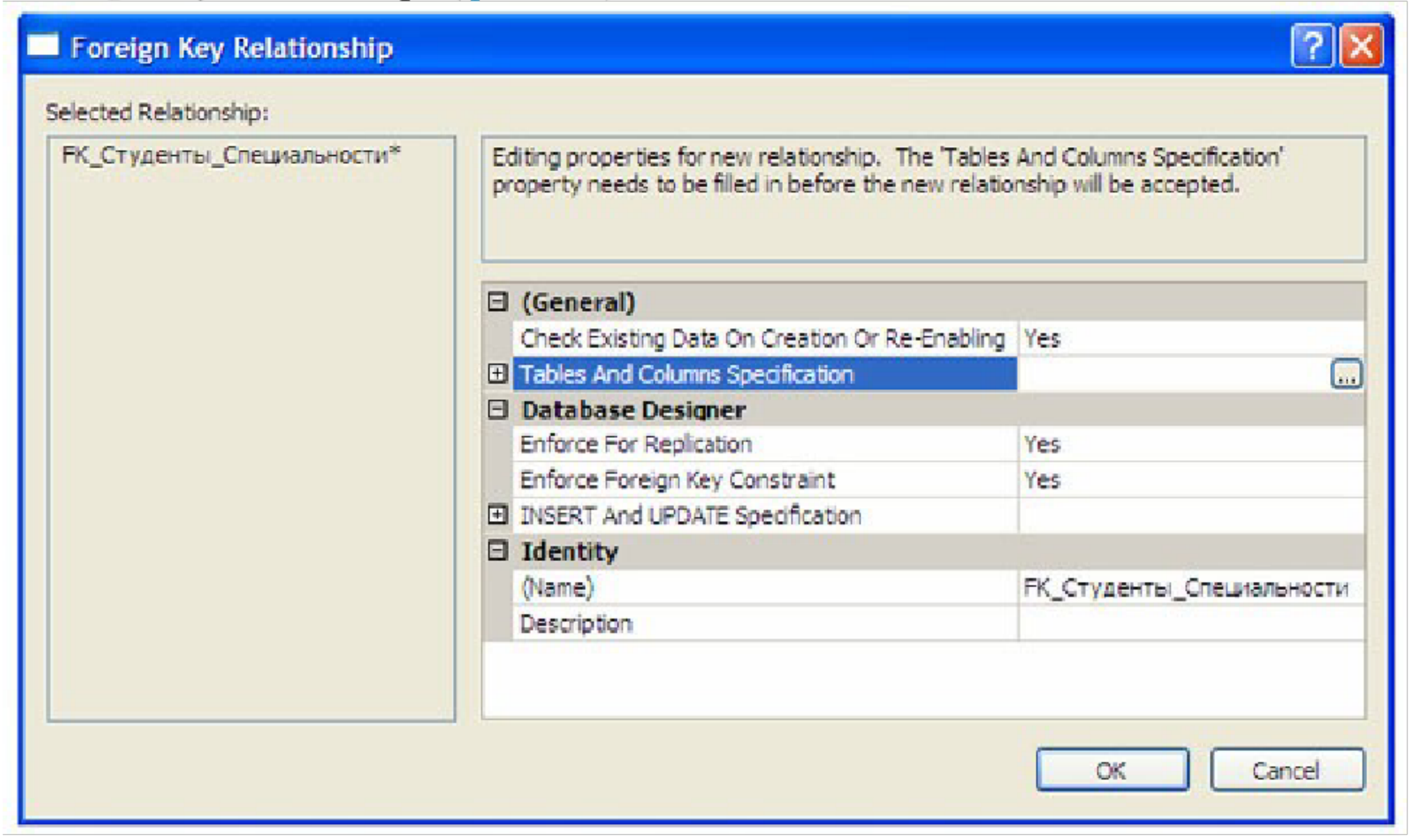


Рис. 14.3.

