УНИВЕРСИТЕТ ИТМО

В.В. Повышев

БАЗЫ ДАННЫХ. ПРАКТИКУМ

Санкт-Петербург
2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

В.В. Повышев

БАЗЫ ДАННЫХ. ПРАКТИКУМ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлению подготовки 09.03.02 Информационные системы и технологии
в качестве учебно-методического пособия для реализации основных
профессиональных образовательных программ высшего образования
бакалавриата

Санкт-Петербург
2020

Повышев В.В. Базы данных. Практикум – СПб: Университет ИТМО, 2019. – 50 с.

Рецензент(ы):
Береснев Артем Дмитриевич, старший преподаватель факультета инфокоммуникационных технологий, Университета ИТМО.

В пособии приведены описание синтаксиса и использования инструкции SELECT языка T-SQL, набор практических заданий с решениями, набор заданий для самопроверки.

УНИВЕРСИТЕТ ИТМО

Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

Содержание

Введение

В курсе Базы данных студент знакомятся с принципами организации работы реляционных баз данных. Важнейшей задачей, осуществляемой системами управления базами данных, является обеспечения доступа до данных. Структурированный язык запросов SQL обеспечивает возможность взаимодействия пользователя с СУБД и получения данных. Наиболее сложной частью языка является написание инструкций SELECT, изучению которой и посвящён данный практикум. В процессе изучения дисциплины студент на практике осваивает следующие умения и навыки: составлять запросы на языке SQL, выбирать структуру запроса для минимизации времени его выполнения, составления эффективных запросов к реляционной СУБД. Полученные навыки необходимы для успешного освоения таких дисциплин как Программирование, Технологии программирования, Информационные системы и т.д.

Практикум состоит из шести работ, которые необходимо изучать последовательно. Каждая последующая работа подразумевает использование навыков и умений, полученных в предыдущей работе. Работы состоят из теоретического материала, содержащего описание инструкций запросов SQL, сопровождаются примерами и пояснениями к ним, так же в работах рассматривается примеры выполнения конкретных задач, и даны задачи для самостоятельной работы.

Для выполнения практических работ используется демонстрационное программное обеспечение компании Microsoft: Microsoft SQL Server Express, Microsoft SQL Server Management Studio и учебная база данных Microsoft Adventure Works. Программное обеспечение и база данных доступна для свободного использования на официальном сайте поддержки компании Microsoft - docs.microsoft.com.

Практическая работа 1

Цель работы: Изучение базового синтаксиса инструкции SELECT, выражения WHERE и ORDER BY.

Базовый синтаксис инструкции SELECT

Общий синтаксис инструкции SELECT весьма сложен, однако достаточное представление дает следующее описание:

```
SELECT лист выборки [INTO новая таблица ]
[ FROM источник данных ] [ WHERE условия поиска ]
[ GROUP BY выражение для группировки ]
[ HAVING условия поиска по группам ]
[ ORDER BY выражение для сортировки [ ASC | DESC ] ]
```

В данном описании надо учитывать, что курсивом указаны пользовательские, обязательные, параметры синтаксиса конструкции на языке T-SQL, а в квадратных скобках, [], – необязательные элементы синтаксиса. Не надо путать использование квадратных скобок [] в описании синтаксиса и в указании идентификаторов. Фигурные скобки, {}, говорят о том, что пользователь обязан выбрать один из вариантов выражений, перечисленных в скобках через символ «вертикальная черта», |. Традиционно все ключевые слова T-SQL пишутся в верхнем регистре, т.е. прописными буквами, но это правило не строгое, язык T-SQL – регистронезависимый.

Инструкция следующего вида вполне допустима:

```
SELECT 'мама мыла раму'
```

В результате выполнения данной инструкции пользователь получит таблицу (результат выполнения инструкции SELECT всегда является таблицей), состоящую из одной строки и одного столбца, при этом столбец не будет иметь имени.

Следующее изменение инструкции позволит получить таблицу с именованным столбцом:

```
SELECT 'мама мыла раму' AS [просто текст]
```

В данном случае ключевое слово AS говорит о том, что далее будет следовать псевдоним для столбца. Псевдонимы могут быть у столбцов, таблиц и результатов выполнения запросов. Псевдоним, в данном примере [*просто текст*], всегда должен подчиняться правилу идентификаторов (начинаться с буквы, возможно, использовать цифры, все символы без пробелов), если же по каким-то причинам правило идентификаторов нарушается, то необходимо использовать квадратные скобки. Существует общая рекомендация повсеместного использования квадратных скобок, но на практике их опускают ради повышения скорости набора кода.

В качестве источника данных в общем случае выступают таблицы базы данных. Также в качестве источника данных могут выступать отдельные таблицы, результат объединения таблиц, представления, табличные переменные, обобщенные табличные выражения, производные таблицы.

Пример выборки всего содержимого из таблицы:

```
SELECT *

FROM [Production].[Product]
```

Необходимо дать несколько пояснений. Символ «звездочка», *, говорит о том, что из таблицы должны быть выбраны все столбцы. Использовать подобную конструкцию надо с крайней осторожностью, так как она создает значительную нагрузку на сервер и сетевую инфраструктуру. В случае если используется выборка из нескольких таблиц использование «звездочки» может привести к ошибке, так как в результате объединения таблиц может возникнуть ситуация наличия двух и более столбцов с одним именем в результирующем множестве (данный вопрос будет рассмотрен в последующих разделах). Конструкция [Production].[Product] представляет собой полное имя таблицы, где имя таблицы – [Product], а [Production] – это название схемы. Схема – это поименованное логическое объединение таблиц, используемое для упрощения понимания архитектуры базы данных.

Для получения данных из столбца необходимо указать соответствующее имя столбца, например, запрос, возвращающий все названия продуктов из таблиц [Product]:

```
SELECT [Name]

FROM [Production].[Product]
```

Рассмотрим еще один вариант написания запроса:

```
SELECT p.[Name]

FROM [Production].[Product] AS p
```

В данном случае для таблицы Product из схемы Production введен псевдоним p. Разумное использование псевдонимов позволяет увеличить скорость написания кода в MS SQL Management Studio.

Следующая инструкция позволяет получить таблицу с двумя столбцами:

```
SELECT [ProductID], [Name]

FROM [Production].[Product]
```

Рассмотрим запрос:

```
SELECT [ProductID], [Name], [ListPrice]-[StandardCost] AS [discount
size]

FROM [Production].[Product]
```

В этом примере пользователь получает таблицу из трех столбцов, данные первых двух столбцов взяты непосредственно из таблицы [Product], а третий столбец является результатом выполнения операции вычитания данных одного столбца из данных другого столбца, с использованием псевдонима для упрощения понимания результата.

Еще один вариант:

```
SELECT [ProductID], [Name], 'empty column', GETDATE()

FROM [Production].[Product]
```

Пользователь может также получать не только столбцы данных из таблицы, но и столбец с константным значением (третий столбец), а также столбец, содержащий результат выполнения функции. В данном случае в четвертом столбце выводится текущее время сервера, которое возвращает функция GETDATE():

Обратите внимание на формат даты и времени, которые вернул сервер. Формат зависит от локальных настроек сервера и соответствует одному из нескольких десятков поддерживаемых стандартов.

По устаревшим стандартам требовалось использование ключевого слова ALL:

```
SELECT ALL [Size]

FROM [Production].[Product]
```

Ключевое слово ALL говорит о том, что в результат выборки войдут все возможные значения столбца [Size], эта конструкция в текущей версии T-SQL является конструкцией по умолчанию и на практике всегда опускается.

Получение данных без повторений:

```
SELECT DISTINCT [Size]

FROM [Production].[Product]
```

Ключевое слово DISTINCT определяет, что в выборке будут данные из столбца [Size] без повторений.

```
SELECT DISTINCT [Color], [Size]

FROM [Production].[Product]
```

Этот запрос вернет все существующие в таблице пары значений [Color] и [Size] без повторений.

Выражение ORDER BY

Учебные примеры могут дать ложную иллюзию того, что данные, получаемые в результате выполнения инструкции SELECT, находятся в каком-то упорядоченном виде. Рассмотренный ранее запрос, SELECT [Name] FROM [Production].[Product],

действительно возвращает название всех продуктов в упорядоченном по алфавиту виде, но это всего лишь стечение обстоятельств. В общем случае запрос возвращает данные в том порядке, который определен их физическим расположением в файлах БД и операциями, которые выполнила СУБД для их получения. Однако есть инструкция, позволяющая получить упорядоченный набор данных.

Упорядоченный вывод:

```
SELECT [Name], [ListPrice]
FROM [Production].[Product]
ORDER BY [ListPrice] DESC
```

Инструкция ORDER BY определяет порядок вывода строк, в данном случае будут выведены названия продукта и его цена в виде таблицы, упорядоченной по убыванию цены. Ключевое слово DESC определяет порядок упорядочивания по убыванию, ASC – по возрастанию. Если направление упорядочивания не указано явно, то упорядочивание будет произведено по возрастанию. Инструкция ORDER BY всегда является завершающей частью запроса.

Упорядочение по нескольким столбцам, с указанием направления для каждого столбца:

```
SELECT [FirstName], [MiddleName], [LastName]
FROM [Person].[Person]
ORDER BY [FirstName] ASC, [MiddleName] DESC, [LastName] ASC
```

Упорядочение по позиции в выводе, в данном примере по второму столбцу:

```
SELECT [Name], [StandardCost] - [ListPrice]
FROM [Production].[Product]
ORDER BY 2
```

Также возможно использование функции для получения параметра упорядочивания:

```
SELECT BusINessEntityID, JobTitle, HireDate
FROM HumanResources.Employee
ORDER BY DATEPART(year, HireDate)
```

Рассмотрим следующий пример:

```
SELECT TOP 3 [Name], [ListPrice]
FROM [Production].[Product]
ORDER BY [ListPrice] DESC
```

Инструкция TOP ограничивает количество строк, которые получит пользователь при выполнении запроса, в примере соответственно будут выведены три продукта и их цены в отсортированном наборе. Необходимо сделать замечание:

возможно, существует некоторый продукт, условно говоря, четвертый, цена которого равна цене третьего продукта, но он не попадет в выборку для пользователя. К сожалению, мы не можем заранее определить, какой из продуктов, третий или четвертый, с равной ценой, в отсортированном списке попадет в пользовательскую выборку при использовании конструкции TOP 3.

Инструкция WITH TIES позволяет выводить все строки, «соперничающие за последнее место»:

```
SELECT TOP 3 WITH TIES [Name], [ListPrice]
FROM [Production].[Product]
ORDER BY [ListPrice] DESC
```

Приведенный пример использования инструкции WITH TIES позволяет вывести на экран не только первые три продукта из набора, отсортированного по убыванию цены, но и все продукты, цена которых равна цене третьего продукта. Данная инструкция работает исключительно с SELECT и только при использовании оператора упорядоченного вывода ORDER BY.

Вывод доли строк из результирующего набора:

```
SELECT TOP 3 PERCENT [Name], [ListPrice]
FROM [Production].[Product]
ORDER BY [ListPrice] DESC
```

В случае использования конструкции TOP *число* PERCENT пользователь получает некоторый процент строк, равный *числу* из результирующего набора. Процент строк округляется до следующего целого.

Использование инструкции INTO позволяет создать новую таблицу и поместить в нее результат выборки:

```
SELECT TOP 3 PERCENT [Name], [ListPrice] INTO Tmp
FROM [Production].[Product]
ORDER BY [ListPrice] DESC
```

Выполнение данного запроса будет разбито на два этапа. На первом этапе будет создана таблица Tmp. На втором этапе в нее будет помещен результат выполнения запроса. Необходимо добавить несколько замечаний. Если по каким-то причинам при выполнении запроса произойдет сбой, то будет создана пустая таблица. Использование инструкции упорядоченного выводы выборки, ORDER BY, не гарантирует сохранения порядка строк в созданной таблице. Если результирующий набор дает набор строк, не обладающий свойством уникальности, то все равно таблица будет создана, данные будут добавлены, но никакие первичные ключи пользователю не будут доступны, хотя на физическом уровне они будут реализованы СУБД. Для использования инструкции INTO необходимо иметь права, аналогичные правам создания обычной таблицы.

Выражение WHERE

Инструкция SELECT не позволяет выбрать отдельные строки из источника, но можно выбрать строки, удовлетворяющие определенным условиям – наличию тех или иных значений в столбцах.

Простой пример использования предложения WHERE:

```
SELECT [Name], [ListPrice]
FROM [Production].[Product]
WHERE [ListPrice]=3578.27
```

В данном случае будут выведены названия продуктов, у которых цена равна 3578.27, а также их цена. Так как столбец [ListPrice] имеет тип mONey, то использование оператора сравнения на равенство допустимо.

В следующем примере ограничения затрагивают два столбца:

```
SELECT [Name], [ListPrice]
FROM [Production].[Product]
WHERE [ListPrice]=3578.27 AND [Size]=62
```

Пользователь получит название таких товаров, у которых цена равна 3578.27 и размер равен 62, и их цену, для этого был использован оператор AND – логическое И. Существует еще два логических оператора, OR – логическое ИЛИ и NOT – логическое отрицание.

