Оглавление

[1. Создание Linked Server. Подключение к внешним данным 2](#_Toc420051512)

[1.1. Тестовое создание Linked Server 2](#_Toc420051513)

[1.2. Доступ к данным из другого экземпляра SQL Server 3](#_Toc420051514)

[1.3. Подключение к XLSX файлу 4](#_Toc420051515)

[2. Использование функции OPENDATASOURCE 5](#_Toc420051516)

[3. Использование функции OPENQUERY 6](#_Toc420051517)

[4. Использование функции OPENROWSET 7](#_Toc420051518)

[4.1. Импорт данных с использованием не XML файла форматирования 7](#_Toc420051519)

[4.2. Импорт данных с использованием XML файла форматирования 8](#_Toc420051520)

[4.3. Запрос данных без импортирования в таблицу 8](#_Toc420051521)

[5. Использование функции OPENXML 9](#_Toc420051522)

[6. Организация Paging’а на стороне сервера 10](#_Toc420051523)

[7. Массовый импорт и экспорт XML данных 10](#_Toc420051524)

[8. Полнотекстовый и нечеткий поиск 12](#_Toc420051525)

[8.1. Тестовый пример 12](#_Toc420051526)

[8.2. Пример полнотекстового поиска на нашей базе 15](#_Toc420051527)

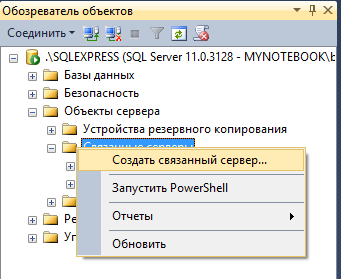
[9. Вывод 15](#_Toc420051528)

# Создание Linked Server. Подключение к внешним данным

# Тестовое создание Linked Server

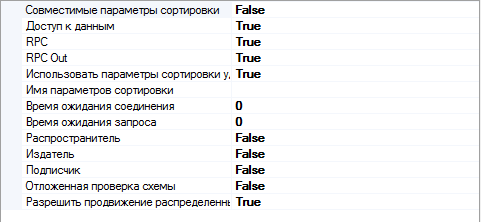
Рассмотрим создание связного сервера и произведем доступ к данным текущего экземпляра SQL Server.

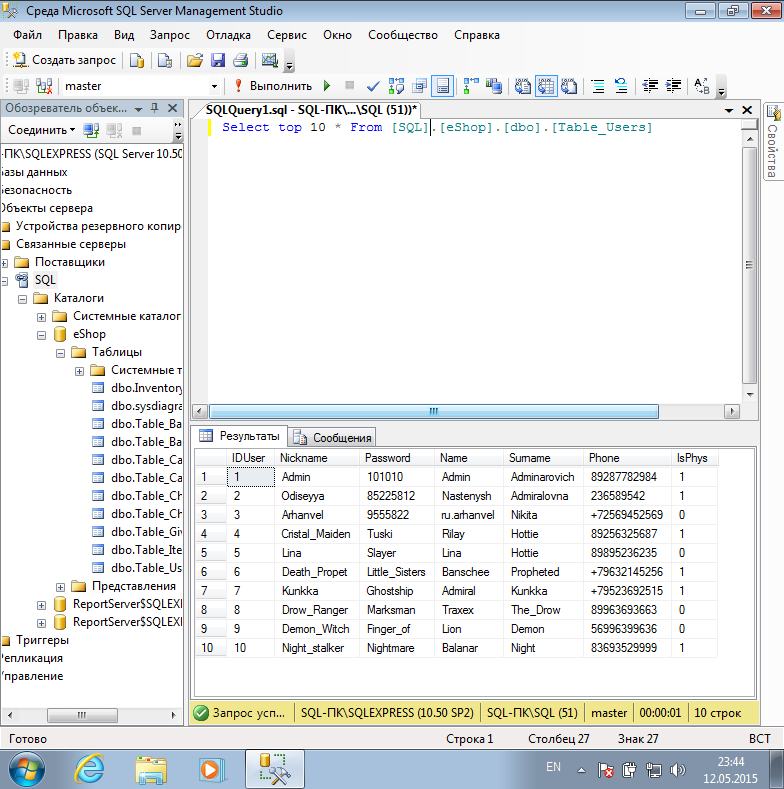
1. Создадим связанный сервер



1. В настройках указываем следующее



1. В качестве параметров сервера укажем следующее
2. Выполним тестовый запрос на выборку

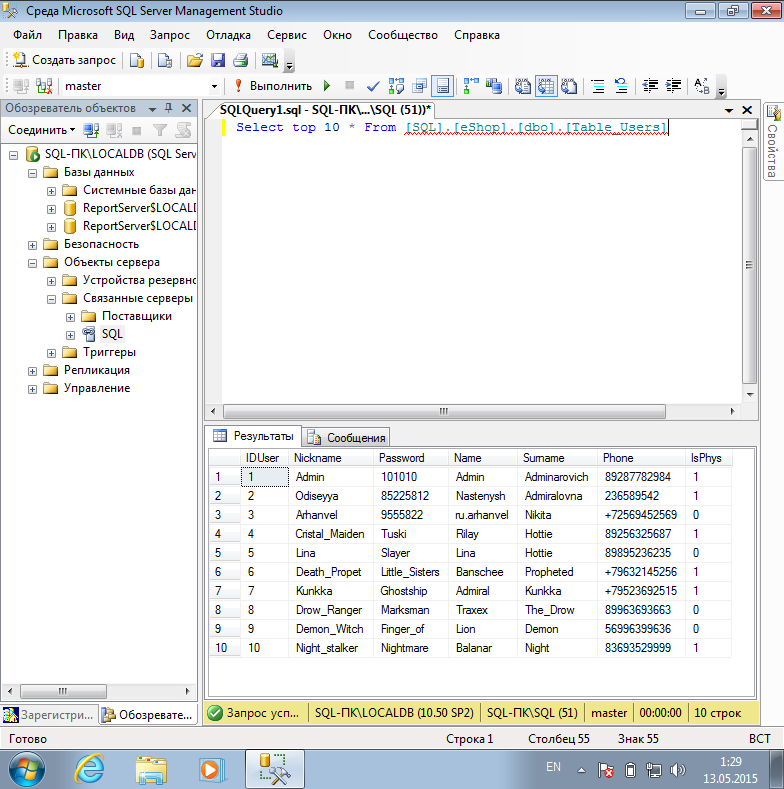


Запрос был выполнен успешно, значит, связанный сервер работает корректно.

# Доступ к данным из другого экземпляра SQL Server

Заметим, что связанный сервер не обязательно должен быть другим экземпляром SQL Server, хотя такой вариант часто встречается.

Создадим связанный сервер с прежними настройками, но на другом сервере (localdb). Также выполним тестовый запрос на выборку.



Запрос был выполнен успешно, значит, связанный сервер работает корректно.

# Подключение к XLSX файлу

На сей раз воспользуемся T-SQL для создания связанного сервера. В качестве провайдера выберем Microsoft.Jet.OLEDB.4.0, в качестве источника данных – книгу Excel (в нее запишем данные о клиентах).

