**ОСНОВЫ ПРОГРАММИРОВАНИЯ С ПОМОЩЬЮ ВСТРОЕННОГО ЯЗЫКА TRANSACT-SQL В MICROSOFT SQL SERVER**

**Цель работы**

* Изучить используемый в реляционных СУБД встроенный язык программирования Transact-SQL для написания программ в MS SQL Server.
* Изучить правила построения идентификаторов, правила объявления переменных и их типов.
* Изучить принципы работы с циклами и ветвлениями. Изучить работу с переменными типа Table.
* Изучить синтаксис и семантику функций и хранимых процедур Transact–SQL: способов их идентификации, методов задания и спецификации параметров и возвращаемых значений и вызовов функций и хранимых процедур.

Для освоения программирования используем пример базы данных**,** которая была создана в предыдущих работах. При выполнении примеров и заданий обращайте внимание на соответствие названий БД, таблиц и других объектов проекта.

**Идентификаторы** - это имена объектов, на которые можно ссылаться в программе, написанной на языке Transact SQL. Первый символ может состоять из букв английского алфавита или "**@**", "**#**". Остальные дополнительно из цифр и «$».

Имя идентификатора не должно совпадать с зарезервированным словом.

Для ограничителей идентификаторов при установленном параметре **SET QUOTED\_IDENTIFIER ON** можно использовать как квадратные скобки, так и одинарные кавычки, а строковые значения только в одинарных кавычках (режим по умолчанию).

Если использовать установленный параметр в режиме **SET QUOTED\_IDENTIFIER OFF**, то в качестве ограничителей идентификаторов можно использовать только квадратные скобки, а строковые значения указываются в одинарных или двойных кавычках.

**Переменные** используются для сохранения промежуточных данных в хранимых процедурах и функциях. Все переменные считаются локальными. Имя переменной должно начинаться с **@**.

**Объявление переменных**

Синтаксис в обозначениях MS SQL Server:

**DECLARE @имя\_переменной1 тип\_переменной, @имя\_переменнойN тип\_переменной**

Если тип переменной предполагает указание размера, то используется следующий синтаксис для объявления переменных:

**DECLARE @ имя\_переменной1 тип\_переменной (размер), .,**

@ имя\_переменнойN тип\_переменной(размер)

**Пример 1**: Введите в редактор SQL запросов следующее объявление переменных:

**DECLARE @a INT, @b numeric(10,2)**

**DECLARE @str CHAR(20)**

**Присвоение значений переменным и вывод значений на экран**

Присвоение с помощью **SET** - обычное присвоение, синтаксис:

**SET @имя\_переменной = значение**

Допишите в редакторе SQL объявление переменных с вводом их значений и выводом результата

**DECLARE @a INT, @b numeric (10,2)**

**SET @a = 20**

**SET @b = (@a+@a)/15**

**SELECT @b --вывод на экран результата**

Запустить эту конструкцию на выполнение.

Присвоение с помощью **SELECT** - помещение результата запроса в переменную. Если в результате выполнения запроса не будет возвращено ни одной строки, то значение переменной не меняется, т.е. остается старым.

Рассмотрим пример, в котором переменной присвоим результат выполнения агрегированной функции над таблицами базы данных.

**Пример 2:** Вывести количество строк в таблице **Студенты** и присвоить это значение переменной **@a.**

**DECLARE @a INT**

**SELECT @a = COUNT(\*) FROM Student**

**SELECT @a**

**Пример 3:**

**DECLARE @str CHAR(30)**

**SELECT @str = Sutname FROM Student**

**SELECT @str**

В данном примере в переменную поместится последнее значение из результата запроса.

**Сочетание ключевых слов SET и SELECT**

Измените код запроса из примера 2 на следующий:

**DECLARE @a INT**

**SET @a = (SELECT COUNT(\*) FROM Authors)**

**SELECT @a**

**Создание временной таблицы через переменную типа TABLE**

Объявляется через DECLARE с указанием в скобках столбцов таблицы, их типов, размеров, значений по умолчанию, а также индексов типа PRIMARY KEY или UNIQUE.

