

Лекция 6. Агрегатные функции (AVG, SUM, MIN, MAX, COUNT). Группировка данных. Предложение GROUP BY. Фильтрация групп. Предложение HAVING.

Расширения SQL Server для группировки (ROLLUP, CUBE, GROUPING SETS, OVER). Разворачивание данных. Табличный оператор PIVOT. Отмена разворачивания данных. Табличный оператор UNPIVOT.

I. Агрегатные функции (AVG, SUM, MIN, MAX, COUNT). Группировка данных. Предложение GROUP BY. Фильтрация групп. Предложение HAVING.

Агрегатные функции используются для получения обобщающих значений.
<https://learn.microsoft.com/ru-ru/sql/t-sql/functions/aggregate-functions-transact-sql?view=sql-server-ver16>

Агрегатные функции выполняют вычисления над значениями в наборе строк. Агрегатные функции оперируют значениями столбцов множества строк.

В T-SQL имеются следующие агрегатные функции:

AVG: находит среднее значение

SUM: находит сумму значений

MIN: находит наименьшее значение

MAX: находит наибольшее значение

COUNT: находит количество строк в запросе

В качестве аргумента все агрегатные функции принимают выражение, которое представляет критерий для определения значений. Зачастую, в качестве выражения выступает название столбца, над значениями которого надо проводить вычисления.

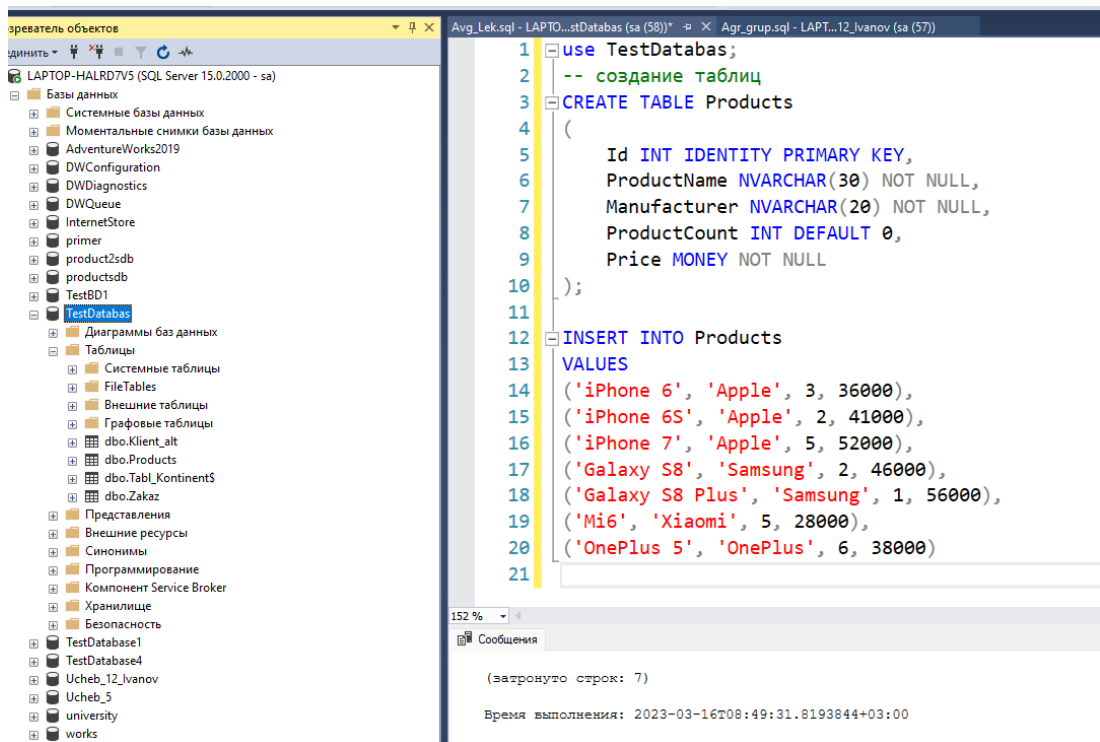
Выражения в функциях AVG и SUM должно представлять числовое значение. Выражение в функциях MIN, MAX и COUNT может представлять числовое или строковое значение или дату.

Все агрегатные функции за исключением COUNT(*) игнорируют значения NULL.

**Для функций SUM и AVG столбец должен содержать числовые значения.
Для функций COUNT() можно указать аргумент * для подсчета всех строк без исключения.**

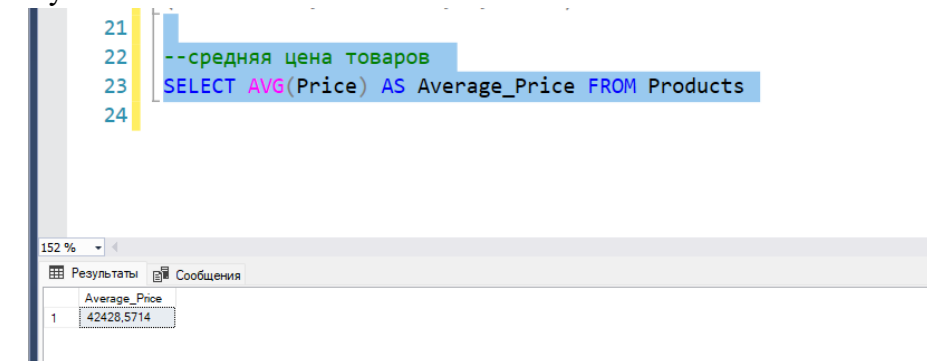
1. Avg. Функция Avg возвращает среднее значение на диапазоне значений столбца таблицы.

