**02 Introduction to T-SQL Querying**

Lesson 1

Introducing T-SQL

About T-SQL

T-SQL-это реализация Microsoft стандартного отраслевого языка структурированных запросов. Язык был первоначально разработан для поддержки новой реляционной модели данных в International Business Machines (IBM) в начале 1970-х годов. С тех пор SQL получил широкое распространение в отрасли. SQL стал стандартом Американского национального института стандартов (ANSI) и Международной организации по стандартизации (ISO) в 1980-х годах.

Стандарт ANSI прошел несколько пересмотров, включая SQL-89 и SQL-92—спецификации полностью или частично поддерживаются T-SQL. SQL Server также реализует функции из более поздних стандартов, таких как ANSI SQL-2008 и ANSI SQL-2011. Microsoft, как и многие другие поставщики, также расширила язык, включив в него функции и функции, специфичные для SQL Server.

Помимо реализации Microsoft в качестве T-SQL в SQL Server, Oracle имеет PL/SQL, IBM имеет SQL PL, а Sybase поддерживает свою собственную операцию T-SQL.

При работе с T-SQL важно понимать, что это не процедурный язык, а язык, основанный на множествах и декларативный язык. Когда вы пишете запрос для получения данных из SQL Server, вы описываете данные, которые хотите отобразить; вы не указываете SQL Server, как именно их получить. Вместо предоставления процедурного списка шагов, которые необходимо выполнить, вы предоставляете атрибуты запрашиваемых данных.

С помощью T-SQL компонент SQL Server Database Engine определит оптимальный путь для доступа к данным и вернет соответствующий набор. Ваша роль состоит в том, чтобы научиться писать эффективный и точный код T-SQL, чтобы вы могли правильно описать набор, который вы хотите получить.

Если у вас есть опыт работы в других средах программирования, принятие нового мышления может стать проблемой. Этот курс был разработан, чтобы помочь вам преодолеть разрыв между процедурным и декларативным T-SQL на основе наборов.

Categories of T-SQL Statements

Операторы T-SQL можно разделить на три категории:

Язык обработки данных (DML)-это набор инструкций T-SQL, который фокусируется на запросе и изменении данных. Это включает в себя SELECT, основной фокус этого курса, и операторы изменения, такие как INSERT, UPDATE и DELETE. Вы узнаете о некоторых утверждениях на протяжении всего этого курса.

Язык определения данных (DDL)-это набор операторов T-SQL, который обрабатывает определение и жизненный цикл объектов базы данных, таких как таблицы, представления и процедуры. Это включает в себя такие операторы, как CREATE, ALTER и DROP.

Язык управления данными (DCL)-это набор инструкций T-SQL, используемых для управления разрешениями безопасности для пользователей и объектов. DCL включает в себя такие операторы, как GRANT, REVOKE и DENY.

**T-SQL Language Elements**

Как и многие языки программирования, T-SQL содержит элементы, которые вы будете использовать в запросах. Вы будете использовать предикаты для фильтрации строк; операторы для выполнения сравнений; функции и выражения для управления данными или извлечения системной информации; а также комментарии для документирования кода. Если вам нужно выйти за рамки написания инструкций SELECT для создания хранимых процедур, триггеров и других объектов, вы можете использовать такие элементы, как операторы управления потоком, переменные для временного хранения значений и разделители пакетов. Следующие несколько тем этого урока познакомят вас со многими из этих элементов.

Цель этого урока-познакомить с общими элементами языка T-SQL, которые будут представлены здесь на высоком концептуальном уровне.

**T-SQL Language Elements: Predicates and Operators**

Язык T-SQL предоставляет элементы для задания и оценки логических выражений. В операторах SELECT можно использовать логические выражения для определения фильтров для предложений WHERE и HAVING.

Вы будете писать эти выражения, используя предикаты и операторы.

Предикаты, поддерживаемые T-SQL, включают следующее:

IN: используется для определения соответствия значения любому значению в списке или подзапросе.

Например, ГДЕ день В (1,5,6,10).

