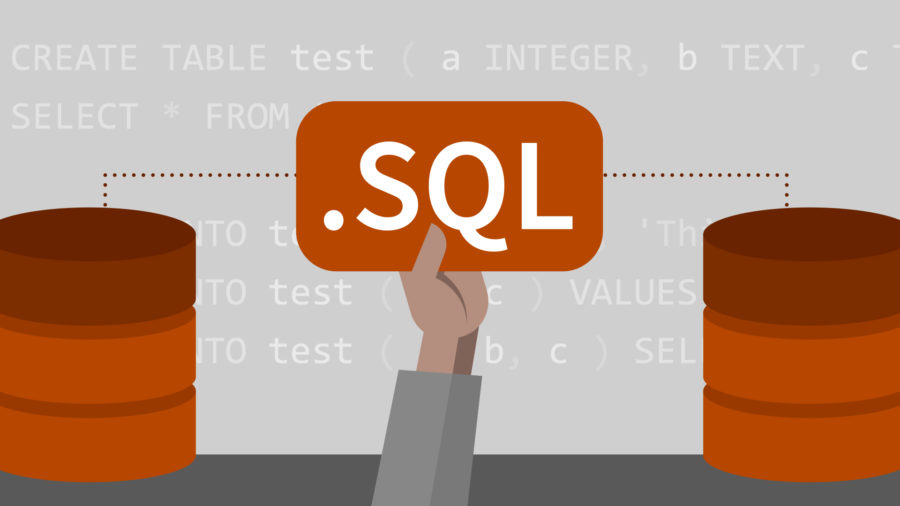
**Зачем аналитику нужно программирование на SQL?**

АВТОР: [РОМАН РОМАНЧУК](http://thisisdata.ru/blog/author/roman-romanchuk/)



Многие аналитики работающие с SQL и занимающиеся анализом данных, никогда не сталкивались с программированием на этом языке, потому что SQL в первую очередь язык структурированных запросов, а не полноценный язык программирования. Однако, программные возможности в нем есть, хотя и весьма скромные.

В тот момент, когда при написании запроса мы сталкиваемся с какими-либо ограничениями, например синтаксическими, нам может пригодиться программирование. Ниже я расскажу о программных конструкциях и модулях существующих в SQL и чем они могут помочь аналитику.

Так как работаю я в основном с Microsoft SQL Server, то в тексте речь пойдет о программных возможностях баз данных построенных именно на этой системе.  
В SQL Azure, Oracle Database, PostgreSQL и прочих реляционных базах синтаксис может отличаться, но принцип остается тем же.

Программные конструкции

Программная конструкция — это часть программы, отвечающая за определенное действие и объясняющая компьютеру как решить задачу. То есть небольшой строительный блок из которых и состоит программный код.

Переменные

Изучение любого из языков программирования, практически всегда начинается со знакомства с переменными.

Переменная — это ячейка в памяти компьютера, используемая для временного хранения какой-либо информации (чисел, строк, дат и т.п.).

У переменной есть три важных атрибута:

* Имя (чтобы можно было к ней обратиться);
* Тип данных (чтобы понимать, что с этой переменной можно делать);
* Область видимости (чтобы понимать, где она будет использоваться).

Также в SQL у переменной есть три фазы:

1. Объявление переменной при помощи инструкции DECLARE;
2. Инициализация при помощи SET или SELECT;
3. Использование переменной при помощи подстановки ее имени в нужное место вашего скрипта.

Давайте создадим переменную:

1. -- Объявляем переменную с именем 'StartDate' и задаем тип данных date
2. **DECLARE** @StartDate **date**
3. -- Инициализируем переменную и присваиваем ей значение '2019-09-30'
4. -- Если нужно объявить сразу несколько переменных, то можно воспользоваться инструкцией SELECT (пример будет ниже)
5. **SET** @StartDate = '2019-09-30'
6. -- Используем переменную в запросе
7. **SELECT** \*
8. **FROM** Orders
9. **WHERE** OrderDate = @StartDate

**Как переменные могут помочь аналитику?**

Первое что приходит на ум — это создание отчета с динамическими периодами, чтобы не редактировать каждый раз даты отчета в условии WHERE:

1. -- Объявляем переменные с именем 'StartDate' и 'EndDate'
2. **DECLARE** @StartDate **date**
3. , @EndDate **date**
4. -- Инициализируем переменные и присваиваем для 'StartDate' значение '2019-09-30', а для @EndDate используем специальную функцию возвращающую текущую дату GETDATE()
5. **SELECT** @StartDate = '2019-09-30', @EndDate = GETDATE()
6. -- Используем переменную в запросе
7. **SELECT** \*
8. **FROM** Orders
9. **WHERE** OrderDate BETWEEN @StartDate AND @EndDate

Что касается области видимости, то нужно запомнить, что переменная видима с момента ее объявления до конца скрипта, после чего она уничтожается.