Можно написать достаточно сложный запрос:

```
SELECT [Name] FROM [Production].[Product]
WHERE
([ListPrice]>20 AND [Color]='Red')
OR
([ListPrice]>25 AND [Color]='Black')
```

В данном случае пользователь получит название таких товаров, у которых цена выше 20 и цвет Red, или цена выше 25 и цвет Black.

При работе с числовыми данными бывают ситуации, когда необходимо использовать некоторый диапазон данных. Например, стоит вопрос – найти названия товаров, цена которых лежит в диапазоне от 20 до 40, включая границы диапазона. Реализовать этот запрос можно с помощью логического оператора AND.

```
SELECT [Name] FROM [Production].[Product]
WHERE [ListPrice]>=20 AND [ListPrice]<=40
```

Однако существует специальный оператор BETWEEN, который определяет диапазон для проверки. Общий синтаксис этого оператора выглядит так:

```
выражение [ NOT ] BETWEEN начало_диапазона AND конец_диапазона
```

Оператор возвращает значение TRUE, если *выражение* входит, или не входит, если используется оператор NOT, в диапазон от *начало_диапазона* до *конец_диапазона* включительно.

С помощью этого оператора предыдущий запрос можно реализовать следующим образом:

```
SELECT [Name] FROM [Production].[Product]
WHERE [ListPrice] BETWEEN 20 AND 40
```

Если стоит задача сравнения выражения с некоторым набором значений, то могут применяться следующие операторы – IN, ANY (или его синоним – SOME), ALL.

Оператор IN определяет принадлежность значения одному и значений в списке.

```
SELECT [Name] FROM [Production].[Product]
WHERE [Color] IN ('Red','Black','Silver')
```

Приведенный выше запрос возвращает название товаров, у которых цвет либо Red, либо Black, либо Silver. Стоит обратить внимание, что в скобках может быть не просто набор значений, но и выражение, которое формирует такой набор, например, оператор SELECT.

Операторы ALL и ANY будут рассмотрены позже, в работе с подзапросами.

При работе со строковыми значениями возникает необходимость проверки на соответствие значения не строке, а шаблону, например, требуется найти все продукты, начинающиеся на букву ‘D’. Для этого используется оператор LIKE:

```
SELECT [Name] FROM [Production].[Product]
WHERE [Name] LIKE 'D%'
```

При использовании оператора LIKE можно использовать оператор NOT. Необходимо помнить, что что существуют параметры сравнения, определяемые настройкой СУБД и отдельных таблиц, но в общем случае СУБД не делает различия между строчными и прописными буквами.

В строковых шаблонах допускаются следующие символы:

% – символ-шаблон, заменяющий любую последовательность символов;

_ (подчеркивание) – символ-шаблон, заменяющий любой одиночный символ;

[] – заменяет одиночный символ, указанный в угловых скобках, можно перечислить символы, или диапазон (через дефис) символов;

[^] – заменяет одиночный символ, не указанный в угловых скобках, можно перечислить символы, или диапазон (через дефис) символов.

Допускается использование ESCAPE последовательностей:

```
WHERE СТРОКА LIKE '%[a-f][^xyz]_30!%%' ESCAPE '!'
```

Данный пример является частью запроса, который, в частности, проверяет, совпадают ли значения в столбце СТРОКА со следующим шаблоном – любое количество символов, далее один из символов диапазона от ‘a’ до ‘f’ включительно, далее любой символ, кроме символов ‘x’, ‘y’ или ‘z’, далее еще один любой символ, далее 30%, и опять последовательность любых символов. Символ ‘!’ является эскейп-символом и говорит о том, что следующий за ним символ, в данном случае ‘%’, не надо рассматривать как управляющий; таким образом, в пример включен символ %, который в общем случае является служебным.

В реляционных базах данных существует особое значение – NULL. Это значение не является нулевым значением в математическом понимании нуля, это неопределённость, которая сообщает пользователям, что в данный момент времени значение атрибута не определено. Для работы с этим значением нельзя использовать оператор сравнения на равенство или любой другой оператор сравнения. Проверка на определенность осуществляется с помощью оператора IS.

```
SELECT [Name] FROM [Production].[Product]

WHERE [Color] IS NULL
```

Данный запрос возвращает название товаров, для которых цвет не определен.

```
SELECT [Name] FROM [Production].[Product]

WHERE [Size] IS NOT NULL
```

Этот запрос возвращает название товаров, у которых определен размер, т.е. он не является NULL.

Примеры запросов с решениями

Описание функций и архитектура приведены в приложении.

1. Получить все названия товаров из таблицы Product.

```
SELECT p.Name
FROM [Production].[Product] AS p
```

В данном случае для таблицы Product из схемы Production введен псевдоним p. Разумное использование псевдонимов позволяет увеличить скорость написания кода в MS SQL Management Studio.

2. Получить все названия товаров в системе, цена которых (listprice) выше 200.

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE p.ListPrice>200
```

3. Получить все названия товаров в системе цена которых (listprice) выше 200 и у которых первая буква в названии “S”.

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE p.ListPrice>200 AND p.Name like 's%'
```

4. Получить все названия товаров в системе, цена которых (listprice) выше 200 и у которых в названии есть сочетание символов “are”.

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE p.ListPrice>200 AND p.Name like '%are%'
```

5. Получить все названия товаров в системе, в названии которых третий символ – либо буква “s”, либо буква “r”. Решить задачу как минимум двумя способами.

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE p.Name like '__s%' or p.Name like '__r%'
```

или

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE p.Name like '__[sr]%'
```

7. Получить все названия товаров в системе, в названии которых ровно 5 символов.

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE p.Name like '_____'
```

Или второй вариант

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE len(p.Name)=5
```

Во втором примере используется строковая функция len. Строковые функции даны в приложении.

8. Найти, без повторений, номера товаров, которые были куплены хотя бы один раз.

```
SELECT DISTINCT sod.ProductID
FROM [Sales].[SalesORDERDetail] AS sod
```

9. Найти список всех возможных стилей (style) продукта, без повторений.

```
SELECT DISTINCT p.Style
FROM [Production].[Product] AS p
WHERE p.Style is NOT null
```

10.Написать запрос, который возвращает названия товаров, которые были произведены между мартом 2011 года и мартом 2012 года включительно (необходимо учитывать формат даты)

```
SELECT  p.Name
FROM [Production].[Product] AS p
WHERE p.SellStartDate>='2011-01-03'
AND p.SellStartDate<='2012-31-03'
```

11. Найти максимальную стоимость товара (отпускная цена ListPrice) из тех, которые были произведены, начиная с марта 2011 года.

```
SELECT  max(p.ListPrice) AS [max price]
FROM [Production].[Product] AS p
WHERE p.SellStartDate>='2011-01-03'
```

12. Вывести название и цвет продукта, отсортированные по названию продукта по возрастанию.

```
SELECT  p.Name, p.Color
FROM [Production].[Product] AS p
ORDER BY p.Name ASc
```

13. Вывести названия продукта, цвет и отпускную цену для таких товаров, у которых цвет определен и цена отлична от нуля, и отсортировать полученный список по возрастанию цвета товара и убыванию отпускной цены.

```
SELECT  p.Name, p.Color, p.ListPrice
FROM [Production].[Product] AS p
WHERE p.Color is NOT null AND p.ListPrice!=0
ORDER BY p.Color,  p.ListPrice desc
```

14. Получить название продукта и разницу между отпускной стандартной ценой продукта и стандартной ценой продукта для тех товаров, у которых эти показатели не равны нулю.

```
SELECT p.Name, p.ListPrice-p.StandardCost
FROM [Production].[Product] AS p
WHERE p.ListPrice!=0 AND p.StandardCost!=0
```

15. Найти название самого дорогого товара, исходя из предположения, что нет двух товаров с одинаковой ценой.

```
SELECT top 1 with ties p.Name
FROM [Production].[Product] AS p
ORDER BY p.ListPrice desc
```

16. Найти список товаров, произведенных в 2005 году.

```
SELECT p.Name
FROM [Production].[Product] AS p
WHERE datepart(YEAR,p.SellStartDate)=2005
```

Задания для самостоятельной работы

1. Найти и вывести на экран названия продуктов, их цвет и размер.

2. Найти и вывести на экран названия, цвет и размер таких продуктов, у которых цена более 100.

3. Найти и вывести на экран название, цвет и размер таких продуктов, у которых цена менее 100 и цвет Black.

4. Найти и вывести на экран название, цвет и размер таких продуктов, у которых цена менее 100 и цвет Black, упорядочив вывод по возрастанию стоимости продуктов.

5. Найти и вывести на экран название и размер первых трех самых дорогих товаров с цветом Black.

6. Найти и вывести на экран название и цвет таких продуктов, для которых определен и цвет, и размер.

7. Найти и вывести на экран не повторяющиеся цвета продуктов, у которых цена находится в диапазоне от 10 до 50 включительно.

8. Найти и вывести на экран все цвета таких продуктов, у которых в имени первая буква ‘L’ и третья ‘N’.

9. Найти и вывести на экран названия таких продуктов, которых начинаются либо на букву ‘D’, либо на букву ‘M’, и при этом длина имени – более трех символов.

10. Вывести на экран названия продуктов, у которых дата начала продаж – не позднее 2012 года.

11. Найти и вывести на экран названия всех подкатегорий товаров.

12. Найти и вывести на экран названия всех категорий товаров.

13. Найти и вывести на экран имена всех клиентов из таблицы Person, у которых обращение (Title) указано как «Mr.».

14. Найти и вывести на экран имена всех клиентов из таблицы Person, для которых не определено обращение (Title).

Практическая работа 2

Цель работы: Изучение выражения GROUP BY – группировка данных

Одной из ключевых задач СУБД является выполнение операций, связанных с обработкой данных. Это необходимо для анализа информации, а также для формирования сложных запросов. Выражение GROUP BY позволяет разделить результаты выполнения оператора SELECT на группы по признаку, с целью выполнения статистических операций над группами.

Простой запрос:

```
SELECT [Name], [ListPrice], [Color] FROM [Production].[Product]
WHERE [Color] IS NOT NULL
```

В результате выполнения этого запроса пользователь получит название, стоимость и цвет таких товаров, у которых цвет определен. Это полезная и важная информация, однако у пользователя может появиться более сложная задача, например, посчитать количество товаров того или иного цвета.

Существует набор функций, которые позволяют выполнить операцию над набором значений в столбце. Такие функции называются агрегирующими:

SUM ([ALL | DISTINCT] expressiON) – сумма значений в столбце;

MAX([ALL | DISTINCT] expressiON) – максимальное значение в столбце;

MIN ([ALL | DISTINCT] expressiON) – минимальное значение в столбце;

AVG ([ALL | DISTINCT] expressiON) – среднее значение в столбце;

COUNT ({ [[ALL | DISTINCT] expressiON] | * }) – количество строк в столбце, с учетом NULL значений.

Приведенный общий синтаксис функций рассмотрим на примере:

```
SELECT COUNT(DISTINCT [Color]) FROM [Production].[Product]
WHERE [Color] IS NOT NULL
```

В данном случае будет выполнен SELECT, который отберет из таблицы все строки, для которых цвет определен, и будет выполнена агрегирующая функция COUNT, которая посчитает количество различных цветов. Инструкция DISTINCT обеспечивает выбор неповторяющихся значений цвета, а функция COUNT считает количество полученных значений.

Запрос следующего вида фактически возвращает количество строк, для которых определен цвет продукта:

```
SELECT COUNT([Color]) FROM [Production].[Product]
WHERE [Color] IS NOT NULL
```

Он полностью эквивалентен следующему запросу:

```
SELECT COUNT(*) FROM [Production].[Product]
WHERE [Color] IS NOT NULL
```

Рассмотрим работу функции MAX:

```
SELECT MAX([ListPrice]) FROM [Production].[Product]
WHERE [Color]='Red'
```

Этот запрос возвращает максимальную цену продукта, у которого цвет ‘Red’.

Выражение GROUP BY позволяет разделить результат выполнения запроса на группы и выполнить ту или иную функцию над каждой из групп.