Пусть в базе данных у нас есть таблица товаров Products, которая описывается следующими выражениями:

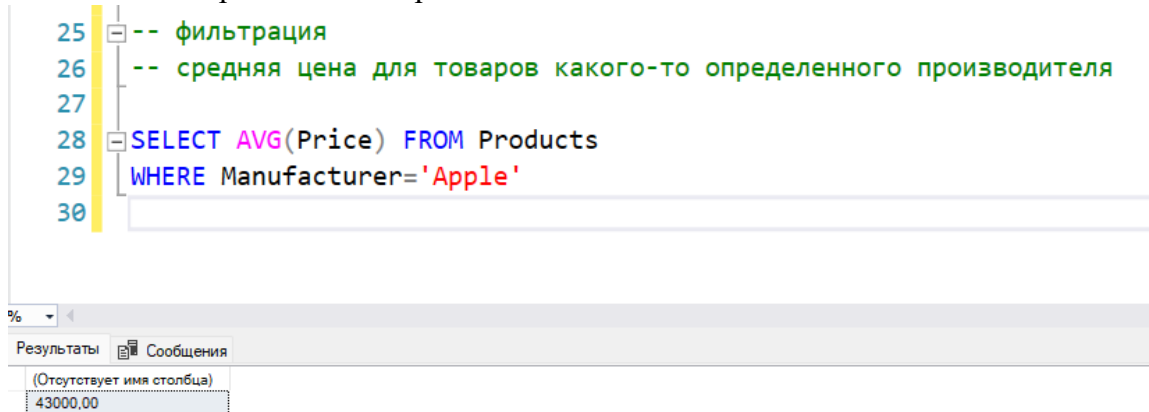


Найдем среднюю цену товаров из базы данных:

Для поиска среднего значения в качестве выражения в функцию передается столбец Price. Для получаемого значения устанавливается псевдоним Average_Price, хотя можно его и не устанавливать.



Также мы можем применить фильтрацию. Например, найти среднюю цену для товаров какого-то определенного производителя



И, кроме того, мы можем находить среднее значение для более сложных выражений. Например, найдем среднюю сумму всех товаров, учитывая их количество:

```
30 |
31 | SELECT AVG(Price * ProductCount) FROM Products;
32 |
```

Результаты	Сообщения
(Отсутствует имя столбца)	
138000.00	

2. Count. Функция Count вычисляет количество строк в выборке.

Есть две формы этой функции. Первая форма COUNT(*) подсчитывает число строк в выборке:

```
33 | --подсчитывает число строк в выборке
34 |
35 | SELECT COUNT(*) FROM Products;
36 |
```

Результаты	Сообщения
(Отсутствует имя столбца)	
7	

Вторая форма функции вычисляет количество строк по определенному столбцу, при этом строки со значениями NULL игнорируются:

```
36 |
37 | --количество строк по определенному столбцу
38 |
39 | SELECT COUNT(Manufacturer) FROM Products
40 |
41 |
```

Результаты	Сообщения
(Отсутствует имя столбца)	
7	

3. Min и Max. Функции Min и Max возвращают соответственно минимальное и максимальное значение по столбцу.

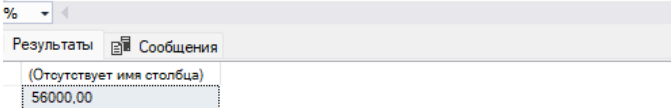
Например, найдем минимальную цену среди товаров:

```
41 | --минимальная цена среди товаров
42 |
43 | SELECT MIN(Price) FROM Products
44 |
45 |
```

Результаты	Сообщения
(Отсутствует имя столбца)	
28000.00	

Поиск максимальной цены:

```
45 --Поиск максимальной цены:
46 SELECT MAX(Price) FROM Products
47
```



The screenshot shows a SQL query window with the query: `SELECT MAX(Price) FROM Products`. The results pane displays a single value: 56000,00.

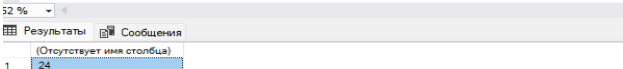
(Отсутствует имя столбца)
56000,00

Данные функции также игнорируют значения NULL и не учитывают их при подсчете.

4. Sum. Функция Sum вычисляет сумму значений столбца.

Например, подсчитаем общее количество товаров:

```
48 --Сумма значений столбца
49 SELECT SUM(ProductCount) FROM Products
50
51
```



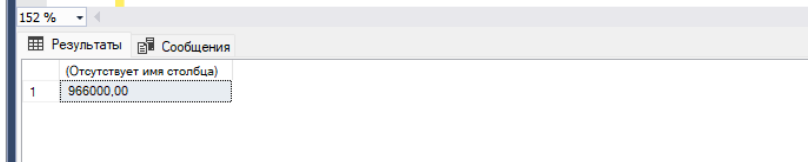
The screenshot shows a SQL query window with the query: `SELECT SUM(ProductCount) FROM Products`. The results pane displays a single value: 24.

(Отсутствует имя столбца)
24

Также вместо имени столбца может передаваться вычисляемое выражение.

Например, найдем общую стоимость всех имеющихся товаров:

```
50
51 --общая стоимость всех имеющихся товаров
52
53 SELECT SUM(ProductCount * Price) FROM Products
54
55
```



The screenshot shows a SQL query window with the query: `SELECT SUM(ProductCount * Price) FROM Products`. The results pane displays a single value: 966000,00.

(Отсутствует имя столбца)
966000,00

5. All и Distinct

По умолчанию все вышеперечисленных пять функций учитывают все строки выборки для вычисления результата. Но выборка может содержать повторяющиеся значения. Если необходимо выполнить вычисления только над уникальными значениями, исключив из набора значений повторяющиеся данные, то для этого применяется оператор DISTINCT.

```
1 SELECT AVG(DISTINCT ProductCount) AS Average_Price FROM Products
```

По умолчанию вместо DISTINCT применяется оператор ALL, который выбирает все строки:

```
1 SELECT AVG(ALL ProductCount) AS Average_Price FROM Products
```

Так как этот оператор неявно подразумевается при отсутствии DISTINCT, то его можно не указывать.