**Команда:**

EXEC sp\_addlinkedserver

@server = 'ExcelServer1',

@srvproduct = 'Excel',

@provider = 'Microsoft.Jet.OLEDB.4.0',

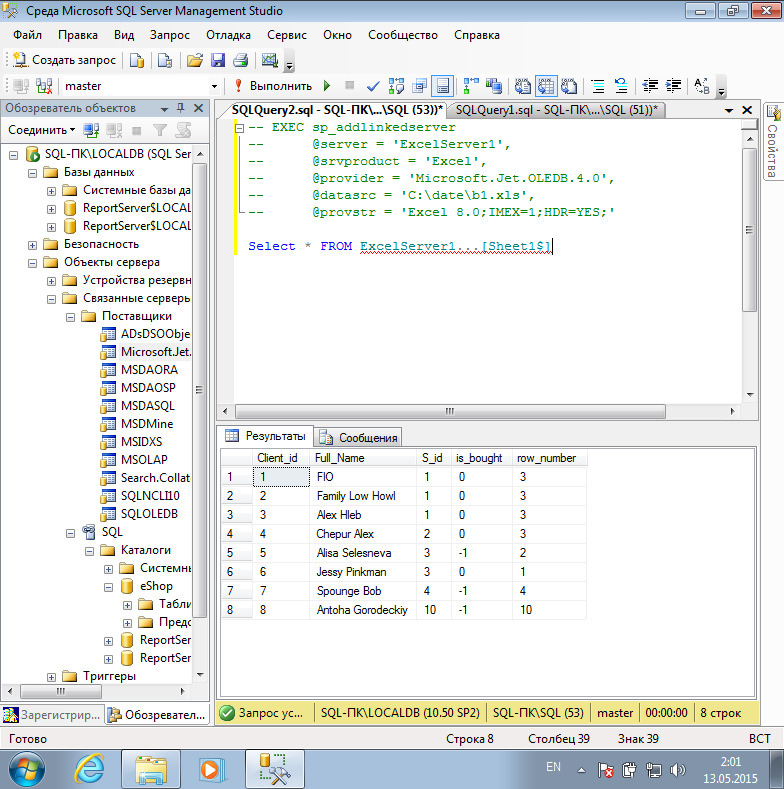
@datasrc = 'C:\date\b1.xls',

@provstr = 'Excel 8.0;IMEX=1;HDR=YES;'

**Затем произведем выборку:**

SELECT \* FROM ExcelServer1...[Sheet1$]

**Результат:**



# Использование функции OPENDATASOURCE

Снова произведем выборку данных из файла EXCEL, но теперь с использованием команды OPENDATASOURCE.

**Команда:**

sp\_configure 'show advanced options',1

reconfigure with override

go

sp\_configure 'Ad Hoc Distributed Queries',1

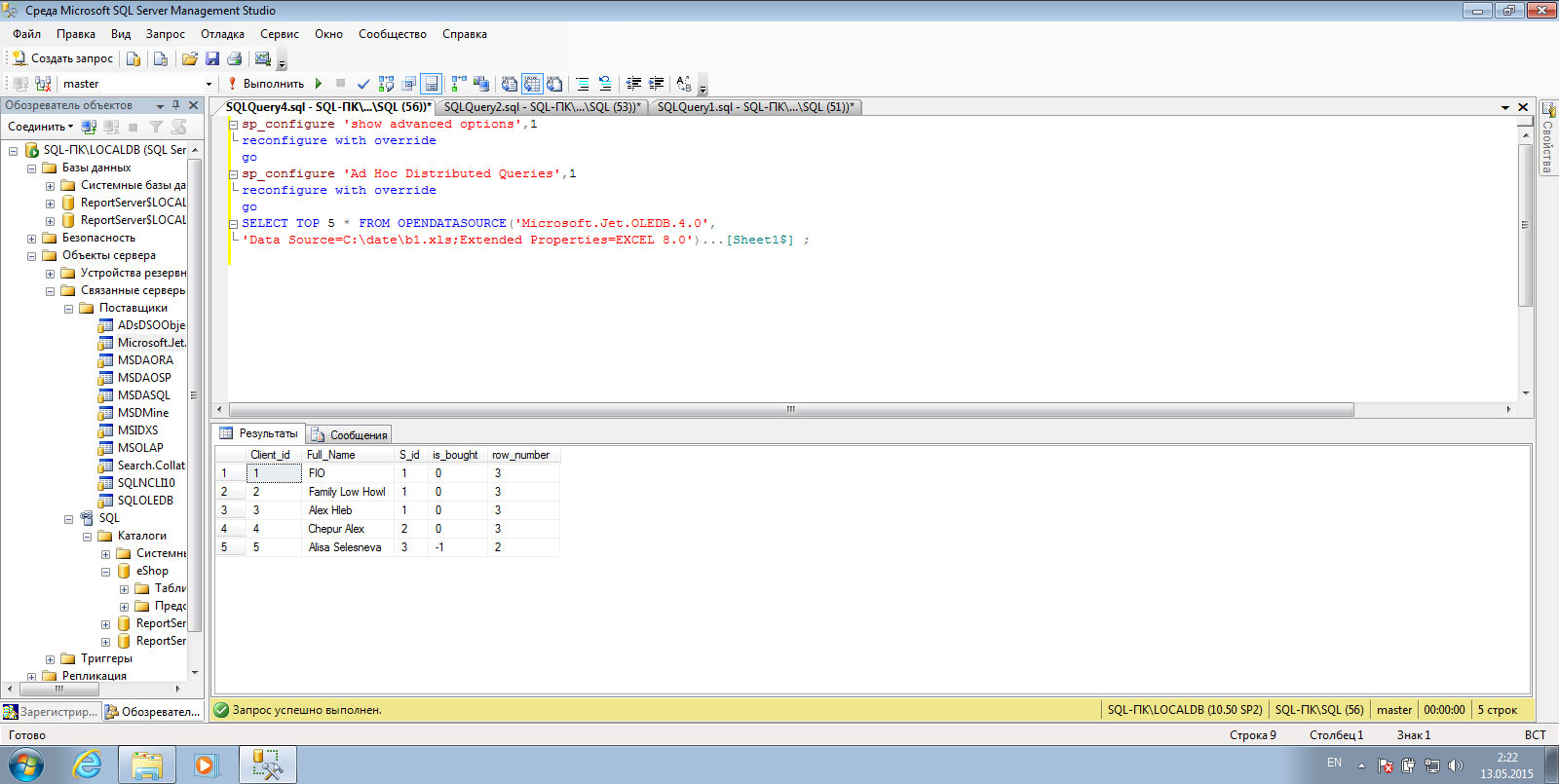
reconfigure with override

go

SELECT TOP 5 \* FROM OPENDATASOURCE('Microsoft.Jet.OLEDB.4.0',

'Data Source=C:\date\b1.xls;Extended Properties=EXCEL 8.0')...[Sheet1$] ;

