**Практическая работа 16 «Пользовательские функции»**

Цель практической работы:

1. Изучить порядок создания пользовательских функций.
2. Освоить применение пользовательских функций.

Практические задания и рекомендации по их выполнению:

ЗАДАНИЕ

Пользовательские функции очень похожи на хранимые процедуры. Так же в них можно передавать параметры и они выполняют некоторые действия, однако их главным *отличием* от хранимых процедур является то, что они выводят (возвращают) какой то результат. Более того, они вызываются только при помощи оператора SELECT, аналогично встроенным функциям. Все пользовательские функции делятся на 2 вида:

1. Скалярные функции — функции, которые возвращают число или текст, то есть одно или несколько значений;
2. Табличные функции — функции, которые выводят результат виде таблицы.

Теперь рассмотрим создание и применение пользовательских функций. В Microsoft SQL Server все пользовательские функции находятся в папке “Functions” («Функции»), расположенной в папке “Programmability” («ПРограммирование») в обозревателе объектов (рис. 69).

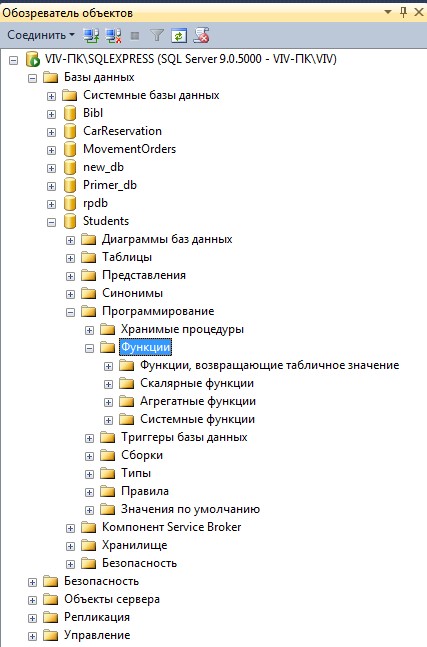
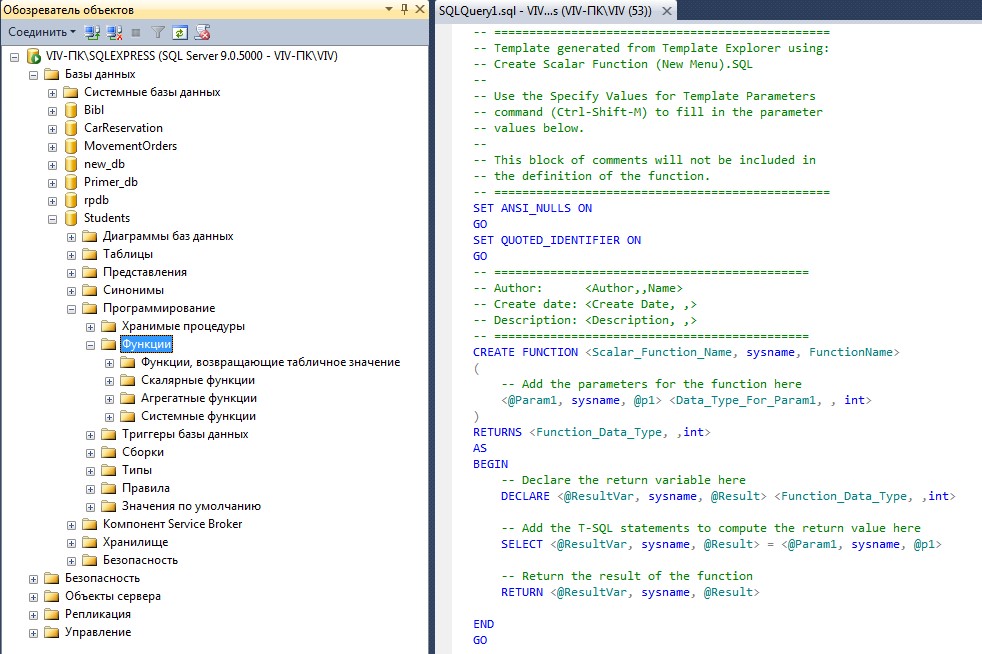


Рисунок 69 — Отображение функций в Обозревателе объектов

Начнём с создания скалярных пользовательских функций. Для создания новой скалярной пользовательской функции в обозревателе объектов в БД “Students” в папке “Programmability” («Программирование»), щёлкните правой кнопкой мыши по папке “Functions” («Функции») и в появившемся меню выберите пункт “New / Scalar-valued Function” («Создать / Скалярная функция»). Появится окно новой скалярной пользовательской функции (рис.

70).