6. Комбинирование функций

Объединим применение нескольких функций:

```
55  --6 Комбинирование функций
56  SELECT COUNT(*) AS ProdCount,
57         SUM(ProductCount) AS TotalCount,
58         MIN(Price) AS MinPrice,
59         MAX(Price) AS MaxPrice,
60         AVG(Price) AS AvgPrice
61  FROM Products
```

%

Результаты Сообщения

ProdCount	TotalCount	MinPrice	MaxPrice	AvgPrice
7	24	28000,00	56000,00	42428,5714

7. Группировка данных. Предложение GROUP BY.

Для группировки данных в T-SQL применяются операторы GROUP BY и HAVING, для использования которых применяется следующий формальный синтаксис:

SELECT столбцы
FROM таблица
[WHERE условие_фильтрации_строк]
[GROUP BY столбцы_для_группировки]
[HAVING условие_фильтрации_групп]
[ORDER BY столбцы_для_сортировки]

Агрегатные функции можно применить не только на всю таблицу, но также на группу значений. Для этого применяется команда **GROUP BY**, которая пишется после WHERE.

После команды **GROUP BY** перечисляется *название столбцов*, по которым следует группировать данные. *Оператор GROUP BY определяет, как строки будут группироваться.*

Предложение **GROUP BY** указывает, что результаты запроса следует разделить на группы, применить агрегатную функцию по отдельности к каждой группе и получить для каждой группы одну строку результатов.

В качестве элемента группировки должен выступать любой возвращаемый элемент, указанный в предложении SELECT, **кроме значений агрегатных функций.**

Если столбец, по которому производится группировка, содержит значение NULL, то строки со значением NULL составят отдельную группу.

7.1 GROUP BY. Оператор GROUP BY определяет, как строки будут группироваться.

Изначальные данные:

.52 %					
Результаты					
	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	3	36000,00
2	2	iPhone 6S	Apple	2	41000,00
3	3	iPhone 7	Apple	5	52000,00
4	4	Galaxy S8	Samsung	2	46000,00
5	5	Galaxy S8 Plus	Samsung	1	56000,00
6	6	Mi6	Xiaomi	5	28000,00
7	7	OnePlus 5	OnePlus	6	38000,00

Например, сгруппируем товары по производителю

62	
63	-- Группировка
64	
65	SELECT Manufacturer, COUNT(*) AS ModelsCount
66	FROM Products
67	GROUP BY Manufacturer
68	
%	
Результаты	
Manufacturer	ModelsCount
Apple	3
OnePlus	1
Samsung	2
Xiaomi	1

Первый столбец в выражении SELECT - **Manufacturer** представляет **название группы**, а второй столбец - ModelsCount представляет результат функции Count, которая вычисляет количество строк в группе.

Стоит учитывать, что любой столбец, который используется в выражении SELECT (не считая столбцов, которые хранят результат агрегатных функций), должны быть указаны после оператора GROUP BY.

Так, например, в случае выше столбец Manufacturer указан и в выражении SELECT, и в выражении GROUP BY.

И если в выражении SELECT производится выборка по одному или нескольким столбцам и также используются агрегатные функции, то необходимо использовать выражение GROUP BY. Так, следующий пример работать не будет, так как он не содержит выражение группировки

```
1 SELECT Manufacturer, COUNT(*) AS ModelsCount
2 FROM Products
```

Другой пример, изначальные данные:

52 %						
		Результаты		Сообщения		
	Id	ProductName	Manufacturer	ProductCount	Price	
1	1	iPhone 6	Apple	3	36000,00	
2	2	iPhone 6S	Apple	2	41000,00	
3	3	iPhone 7	Apple	5	52000,00	
4	4	Galaxy S8	Samsung	2	46000,00	
5	5	Galaxy S8 Plus	Samsung	1	56000,00	
6	6	Mi6	Xiaomi	5	28000,00	
7	7	OnePlus 5	OnePlus	6	38000,00	

добавим группировку по количеству товаров:

```
69 -- добавим группировку по количеству товаров
70 SELECT Manufacturer, ProductCount, COUNT(*) AS ModelsCount
71 FROM Products
72 GROUP BY Manufacturer, ProductCount
73
74
```

152 %				
		Результаты		Сообщения
	Manufacturer	ProductCount	ModelsCount	
1	Samsung	1	1	
2	Apple	2	1	
3	Samsung	2	1	
4	Apple	3	1	
5	Apple	5	1	
6	Xiaomi	5	1	
7	OnePlus	6	1	

Оператор GROUP BY может выполнять группировку по множеству столбцов.

Если столбец, по которому производится группировка, содержит значение NULL, то строки со значением NULL составят отдельную группу.