**Пример 4:** Создание временной таблицы с двумя полями, в которую будут добавлены 2 строки с данными:

**DECLARE @mytable TABLE (id INT, myname CHAR(20) DEFAULT 'Иванов Иван')**

**INSERT INTO @mytable(id) VALUES (1)**

**INSERT INTO @mytable(id, myname) VALUES (2,'Игорь Троцкий')**

**SELECT \* FROM @mytable**

Выполните и сохраните запрос.

**Пример 5:** Создание временной таблицы с двумя полями, в которую будут добавлены строки, как результат выполнения запроса выборки данных из таблицы **факультет.**

**DECLARE @mytable TABLE(id INT, myname CHAR(255) DEFAULT 'Введите название')**

**INSERT @mytable SELECT kod\_faculteta, name\_faculteta FROM facultet**

**SELECT \* FROM @mytable**

**Операторские скобки**

**BEGIN**

/\* в них нельзя помещать команды, изменяющие структуры объектов БД. Операторские скобки должны содержать хотя бы один оператор. Требуются для конструкций поливариантных ветвлений, условных и циклических конструкций

\*/

**END**

**Условная конструкция IF**

Синтаксис:

**IF условие**

Набор операторов**1 ELSE**

**Набор операторов2**

**Пример 6:** Использование операторских скобок и условных конструкций для поиска ответа – количество кафедр больше 10, да или нет

**DECLARE @a INT**

**DECLARE @str CHAR(30)**

**SET @a = (SELECT COUNT(\*) FROM kafedra)**

**IF @a >10 BEGIN**

**SET @str = 'Количество кафедр больше 10'**

**SELECT @str**

**END ELSE**

**BEGIN**

**SET @str = 'Количество кафедр = ' + str(@a)**

**SELECT @str**

**END**

**Цикл WHILE**

Синтаксис:

**WHILE Условие**

**Набор операторов**1 **BREAK**

**Набор опреторов**2 **CONTINUE**

Конструкции **BREAK** и **CONTINUE** являются необязательными.

Цикл можно принудительно остановить, если в его теле выполнить команду **BREAK**. Если же нужно начать цикл заново, не дожидаясь выполнения всех команд в теле, необходимо выполнить команду **CONTINUE**.

**Пример 7:** Объявление переменной **а**, проверка в цикле, чтобы значение **а** не превосходило 100. Переменная **а** увеличивает свое значение в случайном порядке. Также организуется проверка на условие.

DECLARE @a INT

SET @a = 1

WHILE @a <100

BEGIN

PRINT @a -- вывод на экран значения переменной

IF (@a>40) AND (@a<50)

BREAK --выход и выполнение 1-й команды за циклом

ELSE

SET @a = @a+rand()\*10

CONTINUE

END

PRINT @a

**Функции и хранимые процедуры** используются в SQL Server для реализации на языке Transact-SQL сложных часто используемых алгоритмов обработки данных или различных административных действий создания учетных записей, получения информации об объектах базы данных, управления свойствами сервера и баз данных, управления подсистемой репликации и автоматизации и т.д.

**Они хранятся в виде исходного текста и являются программными модулями**, существующими независимо от таблиц или каких-либо других объектов баз данных.

Хранение функций и хранимых процедур в виде исходных модулей языка Transact – SQL на сервере и в соответствующих базах данных **позволяет уменьшить размер запроса**, посылаемого по сети от клиента на сервер, а, следовательно, и **нагрузку на сеть**, что повышает общую производительность системы. Это также позволяет **упростить сопровождение программных комплексов** и внесение изменений в исходный текст модулей, причем большинство изменений не отразится на работоспособности клиентских приложений.

Значительная часть функций и хранимых процедур поставляются в составе SQL Server. Они называются **системными**, или **встроенными (built - in)**.

Кроме того, пользователю предоставляется возможность разрабатывать и включать в свою базу данных собственные, или **пользовательские (user-defined) функции и хранимые процедуры**, реализующие специальные алгоритмы обработки данных.