Условные операторы

Практически все языки программирования предоставляют возможность использовать условные операторы и SQL не исключение.

Условный оператор приходит на помощь в тот момент, когда вам нужно выполнить либо одно, либо другое действие в зависимости от заданного условия.

Синтаксис:

* Ключевое слово IF;
* Условие, в соответствии с которым будет происходить проверка;
* Необязательное ключевое слово ELSE, указывающее на то, что делать если условие не выполнилось.

Давайте разберем на примере:

1. -- Объявляем переменную с именем @var и сразу присваиваем ей значение (так тоже можно)
2. **DECLARE** @var **int** = 4
3. -- Простая форма, если условие не выполняется, то ничего не происходит
4. IF 2 + 2 = @var
5. **SELECT** 'Верно'
6. -- Сложная форма, если условие не выполняется, говорим что делать
7. IF 2 + 2 = @var
8. **SELECT** 'Верно'
9. **ELSE**
10. **SELECT** 'Ошибка'

Чаще всего на практике требуются более сложные составные условия, содержащие различные логические связки.

**Зачем это аналитику?**

Логический оператор IF, сам по себе нельзя использовать в обычном SELECT-е, однако он активно применяется при создании программных модулей, таких как функции и хранимые процедуры о которых мы поговорим ниже.

Но есть логический оператор, который можно использовать внутри запроса и речь идет о операторе CASE он же оператор множественного выбора.

Оператор множественного выбора в зависимости от указанных условий позволяет вернуть одно из множества возможных значений, то есть CASE работает как несколько IF-ов.

Синтаксис:

* Ключевое слово CASE;
* Ключевое слово WHEN, после которого идет условие проверки;
* Ключевое слово THEN, после которого идет действие к выполнению;
* Необязательное ключевое слово ELSE, указывающее на то, что делать если ни одно из условий не выполнилось;
* Ключевое слово END.

На практике все гораздо проще:

1. -- CASE работает только внутри SELECT
2. **SELECT**
3. -- При помощи CASE разобьем все товары на 4 группы в зависимости от цены
4. CASE
5. **WHEN** price <= 99 **THEN** 'Дешевые товары'
6. **WHEN** price BETWEEN 100 AND 500 **THEN** 'Средняя ценовая группа'
7. **WHEN** price > 500 **THEN** 'Дорогие товары'
8. **ELSE** 'Бесплатно'
9. **END**
10. **FROM** Orders

Для аналитика польза очевидна, так как при помощи CASE, можно создавать новые параметры отчетах, а также в зависимости от условий рассчитывать показатели.

Для примера давайте создадим новую группу каналов, ведь часто в отчетах бизнес-пользователи хотят видеть более понятные названия вместо smm\_cpc, cpc или organic.

**SELECT**

CASE

**WHEN** medium = 'smm\_cpc' **THEN** 'Таргетированная реклама'

**WHEN** medium = 'cpc' **THEN** 'Контекстная реклама'

**WHEN** medium = 'organic' **THEN** 'Органический поиск'

**WHEN** medium = 'referral' **THEN** 'Переходы по ссылкам'

**ELSE** 'Прочее'

**END**

**FROM** Orders

Циклы

Ни один уважающий себя язык программирования не обходится без циклов.

Цикл — это инструкция, позволяющая выполнять один и тот же запрос несколько раз, пока условие истинно.

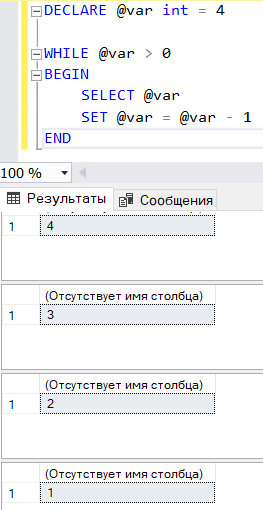
Синтаксис:

* Ключевое слово WHILE;
* Условие в соответствии с которым будет выполняться цикл.