Следует учитывать, что выражение GROUP BY должно идти после выражения WHERE, но до выражения ORDER BY:

Изначальные Данные:

.52 %					
Результаты Сообщения					
	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	3	36000,00
2	2	iPhone 6S	Apple	2	41000,00
3	3	iPhone 7	Apple	5	52000,00
4	4	Galaxy S8	Samsung	2	46000,00
5	5	Galaxy S8 Plus	Samsung	1	56000,00
6	6	Mi6	Xiaomi	5	28000,00
7	7	OnePlus 5	OnePlus	6	38000,00

```
74 --Группировка с выражением WHERE
75
76 SELECT Manufacturer, COUNT(*) AS ModelsCount
77 FROM Products
78 WHERE Price > 30000
79 GROUP BY Manufacturer
80 ORDER BY ModelsCount DESC
81
```

.52 %					
Результаты Сообщения					
	Manufacturer	ModelsCount			
1	Apple	3			
2	Samsung	2			
3	OnePlus	1			

7.2 Фильтрация групп. HAVING

Оператор **HAVING** определяет, **какие группы будут включены в выходной результат, то есть выполняет фильтрацию групп.**

Применение **HAVING** во многом аналогично применению **WHERE**. Только есть **WHERE** применяется к фильтрации строк, то **HAVING** используется для **фильтрации групп.**

Команда HAVING <условие> применяется для **фильтрации строк, возвращаемых при использовании предложения GROUP BY.** Оператор **HAVING** определяет, **какие группы будут включены в выходной результат, то есть выполняет фильтрацию групп.**

HAVING пишется после **GROUP BY**, имеет такой формат, как **WHERE**, но в качестве значения используется значение, возвращаемое агрегатными функциями.

Изначальные данные:

.52 %					
Результаты Сообщения					
	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	3	36000,00
2	2	iPhone 6S	Apple	2	41000,00
3	3	iPhone 7	Apple	5	52000,00
4	4	Galaxy S8	Samsung	2	46000,00
5	5	Galaxy S8 Plus	Samsung	1	56000,00
6	6	Mi6	Xiaomi	5	28000,00
7	7	OnePlus 5	OnePlus	6	38000,00

Например, найдем все группы товаров по производителям, для которых определено более 1 модели:

81	
82	--Применение оператора HAVING
83	--группы товаров по производителям, для которых определено более 1 модели
84	
85	SELECT Manufacturer, COUNT(*) AS ModelsCount
86	FROM Products
87	GROUP BY Manufacturer
88	HAVING COUNT(*) > 1

2 %	
Результаты Сообщения	
Manufacturer	ModelsCount
Apple	3
Samsung	2

При этом в одной команде мы можем использовать выражения WHERE и HAVING:

Изначальные данные:

.52 %					
Результаты Сообщения					
	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	3	36000,00
2	2	iPhone 6S	Apple	2	41000,00
3	3	iPhone 7	Apple	5	52000,00
4	4	Galaxy S8	Samsung	2	46000,00
5	5	Galaxy S8 Plus	Samsung	1	56000,00
6	6	Mi6	Xiaomi	5	28000,00
7	7	OnePlus 5	OnePlus	6	38000,00

То есть в данном примере:

сначала фильтруются строки: выбираются те товары, общая стоимость которых больше 80000.

затем выбранные товары группируются по производителям.

далее фильтруются сами группы - выбираются те группы, которые содержат больше 1 модели.

```

89
90 --в одной команде мы можем использовать выражения WHERE и HAVING:
91 SELECT Manufacturer, COUNT(*) AS ModelsCount
92 FROM Products
93 WHERE Price * ProductCount > 80000
94 GROUP BY Manufacturer
95 HAVING COUNT(*) > 1
96

```

52 %

Результаты Сообщения

	Manufacturer	ModelsCount
1	Apple	3

Если при этом необходимо провести сортировку, то выражение ORDER BY идет после выражения HAVING:

Изначальные данные:

52 %

Результаты Сообщения

	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	3	36000,00
2	2	iPhone 6S	Apple	2	41000,00
3	3	iPhone 7	Apple	5	52000,00
4	4	Galaxy S8	Samsung	2	46000,00
5	5	Galaxy S8 Plus	Samsung	1	56000,00
6	6	Mi6	Xiaomi	5	28000,00
7	7	OnePlus 5	OnePlus	6	38000,00

В данном случае группировка идет по производителям, и также выбирается количество моделей для каждого производителя (Models) и общее количество всех товаров по всем этим моделям (Units). В конце группы сортируются по количеству товаров по убыванию.

```

97 --группировка идет по производителям,
98 --выбирается количество моделей для каждого производителя (Models)
99 --общее количество всех товаров по всем этим моделям (Units)
100 --группы сортируются по количеству товаров по убыванию
101
102 SELECT Manufacturer, COUNT(*) AS Models, SUM(ProductCount) AS Units
103 FROM Products
104 WHERE Price * ProductCount > 80000
105 GROUP BY Manufacturer
106 HAVING SUM(ProductCount) > 2
107 ORDER BY Units DESC
108

```

%

Результаты Сообщения

	Manufacturer	Models	Units
1	Apple	3	10
2	OnePlus	1	6
3	Xiaomi	1	5

II. Расширения SQL Server для группировки (ROLLUP, CUBE, GROUPING SETS, OVER).

Дополнительно к стандартным операторам GROUP BY и HAVING SQL Server поддерживает еще четыре специальных расширения для группировки данных: **ROLLUP**, **CUBE**, **GROUPING SETS** и **OVER**.

1. Оператор ROLLUP добавляет суммирующую строку в результирующий набор

Operator ROLLUP создаёт группу для каждого сочетания выражений столбцов.

Кроме того, выполняет сведение результатов в промежуточные и общие итоги. Для этого запрос перемещается справа налево, уменьшая количество выражений столбцов, по которым он создает группы и агрегаты. Синтаксис команды имеет следующий вид:

GROUP BY ROLLUP(<список столбцов>)

или

GROUP BY <список столбцов> WITH ROLLUP

Порядок столбцов влияет на выходные данные ROLLUP и может отразиться на количестве строк в результирующем наборе.