МЕЖДУ: используется для указания диапазона значений. Например, ГДЕ ставка ОТ 3 ДО 7.

НАПРИМЕР: используется для сопоставления символов с шаблоном.

Например, ГДЕ фамилия ТИПА " %mith%’.

Операторы включают в себя несколько общих категорий:

Сравнение. Для тестов на равенство и неравенство: =, <, >, >=, <=, !=, !>, !< (Обратите внимание, что !>, !< и != не являются стандартом ISO—рекомендуется использовать стандартные параметры, когда они существуют.)

Логический. Для проверки правильности условия: И, ИЛИ, НЕТ.

Арифметика. Для выполнения математических операций: +, -, \*, /, % (по модулю).

Сцепление. Для объединения строк символов: +.

Назначение. Для установки значения: =.

T-SQL Language Elements: Functions

SQL Server предоставляет широкий спектр функций для ваших запросов T-SQL. Они варьируются от скалярных функций, таких как SYSDATETIME, которые возвращают однозначный результат, до других, которые работают и возвращают целые наборы, такие как оконные функции, о которых вы узнаете позже в этом курсе.

Как и в случае с операторами, функции SQL Server могут быть организованы в категории. Вот некоторые общие категории скалярных (однозначных) функций, доступных для написания запросов:

* String functions
* o SUBSTRING, LEFT, RIGHT, LEN, DATALENGTH
* o REPLACE, REPLICATE
* UPPER, LOWER, RTRIM, LTRIM
* o STUFF
* o SOUNDEX
* **Date and time functions**
* o GETDATE, SYSDATETIME, GETUTCDATE
* o DATEADD, DATEDIFF
* o YEAR, MONTH, DAY
* o DATENAME, DATEPART
* o ISDATE
* **Aggregate functions**
* o SUM, MIN, MAX, AVG
* o COUNT, COUNT\_BIG
* o STDEV, STDEVP
* o VAR
* **Mathematical functions**
* o RAND, ROUND, POWER, ABS
* o CEILING, FLOOR

Цель этого урока-познакомить со многими элементами языка T-SQL, который представлен здесь на высоком концептуальном уровне. Последующие модули этого курса покажут более подробные объяснения.

**T-SQL Language Elements: Variables 2-7**

Как и многие языки программирования, T-SQL предоставляет средство временного хранения значения определенного типа данных. Однако, в отличие от других сред программирования, все созданные пользователем переменные являются локальными для пакета T-SQL, который их создал, и видны только этому пакету. Пользователям SQL Server недоступны глобальные или общедоступные переменные.

Чтобы создать локальную переменную в T-SQL, необходимо указать имя, тип данных и начальное значение. Имя должно начинаться с одного символа @ (at), а тип данных должен быть системным или определяемым пользователем и храниться в базе данных, с которой будет выполняться ваш код.

В литературе по SQL Server, на веб-сайтах и т. Д. Вы можете Найти ссылки на так называемые “системные переменные” с двойным именем@@, например @@ERROR. Более точно называть их системными функциями, поскольку пользователи не могут присвоить им значение. Этот курс будет отличать пользовательские переменные с префиксом @ от системных функций с префиксом @@.

Вы можете назвать и инициализировать переменную в одном и том же операторе. Если ваша переменная не инициализирована в инструкции DECLARE, она будет создана со значением NULL, и впоследствии вы можете присвоить значение с помощью инструкции SET.

The following example creates a character variable initialized to the string 'Program%':

Character Variable

**DECLARE @search varchar(30) = 'Match%';**

The following example creates a date variable and assigns the current date:

Date Variable

**DECLARE @CurrentDate date;**

**SET @CurrentDate = GETDATE();**

Вы узнаете больше о различных типах данных, включая даты, и переменных T—SQL позже в этом курсе. Если требуется постоянное хранилище или глобальная видимость значения, рассмотрите возможность создания таблицы. SQL Server обеспечивает как временное, так и постоянное хранение в базах данных. Дополнительные сведения о различных типах таблиц см. в разделе Microsoft Docs:

<http://aka.ms/quew7f>

T-SQL Language Elements: Expressions

T-SQL предоставляет комбинации идентификаторов, символов и операторов, которые вычисляются SQL Server для возврата одного результата. Эти комбинации известны как выражения, предлагая полезный и мощный инструмент для ваших запросов. В операторах SELECT можно использовать выражения:

В предложении SELECT для работы со столбцами и/или управления ими.