**Результат:**



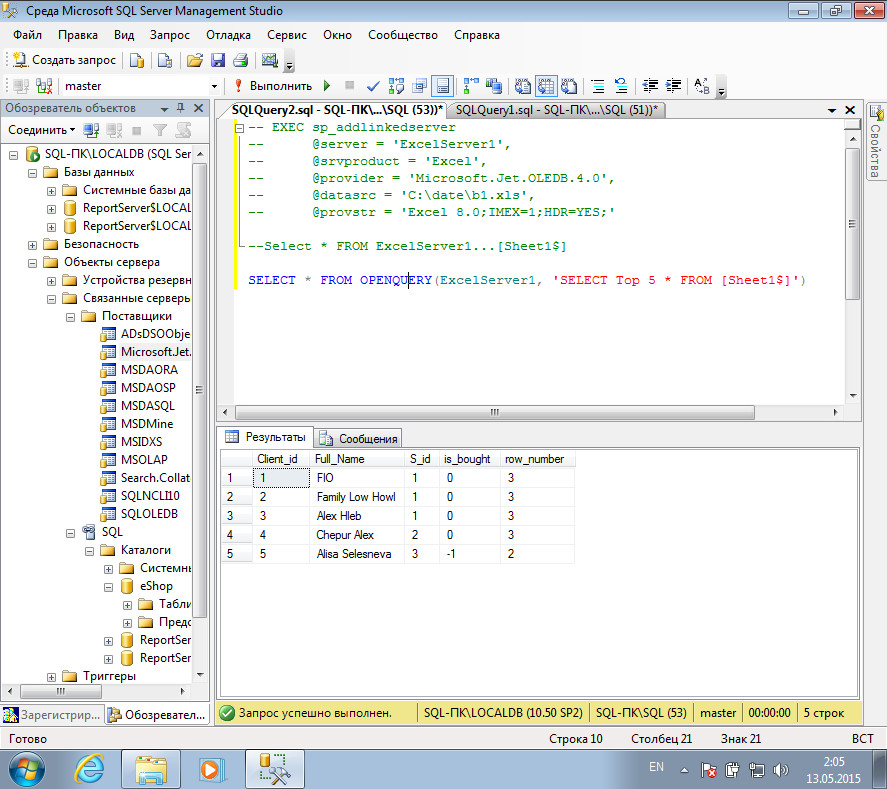
# Использование функции OPENQUERY

Напишем запрос к уже созданному нами связанному серверу на localdb.

**Команда:**

SELECT \* FROM OPENQUERY(ExcelServer1, 'SELECT Top 5 \* FROM [Sheet1$]')

**Результат:**



# Использование функции OPENROWSET

# Импорт данных с использованием не XML файла форматирования

**Команда:**

Use eShop;

GO

INSERT INTO Table\_Items

with (KEEPIDENTITY)

(IDItems,ItemName,ItemCategory,ItemPrice,Avaliable)

SELECT \*

FROM OPENROWSET(BULK 'C:\exportf.dat',

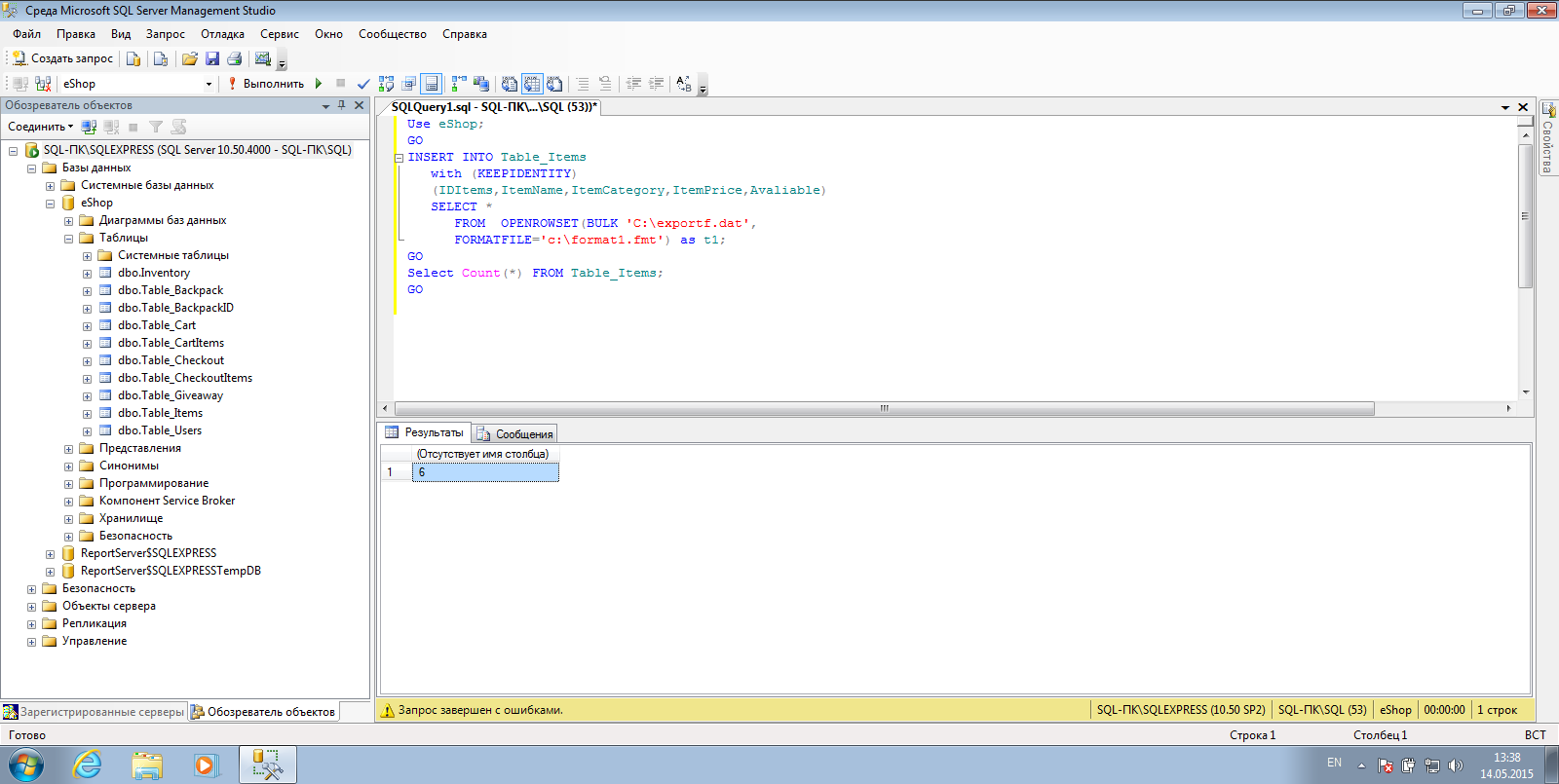
FORMATFILE='c:\format1.fmt') as t1;

GO

Select Count(\*) FROM Table\_Items;