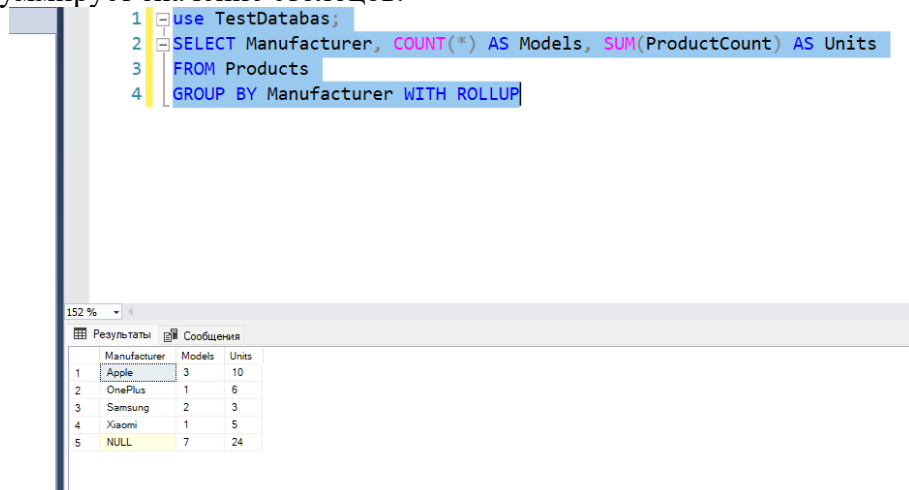
GROUP BY ROLLUP(col1, col2) создает группы для каждой комбинации выражений столбцов в следующих списках:

col1, col2

col1, NULL

NULL, NULL – это общий итог.

Как видно из скриншота, в конце таблицы была добавлена дополнительная строка, которая суммирует значение столбцов.



```
1 use TestDatabases;  
2 SELECT Manufacturer, COUNT(*) AS Models, SUM(ProductCount) AS Units  
3 FROM Products  
4 GROUP BY Manufacturer WITH ROLLUP
```

	Manufacturer	Models	Units
1	Apple	3	10
2	OnePlus	1	6
3	Samsung	2	3
4	Xiaomi	1	5
5	NULL	7	24

Альтернативный синтаксис запроса, который можно использовать, начиная с версии MS SQL Server 2008:

```
SELECT Manufacturer, COUNT(*) AS Models, SUM(ProductCount) AS Units  
FROM Products  
GROUP BY ROLLUP(Manufacturer)
```

При группировке по нескольким критериям ROLLUP будет создавать суммирующую строку для каждой из подгрупп:

1	use TestDatabas;
2	--ROLLUP
3	SELECT Manufacturer, COUNT(*) AS Models, SUM(ProductCount) AS Units
4	FROM Products
5	GROUP BY Manufacturer WITH ROLLUP
6	
7	--группировке по нескольким критериям ROLLUP
8	SELECT Manufacturer, COUNT(*) AS Models, SUM(ProductCount) AS Units
9	FROM Products
10	GROUP BY Manufacturer, ProductCount WITH ROLLUP

Результаты			Сообщения		
	Manufacturer	Models	Units		
	Apple	1	2		
	Apple	1	3		
	Apple	1	5		
	Apple	3	10		
	OnePlus	1	6		
	OnePlus	1	6		
	Samsung	1	1		
	Samsung	1	2		
	Samsung	2	3		
0	Xiaomi	1	5		
1	Xiaomi	1	5		
2	NULL	7	24		

Results		Messages	
	Manufacturer	Models	Units
1	Apple	1	2
2	Apple	1	3
3	Apple	1	5
4	Apple	3	10
5	OnePlus	1	6
6	OnePlus	1	6
7	Samsung	1	1
8	Samsung	1	2
9	Samsung	2	3
10	Xiaomi	1	5
11	Xiaomi	1	5
12	NULL	7	24

При сортировке с помощью ORDER BY следует учитывать, что она применяется уже после добавления суммирующей строки.

- Оператор **CUBE** похож на ROLLUP за тем исключением, что CUBE добавляет суммирующие строки для каждой комбинации групп.

Оператор **CUBE** создает группы для всех возможных сочетаний столбцов. Синтаксис команды имеет следующий вид:

GROUP BY CUBE(<список столбцов>)

или

GROUP BY <список столбцов> WITH CUBE

GROUP BY CUBE(col1, col2) создает группы для каждой комбинации выражений столбцов в следующих списках:

col1, col2

col1, NULL

NULL, col2

NULL, NULL – это общий итог.

Начальные данные

.52 %					
Результаты Сообщения					
	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	3	36000,00
2	2	iPhone 6S	Apple	2	41000,00
3	3	iPhone 7	Apple	5	52000,00
4	4	Galaxy S8	Samsung	2	46000,00
5	5	Galaxy S8 Plus	Samsung	1	56000,00
6	6	Mi6	Xiaomi	5	28000,00
7	7	OnePlus 5	OnePlus	6	38000,00

152 %			
12 --CUBE			
13 SELECT Manufacturer, COUNT(*) AS Models, SUM(ProductCount) AS Units			
14 FROM Products			
15 GROUP BY Manufacturer, ProductCount WITH CUBE			
152 %			
Результаты Сообщения			
	Manufacturer	Models	Units
1	Samsung	1	1
2	NULL	1	1
3	Apple	1	2
4	Samsung	1	2
5	NULL	2	4
6	Apple	1	3
7	NULL	1	3
8	Apple	1	5
9	Xiaomi	1	5
10	NULL	2	10
11	OnePlus	1	6
12	NULL	1	6
13	NULL	7	24
14	Apple	3	10
15	OnePlus	1	6
16	Samsung	2	3
17	Xiaomi	1	5

3. Оператор **GROUPING SETS** аналогично **ROLLUP** и **CUBE** добавляет суммирующую строку для групп. Но при этом он не включает сами группам:

Оператор **GROUPING SETS** позволяет объединять несколько предложений **GROUP BY** в одно предложение **GROUP BY**.

Синтаксис команды имеет следующий вид:

GROUP BY GROUPING SETS(<список столбцов>)

Если параметр **GROUPING SETS** имеет два или более элементов, результатом будет объединение элементов.