В окне создания связи нажмите кнопку **"Ok"**. Появится окно настройки свойств связи **"Foreign Key Relationship"** (рис. 14.4).

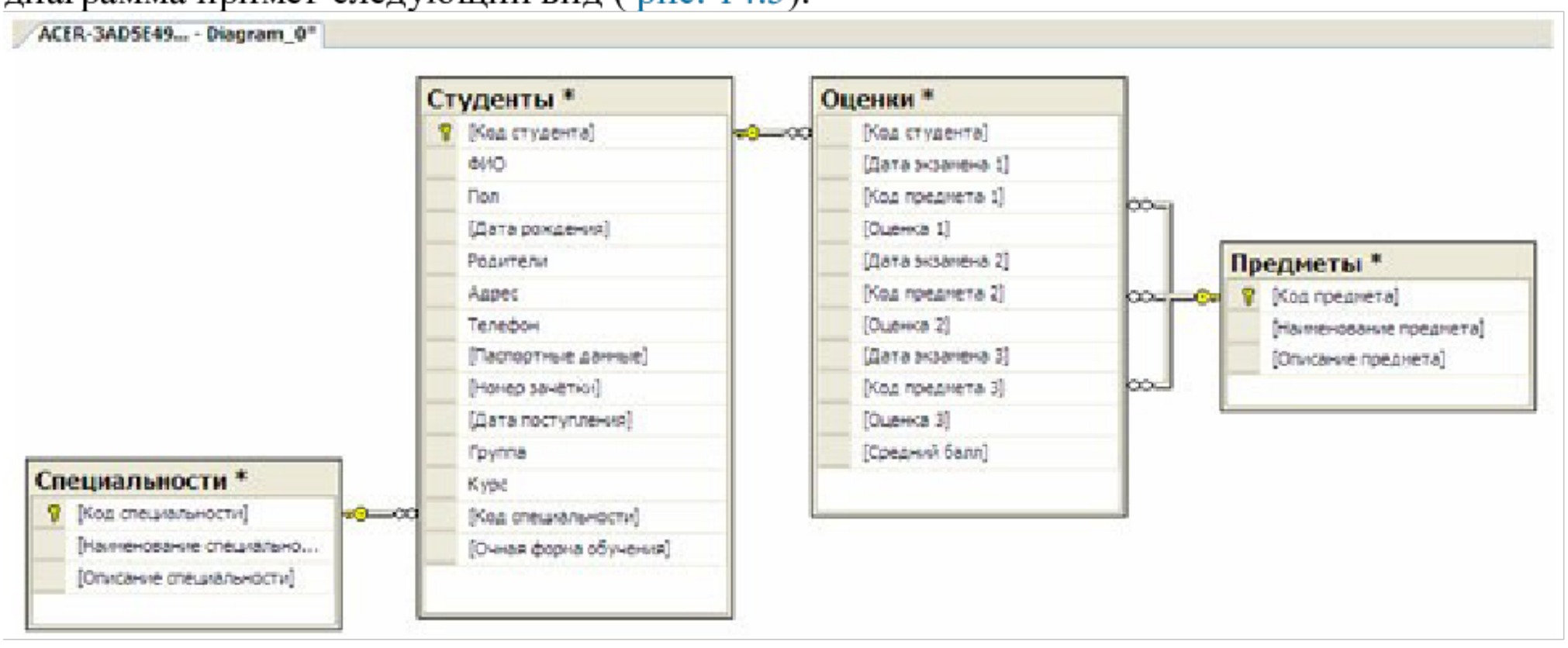


увеличить изображение

Рис. 14.4.

Оставьте свойства связи без изменений и в окне свойств связи нажмите кнопку **"Ok"**.В диаграмме между таблицами **"Студенты"** и **"Специальности"** появится связь в виде ломаной линии (рис. 14.5).