1

2

3

4

5

6

Рисунок 70 — Окно новой скалярной пользовательской функции

Синтаксис скалярной пользовательской функции похож на синтаксис хранимой процедуры (см. лабораторную работу 15). Однако имеется ряд существенных отличий (рис. 70):

1. Область определения имени функции (Scalar\_Function\_Name);
2. Параметры, передаваемые в процедуру (@Param1). Определение параметров аналогично определению параметров в хранимой процедуре (см. лабораторную работу 15);
3. Тип данных значения, возвращаемого процедурой;
4. Область объявления переменных, используемых внутри функции.

Объявление переменных имеет следующий синтаксис:

DECLARE @<Имя переменной> <Тип данных>

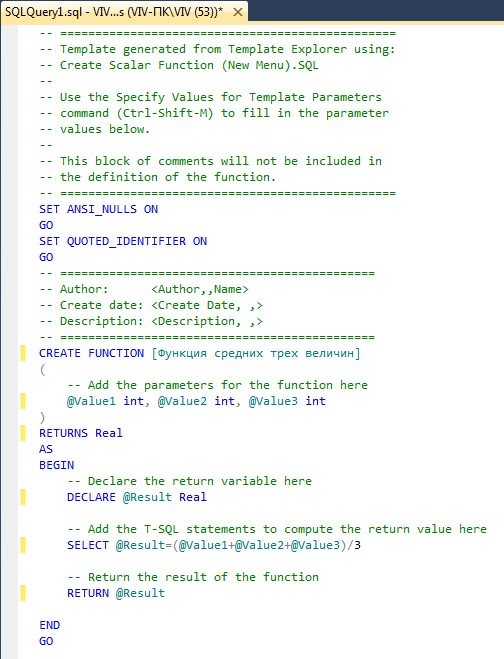
1. Тело самой пользовательской функции, содержит команды языка программирования запросов SQL;
2. Команда RETURN возвращающая результат выполнения функции.

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 трёх величин. В окне новой пользовательской функции наберите код, представленный на рисунке 71.

1

2

3

4

5

6

Рисунок 71 — Текст пользовательской функции, вычисляющей среднее значение трех величин

Рассмотрим более подробно код данной скалярной пользовательской функции (рис. 71):

1. CREATE FUNCTION [Функция средних трёх величин] — определяет имя создаваемой функции как «Функция средних трёх величин»;
2. @Value1 int, @Value2 int, @Value3 int — 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, т.е. дробного числа (тип данных Real);
5. SELECT @Result = (@Valuel + @Value2 + @Value3) /3 — вычисляет среднее и помешает результат в переменную @Result;
6. RETURN @Result — возвращает значение переменной @Result. Остальные фрагменты кода рассмотрены выше (рис. 70).

Для создания функции выполним вышеописанный код, нажав кнопку

) на панели инструментов. В нижней части окна с кодом



(



появиться сообщение “Conimand(s) completed successfully.” («Выполнение команд успешно завершено.»). Закройте окно с кодом, щёлкнув мышью по кнопке закрытия , расположенной в верхнем правом углу окна с кодом функции.

Появится окно сохранения функции. Сохраним функцию под именем «ФункцияСреднихТрехВеличин» в своей папке. Одновременно функция отобразится в окне Обозревателя объектов в папке “Programmability” («Программирование») / “Functions” («Функции») / “Scalar-valued Functions” («Скалярные функции») под именем dbo.[Функция средних трех величин].

Проверим работу созданной скалярной пользовательской функции. Для запуска пользовательской функции необходимо создать новый пустой запрос, нажав на кнопку ( ) на панели инструментов. В появившемся окне с пустым запросом наберите команду:



SELECT dbo.[Функция средних трёх величин] (3, 5, 4)

и нажмите кнопку ) на панели инструментов (рис. 72).



(



В нижней части окна с кодом появится результат выполнения новой скалярной пользовательской функции: 4 (рис. 72).

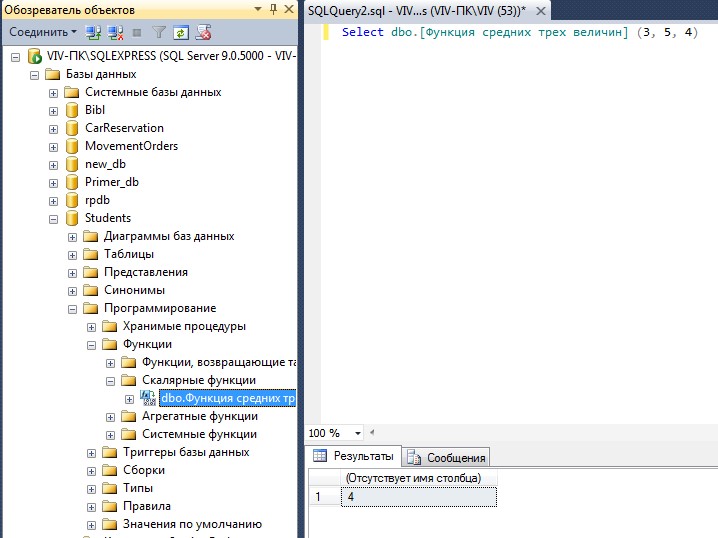


Рисунок 72 — Результат выполнения функции «Функция средних трех величин»