Рассмотрим пример:

1. -- Объявляем переменную с именем @var
2. **DECLARE** @var **int** = 4
3. -- Создаем цикл и рядом пишем условие в соответствии с которым, цикл будет выполняться до тех пор, пока @var не будет равно 0
4. WHILE @var > 0
5. -- Заключаем тело цикла в скобки, в SQL они обозначаются как BEGIN ... END
6. **BEGIN**
7. -- В SELECT выводим текущее значение переменной, а в SET устанавливаем для нее новое значение минус 1
8. **SELECT** @var
9. **SET** @var = @var - 1
10. **END**

В результате запрос, находящийся в операторных скобках, будет выполнен 4 раза:



**В чем же польза для аналитика?**

Так же как и в случае с условным оператором IF, цикл нельзя вставить внутрь запроса и он нужен для создания программных модулей о которых мы сейчас и поговорим.

Программные модули

Программный модуль — это запрос или скрипт сохраненный на сервере и оформленный в виде объектов.

Главная фишка использования программных модулей состоит в том, что они позволяют собирать сложные решения из простых составных частей. То есть позволяют декомпозировать большую задачу на несколько мелких.

Представление

При решении какой-либо задачи, наши запросы могут стать достаточно большими и их контекст сложно постоянно держать в голове. И как только такая ситуация возникнет — лучшим решением будет использование функционала представлений.

Представление — это сохраненный в базе запрос с которым можно работать так, как будто бы это готовая таблица.

У представления есть несколько ограничений:

1. В качестве представления можно сохранить не любой запрос, а только SELECT;
2. В запросе не должно быть сортировки;
3. Все столбцы должны иметь имена.

Синтаксис:

* Ключевое слово CREATE VIEW использующееся для создания представления;
* Ключевое слово AS, после которого должен идти текст запроса.

При создании представления сервер не выполняет запрос, а сохраняет его в базе для последующего использования.

Давайте разберем на примере:

1. -- Создаем представление с именем SalesReport
2. **CREATE** **VIEW** SalesReport
3. **AS**
4. -- Далее указываем наш большой и сложный запрос, текст которого мы хотим сохранить на сервере
5. **SELECT**
6. [**date**]
7. , cost
8. , impressions
9. , clicks
10. , sessions
11. , orders
12. , revenue
13. **FROM** Orders
14. **WHERE** medium = 'cpc'

И теперь, чтобы получить результат выполнения запроса, нам достаточно выполнить следующий код:

1. **SELECT** \*
2. **FROM** SalesReport

**Чем это полезно аналитику?**

Использование представлений позволяет абстрагироваться от сложностей появляющихся при написании запроса, что крайне актуально, когда ваш запрос не помещается в один экран.

И теперь можно дальше усложнять запрос, используя представления как строительные блоки для конечного отчета.

Табличное выражение

Чтобы создавать представления, у вас должны быть права администратора к базе данных. Но что делать когда таких прав нет, но возможности представлений использовать хочется?

Специально для этого был придуман функционал табличного выражения.

Табличное выражение — это представление которое не сохраняется в базе, а существует только в момент выполнения SELECT-а и после самоликвидируется.

Синтаксис:

* Ключевое слово WITH использующееся для создания табличного выражения;
* Ключевое слово AS, после которого должен идти псевдоним;
* В скобочках () указывается текст запроса, который будет выполняться при обращении к псевдониму;
* После табличного выражения должен идти одиночный SELECT.

Пример:

1. -- Создаем табличное выражение с именем Cost
2. **WITH** Cost
3. **AS** (
4. -- Далее в скобках указываем наш запрос
5. **SELECT**
6. [**date**]
7. , sourceMedium
8. , campaign
9. , cost
10. , impressions
11. , clicks
12. **FROM** Advertising
13. )
14. -- Выводим результат выполнения запроса
15. **SELECT** \*
16. **FROM** Cost

**Пока не очень понятно в чем польза и чем это отличается от представления?**

Давайте разберем пример из практики практически любого digital-аналитика:

1. Перед вами поставили задачу — создать отчет для отдела контекстной рекламы с доходом и расходом по рекламным кампаниям.
2. Данные по расходам хранятся в одной базе, а данные по заказам в другой.
3. Ключами позволяющими связать данные между собой являются [UTM-метки](http://thisisdata.ru/blog/vse-chto-vy-khoteli-znat-o-utm-metkakh-no-boyalis-sprosit/).