SQL не консолидирует повторяющиеся группы, созданные для списка **GROUPING SETS**.

GROUP BY GROUPING SETS(col1, col2) создает группы для каждой комбинации выражений столбцов в следующих списках:

col1, NULL
NULL, col2

```

16
17 --GROUPING
18 SELECT Manufacturer, COUNT(*) AS Models, ProductCount
19 FROM Products
20 GROUP BY GROUPING SETS(Manufacturer, ProductCount)
21
22

```

152 %

	Manufacturer	Models	ProductCount
1	NULL	1	1
2	NULL	2	2
3	NULL	1	3
4	NULL	2	5
5	NULL	1	6
6	Apple	3	NULL
7	OnePlus	1	NULL
8	Samsung	2	NULL
9	Xiaomi	1	NULL

При этом его можно комбинировать с ROLLUP или CUBE. Например, кроме суммирующих строк по каждой из групп добавим суммирующую строку для всех групп:

```

22 -- комбинация GROUPING с ROLLUP
23 SELECT Manufacturer, COUNT(*) AS Models,
24       ProductCount, SUM(ProductCount) AS Units
25 FROM Products
26 GROUP BY GROUPING SETS(ROLLUP(Manufacturer), ProductCount)

```

152 %

	Manufacturer	Models	ProductCount	Units
1	NULL	1	1	1
2	NULL	2	2	4
3	NULL	1	3	3
4	NULL	2	5	10
5	NULL	1	6	6
6	NULL	7	NULL	24
7	Apple	3	NULL	10
8	OnePlus	1	NULL	6
9	Samsung	2	NULL	3
10	Xiaomi	1	NULL	5

С помощью скобок можно определить более сложные сценарии группировки:

```

28 --сложные сценарии группировки
29 SELECT Manufacturer, COUNT(*) AS Models,
30       ProductCount, SUM(ProductCount) AS Units
31 FROM Products
32 GROUP BY GROUPING SETS((Manufacturer, ProductCount), ProductCount)
33

```

152 %

	Manufacturer	Models	ProductCount	Units
1	Samsung	1	1	1
2	NULL	1	1	1
3	Apple	1	2	2
4	Samsung	1	2	2
5	NULL	2	2	4
6	Apple	1	3	3
7	NULL	1	3	3
8	Apple	1	5	5
9	Xiaomi	1	5	5
10	NULL	2	5	10
11	OnePlus	1	6	6
12	NULL	1	6	6

4. Выражение OVER позволяет суммировать данные, при этом возвращая те строки, которые использовались для получения суммированных данных.

Например, найдем количество моделей и общее количество товаров этих моделей по производителю:

```
35  --OVER
36  SELECT ProductName, Manufacturer, ProductCount,
37         COUNT(*) OVER (PARTITION BY Manufacturer) AS Models,
38         SUM(ProductCount) OVER (PARTITION BY Manufacturer) AS Units
39  FROM Products
40
41
```

152 %

Результаты Сообщения

	ProductName	Manufacturer	ProductCount	Models	Units
1	iPhone 6	Apple	3	3	10
2	iPhone 6S	Apple	2	3	10
3	iPhone 7	Apple	5	3	10
4	OnePlus 5	OnePlus	6	1	6
5	Galaxy S8	Samsung	2	2	3
6	Galaxy S8 Plus	Samsung	1	2	3
7	Mi6	Xiaomi	5	1	5

Выражение OVER ставится после агрегатной функции, затем в скобках идет выражение PARTITION BY и столбец, по которому выполняется группировка.

То есть в данном случае мы выбираем название модели, производителя, количество единиц модели и добавляем к этому количество моделей для данного производителя и общее количество единиц всех моделей производителя:

	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	3	36000,00
2	2	iPhone 6S	Apple	2	41000,00
3	3	iPhone 7	Apple	5	52000,00
4	4	Galaxy S8	Samsung	2	46000,00
5	5	Galaxy S8 Plus	Samsung	1	56000,00
6	6	Mi6	Xiaomi	5	28000,00
7	7	OnePlus 5	OnePlus	6	38000,00

III. Операторы PIVOT и UNPIVOT

В Transact-SQL для написания перекрестных запросов или кросс табличных выражений существует специальный оператор PIVOT

Реляционные операторы PIVOT и UNPIVOT можно использовать для изменения возвращающего табличное значение выражения в другой таблице.

PIVOT поворачивает возвращающее табличное значение выражение, преобразуя уникальные значения одного столбца выражения в несколько выходных столбцов. В случае необходимости PIVOT также объединяет оставшиеся повторяющиеся значения столбца и отображает их в выходных данных.

UNPIVOT выполняет действия, обратные PIVOT, преобразуя столбцы возвращающего табличное значение выражения в значения столбца.

PIVOT – это оператор Transact-SQL, который поворачивает результирующий набор данных, т.е. происходит транспонирование таблицы, при этом используются агрегатные функции, и данные соответственно группируются.

Другими словами, значения, которые расположены по вертикали, мы выстраиваем по горизонтали.

Данный оператор может потребоваться тогда, когда необходимо, например, предоставить какой либо отчет в наглядной форме по годам, допустим для бухгалтеров и экономистов, так как именно они любят представления данных в таком виде.