Аналогичным образом создайте связь таблицы **"Студенты"** с таблицей **"Оценки"**, перетащив поле **"Код студента"** из таблицы **"Студенты"** на одноименное поле в таблице **"Оценки"**. Затем, свяжите таблицы **"Предметы"** и **"Оценки"**, перетащив поле **"Код предмета"** из таблицы **"Предметы"** на поля **"Код предмета 1"**, **"Код предмета 2"** и **"Код предмета 3"** таблицы **"Оценки"**. После выполнения вышеперечисленных действий диаграмма примет следующий вид (рис. 14.5).



увеличить изображение

Рис. 14.5.

Закройте окно с диаграммой, щелкнув мышью по кнопке закрытия

расположенной в верхнем правом углу окна с диаграммой. Появится окно с вопросом о сохранении новой диаграммы, где необходимо нажать кнопку **"Yes"** (рис. 14.6).

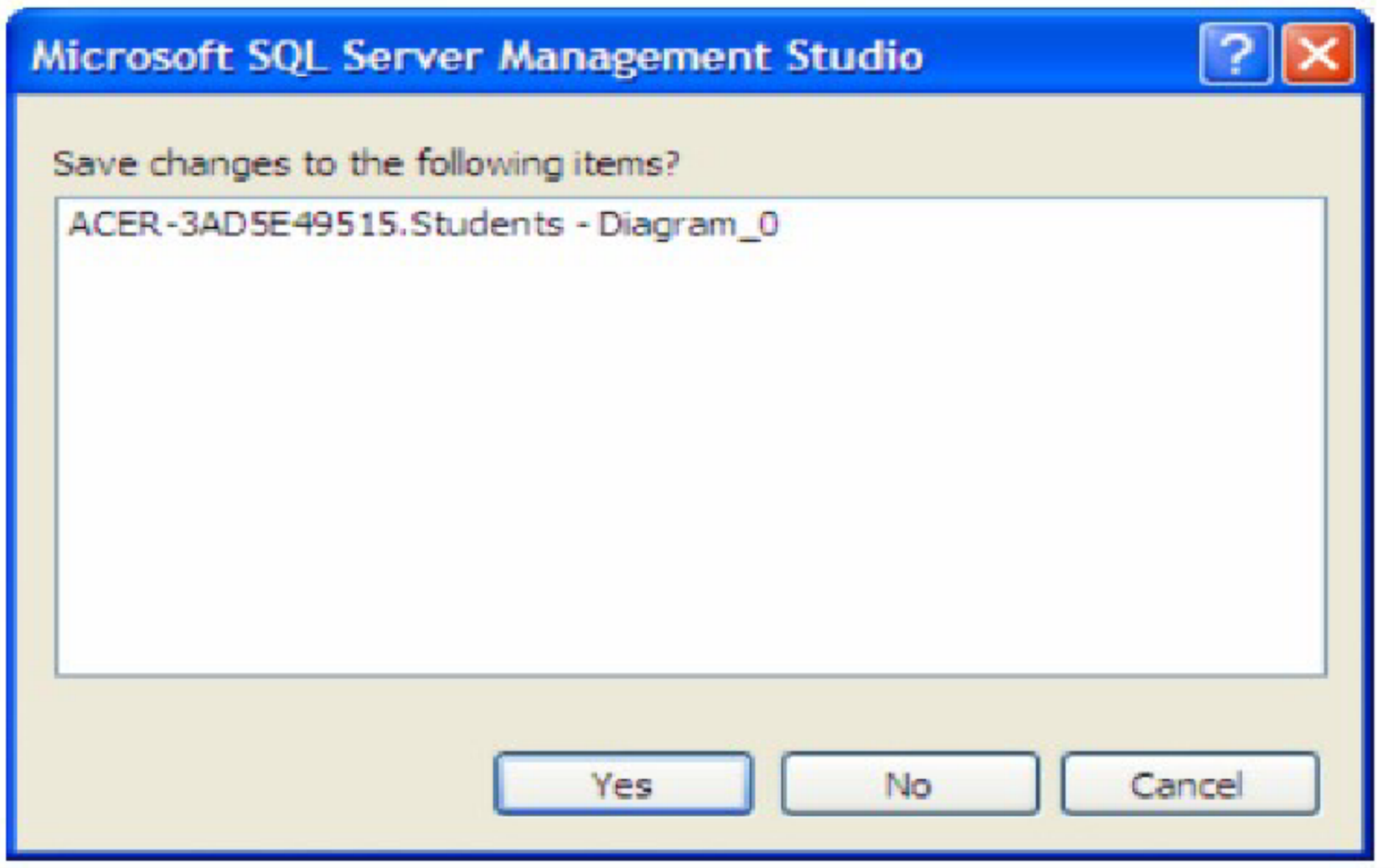


Рис. 14.6.

Появится окно определения имени новой диаграммы **"Choose Name"**. В окне определения имени, задайте имя диаграммы как "Диаграмма БД Студенты" и нажмите кнопку **"Ok"** (рис. 14.7).

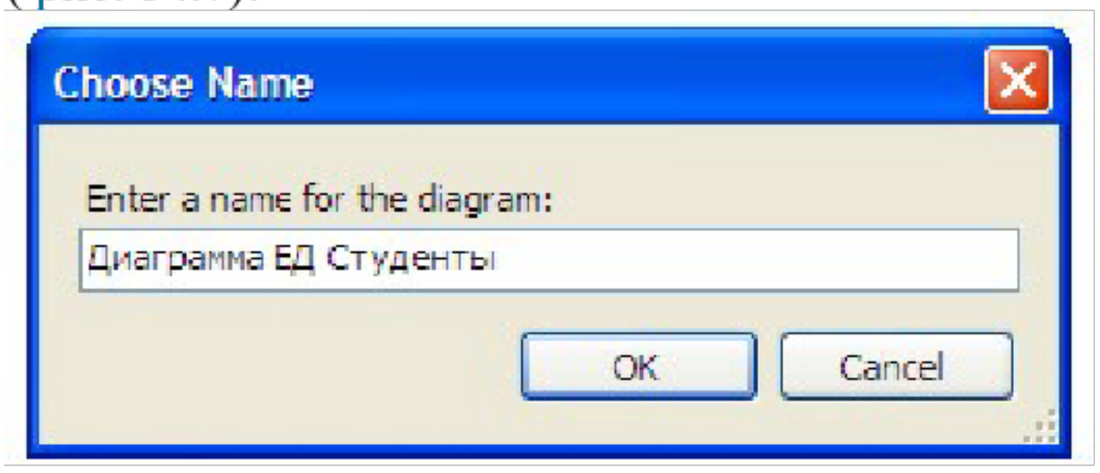


Рис. 14.7.

Появится окно **"Save"** с запросом сохранения таблиц, входящих в диаграмму. В данном окне необходимо нажать кнопку **"Yes"** (рис. 14.8).