```
SELECT [Color], COUNT(*) AS 'Amount'
FROM [Production].[Product]
WHERE [Color] IS NOT NULL
GROUP BY [Color]
```

В данном запросе все строки, для которых выполнено условие «цвет определен», разделены на группы по цвету, и над каждой группой выполнено функция COUNT. Результатом станет два столбца – цвет и количество строк с данным цветом в общей выборке.

Необходимо сделать несколько важных дополнений. При группировке запрос возвращает только те данные, которые имеют одно и то же значение для всей группы. Запрос вида

```
SELECT [Name], [Color], COUNT(*) AS 'Amount'
FROM [Production].[Product]
WHERE [Color] IS NOT NULL
GROUP BY [Color]
```

не будет выполнен, так как в группе может существовать несколько товаров с разными названиями. Даже если фактически в каждой группе с одним и тем же цветом все товары имеют одно и то же название, все равно это будет ошибкой. Так, при группировке нельзя использовать псевдонимы столбцов как признак группировки, нельзя группировать по столбцам с типами данных text, ntext или image, но можно группировать по результату выполнения функции над этими столбцами, например функции приведения типа или строковой функции. Нельзя использовать в качестве признака группировки подзапросы и данные типа XML. Нельзя использовать в качестве признака группировки столбец индексированного представления.

Если столбец, по которому осуществляется группировка, содержит значения NULL, то они будут рассмотрены как идентичные и будут образовывать отдельную группу.

Допускается группировка по результату выполнения простых математических операций, а также группировка по двум и более столбцам:

```
SELECT Столбец1, Столбец2 FROM Таблица GROUP BY Столбец1, Столбец2;

SELECT Столбец1 + Столбец2 FROM Таблица GROUP BY Столбец1, Столбец2;

SELECT Столбец1 + Столбец2 FROM Таблица GROUP BY Столбец1 + Столбец2;

SELECT Столбец1 + Столбец2 + константное_значение FROM Таблица GROUP BY Столбец1, Столбец2.
```

Рассмотрим следующий пример:

```
SELECT [Color], [Style], COUNT(*) AS 'Amount'

FROM [Production].[Product]

WHERE [Color] IS NOT NULL AND [Style] IS NOT NULL

GROUP BY [Color], [Style]
```

Запрос возвращает количество продуктов с общим цветом и стилем, упомянутых в таблице продукты», с учетом того, что у этих товаров и цвет, и стиль определены.

Выражение GROUP BY используется совместно с выражением HAVING, которое определяет условие поиска группы.

```
SELECT [Color], COUNT(*) AS 'Amount'

FROM [Production].[Product]

WHERE [Color] IS NOT NULL

GROUP BY [Color]

HAVING COUNT(*)>10
```

Данный запрос выдаст название цветов, количество товаров данного цвета, но только для тех групп, где это количество больше 10. Естественно, будет выполнено условие, что цвет товаров определен. Условие поиска группы может содержать также логические операторы. Вот пример такой возможности:

```
SELECT [Color], COUNT(*) AS 'Amount'

FROM [Production].[Product]

WHERE [Color] IS NOT NULL

GROUP BY [Color]

HAVING COUNT(*)>10 AND MAX([ListPrice])>3000
```

Стоит отметить, что операция группировки весьма затратна с точки зрения процессорной мощности и времени, неправильная конструкция может существенно увеличить время выполнения запроса. Например, стоит задача, рассмотренная в начале работы: посчитать количество товаров каждого цвета,

исключив товары, для которых цвет не определен. Выше эта задача была решена, однако формально допустимо следующее решение:

```
SELECT [Color], COUNT(*) AS 'Amount'
FROM [Production].[Product]
GROUP BY [Color]
HAVING [Color] IS NOT NULL
```

Такой запрос будет выполняться дольше, так как сначала сформируются все группы, и только потом будет применено условие для выбора группы, т.е. будет отброшена группа, для которой цвет не определён. Время на формирование группы с неопределенным цветом фактически будет потрачено впустую.

Использование GROUP BY допускает использование ORDER BY, TOP.

Существуют специальные конструкции, выполняющие группировку с определёнными правилами. Рассмотрим конструкцию GROUP BY ROLLUP, которая выводить и общие и промежуточные итоги группировки. Частный синтаксис будет такой:

```
SELECT столбец1, столбец2, столбец3, COUNT(*)
FROM таблица
GROUP BY ROLLUP (столбец1, столбец2, столбец3)
```

В данном случае будут выведены итоги общей группировки по трем столбцам, а также добавлены результаты промежуточной группировки. В данном примере промежуточная группировка это строки полученные в результате группировки по первым двум столбцам, третий столбец будет иметь значение NULL, это строки полученные в результате группировки по первому столбцу, второй и третий будут иметь значение NULL, и одна строка где первый, второй и третий столбцы будут иметь значение NULL.

Для понимания ее работы нам понадобится простой пример группировки.

```
SELECT [Color], [Style], [Class], COUNT(*)
FROM [Production].[Product]
GROUP BY [Color], [Style], [Class]
```

В данном случае пользователь увидит окончательный результат работы группировки по трем столбцам, четвертый столбец это число продуктов, в группе с одинаковым классом, стилем и цветом.

Если выполнить запрос с использованием GROUP BY ROLLUP:

```
SELECT [Color], [Style], [Class], COUNT(*)
FROM [Production].[Product]
GROUP BY ROLLUP ([Color], [Style], [Class])
```

то пользователь получит все те же строки, как и при обычном GROUP BY, а также дополнительный набор с группировкой по цвету и стилю, дополнительный набор с группировкой только по цвету и строку без группировки.

Схожим образом работает операция GROUP BY CUBE ( ), только к базовой группировке GROUP BY добавляются все возможные сочетания параметров группировки. Например, группировка GROUP BY CUBE (столбец1, столбец2), даст все возможные сочетания группировок, базовую группировку (столбец1, столбец2), и возможные сочетания (столбец1,NULL), (NULL,столбец2), (NULL, NULL).

Конструкция GROUP BY GROUPING SETS ( ) позволяет объединить несколько операций GROUP BY.

```
SELECT COUNT(*)
FROM [Production].[Product]
GROUP BY GROUPING SETS (([Color]),([Size]))
```

Данный запрос вернет данные, сгруппированные по цвету, и еще набор строк, где данные сгруппированы по размеру.

Примеры запросов с решениями

1. Найти номера первых трех подкатегорий (ProductSubcategoryID) с наибольшим количеством наименований товаров.

```
SELECT TOP WITH TIES 3 [ProductSubcategoryID]
FROM [Production].[Product]
WHERE [ProductSubcategoryID] IS NOT NULL
GROUP BY [ProductSubcategoryID]
ORDER BY COUNT(*) DESC
```

2. Разбить продукты по количеству символов в названии, для каждой группы определить количество продуктов.

```
SELECT LEN([Name]), COUNT(*)
FROM [Production].[Product]
GROUP BY LEN([Name])
```

3. Проверить, есть ли продукты с одинаковым названием, если есть, то вывести эти названия.

```
SELECT [Name]
FROM [Production].[Product]
GROUP BY [ProductID], [Name]
```

```
HAVING COUNT(*)>1
```

Задания для самостоятельной работы

1. Найти и вывести на экран количество товаров каждого цвета, исключив из поиска товары, цена которых меньше 30.

2. Найти и вывести на экран список, состоящий из цветов товаров, таких, что минимальная цена товара данного цвета более 100.

3. Найти и вывести на экран номера подкатегорий товаров и количество товаров в каждой категории.

4. Найти и вывести на экран номера товаров и количество фактов продаж данного товара (используется таблица SalesORDERDetail).

5. Найти и вывести на экран номера товаров, которые были куплены более пяти раз.

6. Найти и вывести на экран номера покупателей, CustomerID, у которых существует более одного чека, SalesORDERID, с одинаковой датой

7. Найти и вывести на экран все номера чеков, на которые приходится более трех продуктов.

8. Найти и вывести на экран все номера продуктов, которые были куплены более трех раз.

9. Найти и вывести на экран все номера продуктов, которые были куплены или три или пять раз.

10. Найти и вывести на экран все номера подкатегорий, в которым относится более десяти товаров.

11. Найти и вывести на экран номера товаров, которые всегда покупались в одном экземпляре за одну покупку.

12 Найти и вывести на экран номер чека, SalesORDERID, на который приходится с наибольшим разнообразием товаров купленных на этот чек.

13 Найти и вывести на экран номер чека, SalesORDERID с наибольшей суммой покупки, исходя из того, что цена товара – это UnitPrice, а количество конкретного товара в чеке – это ORDERQty.

14 Определить количество товаров в каждой подкатегории, исключая товары, для которых подкатегория не определена, и товары, у которых не определен цвет.

15 Получить список цветов товаров в порядке убывания количества товаров данного цвета

16 Вывести на экран ProductID тех товаров, что всегда покупались в количестве более 1 единицы на один чек, при этом таких покупок было более двух.

Практическая работа 3

Цель работы: Выборка данных из нескольких таблиц.

Структурированный язык запросов позволяет получать данные из нескольких таблиц одновременно. Существует простой и понятный синтаксис подобных запросов, но для полного понимания необходимо рассмотреть несколько общих вопросов.

Допустим, пользователь имеет две таблицы. Первая таблица будет называться Корма, и в ней один столбец – название корма. Для примера в таблице будет две строки: сено, мясо. Вторая таблица будет называться Животные и также будет иметь один столбец – название животного, в данном случае будет две строки: собака, лошадь.

Вполне допустим следующий запрос:

```
SELECT [название корма], [название животного] FROM Корма, Животные
```

Однако полученный результат вряд ли устроит пользователя:

сено собака

сено лошадь

мясо собака

мясо лошадь

Чтобы результат соответствовал реальному миру, СУБД должно иметь возможность исключить не существующие в реальном мире варианты сочетания корма и животного. Модифицируем наши таблицы. Добавим в таблицу Корма столбец «номер корма», это будет первичный ключ для этой таблицы. В таблицу Животные необходимо будет добавить два столбца. Столбец «номер животного» станет первичным ключом этой таблицы, а еще один столбец, «номер корма», станет внешним ключом, связанным с первичным ключом таблицы Корма. Естественно, значения в столбце «номер корма» таблицы Животные должен содержать значения, которые удовлетворяют двум требованиям. Во-первых, есть точно такие же значения в столбце «номер корма» таблицы Корма. Во-вторых, эти значения соответствуют реальному миру, т.е. в строке, где содержится информация о лошади, должен стоять число, равное номеру сена, а в строке, где упомянута собака, номер ее корма должен соответствовать номеру мяса.

Используемые определения: суперключ, потенциальный ключ, первичный и внешний ключ.

Суперключ – атрибут или множество атрибутов, единственным образом идентифицирующие кортеж.

Потенциальный ключ – суперключ, который не содержит подмножества, также являющегося суперключом данного отношения. Отношение может иметь

несколько потенциальных ключей. Если потенциальный ключ состоит из нескольких атрибутов, он называется составным ключом.

Первичный ключ – потенциальный ключ, который выбран для уникальной идентификации кортежей внутри отношения.

Внешний ключ – атрибут или множество атрибутов внутри отношения, которое соответствует потенциальному ключу некоторого (может быть, того же самого) отношения. Внешние ключи позволяют описать связь между отношениями.

После добавления столбцов и внесения необходимых данных можно выполнить запрос следующего вида:

```
SELECT [название корма], [название животного] FROM Корма, Животные
WHERE Корма.[Номер корма]=Животные.[Номер корма]
```

Он даст результат, который соответствует реальному миру:

сено лошадь

мясо собака

В данном случае СУБД получила однозначную инструкцию – не включать в результирующий набор данных, полученный из двух таблиц, те строки, для которых не выполняется равенство внешнего и первичного ключей.

Такой способ соединения таблиц допустим. Пользователь может производить соединения двух и более таблиц, используя сравнения на равенство или неравенство значений столбца одной таблицы со значениями столбца другой таблицы. Надо отметить, что подобные соединения не подразумевают, что столбцы обязательно являются первичным и внешним ключом соответственно.

У подобного соединения есть один существенный недостаток. Операция сравнения на равенство не применима к значениям NULL. Допустим, у нас появилось информация о животном, например, шимпанзе, для которого корм не определен. В итоговую выборку это животное не попадет. Для разрешения подобной трудности используется операция соединения JOIN. Общий синтаксис, стандартный для всех реляционных СУБД выглядит так:

```
FROM первая_таблица ТИП_СОЕДИНЕНИЯ вторая_таблица [ON
(условия_соединения)]
```

Порядок следования таблиц при соединении играет важную роль, от этого зависит результат при использовании некоторых типов соединений. Иногда таблицы называют левая и правая таблицы.

Тип соединения определяет принцип соединения таблиц. *Условия соединения* определяют условия сравнения значений в столбцах соединяемых таблиц – равенство или неравенство.

Рассмотрим все типы соединения.

Соединение INNER JOIN.

Рассмотрим запрос следующего вида:

```
SELECT p.Name, s.Name

FROM [Production].[Product] p INNER JOIN [Production].[ProductSubcategory] s

ON p.ProductSubcategoryID=s.ProductSubcategoryID
```

Пользователь получит список названий товаров и названий категорий, к которым относится товар, при этом не будут рассмотрены товары, для которых подкатегория не определена. INNER JOIN не учитывает NULL значения как в левой, так и в правой таблице при объединении. Синтаксически допустимо и традиционно используется сокращение названия соединения INNER JOIN до просто JOIN.

В рассмотренном выше запросе объединяются таблицы Product и ProductSubcategory, каждая из которых имеет столбец с именем Name. Чтобы различать эти столбцы при объединении, были введены псевдонимы таблиц, и к столбцу Name обращение идет через оператор доступ – точка.

Соединение LEFT JOIN.

LEFT JOIN учитывает значения NULL из левой таблицы. Таким образом нижеприведенный запрос вернет названия продуктов и название подкатегорий, в том числе и те случаи, когда у продукта не определена подкатегория. Если подкатегория продукта не определена, то ее название будет NULL.