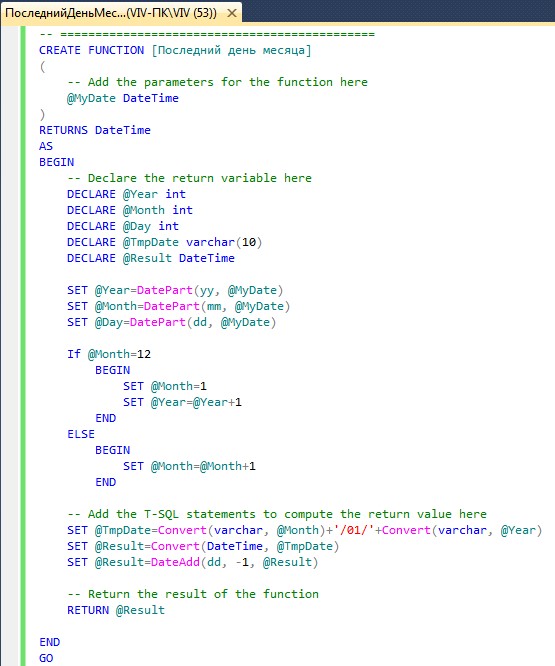
Теперь создадим более сложную скалярную пользовательскую функцию, предназначенную для определения последнего дня месяца введённой даты.

Создайте новую скалярную пользовательскую функцию так, как об этом сказано выше. В окне новой пользовательской функции наберите следующий код (рис. 73).

Перейдём к рассмотрению кода (рис. 73). 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 введённой 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)).

1

2

3

4

5

6

7

8

9

Рисунок 73 — Текст пользовательской функции «Последний день месяца»

DECLARE @Result DateTime — объявляет переменную @Result для хранения результата — даты последнего дня месяца (тип данных DateTime).

1. SET @Year = DatePart(yy, @MyDate), SET @Month = DatePart(mm, @MyDate), SET @Day = DatePart(dd, @MyDate) — определяются части введённой даты и помещаются в переменные @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 (уу — год, mm — месяц, dd — день), «дата» — это дата, части которой определяем.
2. IF @Month=12

BEGIN

SET @Month=l

SET @Year=@Year+l

END

ELSE

BEGIN

SET @Month=@Month+l

END

Данный 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.

1. 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 её в переменную @TmpDate, затем переводит дату в строковом формате в тип данных даты и времени и помешает её 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».

1. 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 введённой дате (параметр «дата»).

1. 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.” («Выполнение команд успешно завершено.») закройте окно с кодом.



(



Появится окно сохранения функции. Сохраним функцию под именем «ПоследнийДеньМесяца» в своей папке. Одновременно функция отобразится в окне Обозревателя объектов в папке “Programmability” («Программирование») / “Functions” («Функции») / “Scalar-valued Functions” («Скалярные функции») под именем dbo.[Последний день месяца].

Проверим работу функции «Последний день месяца», выполнив её. Создайте новый пустой запрос, затем в окне с пустым запросом наберите команду SELECT dbo.[Последний день месяца] ('12/02/14') и нажмите кнопку



(



) на панели инструментов (рис. 74).

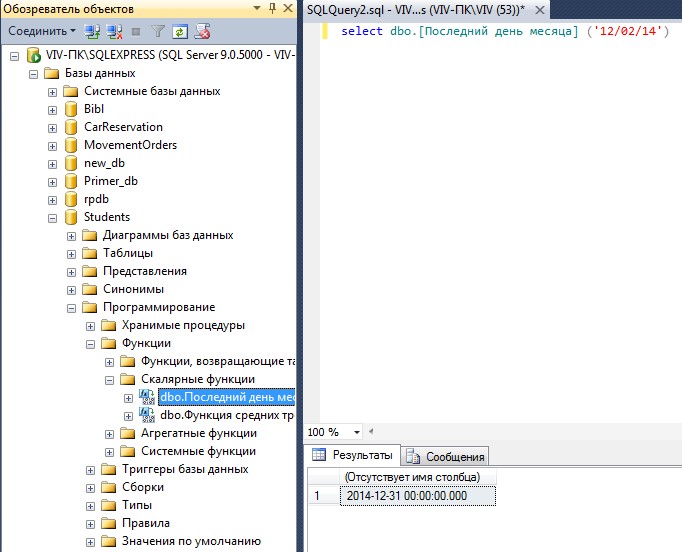
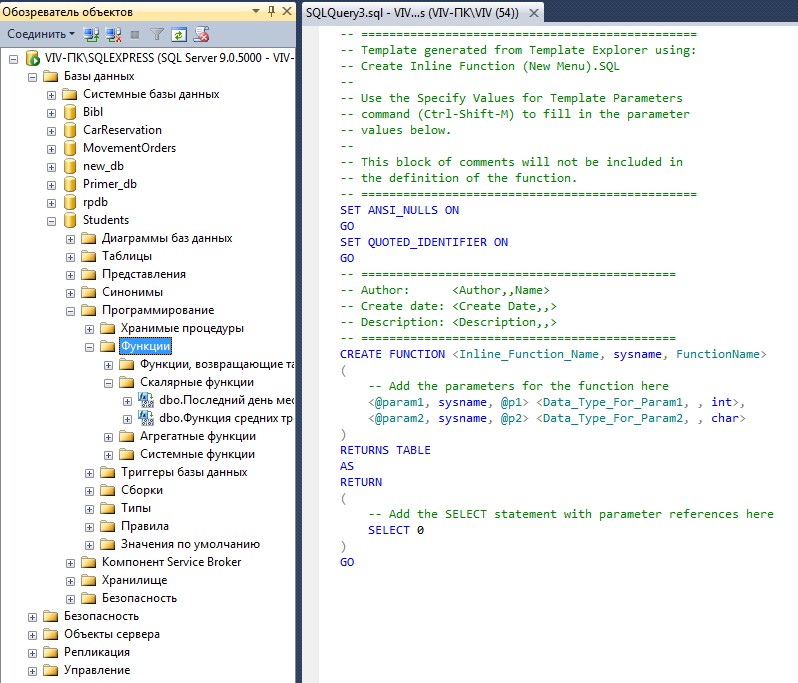