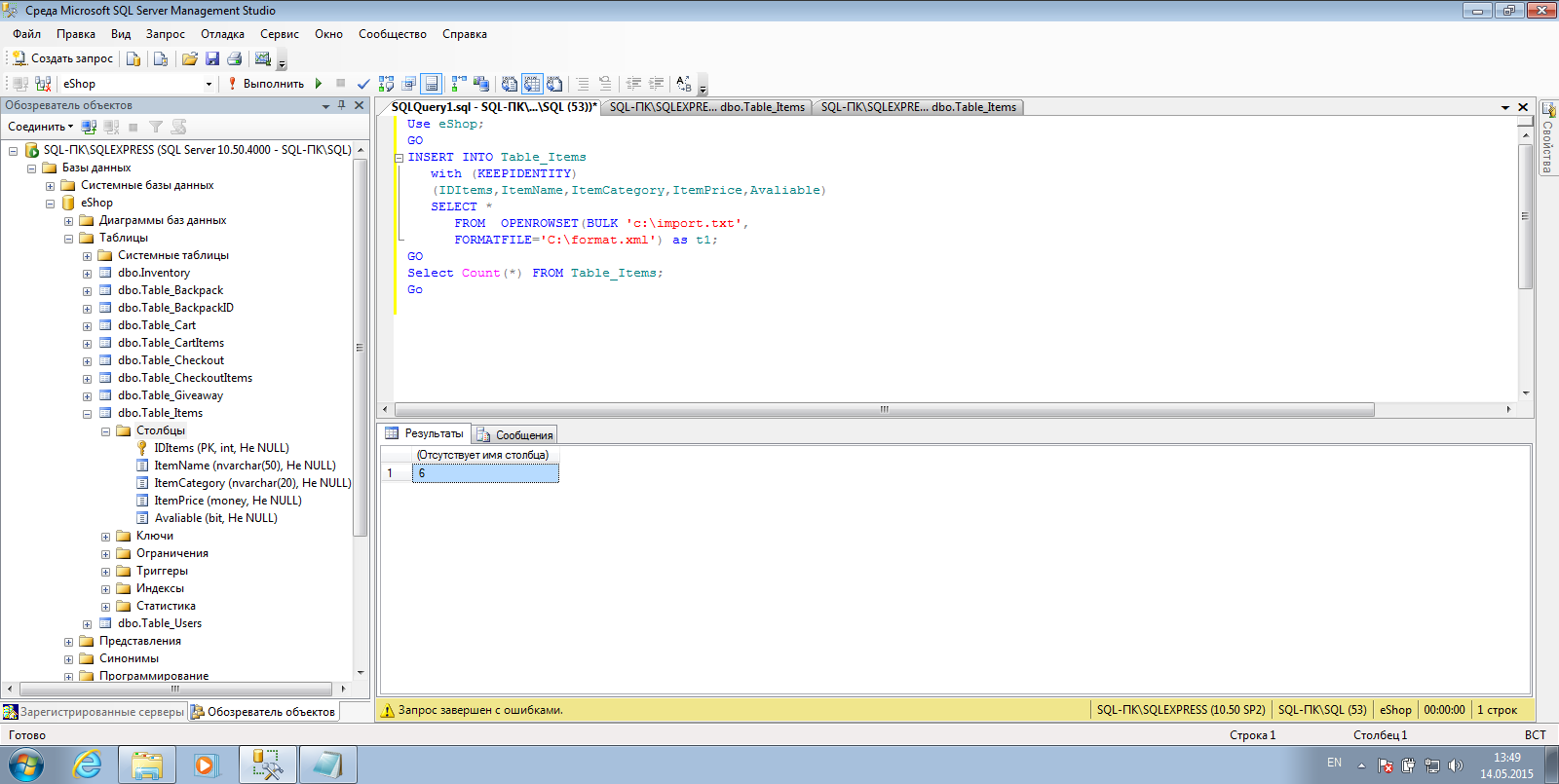
GO

**Результат:**



# Импорт данных с использованием XML файла форматирования

**Результат:**



# Запрос данных без импортирования в таблицу

Посмотрим, что содержит в себе файл Clients.dat. Для этого воспользуемся инструкцией SELECT … FROM OPENROWSET (BULK…)

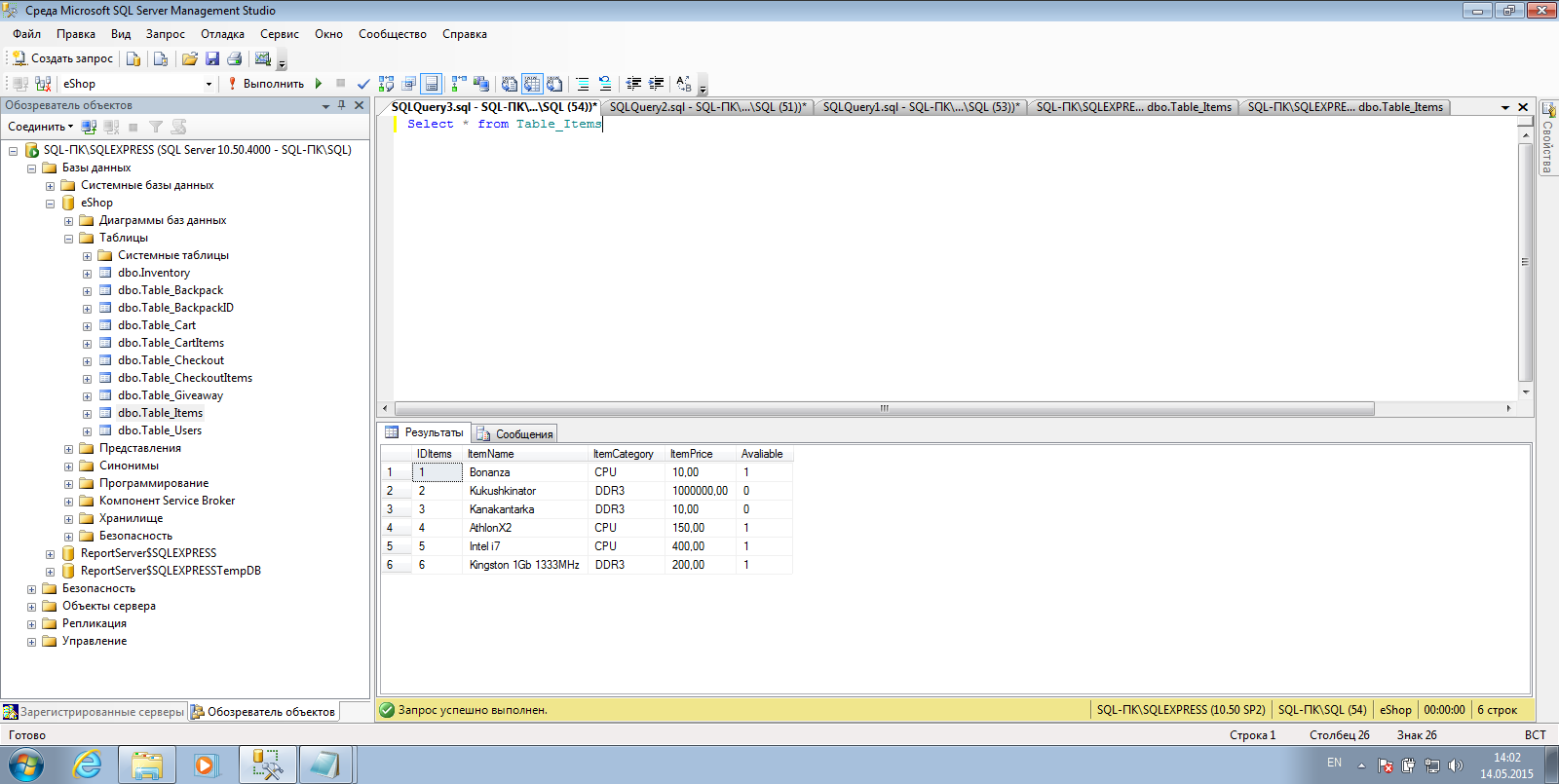
**Команда:**

SELECT t1.\*

FROM OPENROWSET(BULK 'C:\Import.txt',

FORMATFILE='C:\format.xml') as t1;

**Результат:**



# Использование функции OPENXML

Покажем работу с данной функцией на примере тестового xml-файла, содержащего список книг и информацию о каждой из них (books.xml):

<books>

<book author="John Smith"

title="Book #1"

pubyear="2004" />

<book author="Mike Doe"

title="Book #2"

pubyear="2010" />

<book author="Ivan Petrov"

title="Book #3"

pubyear="2014" />

</books>

Затем функцией OPENROWSET произведем выборку:

DECLARE

@iDoc INT,

@Doc XML;

SET @Doc = (

SELECT \* FROM

OPENROWSET (

BULK 'C:\books.xml',

SINGLE\_BLOB

) AS X );

EXEC sp\_xml\_preparedocument @iDoc OUTPUT, @Doc;

SELECT \*

FROM OPENXML (@iDoc, '/books/book', 2)