Таким образом, пользовательские функции и хранимые процедуры становятся объектами той базы данных, в которой они создавались. Поэтому при их создании, если необходимо, требуемую базу данных следует сделать текущей с помощью команды **USE** имя базы данных. Системные же функции хранятся на экземпляре сервера, а системные хранимые процедуры – в базе MASTER этого же экземпляра сервера.

**Функции и хранимые процедуры могут быть вызваны клиентскими программами**, другими функциями или хранимыми процедурами, а также **триггерам**и.

В любом случае необходимо указать имя функции или хранимой процедуры и список аргументов, которые сопоставляются параметрам соответствующей функции или хранимой процедуры, при этом типы аргументов и параметров должны совпадать или допускать автоматические преобразования типов. Если для некоторого параметра задано значение по умолчанию и это значение подходит для данного вызова, то соответствующий аргумент может быть опущен.

Поскольку функция возвращает значение, она используется в качестве операнда некоторого выражения в виде вызова функций, состоящего из имени этой функции и списка аргументов, заключенного в круглые скобки, при этом в качестве аргументов могут быть любые выражения языка Transact – SQL, дающие в результате значения требуемых типов.

Аргументы в вызове функции отделяются запятыми.

Если список аргументов пуст, то круглые скобки после имени функции, как правило, задаются.

Исключения составляют некоторые системные функции, для которых круглые скобки не задаются, когда нет аргументов.

Хранимые процедуры могут вызываться только командой **EXECUTE**, или сокращенно **EXEC**. За этой командой должны быть указаны имя процедуры и через пробел список аргументов, если вызывается процедура с параметрами. Аргументы разделяются запятой. Если для параметра задано значение по умолчанию, то аргумент либо совсем не задается (в конце списка), либо используется слово **DEFAULT**(в середине списка).

Процедура может возвращать результаты только через параметры с ключевым словом **OUTPUT**, при этом и аргумент должен быть задан с таким же ключевым словом.

Создание, изменение и удаление функций и хранимых процедур производится соответственно командами:

**CREATE FUNCTION,**

**CREATE PROCEDURE,**

**ALTER FUNCTION,**

**ALTER PROCEDURE,**

**DROP FUNCTION,**

**DROP PROCEDURE**.

**При создании функции** указывается тип возвращаемого значения и в теле функции обязательно задается команда **RETURN**, за которой следует выражения для вычисления возвращаемого значения.

**В теле процедуры** использование команды **RETURN** (конечно, без последующего выражения) вовсе не обязательно. Когда этой команды нет, выход из процедуры будет происходить после исполнения последней команды процедуры.

Тело, как функции, так и хранимой процедуры начинается ключевым словом **AS.**

Поскольку каждая из них хранится как отдельный объект, то для указания конца тела не требуется записывать какое-либо специальное ключевое слово или знак.

За командами создания функции или хранимой процедуры перечисляются имена параметров, начинающиеся с символа **@**, и их типы, а также важно значение по умолчанию.

Для функции этот список заключается в круглые скобки, после которых записывается ключевое слово **RETURNS** (возвращает) и тип возвращаемого значения.

Для хранимой процедуры круглые скобки не используются, и задавать тип возвращаемого значение не требуется.

Для тела функции часто используют ключевое слово **begin** после ключевого слова **as** и ключевое слово **end** в конце тела.

Дополнительные опции функции или хранимой процедуры задаются ключевым словом **with** до начала тела.

В SQL Server 2003 можно создавать **функции трех классов**:

**- Scalar** – возвращают обычное скалярное значение;

**- Inline** – возвращают таблицу динамической структуры, создаваемую единственной командой тела функции SELECT;

**- Multi – statement** – возвращает обычную таблицу заданной структуры, при этом количество команд в теле функции не ограничивается.