```
SELECT p.Name, s.Name

FROM [Production].[Product] p LEFT JOIN [Production].[ProductSubcategory] s

ON p.ProductSubcategoryID=s.ProductSubcategoryID
```

Соединение RIGHT JOIN учитывает значения NULL из правой таблицы.

Рассмотрим запрос:

```
SELECT p.Name, s.Name

FROM [Production].[Product] p RIGHT JOIN [Production].[ProductSubcategory] s

ON p.ProductSubcategoryID=s.ProductSubcategoryID
```

В данном случае соединение RIGHT JOIN учитывает ситуацию, в столбце правой таблицы, ProductSubcategory, есть значения NULL. Это правило формально соблюдено СУБД, несмотря на то, что в правой таблице, ProductSubcategory, столбец для соединения, ProductSubcategoryID, является первичным ключом.

Соединение FULL OUTER JOIN учитывает значение NULL и в левой, и в правой таблице. На практике синтаксическая единица FULL OUTER JOIN сокращается до FULL JOIN.

Рассмотрим пример. Даны две таблицы, tA и tB, со столбцами [tAvalue] и [tBvalue]. Столбец [tAvalue] имеет следующие значения: 1, 2, 3 и NULL. Столбец [tBvalue] имеет значения: 2, 3, 4, и NULL.

Выполнив запрос

```
SELECT [tAvalue], [tBvalue] FROM [dbo].[tA] FULL JOIN [dbo].[tB] ON tA.[tAvalue]=tB.[tBvalue]
```

пользователь получит следующие пары значений: (1, NULL), (2,3), (3,3), (NULL, NULL), (NULL,4), (NULL, NULL).

В данном случае был рассмотрен пример соединения, в котором условие соединения не связано с первичным или внешним ключом. Также надо учитывать, что запрос вернул две строки со значением NULL в каждом из столбцов.

Соединение CROSS JOIN производит полное декартово произведение двух таблиц и не имеет условия соединения. Рассмотрим запрос к таблицам из предыдущего примера

```
SELECT [tAvalue], [tBvalue] FROM
[dbo].[tA] CROSS JOIN [dbo].[tB]
```

Этот запрос вернет все возможные сочетания значений [tAvalue] и [tBvalue], т.е. шестнадцать строк. Соединение CROSS JOIN может потребовать значительных ресурсов СУБД и серьезно затруднить работу других пользователей.

Соединение таблиц допускает соединение таблицы со своей копией. Стоит задача – найти такие продукты, для которых существуют продукты с таким же названием. Задачу можно решить с использованием выражения GROUP BY, а можно и с использованием самосоединения.

```
SELECT p1.Name
FROM [Production].[Product] p1 JOIN
[Production].[Product] p2
ON p1.Name=p2.Name AND
p1.ProductID!=p2.ProductID
```

Стоит обратить внимание что в данном запросе обязательно использование псевдонимов. Для соединения использованы два условия, одно из которых – на неравенство.

Соединение таблиц допускает использование ключевых слов WHERE, GROUP BY, HAVING, ORDER BY и других. Допускается соединение более чем двух таблиц.

В нижеприведенных примерах используется схема данных из первой лабораторной работы, а также схема, приведенная в приложении к данной работе.

Примеры запросов с решениями

1 Найти название продуктов и название подкатегорий этих продуктов, у которых отпускная цена больше 100, не включая случаи, когда продукт не относится ни к какой подкатегории.

```
SELECT P.Name, PSC.Name
FROM [Production].[Product] AS P INNER JOIN
[Production].[ProductSubcategory] AS PSC
ON P.ProductSubcategoryID=PSC.ProductSubcategoryID
WHERE [ListPrice]>100
```

2 Найти название продуктов и название подкатегорий этих продуктов, у которых отпускная цена больше 100, включая случаи, когда продукт не относится ни к какой категории.

```
SELECT P.Name, PSC.Name
FROM [Production].[Product] AS P LEFT JOIN
[Production].[ProductSubcategory] AS PSC
ON P.ProductSubcategoryID=PSC.ProductSubcategoryID
WHERE [ListPrice]>100
```

3 Найти название продуктов и название категорий из таблицы ProductCategory, с которой связан этот продукт, не включая случаи, когда у продукта нет подкатегории.

```
SELECT P.Name, PC.Name
FROM [Production].[Product] AS P INNER JOIN
[Production].[ProductSubcategory] AS PSC
ON P.ProductSubcategoryID=PSC.ProductSubcategoryID
INNER JOIN [Production].[ProductCategory] AS PC
ON PSC.ProductCategoryID=PC.ProductCategoryID
```

4 Найти название продукта, отпускную цену продукта, а также последнюю отпускную цену этого продукта (LAStReceiptCost), которую можно узнать из таблицы ProductVendor.

```
SELECT P.Name, P.ListPrice, PV.LAStReceiptCost
FROM [Production].[Product] AS P INNER JOIN
[PurchASINg].[ProductVendor] AS PV
```

```
ON P.ProductID=PV.ProductID
```

5 Найти название продукта, отпускную цену продукта, а также последнюю отпускную цену этого продукта (LAStReceiptCost), которую можно узнать из таблицы ProductVendor, для таких продуктов, у которых отпускная цена оказалась ниже последней отпускной цены у поставщика, исключив те товары, для которых отпускная цена равна нулю.

```
SELECT P.Name, P.ListPrice, PV.LAStReceiptCost
FROM [Production].[Product] AS P INNER JOIN
[PurchASINg].[ProductVendor] AS PV
ON P.ProductID=PV.ProductID
WHERE P.ListPrice!=0 AND P.ListPrice<PV.LAStReceiptCost
```

6 Найти количество товаров, которые поставляют поставщики с самым низким кредитным рейтингом (CreditRatINg принимает целые значение от минимального, равного 1, до максимального, равного 5).

```
SELECT COUNT(DISTINCT PV.ProductID)
FROM [PurchASINg].[ProductVendor] AS PV INNER JOIN
[PurchASINg].[Vendor] AS V
ON PV.BusINessEntityID=V.BusINessEntityID
WHERE [CreditRatINg]=1
```

7 Найти, сколько товаров приходится на каждый кредитный рейтинг, т.е. сформировать таблицу, первая колонка которой будет содержать номер кредитного рейтинга, вторая – количество товаров, поставляемых всеми поставщиками, имеющими соответствующий кредитный рейтинг. Необходимо сформировать универсальный запрос, который будет валидным и в случае появления новых значений кредитного рейтинга.

```
SELECT [CreditRatINg], COUNT(DISTINCT PV.ProductID)
FROM [PurchASINg].[ProductVendor] AS PV INNER JOIN
[PurchASINg].[Vendor] AS V
ON PV.BusINessEntityID=V.BusINessEntityID
GROUP BY [CreditRatINg]
```

8 Найти номера первых трех подкатегорий (ProductSubcategoryID) с наибольшим количеством наименований товаров.

```
SELECT TOP 3 [ProductSubcategoryID]
FROM [Production].[Product]
WHERE [ProductSubcategoryID] IS NOT NULL
GROUP BY [ProductSubcategoryID]
```

```
ORDER BY COUNT(*) DESC
```

9 Получить названия первых трех подкатегорий с наибольшим количеством наименований товаров.

```
SELECT TOP 3 PC.ProductCategoryID
FROM [Production].[Product] AS P INNER JOIN
[Production].[ProductSubcategory] AS PSC
ON P.ProductSubcategoryID=PSC.ProductSubcategoryID
INNER JOIN [Production].[ProductCategory] AS PC
ON PSC.ProductCategoryID=PC.ProductCategoryID
GROUP BY PC.ProductCategoryID
ORDER BY COUNT(*) DESC
SELECT top 3 psc.name, count(*)
FROM [Production].[Product] AS p INner JOIN
[Production].[ProductSubcategory] AS psc
ON p.ProductSubcategoryID=psc.ProductSubcategoryID
WHERE p.ProductSubcategoryID is NOT null
GROUP BY p.ProductSubcategoryID, psc.Name
ORDER BY count(*) desc
```

10 Высчитать среднее количество товаров, приходящихся на одну подкатегорию, с точностью минимум до одной десятой.

Вариант 1

```
SELECT 1.0*COUNT(*)/COUNT(DISTINCT [ProductSubcategoryID])
FROM [Production].[Product]
WHERE [ProductSubcategoryID] IS NOT NULL
```

Вариант 2

```
SELECT CAST(COUNT(*) AS FLOAT)/COUNT(DISTINCT
[ProductSubcategoryID])
FROM [Production].[Product]
WHERE [ProductSubcategoryID] IS NOT NULL
```

Вариант 3

```
SELECT cASt((cASt(count(p.ProductID) AS float)/count(DISTINCT
p.ProductSubcategoryID))
AS decimal(6,3))
FROM [Production].[Product] AS p
```

```
WHERE p.ProductSubcategoryID is NOT null
```

11 Вычислить среднее количество товаров, приходящихся на одну категорию, в целых числах.

```
SELECT COUNT(*)/COUNT(DISTINCT PC.ProductCategoryID)
FROM [Production].[Product] AS P INNER JOIN
[Production].[ProductSubcategory] AS PSC
ON P.ProductSubcategoryID=PSC.ProductSubcategoryID
RIGHT JOIN [Production].[ProductCategory] AS PC
ON PSC.ProductCategoryID=PC.ProductCategoryID
```

12 Найти количество цветов товаров, приходящихся на каждую категорию, без учета товаров, для которых цвет не определен.

```
SELECT COUNT(DISTINCT [Color])
FROM [Production].[Product] AS P INNER JOIN
[Production].[ProductSubcategory] AS PSC
ON P.ProductSubcategoryID=PSC.ProductSubcategoryID
RIGHT JOIN [Production].[ProductCategory] AS PC
ON PSC.ProductCategoryID=PC.ProductCategoryID
GROUP BY PC.ProductCategoryID
```

13 Найти средний вес продуктов. Просмотреть таблицу продуктов и убедиться, что есть продукты, для которых вес не определен. Модифицировать запрос так, чтобы при нахождении среднего веса продуктов те продукты, для которых вес не определен, считались как продукты с весом 10.

```
SELECT AVG(ISNULL([Weight],10))
FROM [Production].[Product]
```

14 Вывести названия продуктов и период их активных продаж (период между SellStartDate и SellEndDate) в днях, отсортировав по уменьшению времени продаж. Если продажи идут до сих пор и SellEndDate не определен, то считать периодом продаж число дней с начала продаж и по текущие сутки.

```
SELECT [Name], [SellStartDate], [SellEndDate],
DATEDIFF(D,[SellStartDate],ISNULL([SellEndDate],GETDATE()))
FROM [Production].[Product]
WHERE [SellStartDate] IS NOT NULL
```

15 Разбить продукты по количеству символов в названии, и для каждой группы определить количество продуктов.

```
SELECT LEN([Name]), COUNT(*)
```

```
FROM [Production].[Product]
GROUP BY LEN([Name])
```

16 Найти для каждого поставщика количество подкатегорий продуктов, к которым относится продукты, поставляемые им, без учета ситуации, когда продукт не относится ни к какой подкатегории.

```
SELECT PV.BusINessEntityID, COUNT(DISTINCT P.ProductSubcategoryID)
FROM [Production].[Product] AS P INNER JOIN
[PurchASINg].[ProductVendor] AS PV
ON P.ProductID=PV.ProductID
WHERE P.ProductSubcategoryID IS NOT NULL
GROUP BY PV.BusINessEntityID
```

17 Проверить, есть ли продукты с одинаковым названием, если есть, то вывести эти названия.

Вариант 1

```
SELECT P1.Name
FROM [Production].[Product] AS P1,
[Production].[Product] AS P2
WHERE P1.ProductID!=P2.ProductID AND
P1.Name=P2.Name
```

Вариант 2

```
SELECT [Name]
FROM [Production].[Product]
GROUP BY [ProductID], [Name]
HAVING COUNT(*)>1
```

18 Найти первые 10 самых дорогих товаров, с учетом ситуации, когда цена цены у некоторых товаров могут совпадать.

```
SELECT TOP 10 WITH TIES [Name]
FROM [Production].[Product]
ORDER BY [ListPrice] DESC
```

19 Найти первые 10 процентов самых дорогих товаров, с учетом ситуации, когда цены у некоторых товаров могут совпадать.

```
SELECT TOP 10 PERCENT WITH TIES [Name]
FROM [Production].[Product]
ORDER BY [ListPrice] DESC
```

20 Найти первых трех поставщиков, отсортированных по количеству поставляемых товаров, с учетом ситуации, что количество поставляемых товаров может совпадать для разных поставщиков.

```
SELECT TOP 3 WITH TIES PV.BusINessEntityID
FROM [Production].[Product] AS P INNER JOIN
[PurchASINg].[ProductVendor] AS PV
ON P.ProductID=PV.ProductID
GROUP BY PV.BusINessEntityID
ORDER BY COUNT(P.ProductID) DESC
```

Задания для самостоятельной работы

1 Найти и вывести на экран название продуктов и название категорий товаров, к которым относится этот продукт, с учетом того, что в выборку попадут только товары с цветом Red и ценой не менее 100.

2 Вывести на экран названия подкатегорий с совпадающими именами.

3 Вывести на экран название категорий и количество товаров в данной категории.

4 Вывести на экран название подкатегории, а также количество товаров в данной подкатегории с учетом ситуации, что могут существовать подкатегории с одинаковыми именами.

5 Вывести на экран название первых трех подкатегорий с небольшим количеством товаров.

6 Вывести на экран название подкатегории и максимальную цену продукта с цветом Red в этой подкатегории.

7 Вывести на экран название поставщика и количество товаров, которые он поставляет.

8 Вывести на экран название товаров, которые поставляются более чем одним поставщиком.

9 Вывести на экран название самого продаваемого товара.

10 Вывести на экран название категории, товары из которой продаются наиболее активно.

11 Вывести на экран названия категорий, количество подкатегорий и количество товаров в них.

12 Вывести на экран номер кредитного рейтинга и количество товаров, поставляемых компаниями, имеющими этот кредитный рейтинг.

Практическая работа 4

Цель работы: использование подзапросов.

При написании запросов может возникнуть необходимость в данных, полученных на предыдущем этапе. Встает вопрос об актуальности данных. Рассмотрим следующий пример. Необходимо найти название самого дорогого товара с цветом Red. Пользователь, не знакомый с механизмом использования подзапросов, может написать следующий набор запросов.