Рисунок 74 – Результат выполнения пользовательской функции

«Последний день месяца»

Появится результат выполнения новой скалярной пользовательской функции: 2014-12-31 (рис. 74).

Теперь перейдём к созданию табличных пользовательских функций. Для создания табличной пользовательской функции в обозревателе объектов в БД Students в папке “Programmability” («Программирование») щёлкните правой кнопкой мыши по папке “Functions” («Функции») и в появившемся меню выберите пункт “New Table-valued Function” («Создать / Встроенная функция, возвращающая табличное значение»). Появится окно новой табличной пользовательской функции (рис. 75)

1

2

3

4

Рисунок 75 — Окно новой табличной пользовательской функции

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, @Рагаш2);
3. RETURNS TABLE — показывает, что функция является табличной, т.е. возвращает таблицу;
4. Тело самой пользовательской функции состоит из команды SELECT языка программирования запросов SQL.

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

В заключении 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. 75) наберите следующий код (рис. 76):

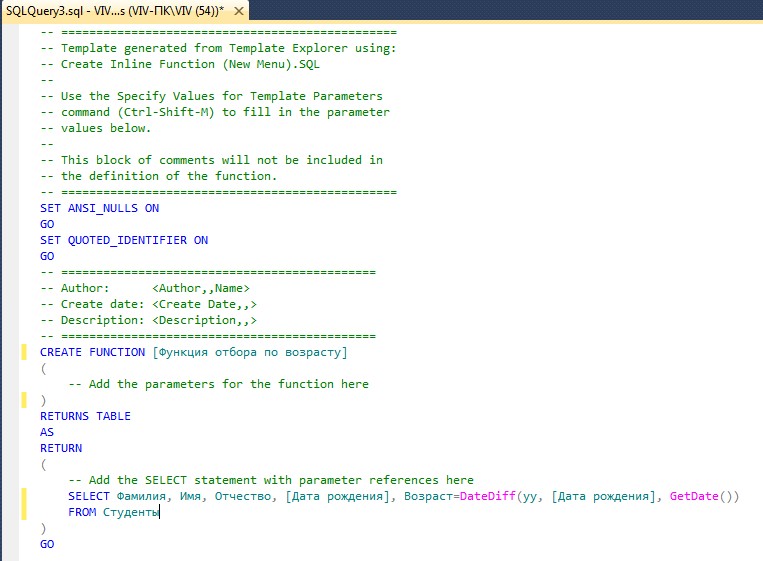


Рисунок 76 — Текст пользовательской функции «Функция отбора по возрасту»

Из кода, представленного на рисунке 76, видно, что данная табличная функция не имеет параметров и реализуется командой:

SELECT Фамилия, Имя, Отчество, [Дата рождения], Возраст = DateDiff(yy, [Дата рождения], GetDate()) FROM Студенты.

Из вышепредставленной команды видно, что из таблицы «Студенты» отображаются поля «Фамилия», «Имя», «Отчество» и «Дата рождения», а также вычислимое поле «Возраст». Поле «Возраст» вычисляется при помощи встроенной функции DateDiff, вычисляющей разницу между датами в определённых 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) (уу — год, mm — месяц, dd — день), «начальная дата» — дата начала периода и «конечная дата» — дата конца периода. В нашем случае в качестве начальной даты берём дату рождения студента, а в качестве конечной даты берём текущую дату (функция

GetDate()).