**СОЗДАНИЕ ФУНКЦИЙ**

Команда:

**CREATE FUNCTION (Transact-SQL)**

создает **определяемую пользователем функцию** в SQL Server. Определяемая пользователем функция представляет собой подпрограмму Transact-SQL или среды CLR, которая принимает параметры, выполняет действия, такие как сложные вычисления, а затем возвращает результат этих действий в виде значения. Возвращаемое значение может быть скалярным значением или таблицей. При помощи этой инструкции можно создать подпрограмму, которую можно повторно использовать следующими способами:

 В инструкциях Transact-SQL, например **SELECT**.

 В приложениях, вызывающих функцию.

 В определении другой пользовательской функции.

 Для параметризации представления или улучшения функциональности индексированного представления.

 Для определения столбца таблицы.

 Для определения ограничения **CHECK** на столбец.

 Для замены хранимой процедуры.

Синтаксис создания скалярной функции.

Аргументы

**schema\_name** - Имя схемы, к которой принадлежит определяемая пользователем функция.

**function\_name** - Имя определяемой пользователем функции. Имена функций должны удовлетворять правилам построения идентификаторов и должны быть уникальными в пределах базы данных и схемы.

**@parameter\_name** - Аргумент пользовательской функции. Может быть объявлен один или несколько аргументов.

Для функций допускается не более 2 100 параметров. При выполнении функции значение каждого из объявленных параметров должно быть указано пользователем, если для них не определены значения по умолчанию.

[ **type\_schema\_name**. ] **parameter\_data\_type -** Тип данных параметра

**[ =default ] -** Значение по умолчанию для аргумента. Если определено значение default, то функция выполняется даже в том случае, если для данного аргумента значение не указано.

**READONLY -** Указывает, что параметр не может быть обновлен или изменен при определении функции. Если тип параметра является определяемым пользователем табличным типом, то должно быть указано ключевое слово READONLY.

**return\_data\_type** - Возвращаемое значение скалярной функции, определяемой пользователем.

**function\_body** - Указывает серию инструкций Transact-SQL

**scalar\_expression** - Указывает скалярное значение, возвращаемое скалярной функцией.

**Предложение EXECUTE AS** - Указывает контекст безопасности, в котором выполняется определяемая пользователем функция. Иными словами, есть возможность управлять тем, какую учетную запись пользователя SQL Server использует при определении разрешений на объекты базы данных, на которые ссылается функция.

**Синтаксис функции, которая возвращает табличное значение**:

**Аргументы:**

**TABLE -** Указывает, что возвращаемым значением функции с табличным значением, является таблица. Функциям с табличным значением, могут передаваться только константы и @local\_variables.

Во встроенных функциях с табличным значением возвращаемое значение TABLE определяется при использовании единственной инструкции **SELECT**. Встроенные функции не имеют соответствующих возвращаемых переменных.

**select\_stmt -** Одиночная инструкция **SELECT**, определяющая возвращаемое значение встроенной функции с табличным значением.

В функциях допустимы следующие инструкции.

 Инструкции присваивания.

 Инструкции управления потоком, за исключением инструкций **TRY...CATCH**.

 Инструкции **DECLARE**, объявляющие локальные переменные и локальные курсоры.

 Инструкции **SELECT**, которые содержат списки выбора с выражениями, присваивающими значения локальным переменным.

 Операции над локальными курсорами, которые объявляются, открываются, закрываются и освобождаются в теле функции. Допустимы только те инструкции **FETCH,** которые предложением **INTO** присваивают значения локальным переменным. Инструкции **FETCH**, возвращающие данные клиенту, недопустимы.

 Инструкции **INSERT, UPDATE** и **DELETE**, которые изменяют локальные табличные переменные.

 Инструкции **EXECUTE**, вызывающие расширенные хранимые процедуры.

**Ограничения**

Определяемые пользователем функция не может выполнять действия, изменяющие состояние базы данных.

Определяемые пользователем функции не могут содержать предложение **OUTPUT INTO**, целью которого является таблица.

Определяемые пользователем функции могут быть **вложенными**, то есть из одной функции может быть вызвана другая. Вложенность определяемых пользователем функций не может превышать 32 уровней.