```
SELECT MAX([ListPrice]) FROM [Production].[Product]
WHERE [Color]='Red’
```

Данный запрос вернет таблицу, одна строка и один столбец без имени, со значением 3578,27. Подставив полученный результат в следующий запрос, пользователь получит искомые данные.

```
SELECT [Name]
FROM [Production].[Product]
WHERE [Color]='Red' AND [ListPrice]=3578.27
```

При выполнении данных операций пользователь, во-первых, вынужден выполнять промежуточные действия вручную, во-вторых, актуальность полученных данных может быть поставлена под сомнение.

Использование подзапросов решит обе эти проблемы.

```
SELECT [Name]
FROM [Production].[Product]
WHERE [Color]='Red'AND [ListPrice]=
(SELECT MAX([ListPrice]) FROM [Production].[Product]
WHERE [Color]='Red')
```

В данном случае показан простейший вариант использования подзапроса. Стоит отметить, что использован оператор сравнения на равенство, так как функция MAX гарантированно возвращает одно значение.

Допускается использование логических операторов для сравнения скалярного значения с результатом выполнения подзапроса. Рассмотрим несколько примеров.

Необходимо получить список товаров, цвет которых может быть любой, кроме Red, а цена равна цене любого товара с цветом Red. Для этого можно использовать запрос следующего вида:

```
SELECT [Name]
FROM [Production].[Product]
WHERE [Color]!='Red'AND [ListPrice] = ANY
```

```
(SELECT [ListPrice] FROM [Production].[Product]
WHERE [Color]='Red')
```

Оператор сравнения использован вместе с логическим оператором ANY, так как подзапрос возвращает потенциально более одного значения. Логический оператор ANY сравнивает скалярное значение с набором значений, состоящим из одного столбца, и условие сравнения должно быть выполнено хотя бы для одного значения из набора.

Необходимо получить список товаров, цена которых больше цены любого из товаров с цветом Red.

```
SELECT [Name]
FROM [Production].[Product]
WHERE [ListPrice] >ALL
(SELECT [ListPrice] FROM [Production].[Product]
WHERE [Color]='Red')
```

Логический оператор ALL сравнивает скалярное значение с набором значений, состоящим из одного столбца, и условие сравнения должно быть выполнено для каждого значения из набора.

Необходимо получить название товаров, чей цвет совпадает с цветом одного из товаров, чья цена больше 3000.

```
SELECT [Name]
FROM [Production].[Product]
WHERE [Color] IN
(SELECT [Color] FROM [Production].[Product]
WHERE [ListPrice]>3000)
```

Логический оператор IN определяет, совпадает ли указанное значение с одним из значений, содержащихся во вложенном запросе или списке.

Подзапросы могут сами содержать подзапросы. Следующий пример находит название категории, где содержится самый дорогой товар.

```
SELECT [Name]
FROM [Production].[ProductCategory]
WHERE [ProductCategoryID] IN
(SELECT [ProductCategoryID]
FROM [Production].[ProductSubcategory]
WHERE [ProductSubcategoryID] IN
(SELECT [ProductSubcategoryID]
```

```
FROM [Production].[Product]
WHERE [ListPrice] =
(SELECT MAX([ListPrice])
FROM [Production].[Product])))
```

Данную задачу можно было бы решить с помощью соединения таблиц, упорядочивания и выражения TOP, но использование подзапросов в данном случае предпочтительней, так как это позволяет избежать ресурсоемкой операции соединения. Это утверждение справедливо только для ситуаций, когда подзапрос простой.

Запрос может использовать более одного подзапроса на одном уровне вложенности. Например, необходимо получить с помощью одного запроса список товаров, у которых цвет совпадает с цветом самого дорогого товара, и стиль совпадает со стилем самого дорого товара.

```
SELECT [Name]
FROM [Production].[Product]
WHERE [Color] IN
(SELECT [Color]
FROM [Production].[Product]
WHERE [ListPrice] =
(SELECT MAX([ListPrice])
FROM [Production].[Product]))
AND
[Style] IN
(SELECT [Style]
FROM [Production].[Product]
WHERE [ListPrice] =
(SELECT MAX([ListPrice])
FROM [Production].[Product]))
```

Все рассмотренные ранее подзапросы являлись простыми, они возвращали фиксированный, неизменный набор данных, вне зависимости от того, с какой из строк запроса верхнего уровня в данный момент работает СУБД.

Подзапросы также используются для формирования выборки с использованием конструкции GROUP BY HAVING. Допустим, необходимо найти номер подкатегории товаров с наибольшим количеством товаров. Данный запрос можно реализовать несколькими способами, в том числе с использованием подзапроса.

```
SELECT [ProductSubcategoryID]
FROM [Production].[Product]
GROUP BY [ProductSubcategoryID]
HAVING COUNT(*)=
(SELECT TOP 1 COUNT(*)
FROM [Production].[Product]
GROUP BY [ProductSubcategoryID]
ORDER BY 1 DESC)
```

Рассмотрим следующую задачу. Необходимо получить список самых дорогих товаров в каждой из подкатегорий. Подобную задачу можно решить с использованием сложного, коррелирующего или связанного, подзапроса. Коррелирующим подзапросом называют такой подзапрос, который формирует связанную выборку, зависимую от данных внешнего запроса. Фактически коррелирующий подзапрос выполняется для каждой строки запроса верхнего уровня.

```
SELECT [Name]
FROM [Production].[Product] AS P1
WHERE [ListPrice]=
(SELECT MAX([ListPrice])
FROM [Production].[Product] AS P2
WHERE P1.ProductSubcategoryID=P2.ProductSubcategoryID)
```

Связанные подзапросы могут использоваться для формирования выводимого столбца. Например, следующий запрос возвращает название продукта и название подкатегории, к которой он относится.

```
SELECT [Name],
(SELECT [Name]
FROM [Production].[ProductSubcategory] AS PS
WHERE P1.ProductSubcategoryID=PS.ProductSubcategoryID)
FROM [Production].[Product] AS P1
```

Примеры запросов с решениями

1 Найти название подкатегории с наибольшим количеством продуктов, без учета продуктов, для которых подкатегория не определена (еще одна возможная реализация).

```
SELECT [Name]
FROM [Production].[ProductSubcategory]
```

```
WHERE [ProductSubcategoryID] IN
(SELECT [ProductSubcategoryID]
FROM [Production].[Product]
WHERE [ProductSubcategoryID] IS NOT NULL
GROUP BY [ProductSubcategoryID]
HAVING COUNT(*)=
(SELECT TOP 1 COUNT(*)
FROM [Production].[Product]
WHERE [ProductSubcategoryID] IS NOT NULL
GROUP BY [ProductSubcategoryID]
ORDER BY 1 DESC
)
)
```

2 Вывести на экран такого покупателя, который каждый раз покупал только одну номенклатуру товаров, не обязательно в одинаковых количествах, т.е. у него всегда был один и тот же «список покупок».

Вариант 1

```
SELECT [CustomerID], count(*)
FROM [Sales].[SalesORDERHeader] AS SOH1
GROUP BY [CustomerID]
HAVING count(*)>1 AND count(*)=all
(SELECT count(*)
FROM [Sales].[SalesORDERHeader] AS SOH INner JOIN
[Sales].[SalesORDERDetail] AS SOD
ON soh.SalesORDERID=sod.SalesORDERID
GROUP BY soh.[CustomerID], sod.ProductID
HAVING soh.CustomerID=soh1.CustomerID)
```

Вариант 2

```
SELECT soh.CustomerID, p.Name
FROM [Sales].[SalesORDERHeader] AS SOH INner JOIN
[Sales].[SalesORDERDetail] AS sod
ON soh.SalesORDERID=sod.SalesORDERID INner JOIN
[Production].[Product] AS p
ON sod.ProductID=p.ProductID
```

```
WHERE soh.CustomerID IN (
SELECT [CustomerID]
FROM [Sales].[SalesORDERHeader] AS SOH1
GROUP BY [CustomerID]
HAVING count(*)>1 AND count(*)=all
(SELECT count(*)
FROM [Sales].[SalesORDERHeader] AS SOH INner JOIN
[Sales].[SalesORDERDetail] AS SOD
ON soh.SalesORDERID=sod.SalesORDERID
GROUP BY soh.[CustomerID], sod.ProductID
HAVING soh.CustomerID=soh1.CustomerID))
ORDER BY 1
```

Вариант 3

```
SELECT soh1.CustomerID
FROM [Sales].[SalesORDERDetail] AS sod1 INner JOIN
[Sales].[SalesORDERHeader] AS soh1
ON sod1.SalesORDERID=soh1.SalesORDERID
GROUP BY soh1.CustomerID
HAVING count(soh1.SalesORDERID)>1 AND
count(DISTINCT sod1.ProductID)>1
 AND count(DISTINCT sod1.ProductID)=all
(
SELECT count(*)
FROM [Sales].[SalesORDERDetail] AS sod2 INner JOIN
[Sales].[SalesORDERHeader] AS soh2
ON sod2.SalesORDERID=soh2.SalesORDERID
GROUP BY soh2.CustomerID, sod2.SalesORDERID
HAVING soh2.CustomerID=soh1.CustomerID
)
```

3 Вывести на экран следующую информацию: название товара (первая колонка), количество покупателей, покупавших этот товар (вторая колонка), количество покупателей, совершавших покупки, но не покупавших товар из первой колонки (третья колонка).

```
SELECT
```

```
p.[ProductID],
(SELECT count(DISTINCT soh.CustomerID)
FROM [Sales].[SalesORDERDetail] AS sod INner JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
WHERE sod.ProductID=p.ProductID),
(SELECT count(DISTINCT soh.CustomerID)
FROM [Sales].[SalesORDERDetail] AS sod INner JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
WHERE soh.CustomerID NOT IN
(SELECT DISTINCT soh.CustomerID
FROM [Sales].[SalesORDERDetail] AS sod INner JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID WHERE
sod.ProductID=p.ProductID))
FROM [Production].[Product] AS p
```

4 Найти такие товары, которые были куплены более чем одним покупателем, при этом все покупатели этих товаров покупали товары только из одной подкатегории.

```
SELECT name
FROM [Production].[Product]
WHERE ProductID IN(
SELECT sod.ProductID
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
WHERE soh.CustomerID IN(
SELECT soh.CustomerID
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID JOIN
[Production].[Product] AS p ON
sod.ProductID=p.ProductID
```

```
GROUP BY soh.CustomerID
HAVING count(DISTINCT p.ProductSubcategoryID)=1)
GROUP BY sod.ProductID
HAVING count(DISTINCT soh.CustomerID)>1)
```

5 Найти покупателя, который каждый раз имел разный список товаров в чеке (по номенклатуре).