WITH (

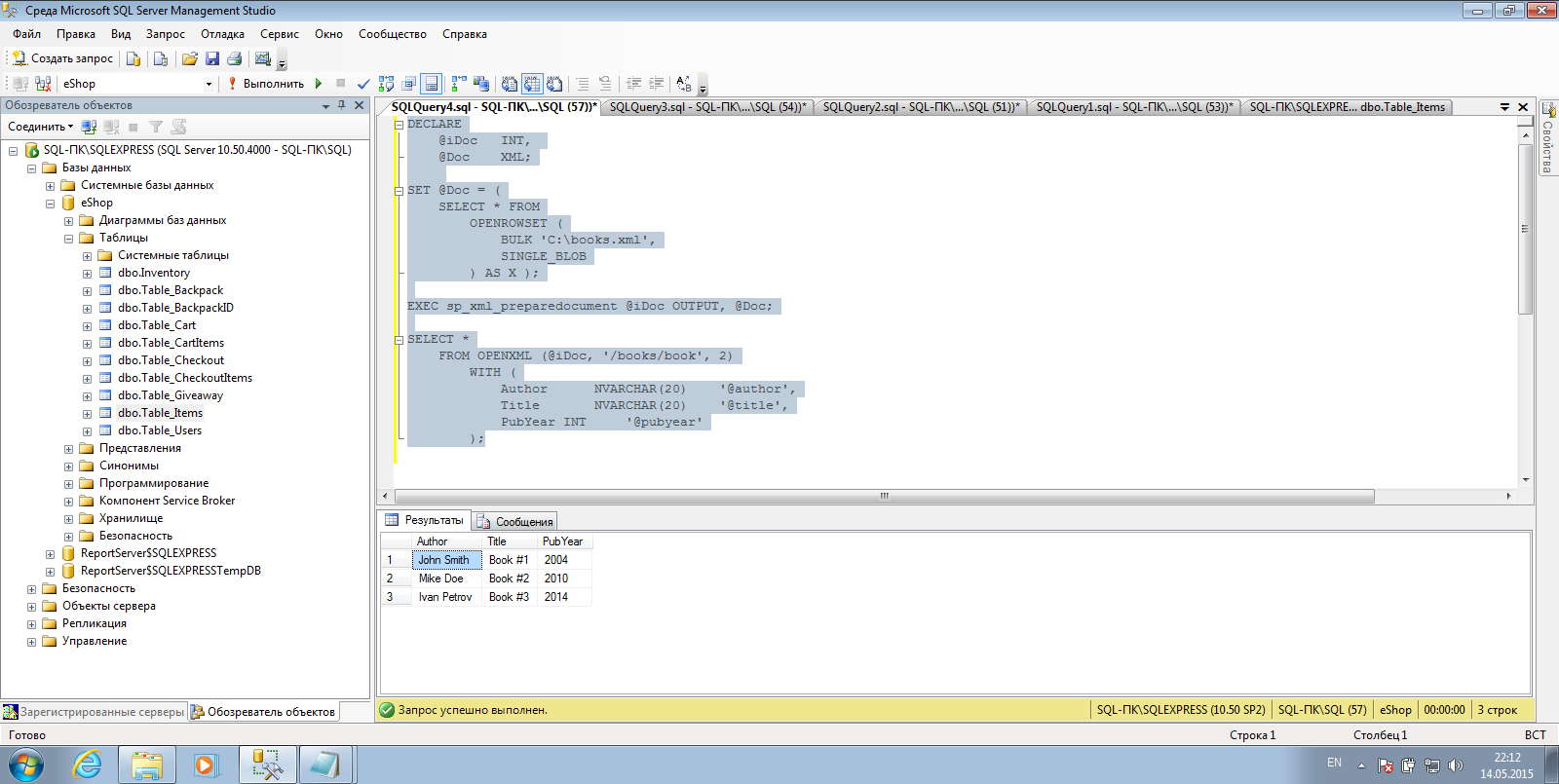
Author NVARCHAR(20) '@author',

Title NVARCHAR(20) '@title',

PubYear INT '@pubyear'

);

**Результат:**



# Организация Paging’а на стороне сервера

Организация постраничной навигации – неотъемлемая часть любой информационной системы, ведь отобразить все в большинстве случаев бывает невозможно. Механизм организации прост и основывается на расчете позиций каждой строки в выборке. Создадим процедуру, принимающую два параметра:

* @Offset–смещение от начала
* @Limit – количество выбираемых строк

**Процедура:**

CREATE PROCEDURE dbo.getUserList

@Offset INT,

@Limit INT

AS

SET NOCOUNT ON;

WITH UserCTE AS

(

SELECT \*, ROW\_NUMBER() OVER(ORDER BY IDUser) AS Num

FROM eShop.dbo.Table\_Users

)

SELECT \* FROM

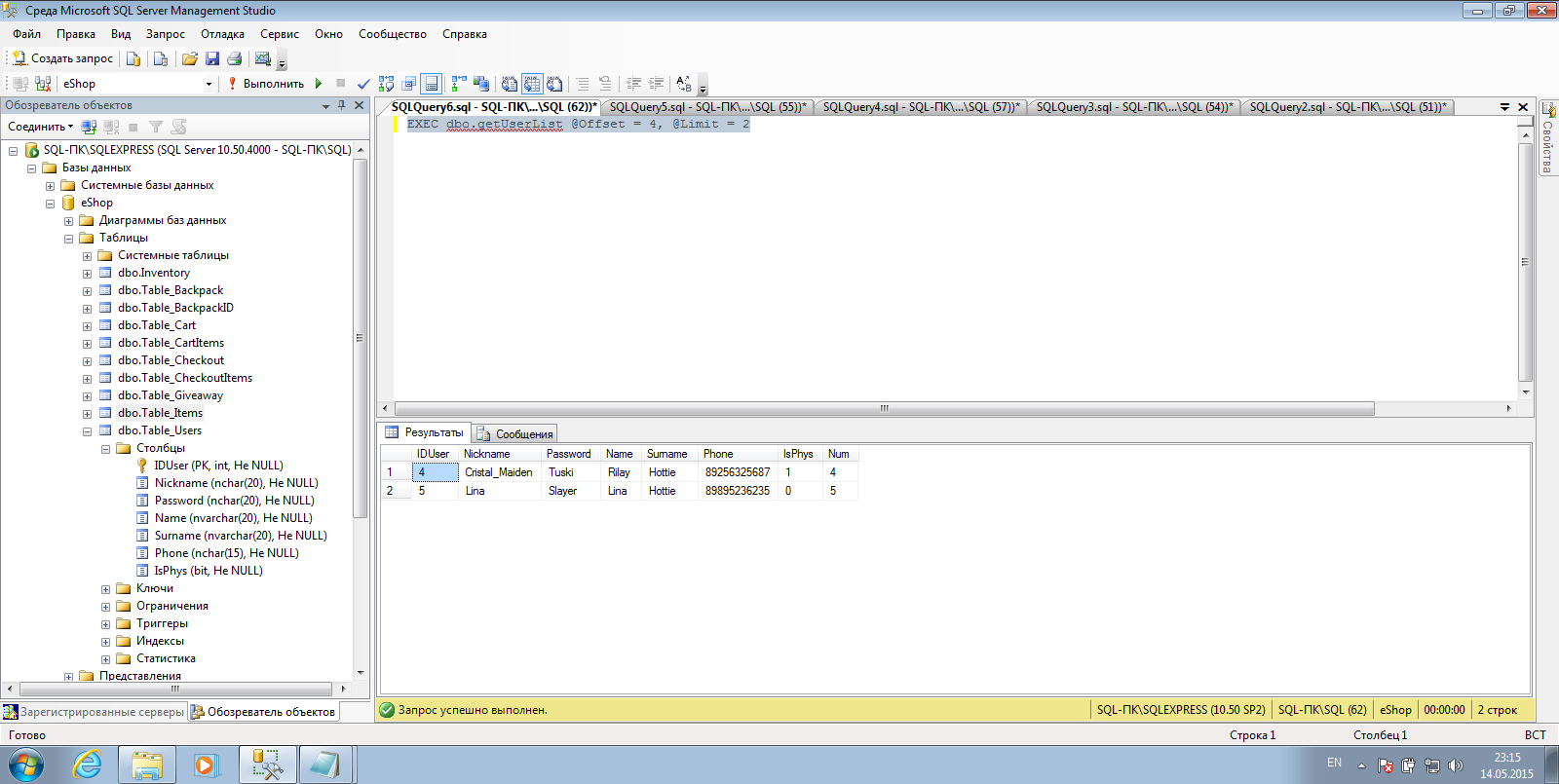
UserCTE

WHERE Num BETWEEN @Offset AND @Offset + @Limit-1;