Для нас теперь это раз плюнуть, вооружимся табличными выражениями и напишем запрос:

1. -- Запрашиваем расходы
2. **WITH** Cost
3. **AS** (
4. **SELECT**
5. [**date**]
6. , sourceMedium
7. , campaign
8. , cost
9. , impressions
10. , clicks
11. **FROM** Advertising
12. )
13. -- Запрашиваем заказы
14. , Orders
15. **AS** (
16. **SELECT**
17. [**date**]
18. , sourceMedium
19. , campaign
20. , orders
21. , revenue
22. **FROM** Crm
23. )
24. -- Создаем итоговый отчет
25. , Report
26. **AS** (
27. **SELECT**
28. Cost.[**date**]
29. , Cost.sourceMedium
30. , Cost.campaign
31. , SUM(Cost.cost) **AS** 'cost'
32. , SUM(Cost.impressions) **AS** 'impressions'
33. , SUM(Cost.clicks) **AS** 'clicks'
34. , SUM(Orders.orders) **AS** 'orders'
35. , SUM(Orders.revenue) **AS** 'revenue'
36. **FROM** Cost
37. -- Соединяем данные между собой по дате и UTM-меткам
38. LEFT JOIN Orders
39. **ON** Cost.[**date**] = Orders.[**date**]
40. AND Cost.sourceMedium = Orders.sourceMedium
41. AND Cost.campaign = Orders.campaign
42. **GROUP** **BY** Cost.[**date**], Cost.sourceMedium, Cost.campaign
43. )
44. -- Выводим результат
45. **SELECT** \*
46. **FROM** Report

Да, конечно, эту задачу можно было решить гораздо меньшим количеством кода.

Но моей целью было показать вам суть табличных выражений — они помогают разбить задачу на маленькие подзадачи, структурировать и упорядочить ваш код, что особенно актуально при написании больших и сложных отчетов.

Процедуры

Как и в случае представления, процедура представляет собой сохраненный в базе запрос, но процедура не является таблицей и поэтому на нее не накладываются такие строгие ограничения, как в представлении.

Основное назначение процедур — это сохранение программных сценариев для выполнения определенных действий в базе данных (UPDATE, DELETE, INSERT и тп.), в том числе для вызова других процедур и функций.

Но у процедуры есть ограничения:

1. Процедуру нельзя использовать в SELECT-е;
2. Процедуру нельзя усложнять «снаружи», то есть ее нельзя JOIN-ть, отфильтровать, группировать и тп.

Синтаксис:

* Ключевое слово CREATE PROCEDURE использующееся для создания процедуры;
* В процедурах можно использовать параметры, название которых указывается после знака @;
* Ключевое слово AS, после которого должен идти программный код;
* Для вызова процедуры используется ключевое слово EXECUTE вместе с ее названием.

Пример:

1. -- Создаем процедуру с именем HumanResourcesProc
2. **CREATE** **PROCEDURE** HumanResourcesProc
3. -- Указываем параметры процедуры, которые будут выступать в качестве переменных
4. @FirstName nvarchar(50),
5. @LastName nvarchar(50)
6. **AS**
7. -- Далее идет текст скрипта
8. **SELECT**
9. FirstName
10. , LastName
11. , Department
12. **FROM** HumanResources
13. **WHERE** FirstName = @FirstName AND LastName = @LastName

И теперь, чтобы вызвать процедуру, достаточно выполнить следующий код:

1. -- Запускаем выполнение скрипта процедуры для вывода информации по определенному сотруднику
2. **EXECUTE** HumanResourcesProc 'Иван', 'Иванов'

**Чем процедуры могут помочь аналитику?**

Функционал процедур можно задействовать для построения на основе вашей базы данных полноценного ETL-решения — то есть процесса по извлечению данных из внешних источников, их трансформации и очистке, а также загрузке в хранилище.

А подключая к хранилищу [Python или R-скрипты](https://docs.microsoft.com/ru-ru/sql/advanced-analytics/what-is-sql-server-machine-learning?view=sql-server-ver15), вы сможете обучать, оценивать и развертывать модели машинного обучения прямо в базе данных.

Функции

Зачастую нам нужно повторять одно и то же действие во многих частях программы или даже в разных отчетах. И чтобы не писать один и то же код в разных местах были придуманы функции.

Функции — это конструкции, содержащие исполняемый код и являющиеся основными строительными блоками программы.

В некоторых языках программирования нет деления на функции и процедуры, так в чем же отличия?

Их несколько:

1. Функция может быть усложнена как «внутри», так и «снаружи», то есть она поддается сортировке, группировке и JOIN-у.
2. Функция всегда возвращает результат определенного типа, например, встроенная функция SUM — всегда вернет число. Процедура же может выполниться и ничего не вернуть или вернуть число, строку, дату и даже таблицу в зависимости от того, как она запрограммирована.

И как тогда определиться что использовать? Все просто.

Процедуру стоит использовать, если вы собираетесь менять базу или для оформления конечного результата, не требующего дополнительной обработки, во всех остальных случаях используйте функции.

Синтаксис:

* Ключевое слово CREATE FUNCTION использующееся для создания функции;
* Параметры функции указываются в скобках () и после знака @;
* Ключевое слово RETURNS указывает на тип возвращаемых функцией данных;
* Ключевое слово AS, после которого должен идти программный код;
* Ключевое слово RETURN, сообщающее функции, что нужно вернуть значение.