```
SELECT DISTINCT CustomerID
FROM [Sales].[SalesORDERHeader]
WHERE CustomerID NOT IN (
SELECT soh.Customerid
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
WHERE
exists(SELECT ProductID
FROM [Sales].[SalesORDERDetail] AS sod1 JOIN
[Sales].[SalesORDERHeader] AS soh1
ON sod.SalesORDERID=soh.SalesORDERID
WHERE soh1.CustomerID=soh.CustomerID AND
sod1.ProductID=sod.ProductID                                    AND
sod.SalesORDERID!=sod1.SalesORDERID
))
```

6 Найти такого покупателя, что все купленные им товары были куплены только им и никогда не покупались другими покупателями.

Вариант 1

```
SELECT soh1.CustomerID
FROM[Sales].[SalesORDERDetail] AS sod1 INner JOIN
[Sales].[SalesORDERHeader] AS soh1
ON sod1.SalesORDERID=soh1.SalesORDERID
GROUP BY soh1.CustomerID
HAVING count(DISTINCT sod1.productid)=
(SELECT count(DISTINCT sod.ProductID)
FROM [Sales].[SalesORDERDetail] AS sod INner JOIN
[Sales].[SalesORDERHeader] AS soh
```

```
ON sod.SalesORDERID=soh.SalesORDERID
WHERE soh.CustomerID=soh1.CustomerID
AND sod.ProductID IN
(SELECT sod.ProductID
FROM[Sales].[SalesORDERDetail] AS sod INner JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
GROUP BY sod.ProductID
HAVING count(DISTINCT soh.CustomerID)=1))
```

Вариант 2

```
SELECT DISTINCT soh.CustomerID
FROM [Sales].[SalesORDERHeader] AS soh
WHERE soh.CustomerID NOT IN(
SELECT DISTINCT soh.CustomerID
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
WHERE ProductID NOT IN(
SELECT sod.ProductID
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
GROUP BY sod.ProductID
HAVING count(DISTINCT soh.CustomerID)=1))
```

Задания для самостоятельной работы

1 Найти название самого продаваемого продукта.

2 Найти покупателя, совершившего покупку на самую большую сумм, считая сумму покупки исходя из цены товара без скидки (UnitPrice).

3 Найти такие продукты, которые покупал только один покупатель.

4 Вывести список продуктов, цена которых выше средней цены товаров в подкатегории, к которой относится товар.

5 Найти такие товары, которые были куплены более чем одним покупателем, при этом все покупатели этих товаров покупали товары только одного цвета и товары не входят в список покупок покупателей, купивших товары только двух цветов.

6 Найти такие товары, которые были куплены такими покупателями, у которых они присутствовали в каждой их покупке.

7 Найти покупателей, у которых есть товар, присутствующий в каждой покупке/чеке.

8 Найти такой товар или товары, которые были куплены не более чем тремя различными покупателями.

9 Найти все товары, такие что их покупали всегда с товаром, цена которого максимальна в своей категории.

10 Найти номера тех покупателей, у которых есть как минимум два чека, и каждый из этих чеков содержит как минимум три товара, каждый из которых как минимум был куплен другими покупателями три раза.

11 Найти все чеки, в которых каждый товар был куплен дважды этим же покупателем.

12 Найти товары, которые были куплены минимум три раза различными покупателями.

13 Найти такую подкатегорию или подкатегории товаров, которые содержат более трех товаров, купленных более трех раз.

14 Найти те товары, которые не были куплены более трех раз, и как минимум дважды одним и тем же покупателем.

Практическая работа 5

Цель работы: использование результатов выборки как источник данных, использование обобщенного табличного выражения (ОТВ).

В подавляющем большинстве случаев операция выборки предполагает, что источником данных будут служить таблицы. В некоторых случаях это создает определенные неудобства, в частности, не позволяет получить выборку из результата выполнения предыдущего запроса. Решить эту задачу можно несколькими способами.

Представление. Представление является именованным результатом выполнения запроса. Поскольку представление – это объект базы данных, то для его создания и использования требуются дополнительные права.

Альтернативным вариантом, не требующим дополнительных прав, является использование запросов как источников данных или обобщенных табличных выражений.

Использование запроса как источника данных подразумевает, что пользователь создает запрос, дает ему псевдоним и использует этот псевдоним аналогично имени таблицы.

Рассмотрим сложный и объемный пример. Необходимо найти для каждой подкатегории количество товаров, у которых цена выше средней цены в подкатегории, и количество товаров, у которых цена ниже средней цены в подкатегории. Оформить вывод в виде трех столбцов: номер подкатегории, первый показатель, второй показатель. Решить подобную задачу традиционным способом достаточно сложно, но она относительно легко решается с использованием псевдонимов для результата выполнения запроса.

```
SELECT T1.PS, T1.c, T2.c FROM
(SELECT COUNT(*) AS c, [ProductSubcategoryID] AS PS
FROM [Production].[Product] AS P
WHERE [ListPrice]<(SELECT avg([ListPrice])
FROM [Production].[Product] AS PT
WHERE pt.ProductSubcategoryID=p.ProductSubcategoryID)
GROUP BY [ProductSubcategoryID]) AS T1 INNER JOIN
(SELECT count(*) AS c, [ProductSubcategoryID] AS PS
FROM [Production].[Product] AS P
WHERE [ListPrice]>=(SELECT avg([ListPrice])
FROM [Production].[Product] AS PT
WHERE pt.ProductSubcategoryID=p.ProductSubcategoryID)
GROUP BY [ProductSubcategoryID]) AS T2 ON
```

```
T1.PS=T2.PS
```

В данном примере созданы два независимых запроса, один из которых находит номер категории и количество товаров, цена у которых ниже средней цены в той же категории, к которой он относится, второй – такой же, но знак заменен на «больше или равно». Каждый запрос получил свой псевдоним, T1 и T2 соответственно, и каждому столбцу в запросе дан свой псевдоним. Запрос верхнего уровня использует запросы T1 и T2 точно так же, как обычные таблицы, проводит операцию INNER JOIN и выводит искомый результат.

К сожалению, использование такого подхода удобно, когда запрос, формирующий источник данных, не очень длинный. В противном случае читаемость кода резко падает, и в этом случае удобнее использовать обобщенное табличное выражение. Также обобщенное табличное выражение можно использовать при выполнении однократной операции модификации данных.

Общий синтаксис:

```
WITH <имя ОТВ>
( столбец 1, столбец 2, … ]
AS
(запрос)
```

В нерекурсивных ОТВ недопустимо использовать операцию упорядочивания, за исключением случаев использования инструкции TOP.

Рассмотрим пример, в котором необходимо найти количество чеков, приходящихся на одного покупателя на каждый год.

```
WITH Sales_CTE (SalesPersonID, SalesORDERID, SalesYear)
AS
(
    SELECT SalesPersonID, SalesORDERID, YEAR(ORDERDate) AS SalesYear
    FROM Sales.SalesORDERHeader
    WHERE SalesPersonID IS NOT NULL
)
SELECT SalesPersonID, COUNT(SalesORDERID) AS TotalSales, SalesYear
FROM Sales_CTE
GROUP BY SalesYear, SalesPersonID
ORDER BY SalesPersonID, SalesYear
```

В данном пример определено ОТВ с именем Sales_CTE, которое использовано как источник данных для запроса.

Если необходимо использовать несколько ОТВ в одном запросе, то их можно определить следующим образом.

```
WITH <имя ОТВ1>
( столбец 1, столбец 2, … ]
AS
(запрос),
<имя ОТВ2>
( столбец 1, столбец 2, … ]
AS
(запрос)
```

Примеры запросов с решениями

1 Найти покупателя, который каждый раз имел разный список товаров в чеке (по номенклатуре)

```
SELECT tmp.c
FROM
(SELECT soh.CustomerID AS c
, soh.SalesORDERID AS o
, CHECKSUM_AGG(sod.ProductID) AS ch
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=soh.SalesORDERID
GROUP BY soh.CustomerID, soh.SalesORDERID) tmp
GROUP BY tmp.c
HAVING count(tmp.ch)=count(DISTINCT tmp.ch)
AND count(tmp.ch)>1
```

2 Найти пары таких покупателей, что список названий товаров, которые они когда-либо покупали, не пересекается ни в одной позиции.

```
SELECT top 3 t1.c, t2.c
FROM
(SELECT soh.CustomerID AS c,
sod.ProductID AS p
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=sod.SalesORDERID) t1,
```

```
(SELECT soh.CustomerID AS c,
sod.ProductID AS p
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=sod.SalesORDERID) t2
WHERE t1.p!=all(SELECT sod.ProductID AS p
FROM [Sales].[SalesORDERDetail] AS sod JOIN
[Sales].[SalesORDERHeader] AS soh
ON sod.SalesORDERID=sod.SalesORDERID
WHERE soh.CustomerID=t2.c)
```

3 Вывести номера продуктов, таких, что их цена выше средней цены продукта в подкатегории, к которой относится продукт. Запрос реализовать двумя способами. В одном из решений допускается использование обобщенного табличного выражения.

Вариант 1

```
SELECT p1.ProductID
FROM [Production].[Product] AS p1
WHERE p1.ListPrice>
(
SELECT avg(p2.[ListPrice])
FROM [Production].[Product] AS p2
WHERE p2.ProductSubcategoryID=p1.ProductSubcategoryID
)
```

Вариант 2

```
with tmp (pscid, acgLP) AS
(SELECT p.ProductSubcategoryID, avg([ListPrice])
FROM [Production].[Product] AS p
GROUP BY p.ProductSubcategoryID)

SELECT p.ProductID
FROM [Production].[Product] AS p JOIN
tmp ON p.ProductSubcategoryID=tmp.pscid
WHERE [ListPrice]>tmp.acgLP
```

Задания для самостоятельной работы

1 Найти среднее количество покупок на чек для каждого покупателя (2 способа).

2 Найти для каждого продукта и каждого покупателя соотношение количества фактов покупки данного товара данным покупателем к общему количеству фактов покупки товаров данным покупателем

3 Вывести на экран следящую информацию: Название продукта, Общее количество фактов покупки этого продукта, Общее количество покупателей этого продукта

4 Вывести для каждого покупателя информацию о максимальной и минимальной стоимости одной покупки, чеке, в виде таблицы: номер покупателя, максимальная сумма, минимальная сумма.

5 Найти номера покупателей, у которых не было нет ни одной пары чеков с одинаковым количеством наименований товаров.

6 Найти номера покупателей, у которых все купленные ими товары были куплены как минимум дважды, т.е. на два разных чека.

Практическая работа 6

Цель работы: Использование оконных функций и предложения OVER.

Предложение OVER определяет секционирование и упорядочение набора строк до применения соответствующей оконной функции.

```
OVER (
  [ <PARTITION BY столбец> ]
  [ <ORDER BY столбец> ]
  [ <ROW или RANGE столбец> ]
)
```

PARTITION BY разделяет результирующий набор запроса на секции. Оконная функция применяется к каждой секции отдельно, и вычисление начинается заново для каждой секции.

ORDER BY определяет логический порядок строк в каждой секции результирующего набора.

ROW or RANGE ограничивает строки в пределах секции, указывая начальную и конечную точки.

Параметр CURRENT ROW указывает, что окно начинается или заканчивается на текущей строке при использовании совместно с предложением ROWS или что окно заканчивается на текущем значении при использовании с предложением RANGE. CURRENT ROW может быть задана и как начальная, и как конечная точка.

Параметр BETWEEN <граница рамки окна > AND <граница рамки окна > используется совместно с предложением ROWS или RANGE для указания нижней (начальной) или верхней (конечной) граничной точки окна. <граница рамки окна> определяет граничную начальную точку, а <граница рамки окна> определяет граничную конечную точку.

Параметр UNBOUNDED FOLLOWING указывает, что окно заканчивается на последней строке секции. UNBOUNDED FOLLOWING может быть указано только как конечная точка окна.

Пример:

```
RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
```

В данном случае параметр определяет, что окно начинается на текущей строке и заканчивается на последней строке секции.

```
ROWS BETWEEN 2 FOLLOWING AND 10 FOLLOWIN
```

В этом примере параметр определяет, что окно начинается на второй строке после текущей и заканчивается на десятой строке после текущей строки. Эта спецификация не допускается в предложении RANGE.

Фактически предложение OVER виртуально разбивает выбранные строки на наборы, окна, определяемые в условии PARTITION BY, упорядочивает эти строки по столбцам определенным ORDER BY. В рамках этих наборов, окон, выполняются те или иные агрегирующие, статистические и иные функции. Результат выполнения этих функций формирует отдельный столбец с одинаковыми значениями для каждой строки в наборе, окне. Однако можно для каждой строки в наборе формировать свое значение функции, для чего используют параметр ROW или RANGE, который определяет диапазон строк, в наборе, окне, с которыми будет работать функция.

```
SELECT SalesORDERID, ProductID, ORDERQty ,SUM(ORDERQty)
OVER(PARTITION BY SalesORDERID) AS Total
,AVG(ORDERQty) OVER(PARTITION BY SalesORDERID) AS "Avg"
,COUNT(ORDERQty) OVER(PARTITION BY SalesORDERID) AS "Count"
,MIN(ORDERQty) OVER(PARTITION BY SalesORDERID) AS "MIN"
,MAX(ORDERQty) OVER(PARTITION BY SalesORDERID) AS "Max"
FROM Sales.SalesORDERDetail
WHERE SalesORDERID IN(43659,43664)
```

Следующий запрос использует приведение типа.

```
SELECT SalesORDERID, ProductID, ORDERQty
,SUM(ORDERQty) OVER(PARTITION BY SalesORDERID) AS Total
,CAST(1. * ORDERQty / SUM(ORDERQty) OVER(PARTITION BY SalesORDERID)
*100 AS DECIMAL(5,2))AS "Percent BY ProductID"
FROM Sales.SalesORDERDetail
WHERE SalesORDERID IN(43659,43664)
```

Можно выполнять функцию не на всем наборе, но на части набора. Часть определяется в зависимости от строки и ее положения в наборе.

```
SELECT BusINessEntityID, TerritoryID
,CONVERT(varchar(20),SalesYTD,1) AS  SalesYTD
,DATEPART(yy,ModifiedDate) AS SalesYear
,CONVERT(varchar(20),SUM(SalesYTD) OVER (PARTITION BY TerritoryID
ORDER BY DATEPART(yy,ModifiedDate)
ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING ),1) AS CumulativeTotal
FROM Sales.SalesPerson
WHERE TerritoryID IS NULL OR TerritoryID < 5
```

Аналитические функции.

Функция FIRST_VALUE возвращает первое значение из упорядоченного набора значений.

Синтаксис

```
FIRST_VALUE ( [scalar_expressiON ] ) OVER ( [ partitiON_BY_clause ] ORDER_BY_clause [ rows_range_clause ] )
```

scalar_expressiON может быть столбцом, вложенным запросом.

Получение имени самого дешевого продукта в заданной категории продуктов.