Также он может пригодиться и просто для представления какой-либо статистики/

Синтаксис PIVOT имеет следующий вид:

```
SELECT < столбцы для группировки>, <пивируемые столбцы> FROM  
(<запрос возвращающий данных>)  
AS <псевдоним>  
PIVOT  
(<агрегирующая функция>(<столбец>)  
FOR [<столбец, значения которого будут заголовками>]  
IN (<список пивируемых столбцов>)  
) AS <псевдоним для пивот-таблицы>
```

Синтаксис UNPIVOT имеет следующий вид:

```
SELECT <список столбцов>  
FROM  
<таблица>  
UNPIVOT  
(<столбец значения строк> FOR [<столбец, значения заголовков>]  
IN (<список анпивируемых столбцов>)  
) AS <псевдоним для анпивот-таблицы>
```


Пример использования оператора PIVOT

Допустим, у нас есть таблица вот с такой структурой:

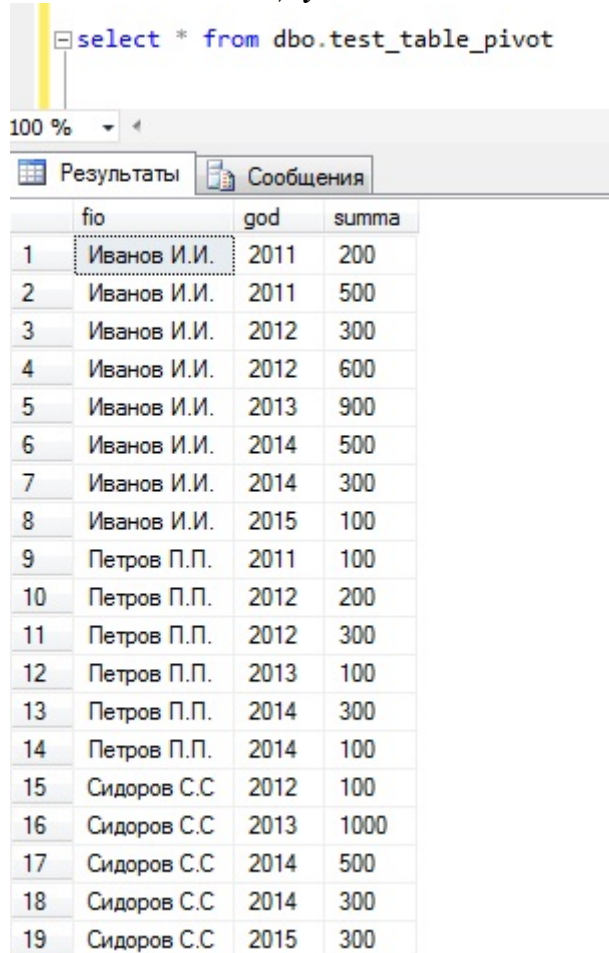
```
CREATE TABLE test_table_pivot(  
    fio varchar(50) NULL,  
    god int NULL,  
    summa float NULL  
)
```

Где, *fio* — это ФИО сотрудника,

god — год, в котором он получал премию,

summa — соответственно сумма премии, вот такой незамысловатый пример, так как в плоскости времени наглядней видна работа оператора PIVOT.

И в данной таблице у нас есть тестовые данные



The screenshot shows a SQL query window with the command `select * from dbo.test_table_pivot`. Below the query, the results are displayed in a table grid. The table has four columns: an index, *fio*, *god*, and *summa*. The data is as follows:

	fio	god	summa
1	Иванов И.И.	2011	200
2	Иванов И.И.	2011	500
3	Иванов И.И.	2012	300
4	Иванов И.И.	2012	600
5	Иванов И.И.	2013	900
6	Иванов И.И.	2014	500
7	Иванов И.И.	2014	300
8	Иванов И.И.	2015	100
9	Петров П.П.	2011	100
10	Петров П.П.	2012	200
11	Петров П.П.	2012	300
12	Петров П.П.	2013	100
13	Петров П.П.	2014	300
14	Петров П.П.	2014	100
15	Сидоров С.С.	2012	100
16	Сидоров С.С.	2013	1000
17	Сидоров С.С.	2014	500
18	Сидоров С.С.	2014	300
19	Сидоров С.С.	2015	300

А теперь представим, что нам необходимо сделать отчет, скажем для начальника, о размере премии, которую получал каждый сотрудник за год, в течение нескольких лет.

Самым простым способом будет конечно просто использовать конструкцию GROUP BY, например

```
SELECT fio, god, sum(summa) AS summa  
FROM dbo.test_table_pivot  
GROUP BY fio, god
```

```

select fio, god, sum(summa) as summa
from dbo.test_table_pivot
group by fio, god

```

	fio	god	summa
1	Иванов И.И.	2011	700
2	Петров П.П.	2011	100
3	Иванов И.И.	2012	900
4	Петров П.П.	2012	500
5	Сидоров С.С	2012	100
6	Иванов И.И.	2013	900
7	Петров П.П.	2013	100
8	Сидоров С.С	2013	1000
9	Иванов И.И.	2014	800
10	Петров П.П.	2014	400
11	Сидоров С.С	2014	800
12	Иванов И.И.	2015	100
13	Сидоров С.С	2015	300

На что нам начальник скажет, что это такое? ничего не понятно? не наглядно?

Улучшить ситуацию можно, добавив еще и сортировку ORDER BY, допустим сначала по фамилии, а затем по году

```

SELECT fio, god, sum(summa) as summa
FROM dbo.test_table_pivot
GROUP BY fio, god
ORDER BY fio, god

```

Если будем использовать оператор **PIVOT**, например вот таким образом
Синтаксис оператора PIVOT

SELECT столбец для группировки, [значения по горизонтали],...

FROM таблица или подзапрос

PIVOT(агрегатная функция

FOR столбец, содержащий значения, которые станут именами столбцов
IN ([значения по горизонтали],...)