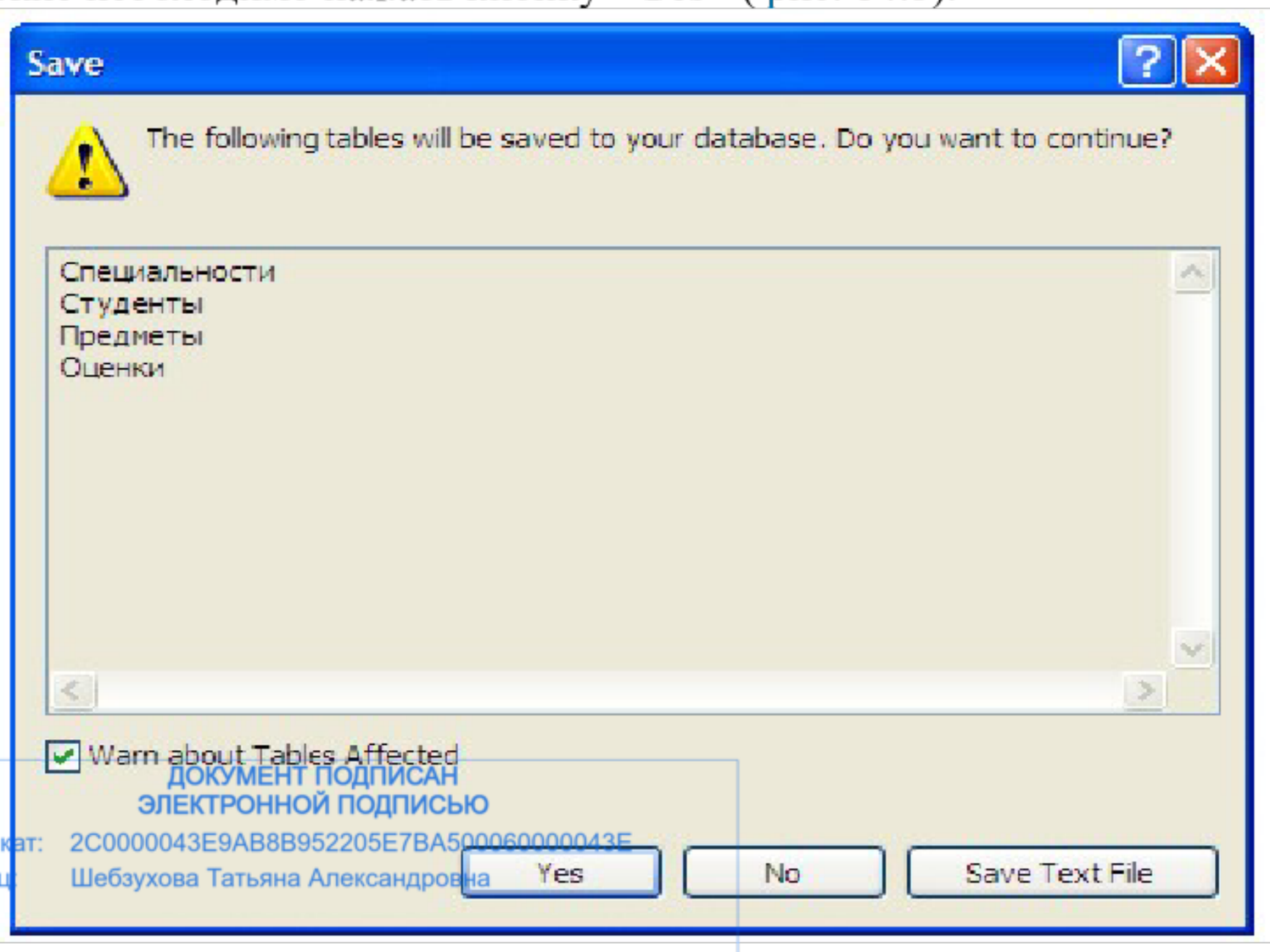


Рис. 14.8.

Перейдем к созданию триггеров. Создадим триггеры для таблицы **"Студенты"**. Триггеры создаются отдельно для каждой таблицы и располагаются в обозревателе объектов в папке **"Triggers"**. В нашем случае, папка **"Triggers"** входит в состав таблицы **"Студенты"** (рис. 14.9).