GO

**Выполним ее:** EXEC dbo.getUserList @Offset = 4, @Limit = 2

**Результат:**



# Массовый импорт и экспорт XML данных

Для того, что экспортировать данные из БД в XML-формат, можно использовать следующую конструкцию:

SELECT top 10

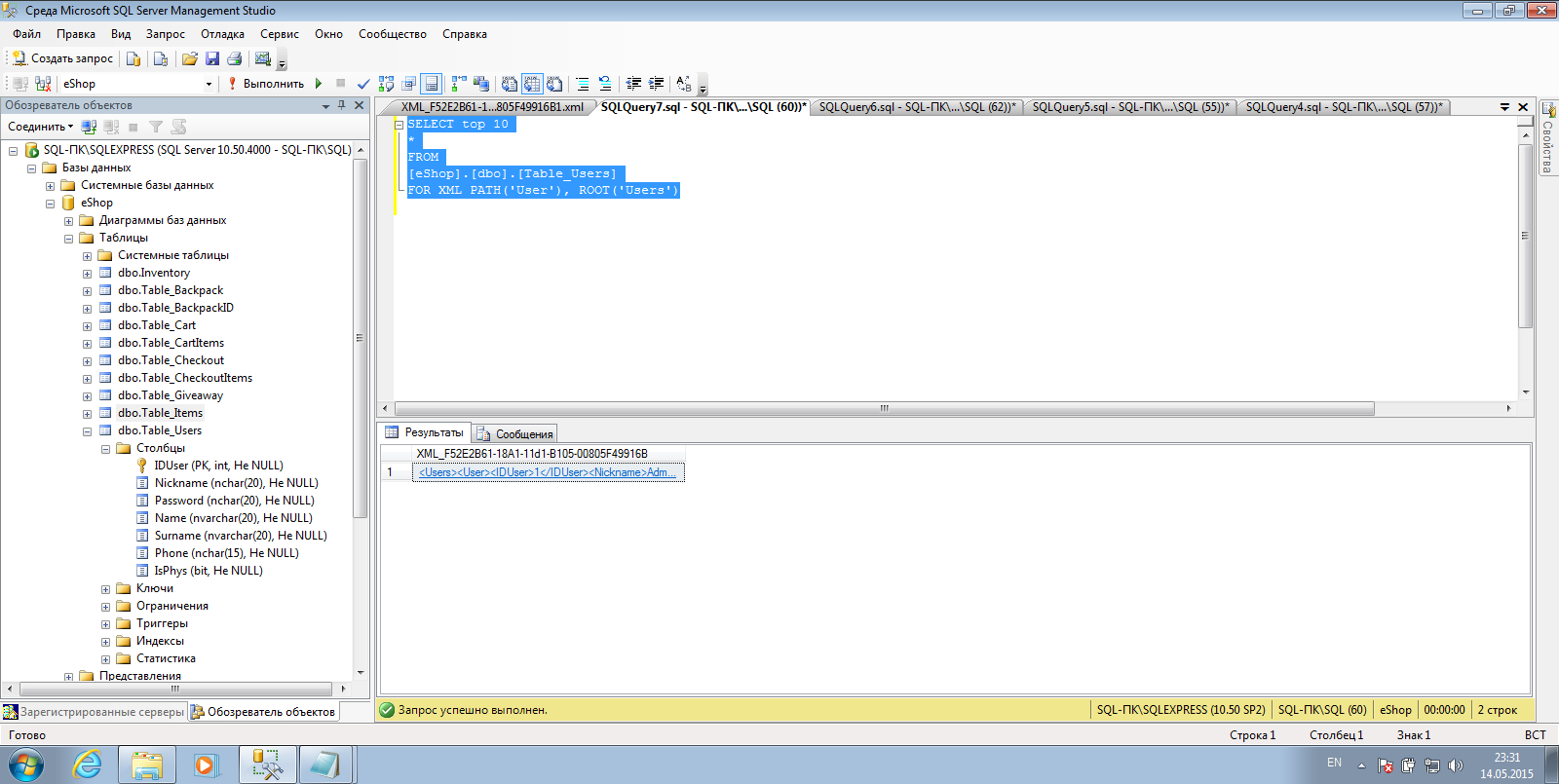
\*

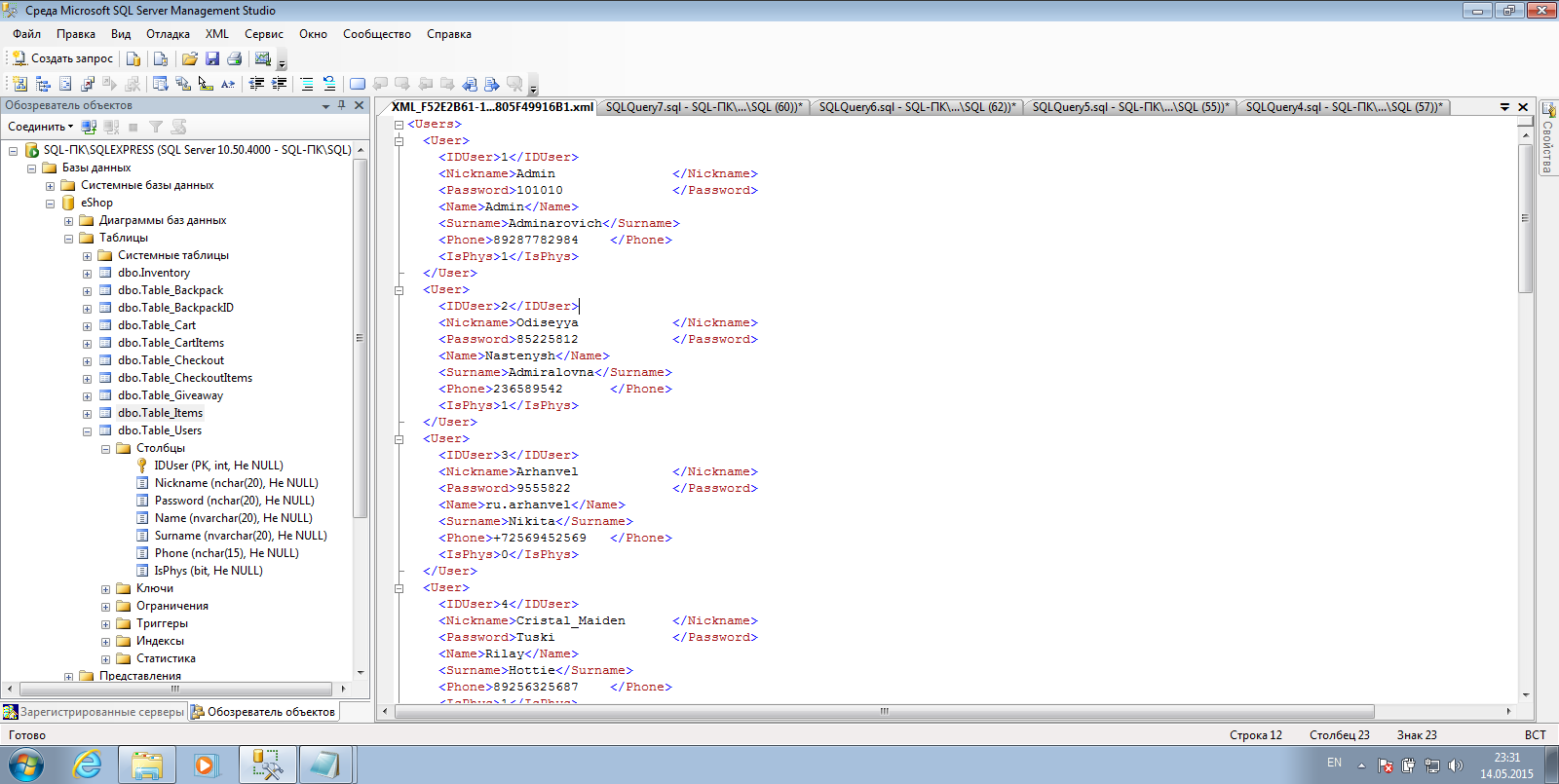
FROM

[eShop].[dbo].[Table\_Users]

FOR XML PATH('User'), ROOT('Users')

В результате мы получаем скалярное значение в строки, которая содержит данные в XML-формате:





Импорт данных из XML-Формата производится с помощью функции OPENXML. Пример:

DECLARE @idoc int

DECLARE @doc varchar(1500)

SET @doc='

<Users>

<User>

<UserId>1</UserId>

<FirstName>Kazak</FirstName>

</User>

<User>

<UserId>2</UserId>

<FirstName>Chepur</FirstName>

</User>

<User>

<UserId>3</UserId>

<FirstName>Denis</FirstName>

</User>

</Users>

';

--Create an internal representation of the XML document.

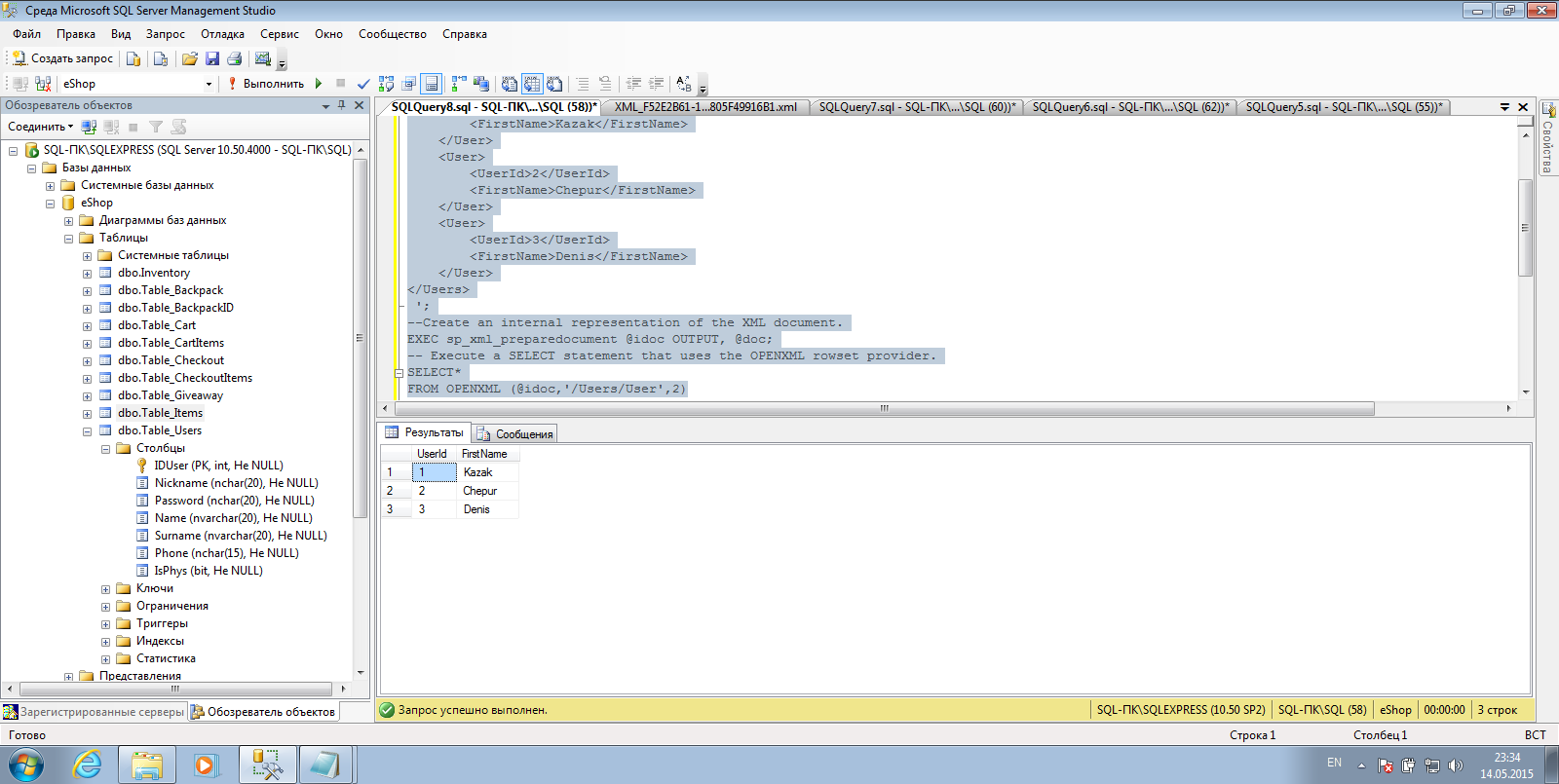
EXEC sp\_xml\_preparedocument @idoc OUTPUT, @doc;

-- Execute a SELECT statement that uses the OPENXML rowset provider.

SELECT\*

FROM OPENXML (@idoc,'/Users/User',2)

**Результат:**



# Полнотекстовый и нечеткий поиск

# Тестовый пример

Для активации функциональных возможностей FTS необходимо создать:

* полнотекстовый каталог,
* полнотекстовый индекс,
* полнотекстовый запрос.

Создадим тестовую таблицу **Stories**, содержащую краткие рассказы классиков о лете.

Структура таблицы следующая:

|  |  |
| --- | --- |
| **Поле** | **Тип данных** |
| Id | INT |
| Title | NVARCHAR(100) |
| Text | TEXT |

**Создадим полнотекстовый каталог TestFT командой:**

CREATE FULLTEXT CATALOG TestFT AS DEFAULT;

**Теперь создадим полнотекстовый индекс на Text, предварительно создав индекс на поле Id:**

CREATE UNIQUE INDEX StoriesId

ON Stories(Id)

GO

CREATE FULLTEXT INDEX

ON Stories

(

Text LANGUAGE 1049

)

KEYINDEX StoriesId

ON TestFT

WITH

(

CHANGE\_TRACKING = AUTO,

STOPLIST = SYSTEM

)

Все, индексы созданы, теперь можно писать запросы с использованием таких предикатов, как CONTAINS и FREETEXT

**CONTENTS**

Напишем запрос, который ищет рассказы, содержащие слово **«лето»**:

SELECT \*

FROM Stories

WHERE CONTAINS(Text, 'лето');

Несмотря на то, что тематика рассказов – лето, предикат CONTAINS не обнаружил точных совпадений:



Как видно из результатов, предикат CONTAINS рассчитан на поиск строк с точным совпадение формы слова.