Как выражения CASE для замены значений, соответствующих логическому выражению, другим значением.

В предложении WHERE для построения предикатов для фильтрации строк.

В качестве табличных выражений для создания временных наборов, используемых для дальнейшей обработки.

Цель этого урока-познакомить со многими элементами языка T-SQL,

которые будут представлены здесь на высоком концептуальном уровне.

Выражения могут быть основаны на скалярной (однозначной) функции, на постоянном значении или на переменных. Несколько выражений могут быть объединены с помощью операторов, если они имеют один и тот же тип данных или если тип данных может быть преобразован из более низкого приоритета в более высокий приоритет (например, int в money).

Следующий пример выражения работает со столбцом, чтобы добавить целое число к результатам функции YEAR в столбце datetime:

Expression

SELECT YEAR(orderdate) AS currentyear, YEAR(orderdate) + 1 AS nextyear

FROM Sales.Orders;

**T-SQL Language Elements: Control of Flow, Errors, and Transactions**

Хотя T-SQL в основном является языком поиска данных, а не процедурным языком, он поддерживает ограниченный набор операторов, которые обеспечивают некоторый контроль потока во время выполнения.

Некоторые из часто используемых операторов управления потоком включают в себя:

• ЕСЛИ ... В противном случае, для обеспечения управления ветвлением на основе логического теста.

• WHILE-для повторения оператора или блока операторов, когда условие истинно.

• НАЧИНАЙТЕ ... КОНЕЦ, для определения экстентов блока операторов T-SQL.

• ПОПРОБУЙТЕ ... CATCH, для определения структуры обработки исключений (обработки ошибок).

• THROW, для создания исключения и передачи выполнения в блок CATCH.

• НАЧАТЬ ТРАНЗАКЦИЮ, чтобы пометить блок операторов как часть явной транзакции. Завершается ТРАНЗАКЦИЕЙ ФИКСАЦИИ или ТРАНЗАКЦИЕЙ ОТКАТА.

**T-SQL Language Elements: Comments**  2-9

В T-SQL есть два метода документирования кода или указания ядру базы данных игнорировать определенные операторы. Какой метод вы используете, как правило, зависит от количества строк кода, которые вы хотите игнорировать:

Для отдельных строк или очень немногих строк кода используйте -- (двойной тире) перед текстом, который будет помечен как комментарий. Любой текст, следующий за тире, будет проигнорирован SQL Server.

Для более длинных блоков кода заключите текст между символами /\* и\*/. Любой код между символами будет игнорироваться SQL Server.

В следующем примере используется метод -- (двойной тире) для пометки комментариев:

Пример однострочного комментария:

Однострочные комментарии

-- Вся эта строка-комментарий

ОБЪЯВИТЕ @search varchar(30) = 'Match%'; -- конец строки

В следующем примере используется метод /\* comment block \*/ для пометки комментариев:

Пример комментария к блоку:

Комментарий к блоку

/\*

Весь текст в этом абзаце будет рассматриваться как комментарии

с помощью SQL Server.

\*/

Многие инструменты редактирования запросов, такие как SQL Server Management Studio (SSMS), Visual Studio ® или SQLCMD, будут окрашивать текст с комментариями в цвет, отличный от окружающего кода T-SQL. В SSMS используйте диалоговое окно Инструменты, параметры для настройки цветов и шрифтов в редакторе сценариев T-SQL.

T-SQL Language Elements: Batch Separators

Клиентские средства SQL Server, такие как SSMS, отправляют команды в компонент database engine в наборах, называемых пакетами. Если вы выполняете код вручную, например в редакторе запросов, вы можете выбрать, следует ли отправлять весь текст в сценарии в виде одного пакета. Вы также можете вставить разделители между определенными разделами кода.