Для создания функции, выполним вышеописанный код, как и в случае с предыдущей функцией. После появления сообщения “Command(s) completed successfully.” («Выполнение команд успешно завершено.») закройте окно с кодом.

Появится окно сохранения функции. Сохраним функцию под именем «ФункцияОтбораПоВозрасту» в своей папке. Одновременно функция отобразится в окне Обозревателя объектов в папке “Programmability” («Программирование») / “Functions” («Функции») / “Table-valued Functions” («Функции, возвращающие табличное значение») под именем dbo.[Функция отбора по возрасту].

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. 77).



(



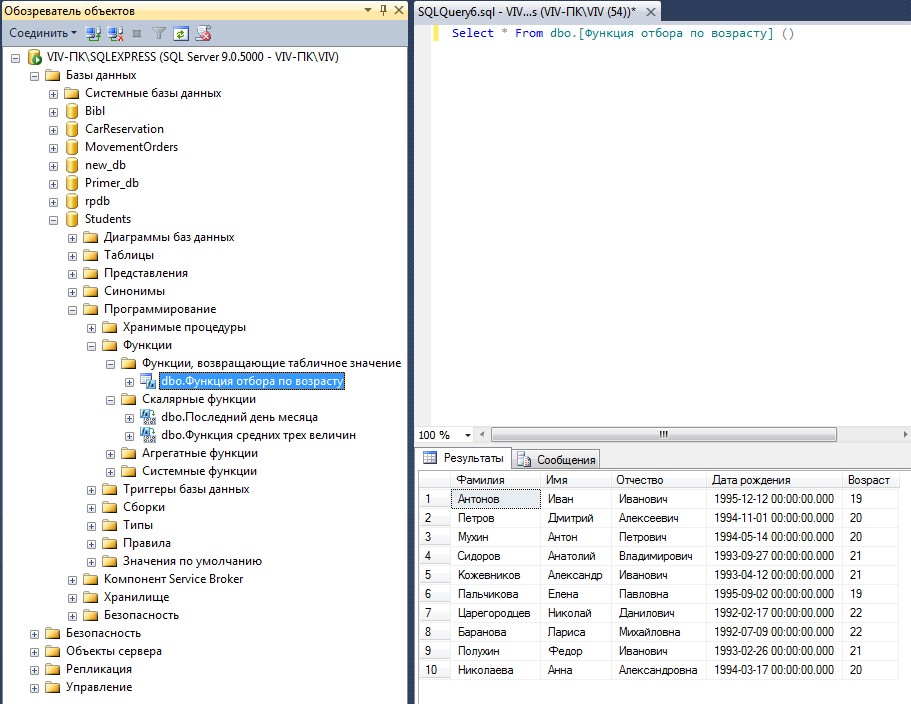


Рисунок 77 — Результат выполнения пользовательской функции

«Функция отбора по возрасту»

В нижней части окна появится таблица с фамилиями, именами, отчествами, датами рождения и возрастом студентов на данный момент времени (рис. 77).

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

Контрольные вопросы:

1. Что такое пользовательская функция?
2. В чем отличие пользовательской функции от хранимой процедуры?
3. Как создается пользовательских функций?
4. Как применяется пользовательская функция? 5) Как вызывается пользовательская функция?

Практическая работа №16 *«Триггеры»*

Цель практической работы:

1. Изучить порядок создания триггеров.

Практические задания и рекомендации по их выполнению:

ЗАДАНИЕ

Триггеры служат для обеспечения целостности данных в MS SQL Server также, как и диаграммы.

**Триггеры** — это аналог процедур обработчиков событий в языках программирования. T.е. они выполняют команды SQL, если происходят какие либо действия с таблицей (например, добавление, изменение или удаление записей). При помощи триггеров можно организовать автоматическое удаление записей из вторичной таблицы при удалении связанной с ними записи из первичной таблицы.

Рассмотрим создание триггеров при помощи языка SQL.

**Создание триггеров** В SQL Server существуют два вида триггеров:

1. Триггеры, выполняемые после события, произошедшего с таблицей (полный аналог процедур событий в Visual Basic).
2. Триггеры, выполняемые вместо события, происходящего с таблицей. В этом случае событие (добавление, изменение или удаление записей) не выполняется, а вместо него выполняются SQL команды, заданные внутри триггера.

Первый вид триггеров применяется для обработки событий таблиц, а второй — для обеспечения целостности данных, т.е. удаление записей из вторичной таблицы при удалении связанной с ними записи из первичной таблицы.

**Замечание**: Триггеры создаются для конкретной таблицы и выполняются автоматически, если с таблицей, для которой они были созданы, происходит событие (добавление, изменение или удаление записей).

Перейдём к созданию триггеров. Создадим триггеры для таблицы «Студенты». Триггеры создаются отдельно для каждой таблицы и располагаются в обозревателе объектов в папке “Triggers” («Триггеры»). В нашем случае, папка “Triggers” («Триггеры») входит в состав таблицы «Студенты» (рис. 78).