**FREETEXT**

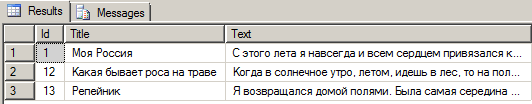
Повторим первый запрос с предикатом FREETEXT:

SELECT \*

FROM Stories

WHERE FREETEXT(Text, 'лето');

Результат:



Данный предикат нашел строки с подходящим по **смыслу** текстом.

**Нечеткий поиск.**

**CONTAINSTABLE**

Запросим рассказы, в которых содержится словосочетание «темный лес»:

SELECT \*

FROM Stories AS FT\_TBL

INNER JOIN CONTAINS TABLE

(

Stories,

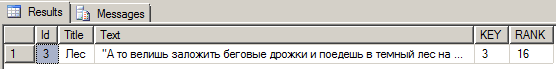
Text,

'темныйNEARлес'

) AS KEY\_TBL

ON FT\_TBL.Id = KEY\_TBL.[KEY]

В данном случае мы использовали ключевое слово NEAR, которое показывает, что рядом со словом «темный» должно быть слово «лес». Результат:



Была найдена одна строка с точным совпадением.

**FREETEXTTABLE**

Повторим предыдущий запрос, используя данный предикат:

SELECT \*

FROM Stories AS FT\_TBL

INNER JOIN FREETEXTTABLE

(

Stories,

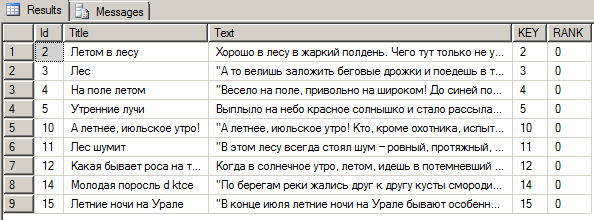
Text,

'темный NEAR лес'

) AS KEY\_TBL

ON FT\_TBL.Id = KEY\_TBL.[KEY]

**Результат:**



Строк в результате заметно прибавилось, что говорит о том, что семантический анализ строк дает более развернутые результаты, нежели поиск по точному совпадению.

# Пример полнотекстового поиска на нашей базе

Вначале создадим полнотекстовый каталог и полнотекстовый индекс на таблице пользователей (Table\_Users).

USE eShop;

CREATE FULLTEXT CATALOG eShopFTCTG AS DEFAULT;

CREATE UNIQUE INDEX UseID

ON Table\_Users(IDUser)

GO

CREATE FULLTEXT INDEX

ON Table\_Users

(

Nickname LANGUAGE 1033

)

KEY INDEX UseId

ON eShopFTCTG

WITH

(

CHANGE\_TRACKING = AUTO,

STOPLIST = SYSTEM

)

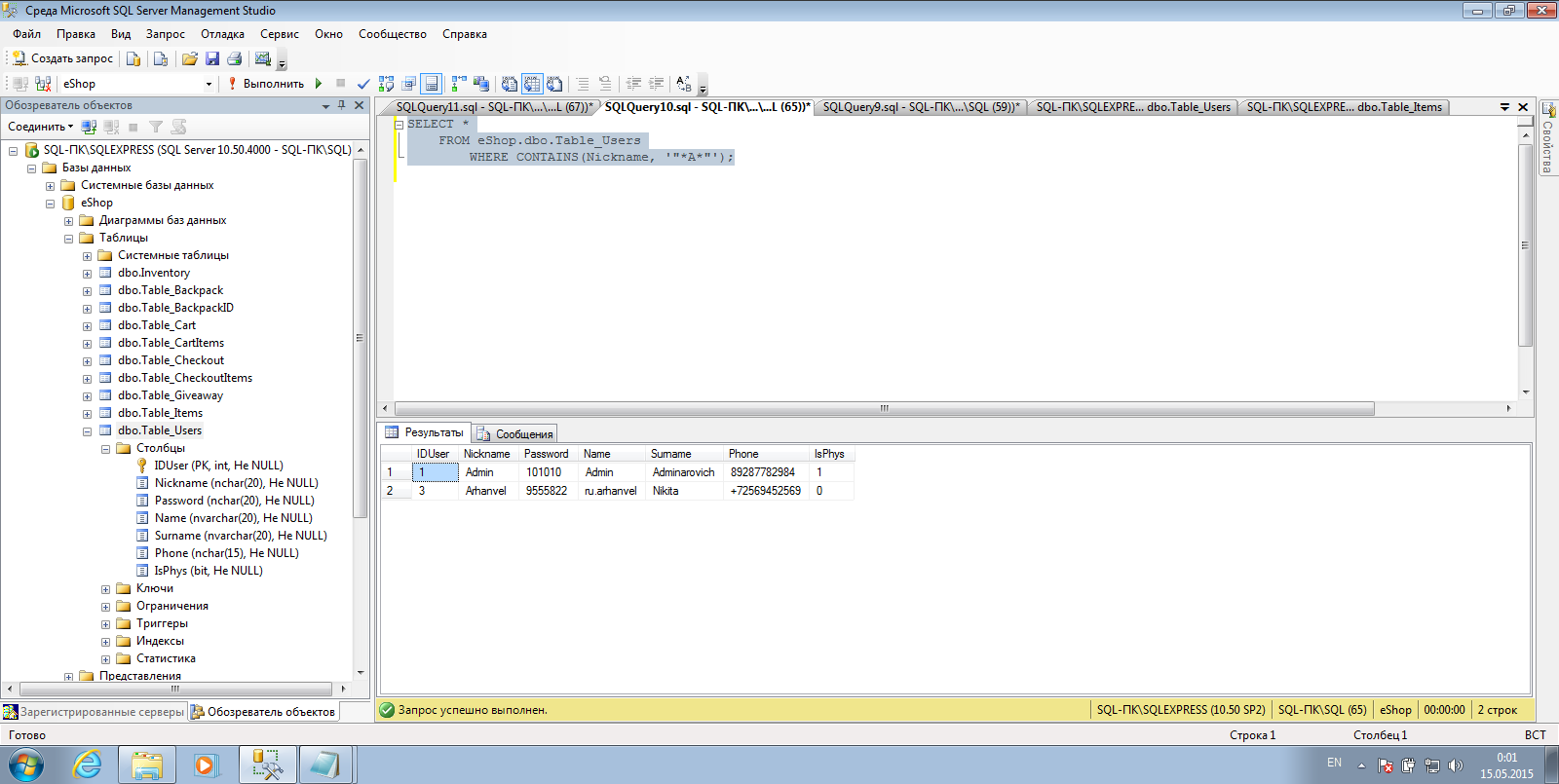
Напишем запрос, который ищет пользователей, начинающихся с **«A»**. Для этого воспользуемся регулярным выражением, заключив строку поиска в кавычки.

SELECT \*

FROM eShop.dbo.Table\_Users

WHERE CONTAINS(Nickname, '"\*A\*"');

**Результат:**



# Вывод

В данной лабораторной работе мы освоили следующие навыки:

1. Подключение к внешним источникам данных. Создание LinkedServer,
2. Использование функции OPENROWSET, OPENROWSET(BULK…) для вставки из файла
3. Использование функции OPENDATASOURCE,
4. Использование функции OPENQUERY
5. Использование функции OpenXML.
6. Массовый импорт и экспорт XML-данных. Разбор XML данных,
7. Организация Paging на стороне SQL server, используя RowNumber / RANK
8. Построение и использование полнотекстового поиска,
9. Нечеткий поиск. Функция CONTAINSTABLE, FULLTEXTTABLE