Спецификация разделителя пакетов обрабатывается вашим клиентским инструментом. Например, ключевое слово GO является разделителем пакетов по умолчанию в SSMS. Вы можете изменить это для текущего запроса в разделе Запрос | Параметры запроса или глобально в разделе Инструменты | Параметры | Выполнение запроса.

Для большинства простых запросов разделители пакетов не используются, поскольку вы будете отправлять один запрос за раз. Однако, когда вам нужно создавать объекты и манипулировать ими, вам может потребоваться разделить операторы на отдельные пакеты. Например, оператор CREATE VIEW может не включаться в тот же пакет, что и другие операторы.

Ниже приведен пример инструкции CREATE TABLE и CREATE VIEW в одном пакете:

Code That Requires Multiple Batches

CREATE TABLE table1 (col1 int);

CREATE VIEW view1 as SELECT \* FROM table1;

Обратите внимание, что объявленные пользователем переменные считаются локальными для пакета, в котором они объявлены. Если переменная объявлена в одном пакете и на нее ссылаются в другом, второй пакет завершится ошибкой. Вставьте разделитель пакетов GO между двумя операторами CREATE, чтобы устранить предыдущую ошибку.

Например, следующие операторы работают правильно при отправке вместе как один пакет:

DECLARE @cust int = 5;

SELECT custid, companyname, contactname

FROM Sales.Customers

WHERE custid = @custid;

Однако, если бы разделитель пакетов был вставлен между объявлением переменной и запросом, в котором используется переменная, произошла бы ошибка.

В следующем примере объявление переменной отделяется от ее использования в запросе:

DECLARE @cust int = 5;

GO

SELECT custid, companyname, contactname

FROM Sales.Customers

WHERE custid = @custid;

The previous example returns the following error:

Msg 137, Level 15, State 2, Line 5

Must declare the scalar variable "@custid".

Demo 1.

**Lesson 2 2-13**

**Understanding Sets**

Set Theory and SQL Server

Теория множеств является одной из математических основ реляционной модели и имеет фундаментальное значение для

работы с SQL Server. Хотя вы, возможно, сможете добиться прогресса в написании запросов на T-SQL без оценки наборов, в конечном итоге у вас могут возникнуть трудности с выражением некоторых из них в одном хорошо выполняемом операторе.

Этот урок подготовит почву для того, чтобы вы начали "мыслить множествами" и понимать их природу.

В свою очередь, это облегчит вам:

Воспользуйтесь преимуществами операторов на основе наборов в T-SQL.

Поймите, почему вам все еще нужно сортировать выходные данные запроса.

Поймите, почему декларативный подход, основанный на множествах, а не процедурный, лучше всего работает с SQL Server.

Для наших целей, не углубляясь в математику, поддерживающую теорию множеств, множество определяется как "совокупность определенных, различных объектов, рассматриваемых как единое целое." В терминах, применяемых к базам данных SQL Server, набор можно рассматривать как единое целое (например, таблицу), содержащее ноль или более элементов одного типа. Например, таблица клиентов представляет набор—в частности, набор всех клиентов. Вы увидите, что результаты оператора SELECT также образуют набор, который будет иметь важные последствия при изучении подзапросов и табличных выражений.

По мере того, как вы узнаете больше о некоторых операторах запросов T-SQL, важно всегда думать обо всем наборе, а не об отдельных членах. Это позволит вам лучше писать код на основе наборов, а не думать по одной строке за раз. Работа с множествами требует мышления в терминах операций, которые происходят "все сразу", а не по одному за раз. В зависимости от вашего фона, это может потребовать корректировки.

После "коллекции" следующий критический термин в нашем определении - "отдельный", или уникальный. Все члены набора должны быть уникальными. В SQL Server уникальность обычно реализуется с помощью ключей, таких как столбец первичного ключа.

Однако, когда вы начинаете работать с подмножествами данных, важно знать, как вы можете однозначно обращаться к каждому члену набора.