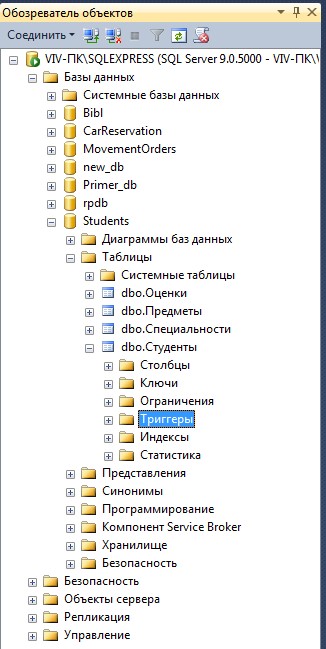


Рисунок 78 — Папка «Триггеры» для таблицы «Студенты» в окне обозревателя объектов

Для начала создадим триггер, выводящий сообщение «Запись добавлена» при добавлении записи в таблицу «Студенты». Создадим новый триггер, щёлкнув правой кнопкой мыши по папке “Triggers” («Триггеры») в таблице «Студенты» и в появившемся меню выбрав пункт “New Trigger” («Создать триггер»). Появится следующее окно с новым триггером (рис. 79):

1

|  |
| --- |
| 2 |

|  |
| --- |
| 3 |

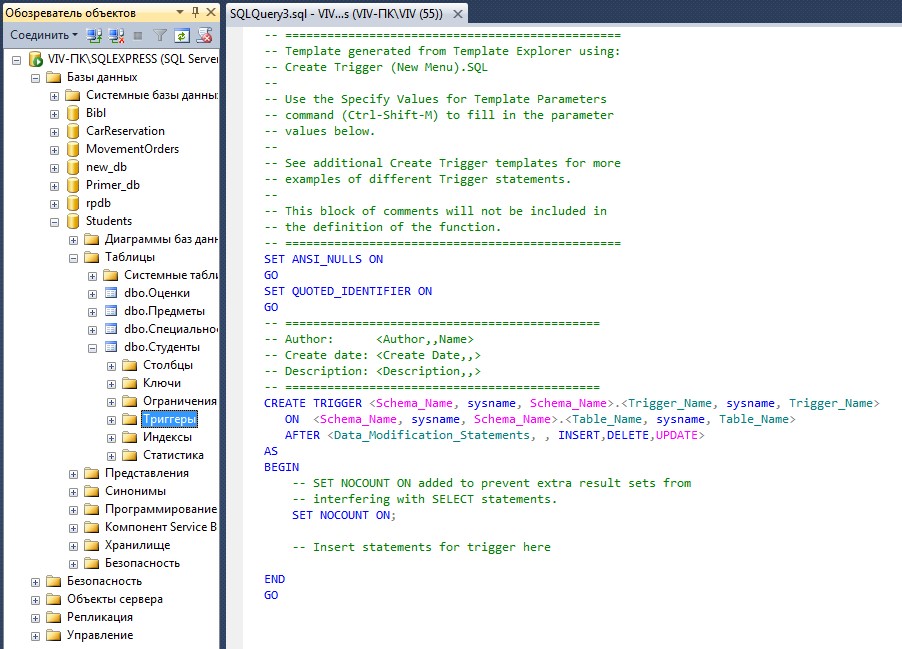
4

Рисунок 79 — Окно с новым триггером

Рассмотрим структуру триггеров:

1. Область определения имени функции (Trigger\_Name);
2. Область, показывающая, для какой таблицы создаётся триггер (Table\_Name);
3. Область, показывающая, когда выполнять триггер (INSERT — при создании записи в таблице, DELETE — при удалении и UPDATE — при изменении) и как его выполнять (ALTER — после выполнения операции, INSTEAD OF — вместо выполнения операции);
4. Тело триггера, содержит команды языка программирования запросов SQL.

В окне нового триггера наберите код, как показано на рисунке 80.

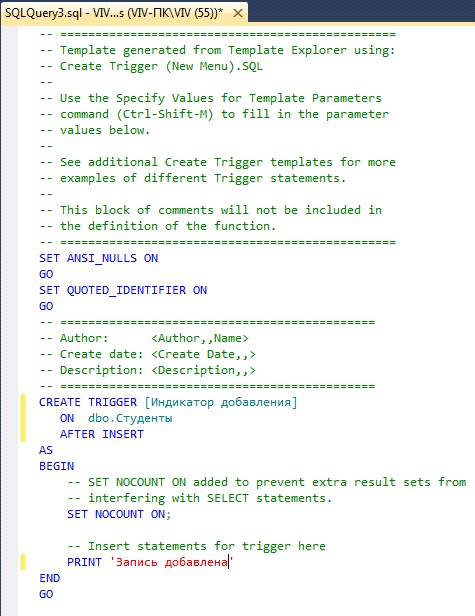


Рисунок 80 — Текст триггера «Индикатор добавления» для таблицы «Студенты»

Из рисунка 80 видно, что создаваемый триггер «Индикатор добавления» выполняется после добавления записи (AFTER INSERT) в таблицу «Студенты» (ON dbo.Студенты). После добавления записи триггер выведет на экран сообщение «Запись добавлена» (PRINT 'Запись добавлена').