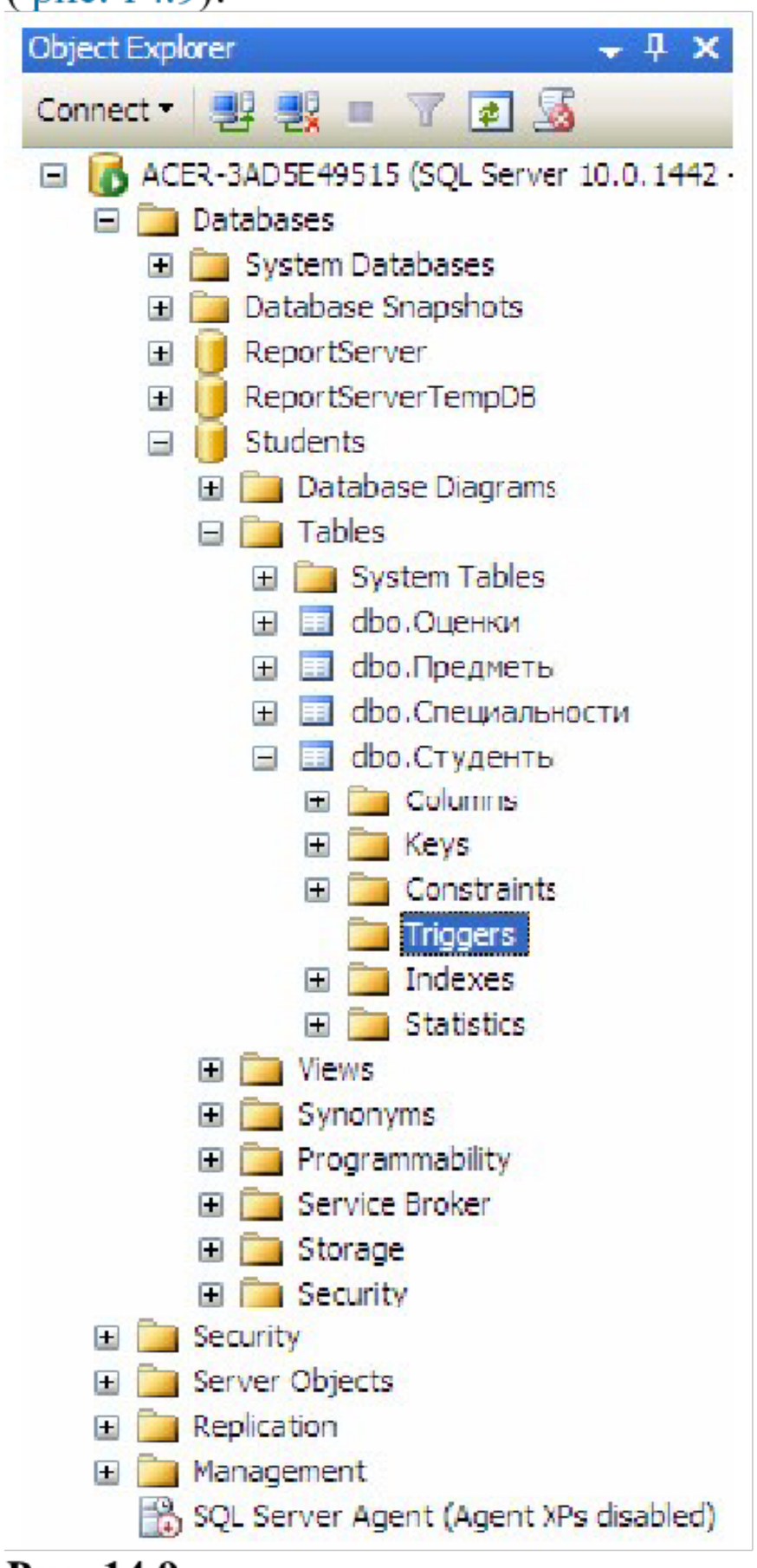


Рис. 14.9.

Для начала создадим триггер, выводящий сообщение **"Запись добавлена"** при добавлении записи в таблицу **"Студенты"**. Создадим новый триггер, щелкнув **ПКМ** по папке **"Triggers"** в таблице **"Студенты"** и в появившемся меню выбрав пункт **"New Trigger"**. Появится следующее окно с новым триггером (рис. 14.10):