**2-14**

Это возвращает нас к рассмотрению множества как "целого"." Известный автор языка SQL Джо Селко предлагает мысленно добавить фразу "Набор всех..." перед именами объектов SQL, представляющих наборы (например, "набор всех клиентов"). Это поможет вам помнить, что при написании кода T-SQL вы обращаетесь к коллекции элементов, а не только к одному элементу за раз.

Одним из важных соображений является то, что упущено из теории множеств—любое требование, касающееся порядка элементов в множестве. Короче говоря, в наборе нет предопределенного порядка. Элементы могут быть адресованы (и извлечены) в любом порядке. Применительно к вашим запросам это означает, что, если вам нужно вернуть результаты в определенном порядке, вы должны использовать предложение ORDER BY в операторах SELECT.

**Set Theory Applied to SQL Server Queries**

Учитывая основанную на множествах основу баз данных, есть несколько соображений и рекомендаций, которые следует учитывать при написании эффективных запросов T-SQL:

Действуйте сразу на всей съемочной площадке. Это означает запрос всей таблицы сразу, а не на основе курсора или итеративной обработки.

Используйте декларативную обработку на основе наборов. Сообщите SQL Server, что вы хотите получить, описав его атрибуты, а не перейдя к его положению.

По возможности убедитесь, что вы обращаетесь к элементам с помощью их уникальных идентификаторов, таких как ключи. Например, напишите предложения СОЕДИНЕНИЯ, ссылающиеся на уникальные ключи на одной стороне связи.

Предоставьте свои собственные инструкции по сортировке, поскольку результирующие наборы не гарантированно будут возвращены в любом порядке.

**DEMO: Create table**

**Lesson 3 2-16**

**Understanding Predicate Logic**

Логика предикатов и SQL Server

Теоретически логика предикатов-это структура для выражения логических тестов, которые возвращают true или false. Предикат-это свойство или выражение, которое является истинным или ложным. Возможно, вы слышали, что это называется логическим выражением.

Взятые сами по себе, предикаты делают сравнения и выражают результаты как истинные или ложные. Однако в T-SQL предикаты не стоят отдельно. Они обычно встроены в оператор, который делает что-то с результатом true или false, например, предложение WHERE для фильтрации строк, выражение CASE для сопоставления значения или даже ограничение столбца, регулирующее диапазон допустимых значений для этого столбца в определении таблицы.

В формальном определении предиката есть одно важное упущение—как обрабатывать неизвестные или отсутствующие значения. Если база данных настроена таким образом, что пропущенные значения недопустимы (из-за ограничений или назначений значений по умолчанию), то, возможно, это не является важным упущением. Однако в большинстве реальных сред вам необходимо учитывать отсутствующие или неизвестные значения и расширять свое понимание предикатов с двух возможных исходов (истинных или ложных) до трех—истинных, ложных или неизвестных.

Использование нулей в качестве отметки для отсутствующих данных будет обсуждаться далее в следующем разделе, а затем в этом курсе.

**Predicate Logic Applied to SQL Server Queries**

Как вы уже поняли, возможность использовать предикаты для выражения сравнений в терминах true, false или unknown жизненно важна для написания эффективных запросов в SQL Server. Хотя они обсуждали их отдельно, предикаты не стоят отдельно, синтаксически говоря. Как правило, вы будете использовать предикаты в любой из следующих ролей в своих запросах:

Фильтрация данных (в предложениях WHERE и HAVING).

Предоставление условной логики выражениям РЕГИСТРА.

Объединение таблиц (в фильтре ON).

Определение подзапросов (например, в тестах EXISTS).

Кроме того, предикаты используются вне операторов SELECT, например, в ограничениях ПРОВЕРКИ для ограничения значений, разрешенных в столбце, и в элементах управления потоком, таких как оператор IF.