Выполните набранный код, нажав кнопку ) на панели инструментов. В нижней части окна с кодом появится сообщение



(



“Command(s) completed successfully.” («Выполнение команд успешно завершено.»).

Проверим, как работает новый триггер. Создайте новый пустой запрос и в нём наберите следующую команду для добавления новой записи в таблицу «Студенты» (рис. 81):

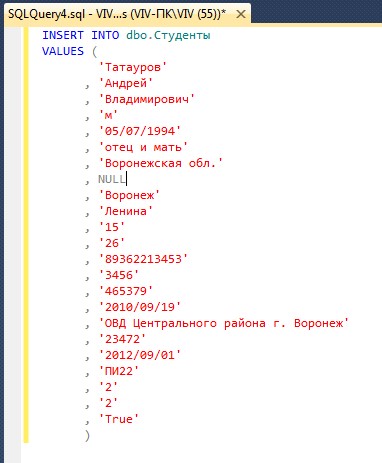


Рисунок 81 — Текст запроса для проверки работы триггера «Индикатор добавления» в таблицу «Студенты»

Выполните набранную команду, нажав кнопку ) на панели инструментов. В таблицу будет добавлена новая запись, и триггер выведет сообщение «Запись добавлена» в нижней части экрана.



(



Теперь создадим триггер, отображающий сообщение «Запись изменена». Создайте новый триггер, как в предыдущем случае. В окне нового триггера наберите следующий код (рис. 82):

Из рисунка 82 видно, что новый триггер «Индикатор изменения» выполняется после изменения записи (AFTER UPDATE) в таблице «Студенты» (ON dbo.Студенты). После изменения записи триггер выведет на экран сообщение «Запись изменена» (PRINT "Запись изменена").

Выполните набранный код. В нижней части окна с кодом появится сообщение «Command(s) completed successfully.» («Выполнение команд успешно завершено.»).

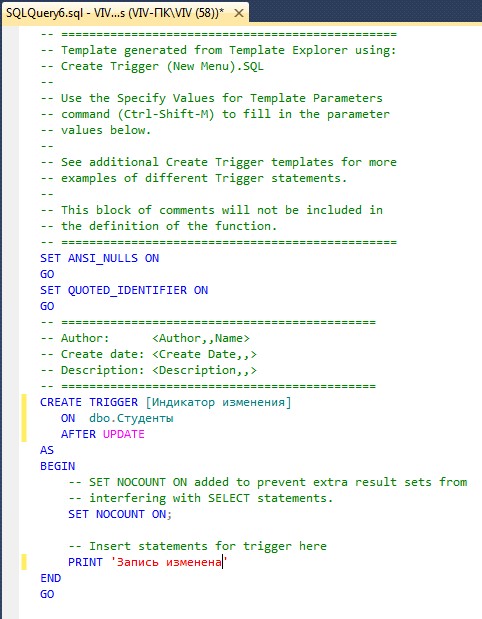


Рисунок 82 — Текст триггера «Индикатор изменения» для таблицы «Студенты»

Проверим работоспособность созданного триггера. Создайте новый запрос и в нём наберите команду, представленную на рисунке 83:

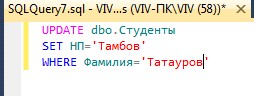


Рисунок 83 — Текст запроса для проверки работы триггера «Индикатор изменения» для таблицы «Студенты» Выполните набранную команду, нажав кнопку ) на панели инструментов.



(



В таблице будет изменена указанная запись, и триггер выведет сообщение «Запись изменена».

Для полноты картины создадим триггер, выводящий сообщение при удалении записи из таблицы «Студенты». Создайте новый триггер и в нём наберите код, показанный на рисунке 84.

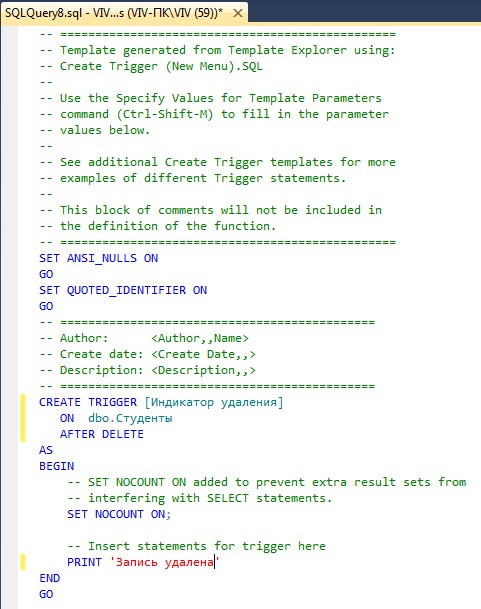


Рисунок 84 — Текст триггера «Индикатор удаления» для таблицы «Студенты»

Создаваемый триггер «Индикатор удаления» выполняется после удаления записи (ALTER DELETE) из таблицы «Студенты» (ON dbo.Студенты). После удаления записи триггер выводит сообщение «Запись удалена» (PRINT "Запись удалена").