```
SELECT Name, ListPrice,

FIRST_VALUE(Name) OVER (ORDER BY ListPrice ASC) AS LeAStExpensive

FROM Production.Product

WHERE ProductSubcategoryID = 37
```

Функция LAST_VALUE возвращает последнее значение из упорядоченного набора значений.

Синтаксис

```
LAST_VALUE ( [ scalar_expressiON ] ) OVER ( [ partitiON_BY_clause ] ORDER_BY_clause rows_range_clause )
```

scalar_expressiON может быть столбцом, вложенным запросом.

Возвращение даты найма последнего сотрудника каждого отдела для указанной заработной платы (Rate). Предложение PARTITION BY разделяет сотрудников по отделам, а функция LAST_VALUE применяется к каждой секции в отдельности. Предложение ORDER BY, указанное в предложении OVER, определяет логический порядок, в котором функция LAST_VALUE применяется к строкам каждой секции.

```
SELECT Department, LastName, Rate, HireDate,

LAST_VALUE(HireDate) OVER (PARTITION BY Department ORDER BY Rate) AS LAStValue

FROM HumanResources.vEmployeeDepartmentHistory AS edh

INNER JOIN HumanResources.EmployeePayHistory AS eph

ON eph.BusINessEntityID = edh.BusINessEntityID

INNER JOIN HumanResources.Employee AS e

ON e.BusINessEntityID = edh.BusINessEntityID

WHERE Department IN (N'INformatiON Services',N'Document CONtrol')
```

Функция LAG обеспечивает доступ к строке с заданным физическим смещением перед началом текущей строки.

```
LAG (scalar_expressiON [,offset] [,default])
```

```
OVER ( [ partitiON_BY_clause ] ORDER_BY_clause )
```

scalar_expressiON – возвращаемое значение основано на указанном смещении.

offset – количество строк до строки перед текущей строкой, из которой необходимо получить значение.

default – возвращаемое значение, когда offset находится за пределами секции.

Нахождение квоты для работника за год и предыдущий год.

Квоты менялись несколько раз в год, но для первой установленной в году квоты нет предыдущего значения от 2010 года, их не включают в выборку.

```
SELECT BusINessEntityID, YEAR(QuotaDate) AS SalesYear, SalesQuota AS
CurrentQuota,

LAG(SalesQuota, 1,0) OVER (ORDER BY YEAR(QuotaDate)) AS
PreviousQuota

FROM Sales.SalesPersonQuotaHistory

WHERE BusINessEntityID = 275 AND YEAR(QuotaDate) IN ('2011','2012')
```

Функция LEAD – доступ к строке на заданном физическом смещении после текущей строки.

Синтаксис

```
LEAD ( scalar_expressiON [ ,offset ] , [ default ] ) OVER ( [
partitiON_BY_clause ] ORDER_BY_clause )
```

Пример, получение квот продаж для указанного работника за последующие годы.

```
SELECT BusINessEntityID, YEAR(QuotaDate) AS SalesYear, SalesQuota AS
CurrentQuota,

LEAD(SalesQuota, 1,0) OVER (ORDER BY YEAR(QuotaDate)) AS NextQuota

FROM Sales.SalesPersonQuotaHistory

WHERE BusINessEntityID = 275 AND YEAR(QuotaDate) IN ('2011','2012')
```

Функция NTILE распределяет строки упорядоченной секции в заданное количество групп. Группы нумеруются, начиная с единицы. Для каждой строки функция NTILE возвращает номер группы, которой принадлежит строка.

Синтаксис

NTILE (INteger_expressiON) OVER ([<partitiON_BY_clause>] < ORDER_BY_clause >)

INteger_expressiON – положительное целое выражение, указывающее число групп, на которые необходимо разделить каждую секцию.

Функция ROW_NUMBER нумерует выходные данные результирующего набора.

```
ROW_NUMBER ( )
OVER ( [ PARTITION BY value_expressiON , ... [ n ] ] ORDER_BY_clause
)
```

Пример использования функций.

```
SELECT p.FirstName, p.LastName
,ROW_NUMBER() OVER (ORDER BY a.PostalCode) AS "Row Number"
,NTILE(4) OVER (ORDER BY a.PostalCode) AS Quartile
,s.SalesYTD
,a.PostalCode
FROM Sales.SalesPerson AS s
INNER JOIN Person.Person AS p
ON s.BusINessEntityID = p.BusINessEntityID
INNER JOIN Person.Address AS a
ON a.AddressID = p.BusINessEntityID
WHERE TerritoryID IS NOT NULL AND SalesYTD <> 0
```

Примеры запросов с решениями

1 Найти долю затрат каждого покупателя на каждый купленный им продукт среди общих его затрат в данной сети магазинов. Можно использовать обобщенное табличное выражение.

```
SELECT [SalesORDERID], p.[ProductID],
[ProductSubcategoryID],
[ORDERQty]*[UnitPrice],
[ORDERQty]*[UnitPrice]/sum([ORDERQty]*[UnitPrice])
OVER(partitiON BY [SalesORDERID]
, [ProductSubcategoryID])
FROM [Sales].[SalesORDERDetail] AS SOD INner JOIN
[Production].[Product] AS p
ON SOD.ProductID=p.ProductID
```

2 Для одного выбранного покупателя вывести, для каждой покупки (чека), разницу в деньгах между этой и следующей покупкой.

Вариант 1

```
with tmp (customer, ORDERid, total)  AS
(SELECT soh.CustomerID, soh.SalesORDERID,
sum(sod.[ORDERQty]*[UnitPrice]) AS total
```

```
FROM [Sales].[SalesORDERHeader] AS SOH INner JOIN
[Sales].[SalesORDERDetail] AS SOD
ON soh.SalesORDERID=sod.SalesORDERID
GROUP BY soh.CustomerID, soh.SalesORDERID)
SELECT customer, ORDERid, total,
total-LEAD(total,1,0)
OVER(partitiON BY customer ORDER BY ORDERid)
FROM tmp
```

3 Вывести следующую информацию: номер покупателя, номер чека этого покупателя, отсортированные по покупателям, номерам чека (по возрастанию). Третья колонка должна содержать в каждой своей строке сумму текущего чека покупателя со всеми предыдущими чеками этого покупателя.

Вариант 1

```
with tmp (cus, ord, ORDERsum)
AS (SELECT oh.CustomerID, od.SalesORDERID,
sum(od.[UnitPrice]*[ORDERQty])
FROM [Sales].[SalesORDERDetail] AS OD
INner JOIN
[Sales].[SalesORDERHeader] AS OH
ON od.SalesORDERID=oh.SalesORDERID
GROUP BY oh.CustomerID, od.SalesORDERID)
SELECT
cus, ord, ORDERsum,
sum(ORDERsum)
OVER(partitiON BY cus ORDER BY ord desc
RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)
FROM tmp
```

Задания для самостоятельной работы

1 Найти долю продаж каждого продукта (цена продукта * количество продукта), на каждый чек, в денежном выражении.

2 Вывести на экран список продуктов, их стоимость, а также разницу между стоимостью этого продукта и стоимостью самого дешевого продукта в той же подкатегории, к которой относится продукт.

3 Вывести три колонки: номер покупателя, номер чека покупателя (отсортированный по возрастанию даты чека) и искусственно введенный порядковый номер текущего чека, начиная с 1, для каждого покупателя.

4 Вывести номера продуктов, таких что их цена выше средней цены продукта в подкатегории, к которой относится продукт. Запрос реализовать двумя способами. В одном из решений допускается использование обобщенного табличного выражения.

5 Вывести на экран номер продукта, название продукта, а также информацию о среднем количестве этого продукта, приходящихся на три последних по дате чека, в которых был этот продукт.

Приложения

Ниже приведена таблица со всеми логическими операторами, их использование будет иллюстрировано на практических примерах.

Оператор	Синтаксис	Комментарий
ALL	*скаляр* { = | <> | != | > | >= | !> | < | <= | !< } ALL (*подзапрос*)	Сравнивает *скалярное* значение с набором значений, находящиеся в столбце, который вернул *подзапрос*. *Скаляр* должен удовлетворять условию для всех значений.
AND	*выражение* AND *выражение*	Стандартная конъюнкция
SOME, ANY	*скаляр* { = | <> | != | > | >= | !> | < | <= | !< } { SOME | ANY } (подзапрос)	Сравнивает *скалярное* значение с набором значений, находящиеся в столбце, который вернул *подзапрос*. *Скаляр* должен удовлетворять условию хотя бы одного из значений. SOME и ANY – эквиваленты.
BETWEEN	*выражение* [NOT] BETWEEN *выражение1* AND *выражение2*	Определяет диапазон для проверки, *выражение* должно находиться в диапазоне от *выражения1* до *выражения2* включительно. Выражения должны иметь один формат.
EXISTS	EXISTS(подзапрос)	Возвращает истину, если подзапрос возвращает хотя бы одно значение, в противном случае возвращает ложь. Особенность данного оператора в том, что подзапрос прекращается после нахождения первого значения, что существенно экономит ресурсы.
IN	*выражение* [NOT] IN (*подзапрос* | *список[,…n]*)	Определяет, совпадает ли указанное значение *выражения* с одним из значений возвращаемым *подзапросом* или содержащимся в *списке*.
LIKE	*выражение* [NOT] LIKE *шаблон*	Определяет, совпадает ли символьное *выражение* с указанным *шаблоном*. Для уточнения параметров шаблона см. пояснения после таблицы.
NOT	NOT *логическое выражение*	Стандартное логическое отрицание.
OR	*выражение* OR *выражение*	Стандартная дизъюнкция

В строковых шаблонах допускаются следующие символы:

% – символ-шаблон, заменяющий любую последовательность символов;

_ (подчеркивание) – символ-шаблон, заменяющий любой одиночный символ;

[] – заменяет одиночный символ, указанный в угловых скобках, можно перечислить символы или диапазон (через дефис) символов;

[^] – заменяет одиночный символ, не указанный в угловых скобках, можно перечислить символы или диапазон (через дефис) символов.

Допускается использование ESCAPE последовательностей:

```
WHERE СТРОКА LIKE '%[a-f][^xyz]_30!%%' ESCAPE '!'
```

Данный пример является частью запроса, который, в частности, проверяет, совпадают ли значения в столбце СТРОКА со следующим шаблоном: любое количество символов, далее один из символов в диапазоне от ‘a’ до ‘f’ включительно, далее любой символ, кроме символов ‘x’, ‘y’ или ‘z’, далее еще один любой символ, далее 30%, и опять последовательность любых символов. Символ ‘!’ является эскейп-символом и говорит о том, что следующий за ним символ, в данном случае ‘%’, не надо рассматривать как управляющий, таким образом, в пример включен символ %, который в общем случае является служебным.

Строковые функции

Функция	Синтаксис	Комментарий
ASCII	ASCII(*строковое выражение*)	Функция возвращает код ASCII первого символа *строкового выражения*.
CHAR	CHAR(*числовое выражение*)	Преобразует код ASCII символа, *числовое выражение*, в символ.
CHARINDEX	CHARINDEX(строка для поиска, строка поиска [, *номер начала поиска*])	Функция выполняет поиск строки, *строка для поиска*, в строке, *строка поиска*. Можно указать номер символа, *номер начала поиска*, с которого начать поиск в строке, *строке поиска*. Функция возвращает номер позиции, *строки для поиска*, если таковая найдена.
CONCAT	CONCAT(*строка1*, *строка2*[,*строкаN*])	Возвращает строку, результат объединения строки1, строки2 … строкиN.
CONCAT_WS	CONCAT_WS(*разделитель*, *строка1*, *строка2*[,*строкаN*])	Возвращает строку, результат объединения *строки1*, *строки2*

		… *строкиN*, разделенные *разделителем*.
DIFFERENCE	DIFFERENCE(*строка1*, *строка2*)	Возвращает число, разницу между значениями SOUNDEX *строки1* и *строки2*.
FORMAT	FORMAT(*значение*, *формат* [, *язык*])	Возвращает *значение* в указанном *формате*, возможно указание языкового / регионального параметра, *языка*.
LEFT	LEFT(*строка*, *число*)	Возвращает указанное *число* символов *строки* слева.
LEN	LEN(*строка*)	Возвращает длину *строки*.
LOWER	LOWER(*строка*)	Возвращает *строку*, все символы которой преобразованы в те же символы нижнего регистра
LTRIM	LTRIM(*строка*)	Возвращает *строку*, у которой удалены начальные пробелы.
NCHAR	NCHAR(*число*)	Возвращает символ Юникода с указанным номером.
PATINDEX	PATINDEX(*шаблон*, *выражение*)	Возвращает начальную позицию первого вхождения *шаблона* в *выражение* или ноль, если такого нет. Выражение – это строка или столбец.
QUOTENAME	QUOTENAME(*строка* [, *разделитель*])	Возвращает *строку* с *разделителями* в виде правильного идентификатора SQL Server.
REPLACE	REPLACE(*строка*, *шаблон*, *замена*)	Заменяет в *строке* все последовательности символов по *шаблону* на строку *замены*.
REVERSE	REVERSE(*строка*)	Возвращает строку, где символы переставлены в обратном порядке относительно *строки* параметра.
RIGHT	RIGHT(*строка*, *число*)	Возвращает указанное *число* символов *строки* справа.
RTRIM	RTRIM(*строка*)	Возвращает *строку*, в которой удалены все завершающие пробелы.
SOUNDEX	SOUNDEX(*строка*)	Возвращает четырехсимвольный код *строки*

SPACE	SPACE(*число*)	Возвращает строку из пробелов. Количество пробелов определяется *числом*.
STR	STR(*число*[, *длинна*[, *количество*]])	Возвращает символьные денные преобразованные из *числа* заданной *длины*, с заданным *количеством* знаков после запятой.
STRING_AGG	STRING_AGG(*выражение*, *разделитель*)	Сцепляет строковые *выражения*, используя *разделитель*. Допускается использование с GROUP BY.
STRING_ESCAPE	STRING_ESCAPE (*строка*, *тип*)	Возвращает *строку* с экранированными по *типу* символами.
STRING_SPLIT	STRING_SPLIT(*строка*, *разделитель*)	Возвращает таблицу, созданную из подстрок *строки* по *разделителю*.
STUFF	STUFF(*строка*, начало, *длинна*, *выражение*)	Вставляет одно *выражение* в *строку. Начало* определяет, с какого символа начнется вставка, и какова будет *длина* удаленной подстроки.
SUBSTRING	SUBSTRING(*строка*, *начало*, *длинна*)	Возвращает подстроку из *строки,* указанной *длины* с позиции *начала*.
TRANSLATE	TRANSLATE(*аргумент1*, *аргумент2*, *аргумент3*)	Возвращает строку, представленную в качестве *аргумента1*, после преобразования символов, *аргумент2*, в конечный набор символов, *аргумент3*.
TRIM	TRIM([*символы* FROM] *строка*)	Удаляет указанные *символы* из начала и конца *строки*.
UNICODE	UNICODE(*строка*)	Возвращает целочисленное значение, соответствующее стандарту Юникод для первого символа *строки*.
UPPER	UPPER(*строка*)	Возвращает *строку*, преобразованную в строку, где все символы переведены в верхний регистр.

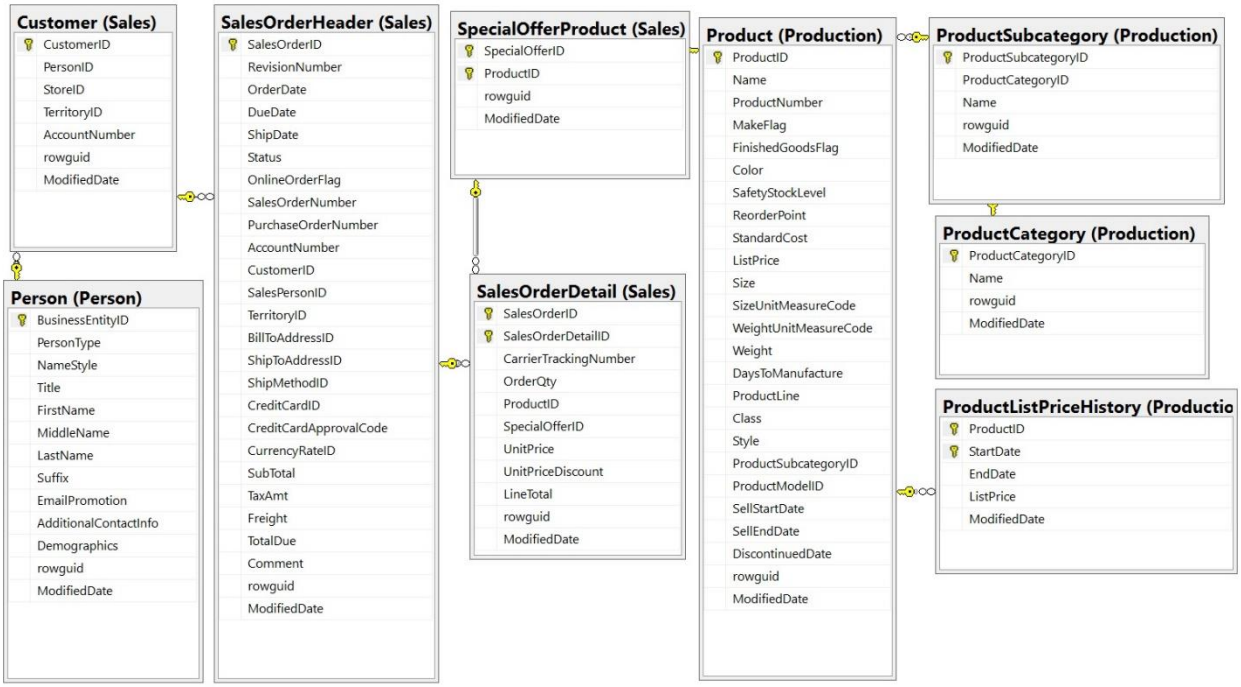
Функции работы с датой и временем

Функция	Синтаксис	Комментарий
SYSDATETIME	SYSDATETIME ()	Возвращает системное время компьютера, на котором запущен экземпляр СУБД. Возвращаемый тип данных – datetime2(7).
SYSDATETIMEOFFSET	SYSDATETIMEOFFSET ( )	Возвращает системное время компьютера, на котором запущен экземпляр СУБД. Возвращаемый тип данных – datetimeoffset(7).
SYSUTCDATETIME	SYSUTCDATETIME ( )	Возвращает системное время компьютера, на котором запущен экземпляр СУБД. Возвращаемый тип данных – datetime2(7) в формате UTC.
CURRENT_TIMESTAMP	CURRENT_TIMESTAMP	Возвращает системное время компьютера, на котором запущен экземпляр СУБД. Возвращаемый тип данных – datetime. Данная конструкция не является функцией, это эквивалент функции GETDATE ( ).
GETDATE	GETDATE ( )	Возвращает системное время компьютера, на котором запущен экземпляр СУБД. Возвращаемый тип данных – datetime.
GETUTCDATE	GETUTCDATE ( )	Возвращает системное время компьютера, на котором запущен экземпляр СУБД. Возвращаемый тип данных – datetime2 в формате UTC.
DATENAME	DATENAME (*часть* , *дата*)	Возвращает строку символов, которая является *частью даты*. Формат *части* даты: • year или yy, yyyy • quarter или qq, q • mONth или mm, m • dayofyear или dy, y • day или dd, d • week или wk, ww • weekday или dw, w

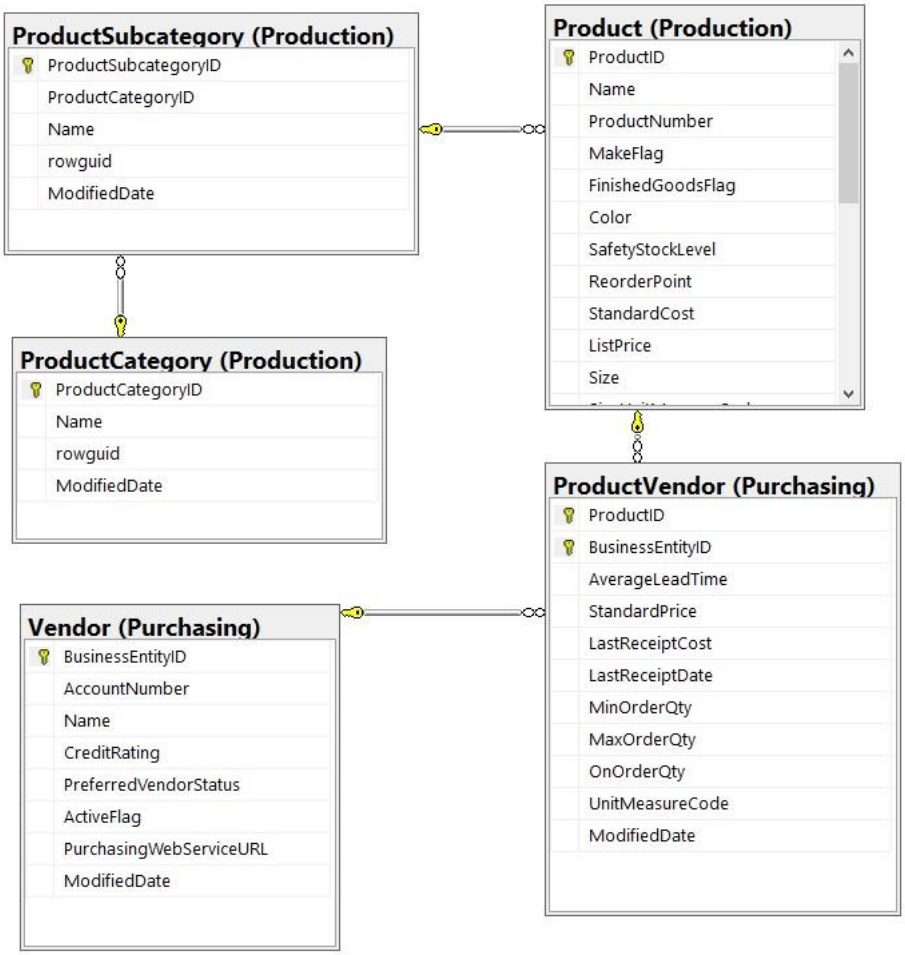
		• hour или hh • mINute или mi, n • secONd или ss, s • millisecONd или ms • microsecONd или mcs • nanosecONd или ns • TZoffset или tz • ISO_WEEK или ISOWK, ISOWW
DATEPART	DATEPART (*часть* , дата)	Возвращает целое число, являющееся *частью даты*. Формат *части* такой же, как и у функции DATENAME
DAY	DAY (*дата*)	Возвращает целое число, являющееся *частью даты*.
MONTH	MONTH (*дата*)	Возвращает целое число, являющееся *частью даты*.
YEAR	YEAR (*дата*)	Возвращает целое число, являющееся *частью даты*.
DATEFROMPARTS	DATEFROMPARTS (*год* , *месяц* , *день*)	Возвращает дату в формате date, соответствующую *году*, *месяцу* и *дню*
DATETIME2FROMPARTS	DATETIME2FROMPARTS (*год*, *месяц*, *день*, *час*, *минуты*, *секунды*, *доли_секунд*, *точность*)	Возвращает дату в формате datetime2, соответствующую *году*, *месяцу*, *дню, часу, минуте, секунде, доли секунды* с заданной *точностью.*
DATETIMEFROMPARTS	DATETIMEFROMPARTS (*год*, *месяц*, *день*, *час*, *минуты*, *секунды*, *миллисекунды*)	Возвращает дату в формате datetime, соответствующую *году*, *месяцу*, *дню, часу, минуте, секунде, миллисекунде.*
DATETIMEOFFSETFROMPARTS	DATETIMEOFFSETFROMPARTS (*год*, *месяц*, *день*, *час*, *минуты*, *секунды*, *доли_секунд*, смещение_в_часах, смещение_в_минутах, *точность*)	Возвращает дату в формате datetimeoffset, соответствующую *году*, *месяцу*, *дню, часу, минуте, секунде, доли секунды* с заданной *точностью*, с учетом *смещения в часах и минутах.*
SMALLDATETIMEFROMPARTS	SMALLDATETIMEFROMPARTS (*год*, *месяц*, *день*, *час*, *минуты*)	Возвращает дату в формате smalldatetime, соответствующую *году*, *месяцу*, *дню, часу, минуте.*
TIMEFROMPARTS	TIMEFROMPARTS (*час*, *минуты*, *секунды*, *доли_секунд*, *точность*)	Возвращает дату в формате time, соответствующую *часу, минуте, секунде, доли секунды* с заданной *точностью.*

DATEDIFF	DATEDIFF (*часть* , начальная_дата , конечная_дата)	Возвращает целое число, INt, разницу между *начальной датой* и *конечной датой* в *частях*. Формат *части* аналогичен формату части для функции DATENAME.
DATEDIFF_BIG	DATEDIFF_BIG (*часть* , начальная_дата , конечная_дата)	Возвращает целое число, bigINt, разницу между *начальной датой* и *конечной датой* в *частях*. Формат *части* аналогичен формату части для функции DATENAME.
DATEADD	DATEADD (*часть* , *число* , *дата*)	Возвращает новую дату, типа datetime, соответствующую *дате* увеличенной на то количество *частей*, которое определено *числом*.
EOMONTH	EOMONTH (*дата* [, *число*])	Возвращает дату последнего дня того месяца, который содержит указанную *дату*. К указанной дате можно добавить *число*, определяющее, на сколько месяцев сместить *дату*.
SWITCHOFFSET	SWITCHOFFSET (*дата* , *зона*)	Возвращает дату, типа datetimeoffset, соответствующую введенной *дате*, типа datetimeoffset, со смещением на часовую *зону*.
TODATETIMEOFFSET	TODATETIMEOFFSET (дата , зона)	Возвращает дату, типа datetimeoffset, соответствующую введенной *дате*, типа datetime2, со смещением на часовую *зону*.
ISDATE	ISDATE (*выражение*)	Возвращает 1, если *выражение* соответствует формату datetime или smalldatetime, и 0 во всех других случаях.

Модель данных



Модель данных



Повышев Владислав Вячеславович

Практикум Базы данных.

Практикум

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверский пр., 49