В математике вам нужно учитывать только те значения, которые присутствуют, поэтому предикаты могут приводить только к истинным или ложным значениям (известным в логике предикатов как “закон исключенной середины”). Однако в базах данных вам, скорее всего, придется учитывать пропущенные значения; взаимодействие предикатов T-SQL с пропущенными значениями приводит к неизвестному. При разработке логики запросов убедитесь, что вы учли все три возможных варианта.outcomes—true, false, or unknown.

**Lesson 4 2-18**

**Understanding the Logical Order of Operations in SELECT Statements**

Elements of a SELECT Statement

Чтобы понять логический порядок операций, необходимо рассмотреть инструкцию SELECT в целом, включая ряд необязательных элементов. Однако этот урок не предназначен для предоставления

подробной информации об этих элементах—каждая часть инструкции SELECT будет обсуждаться в

последующих модулях. Например, понимание деталей предложения WHERE не требуется для определения его места в последовательности событий.

Оператор SELECT состоит из обязательных и необязательных элементов. Строго говоря, SQL Server требует только предложения SELECT для выполнения без ошибок. Предложение SELECT без предложения FROM работает так, как если бы выбор производился из воображаемой таблицы, содержащей одну строку. Вы увидите это поведение при тестировании переменных позже в этом курсе. Однако, поскольку предложение SELECT без предложения FROM не может извлекать данные из таблицы, вы будете рассматривать отдельные предложения SELECT как особый случай, не имеющий прямого отношения к этому уроку. Давайте рассмотрим элементы, их роль высокого уровня в операторе SELECT и порядок, в котором они оцениваются SQL Server.

Не все элементы будут присутствовать в каждом запросе SELECT. Однако, когда элемент присутствует, он всегда будет оцениваться в том же порядке, что и другие присутствующие. Например, предложение WHERE всегда будет оцениваться после предложения FROM и перед предложением GROUP BY, если таковое существует.

Для целей этого урока дополнительные необязательные элементы, такие как, DISTINCT,

OVER, and TOP, опущены.

Logical Query Processing

The order in which a SELECT statement is written is not that in which it is evaluated and processed by the SQL Server Database Engine.

Logical Query Processing

USE TSQL;

SELECT EmployeeId, YEAR(OrderDate) AS

OrderYear

FROM Sales.Orders

WHERE CustomerId = 71

GROUP BY EmployeeId, YEAR(OrderDate)

HAVING COUNT(\*) > 1

ORDER BY EmployeeId, OrderYear;

Прежде чем вы изучите порядок операций во время выполнения, давайте кратко рассмотрим, что делает запрос, хотя подробные сведения о многих предложениях должны будут подождать до соответствующего модуля. Первая строка гарантирует, что вы подключены к правильной базе данных для запроса. Эта строка не проверяется на предмет ее порядка во время выполнения.

При необходимости это необходимо выполнить до выполнения основного запроса SELECT:

Изменение подключения к базе данных

ИСПОЛЬЗУЙТЕ TSQL; -- измените контекст подключения к базе данных с именем TSQL.

Следующая строка-это начало инструкции SELECT в том виде, в каком вы ее написали, но, как вы увидите, она не будет первой оцениваемой строкой.

Предложение SELECT возвращает столбец EmployeeID и извлекает только год из столбца OrderDate:

**SELECT EmployeeId, YEAR(OrderDate) AS OrderYear**

The FROM clause identifies which table is the source of the rows for the query—in this case Sales.Orders:

**FROM Sales.Orders**

The WHERE clause filters the rows out of the Sales.Orders table, keeping only those that satisfy the predicate—in this case, a customer with an ID of 71:

**WHERE CustomerId = 71**

The GROUP BY clause groups together the remaining rows by EmployeeId, and then by the year of the order:

**GROUP BY EmployeeId, YEAR(OrderDate)**

После создания групп предложение HAVING фильтрует их на основе своего предиката. Только сотрудники с более чем одной продажей на одного клиента в данном году пройдут этот фильтр:

**HAVING COUNT(\*) > 1 2-20**

For the purposes of previewing this query, the final clause is the ORDER BY, which sorts the output by EmployeeId, and then by year:

**ORDER BY EmployeeId, OrderYear;**

Теперь, когда вы установили, что делает каждое предложение, давайте рассмотрим порядок, в котором SQL Server должен их оценивать:

1. Сначала вычисляется предложение FROM, чтобы предоставить исходные строки для остальной части инструкции. Позже в этом курсе вы увидите, как объединить несколько таблиц вместе в предложении FROM. Виртуальная таблица создается и передается на следующий шаг.