Давайте переделаем процедуру из предыдущего примера в функцию:

1. -- Создаем функцию с именем HumanResourcesFunc
2. **CREATE** **FUNCTION** HumanResourcesFunc (
3. -- В скобочках указываем параметры функции, которые будут выступать в качестве переменных
4. @FirstName nvarchar(50),
5. @LastName nvarchar(50)
6. )
7. -- Указываем что в данном случае функция должна вернуть нам таблицу
8. **RETURNS** **TABLE**
9. **AS**
10. **RETURN**
11. -- Далее идет текст скрипта
12. **SELECT**
13. FirstName
14. , LastName
15. , Department
16. **FROM** HumanResources
17. **WHERE** FirstName = @FirstName AND LastName = @LastName

Для вызова созданной функции используем следующий код:

1. -- Запускаем выполнение скрипта функции для вывода информации по определенному сотруднику
2. **SELECT** \* **FROM** HumanResourcesFunc ('Иван', 'Иванов')

**Как функции могут облегчить жизнь аналитику?**

Помните, чуть выше в блоке про оператор множественного выбора, мы разбирали [пример](http://thisisdata.ru/blog/zachem-analitiku-nuzhno-programmirovaniye-na-sql/#case) с новой группой каналов?

А теперь, представьте, что это действие — присваивание понятного названия канала, нам нужно сделать не один раз в одном месте, а много и в нескольких отчетах.

Чтобы не повторять один и тот же код, можно написать функцию, которая заменит CASE:

1. -- Создаем функцию с именем ChannelFunc
2. **CREATE** **FUNCTION** ChannelFunc (
3. -- В скобочках указываем параметр функции, который будет выступать в качестве переменной
4. @medium nvarchar(50)
5. )
6. -- Указываем, что в данном случае функция должна вернуть нам одно значение в виде строки
7. **RETURNS** nvarchar (100)
8. **AS**
9. -- В операторных скобкам описываем логику при помощи IF
10. **BEGIN**
11. IF @medium = 'smm\_cpc'
12. **RETURN** 'Таргетированная реклама'
13. IF @medium = 'cpc'
14. **RETURN** 'Контекстная реклама'
15. IF @medium = 'organic'
16. **RETURN** 'Органический поиск'
17. IF @medium = 'referral'
18. **RETURN** 'Переходы по ссылкам'
19. **RETURN** 'Прочее'
20. **END**

И теперь, если мы захотим в каком-нибудь отчете заменить стандартные каналы на их русские названия, то просто применим функцию:

1. **SELECT**
2. [**date**]
3. -- Вместо CASE вызываем функцию ChannelFunc с параметром medium
4. , ChannelFunc([medium]) **AS** 'newChannel'
5. , campaign
6. , cost
7. , impressions
8. , clicks
9. **FROM** Advertising

Что дальше?

В [следующей статье](http://thisisdata.ru/blog/avtomatizatsiya-otchetnosti-pri-pomoshchi-sql-i-power-bi/) я расскажу как, используя программные возможности языка SQL, можно создать автообновляемый отчет на основе стека технологий MS SQL Server и Power BI.

Документация Microsoft (Transact-SQL):

* [Типы данных](https://docs.microsoft.com/ru-ru/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-2017);
* [Переменные](https://docs.microsoft.com/ru-ru/sql/t-sql/language-elements/variables-transact-sql?view=sql-server-2017);
* [Оператор IF…ELSE](https://docs.microsoft.com/ru-ru/sql/t-sql/language-elements/if-else-transact-sql?view=sql-server-ver15);
* [Оператор CASE](https://docs.microsoft.com/ru-ru/sql/t-sql/language-elements/case-transact-sql?view=sql-server-ver15);
* [Цикл WHILE](https://docs.microsoft.com/ru-ru/sql/t-sql/language-elements/while-transact-sql?view=sql-server-ver15);
* [Представления](https://docs.microsoft.com/ru-ru/sql/t-sql/statements/create-view-transact-sql?view=sql-server-ver15);
* [Табличное выражение](https://docs.microsoft.com/ru-ru/sql/t-sql/queries/with-common-table-expression-transact-sql?view=sql-server-ver15);
* [Процедуры](https://docs.microsoft.com/ru-ru/sql/t-sql/statements/create-procedure-transact-sql?view=sql-server-ver15);
* [Функции](https://docs.microsoft.com/ru-ru/sql/t-sql/statements/create-function-transact-sql?view=sql-server-ver15).