)AS псевдоним таблицы (обязательно)

```

select fio, [2011],[2012],[2013],[2014],[2015]
from dbo.test_table_pivot
PIVOT (sum(summa)for god in ([2011],[2012],[2013],[2014],[2015])
) as test_pivot

```

	fio	2011	2012	2013	2014	2015
1	Иванов И.И.	700	900	900	800	100
2	Петров П.П.	100	500	100	400	NULL
3	Сидоров С.С	NULL	100	1000	800	300

Здесь у нас:

fio — столбец, по которому мы будем осуществлять группировку;
[2011],[2012],[2013],[2014],[2015] — названия наших столбцов по горизонтали, ими выступают значения из колонки *god*;
sum(summa) — агрегатная функция по столбцу *summa*;
for god in ([2011],[2012],[2013],[2014],[2015]) — тут мы указываем колонку, в которой содержатся значения, которые будут выступать в качестве названия наших результирующих столбцов, по факту в скобках мы указываем то же самое, что и чуть выше в *select*;
as test_pivot — это обязательный псевдоним, не забывайте его указывать, иначе будет ошибка.

Переходим к UNPIVOT.

Оператор UNPIVOT

UNPIVOT – это оператор *Transact-SQL*, который выполняет действия, обратные **PIVOT**, он разворачивает таблицу в обратную сторону, но в отличие от оператора **PIVOT** он ничего не агрегирует и уж тем более не раз агрегирует.

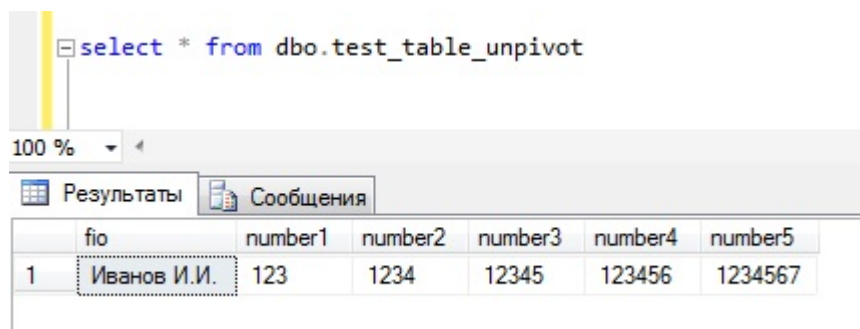
UNPIVOT требуется еще реже, чем **PIVOT**, но о нем также необходимо знать.

Пример использования UNPIVOT

Допустим, таблица имеет следующую структуру:

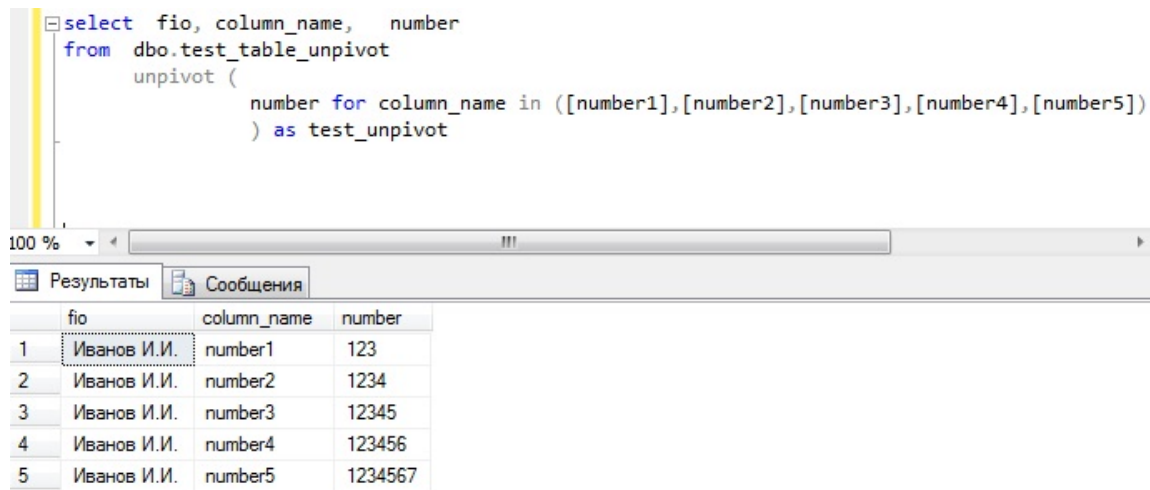
```
CREATE TABLE test_table_unpivot(  
    fio varchar(50) NULL,  
    number1 int NULL,  
    number2 int NULL,  
    number3 int NULL,  
    number4 int NULL,  
    number5 int NULL,  
)
```

Где, *fio* — ФИО сотрудника, а *number1, number2...* и так далее это какие-то номера этого сотрудника:



The screenshot shows a SQL query window with the query: `select * from dbo.test_table_unpivot`. Below the query, the 'Results' tab is active, displaying a table with 7 columns: *fio*, *number1*, *number2*, *number3*, *number4*, and *number5*. The first row of data shows the value '1' in the first column, 'Иванов И.И.' in the second column, and numerical values in the remaining columns.

	fio	number1	number2	number3	number4	number5
1	Иванов И.И.	123	1234	12345	123456	1234567



The screenshot shows a SQL query window with the following text:

```

select fio, column_name, number
from dbo.test_table_unpivot
unpivot (
    number for column_name in ([number1],[number2],[number3],[number4],[number5])
) as test_unpivot
  
```

Below the query window, the 'Results' tab is active, displaying a table with 5 rows and 3 columns: fio, column_name, and number.

	fio	column_name	number
1	Иванов И.И.	number1	123
2	Иванов И.И.	number2	1234
3	Иванов И.И.	number3	12345
4	Иванов И.И.	number4	123456
5	Иванов И.И.	number5	1234567

Где,

fio – столбец с ФИО, он в принципе не изменился;

column_name – псевдоним столбца, который будет содержать названия наших колонок;

number – псевдоним для значений из столбцов *number1*, *number2*...