2. Далее следует вычислить предложение WHERE, отфильтровав те строки из исходной таблицы, которые соответствуют предикату. Отфильтрованная виртуальная таблица передается на следующий шаг.

3. Далее следует ГРУППИРОВАТЬ ПО, упорядочивая строки в виртуальной таблице в соответствии с уникальными значениями, найденными в списке ГРУПП ПО. Создается новая виртуальная таблица, содержащая список групп, и передается на следующий шаг.

С этого момента в потоке операций другие элементы могут ссылаться только на столбцы в ГРУППЕ ПО списку или агрегатным функциям. Это окажет значительное влияние на список ИЗБРАННЫХ.

4. Затем вычисляется предложение HAVING, отфильтровывающее целые группы на основе его предиката. Виртуальная таблица, созданная на шаге 3, фильтруется и передается на следующий шаг.

5. Наконец, выполняется предложение SELECT, определяющее, какие столбцы будут отображаться в результатах запроса.

Поскольку предложение SELECT вычисляется после других шагов, любые псевдонимы столбцов (в

нашем примере OrderYear), созданные там, не могут использоваться в предложении GROUP BY или HAVING.

6. В нашем примере предложение ORDER BY выполняется последним, сортируя строки, определенные в списке столбцов.

Чтобы применить это к нашему примеру запроса, вот логический порядок во время выполнения, с оператором USE, опущенным для ясности:

Logical Order

FROM Sales.Orders

WHERE CustomerId = 71

GROUP BY EmployeeId, YEAR(OrderDate)

HAVING COUNT(\*) > 1

SELECT EmployeeId, YEAR(OrderDate) AS OrderYear

ORDER BY EmployeeId, OrderYear;

Как вы уже видели, вы не пишете запросы T-SQL в том же порядке, в котором они логически оцениваются. Поскольку порядок оценки во время выполнения определяет, какие данные доступны для предложений ниже по потоку друг от друга, важно понимать истинный логический порядок при написании запросов.

Applying the Logical Order of Operations to Writing SELECT Statements **2-21**

Теперь, когда вы изучили логический порядок операций при оценке и обработке запроса SELECT, помните о следующих соображениях при написании запроса. Обратите внимание, что некоторые из них могут относиться к деталям, которые вы узнаете в последующих модулях:

Решите, какие таблицы запрашивать в первую очередь, в дополнение к любым псевдонимам таблиц, которые вы будете применять. Это определит предложение FROM.

Решите, какой набор или подмножество строк будет извлечен из таблицы(таблиц) в предложении FROM и как вы будете выражать свой предикат. Это определит ваше предложение WHERE.

Если вы собираетесь сгруппировать строки, решите, какие столбцы будут сгруппированы. Помните, что только столбцы в предложении GROUP BY, в дополнение к агрегатным функциям, таким как COUNT, в конечном итоге могут быть включены в предложение SELECT.

Если вам нужно отфильтровать группы, определите свой предикат и создайте предложение HAVING. Результаты этой фазы становятся входными данными для предложения SELECT.

Если вы не используете GROUP BY, определите, какие столбцы из исходных таблиц вы хотите отобразить, и используйте любые созданные вами псевдонимы таблиц для ссылки на них. Это станет ядром вашего предложения SELECT. Если вы использовали предложение GROUP BY, выберите один из столбцов в предложении GROUP BY и добавьте любые дополнительные агрегаты в список ВЫБОРА.

Наконец, помните, что наборы не включают никакого упорядочения—вам нужно будет добавить предложение ORDER BY, чтобы гарантировать порядок сортировки, если это необходимо.

Demo.

Lab