**Пример 1.** Применение скалярной определяемой пользователем функции, вычисляющей месяц.

В следующем примере показано создание определяемой пользовательской функции **ISOweek,** которая получает в качестве аргумента дату и вычисляет номер месяца, а затем по номеру определяет название месяца. Для правильной работы этой функции перед ее вызовом должна быть выполнена инструкция **SET DATEFIRST 1.**

Следующий пример также показывает использование предложения **EXECUTE AS** для указания контекста безопасности, в котором может быть выполнена хранимая процедура. В этом примере параметр **CALLER** указывает, что процедура будет выполнена в контексте пользователя, который ее вызывает.

В начале, прежде чем создавать функцию, будет выполнена проверка на наличие в базе данных функции под таким же именем. Если функция с таким именем уже была создана ранее, то она будет удалена с помощью команды **DROP FUNCTION** и создана заново **CREATE FUNCTION**.

Откройте sql-редактор. Создайте новый запрос.

**USE University;**

**GO**

**IF OBJECT\_ID (N'dbo.ISOweek', N'FN') IS NOT NULL**

**DROP FUNCTION dbo.ISOweek;**

**GO**

**CREATE FUNCTION dbo.ISOweek (@DATE date)**

**RETURNS CHAR(15)**

**WITH EXECUTE AS CALLER**

**AS**

**BEGIN**

**DECLARE @man int;**

**DECLARE @ISOweek char(15);**

**SET @man= MONTH(@DATE)**

**IF (@man=1) SET @ISOweek='Январь';**

**IF (@man=2) SET @ISOweek='Февраль';**

**IF (@man=3) SET @ISOweek='Март';**

**IF (@man=4) SET @ISOweek='Апрель';**

**IF (@man=5) SET @ISOweek='Май';**

**IF (@man=6) SET @ISOweek='Июнь';**

**IF (@man=7) SET @ISOweek='Июль';**

**IF (@man=8) SET @ISOweek='Август';**

**IF (@man=9) SET @ISOweek='Сентябрь';**

**IF (@man=10) SET @ISOweek='Октябрь';**

**IF (@man=11) SET @ISOweek='Ноябрь';**

**IF (@man=12) SET @ISOweek='Декабрь';**

**RETURN(@ISOweek);**

**END;**

Для того чтобы увидеть результат, выведем на экран вычисление значение функции от произвольной даты, например, '12.04.2004'. для этого ниже в sql-редакторе пишем:

**GO**

**SET DATEFIRST 1;**

**SELECT dbo.ISOweek('12.04.2004') AS 'Месяц';**

Ниже приводится результирующий набор.

Просмотреть все функции, которые пользователь может создавать, откройте в окне обозревателя объектов в базе данных **University** группу **Программирование/Функции/Скалярные функции**.

**Пример 2. Создание пользовательской функции с табличным значением**.

Например, создадим пользовательскую функцию, которая будет разрешена к использованию всеми пользователями с правами роли «**dekan**». Функция будет возвращать результат в виде **таблицы** – вывод на всех кафедрах суммирующей зарплаты по каждой должности всех преподавателей. При этом функция имеет один параметр **@storeid**, с помощью которого введем ограничение на вычисление, а именно зарплата должна быть больше 100.

Создайте новый запрос:

**USE University;**

**GO**

**IF OBJECT\_ID (N'ufn\_SalesByStore', N'IF') IS NOT NULL**

**DROP FUNCTION dekan.ufn\_SalesByStore;**

**GO**

**CREATE FUNCTION dekan.ufn\_SalesByStore(@storeid int)**

**RETURNS TABLE**

**AS**

**RETURN**

**(**

**SELECT d.Name\_kafedru AS "Кафедра",**

**t.Dolgnost AS "Должность",** 164

**SUM(t.Salary + t.RISE) AS "Сумма зарплаты"**

**FROM KAFEDRA d, TEACHER t**

**WHERE d.KOD\_kafedru =t.KOD\_kafedru**

**and t.salary>@storeid**

**GROUP BY d.Name\_kafedru, t.Dolgnost**

**);**

Для вызова этой функции нужно выполнить следующий запрос:

**GO**

**SELECT \* from dekan.ufn\_SalesByStore(100);**

Таблица Orders представляет заказы, а столбец CreatedAt - дату заказов. *Узнаем, были ли заказы за последние 10 дней:*

DECLARE @lastDate DATE

SELECT @lastDate = MAX(CreatedAt) FROM Orders

IF DATEDIFF(day, @lastDate, GETDATE()) > 10

PRINT 'За последние десять дней не было заказов'

Добавим выражение ELSE:

DECLARE @lastDate DATE

SELECT @lastDate = MAX(CreatedAt) FROM Orders

IF DATEDIFF(day, @lastDate, GETDATE()) > 10

PRINT 'За последние десять дней не было заказов'

ELSE

PRINT 'За последние десять дней были заказы'

Если после IF или ELSE идут две и более инструкций, то они заключаются в блок BEGIN...END:

DECLARE @lastDate DATE, @count INT, @sum MONEY

SELECT @lastDate = MAX(CreatedAt),

@count = SUM(ProductCount) ,

@sum = SUM(ProductCount \* Price)

FROM Orders

IF @count > 0

BEGIN

PRINT 'Дата последнего заказа: ' + CONVERT(NVARCHAR, @lastDate)

PRINT 'Продано ' + CONVERT(NVARCHAR, @count) + ' единиц(ы)'

PRINT 'На общую сумму ' + CONVERT(NVARCHAR, @sum)

END;

ELSE

PRINT 'Заказы в базе данных отсутствуют'

Для выполнения повторяющихся операций в T-SQL применяются циклы. В частности, в T-SQL есть цикл WHILE. Этот цикл выполняет определенные действия, пока некоторое условие истинно.

WHILE условие

{инструкция|BEGIN...END}

Если в блоке WHILE необходимо разместить несколько инструкций, то все они помещаются в блок BEGIN...END.

Например, вычислим факториал числа:

DECLARE @number INT, @factorial INT

SET @factorial = 1;

SET @number = 5;

WHILE @number > 0

BEGIN

SET @factorial = @factorial \* @number

SET @number = @number - 1

END;

PRINT @factorial

То есть в данном случае пока переменная @number не будет равна 0, будет продолжаться цикл WHILE. Так как @number равна 5, то цикл сделает пять проходов. Каждый проход цикла называется итерацией. В каждой итерации будет переустанавливаться значение переменных @factorial и @number.

Другой пример - рассчитаем баланс счета через несколько лет с учетом процентной ставки:

USE productsdb;

CREATE TABLE #Accounts ( CreatedAt DATE, Balance MONEY)

DECLARE @rate FLOAT, @period INT, @sum MONEY, @date DATE

SET @date = GETDATE()

SET @rate = 0.065;

SET @period = 5;

SET @sum = 10000;

WHILE @period > 0

BEGIN

INSERT INTO #Accounts VALUES(@date, @sum)

SET @period = @period - 1

SET @date = DATEADD(year, 1, @date)

SET @sum = @sum + @sum \* @rate

END;

SELECT \* FROM #Accounts

Здесь создается временная таблица #Accounts, в которую добавляется в цикле пять строк с данными.

Операторы BREAK и CONTINUE

Оператор BREAK позволяет завершить цикл, а оператор CONTINUE - перейти к новой итерации.

DECLARE @number INT

SET @number = 1

WHILE @number < 10

BEGIN

PRINT CONVERT(NVARCHAR, @number)

SET @number = @number + 1

IF @number = 7

BREAK;

IF @number = 4

CONTINUE;

PRINT 'Конец итерации'

END;

Когда переменная @number станет равна 4, то с помощью оператора CONTINUE произойдет переход к новой итерации, поэтому последующая строка PRINT 'Конец итерации' не будет выполняться, хотя цикл продолжится.

Когда переменная @number станет равна 7, то оператор BREAK произведет выход из цикла, и он завершится.