Выполните код, представленный на рисунке 84. В нижней части окна с кодом появится сообщение «Command(s) completed successfully.» («Выполнение команд успешно завершено.»).

Проверим работу триггера «Индикатор удаления», удалив созданную ранее запись из таблицы «Студенты». Для этого создайте новый запрос и в нём наберите следующую команду (рис. 85):

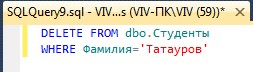


Рисунок 85 — Текст запроса для проверки работы триггера «Индикатор удаления» из таблицы «Студенты»

Выполните вышеприведённую команду. После удаления записи триггер «Индикатор удаления» отобразит сообщение «Запись удалена».

В заключение рассмотрим пример применения триггеров для обеспечения целостности данных. Создадим триггер «Удаление студента», который при удалении записи из таблицы «Студенты» сначала удаляет все связанные с ней записи из таблицы «Оценки», а затем удаляет саму запись из таблицы «Студенты», тем самым обеспечивается целостность данных.

Создайте новый триггер и в нём наберите следующий код (рис. 86):

Создаваемый триггер «Удаление студента» выполняется вместо удаления записи (INSTEAD OF DELETE) из таблицы «Студенты» (ON dbo.Студенты).

**Замечание**: При срабатывании триггера вместо удаления записи создаётся временная константа Deleted, содержащая имя таблицы, из которой должно было быть произведено удаление.

После срабатывания триггера из таблицы «Оценки» удаляется запись, у которой значение поля «Код студента» равно значению такого же поля у удаляемой записи из таблицы «Студенты». Эту операцию выполняют следующие команды:

DELETE dbo.Оценки

FROM Deleted

WHERE Deleted.[Код студента] = Оценки.[Код студента]

Затем удаляется запись из таблицы «Студенты», которую удаляли до срабатывания триггера. Удаление выполняется следующими командами:

DELETE dbo.Студенты

FROM Deleted

WHERE Deleted.[Код студента] = Студенты.[Код студента]

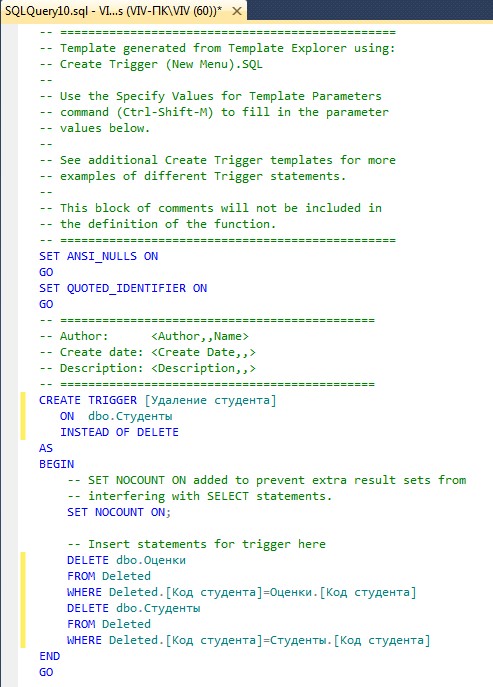


Рисунок 86 — Текст триггера «Удаление студента»

Выполните код, представленный на рисунке 86. В нижней части окна с кодом появится сообщение «Command(s) completed successfully.» («Выполнение команд успешно завершено.»).

Проверим, как работает триггер «Удаление студента». Для этого создайте новый запрос и в нём наберите следующий код (рис. 87):

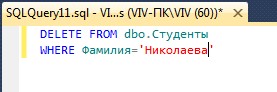


Рисунок 87 — Текст запроса для проверки работы триггера «Удаление студента»

При срабатывании триггера сначала из таблицы «Оценки» удалятся все связанные с удаляемой записью записи, а затем удаляется сама удаляемая запись из таблицы «Студенты», при этом сохраняется целостность данных. **Замечание**: Хотелось бы заметить, что без использования триггера «Удаление студента» нам бы не удалось удалить запись из таблицы «Студенты». Команда удаления была бы заблокирована диаграммой базы данных во избежание нарушения целостности данных.

Контрольные вопросы:

1. Что такое триггер?
2. Какие типы триггеров бывают?
3. Где сохраняются триггеры?
4. Какой порядок создания триггеров? 5) Какие параметры имеет триггер?