### Лекция 14:

1. Aggregate window functions
2. Offset window functions
3. Функции и процедуры
4. CROSS APPLY

**Aggregate window functions**

Полезными для отображения статистики могут быть и, например, SUM и AVG. Например:

DECLARE @T TABLE (id int, val float)

INSERT INTO @T

VALUES (1, 2), (1, 3), (2, 4)

SELECT \*,

SUM(val) OVER(PARTITION BY id) СуммаВГруппе, SUM(val) OVER() СуммаВсего,

AVG(val) OVER(PARTITION BY id) СреднееВГруппе, AVG(val) OVER() СреднееВЦелом

FROM @T

Итог:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id | val | СуммаВГруппе | СуммаВсего | СреднееВГруппе | СреднееВЦелом |
| 1 | 2 | 5 | 9 | 2.5 | 3 |
| 1 | 3 | 5 | 9 | 2.5 | 3 |
| 2 | 4 | 4 | 9 | 4 | 3 |

СуммаВГруппе – сумма val внутри группы с одинаковыми значениями id (так как указан partition). СуммаВсего – сумма всех val для всей таблицы.

Понятно, что это всё Вы можете сделать и без использования aggregate window function, но согласитесь, что использование этих функций в некоторых случаях значительно упрощает написание запросов.

Кроме упомянутых SUM и AVG, Вы можете так же использовать прочие агрегатные функции: MIN, MAX, COUNT и т.д.

**Изменения в 2012 MS SQL Server**

С версии 2012 появились следующие улучшения, которые тоже можно использовать и писать при помощи них более простые внешне конструкции, которые Вы могли бы реализовать средствами стандартного SQL, но более длинным образом:

DECLARE @T TABLE (id int)

INSERT INTO @T VALUES (1), (2), (4), (5), (6)

SELECT id, SUM(id) OVER(ORDER BY id ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS runid

FROM @T

|  |  |
| --- | --- |
| id | runid |
| 1 | 1 |
| 2 | 3 |
| 4 | 7 |
| 5 | 12 |
| 6 | 18 |

SELECT id, SUM(id) OVER(ORDER BY id ROWS BETWEEN 2 PRECEDING AND 1 FOLLOWING) AS runid

FROM @T

|  |  |
| --- | --- |
| id | runid |
| 1 | 3 |
| 2 | 7 |
| 4 | 12 |
| 5 | 17 |
| 6 | 15 |

Общий синтаксис для оформления предложения ROWS:

ROWS BETWEEN

UNBOUNDED PRECEDING | <n> PRECEDING | <n> FOLLOWING | CURRENT ROW

AND

UNBOUNDED FOLLOWING | <n> PRECEDING |<n> FOLLOWING |CURRENT ROW

Обратите внимание, что вместо «ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW» Вы можете написать просто «ROWS UNBOUNDED PRECEDING»

Так же можно использовать предложение PARTITION (перед ORDER BY). Как простой пример:

DECLARE @T TABLE (id int, val int)

INSERT INTO @T VALUES (1, 1), (1, 2), (4, 1), (4, 2), (4, 3)

SELECT id, val, SUM(val) OVER(PARTITION BY id ORDER BY val ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS runval

FROM @T

|  |  |  |
| --- | --- | --- |
| id | val | runval |
| 1 | 1 | 3 |
| 1 | 2 | 3 |
| 4 | 1 | 3 |
| 4 | 2 | 6 |
| 4 | 3 | 5 |

Посмотрите, понятен ли Вам результат?

Кроме ROWS можно использовать RANGE. Но, в 2012 и 2014 для RANGE реализованы только UNBOUND и CURRENT ROW. Поэтому пока более подробно рассматривать не будем.

**Offset Functions**

**FIRST\_VALUE, LAST\_VALUE:**

Эти функции возвращают первое и последнее значение из отсортированного набора значений. Но с одной тонкостью. Если Вы используете синтаксис ниже, то получите результат как в таблице.

DECLARE @T TABLE (id int, val int)

INSERT INTO @T VALUES (1, 1), (1, 5), (2, 2), (2, 6), (4, 1)

SELECT id,

FIRST\_VALUE(val) OVER(ORDER BY val) f1, LAST\_VALUE (val) OVER(ORDER BY val) l1

FROM @T

ORDER BY id

|  |  |  |
| --- | --- | --- |
| id | f1 | l1 |
| 1 | 1 | 1 |
| 1 | 1 | 5 |
| 2 | 1 | 2 |
| 2 | 1 | 6 |
| 4 | 1 | 1 |

Если в случае FIRST всё понятно, Вы получили ровно то, что ожидали, то в случае в LAST кажется, что результат какой-то странный. На самом деле, так как FIRST и LAST поддерживают синтаксис оконных функций, то то, что сейчас написано подразумевает под собой синтаксис: «ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW». И если для FIRST – это то, что нужно, то для LAST это не так. Поэтому лучше всегда явно писать, в каком диапазоне Вы ищете первое и последнее значения:

SELECT id,

FIRST\_VALUE(val) OVER(ORDER BY val ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) f1,

LAST\_VALUE (val) OVER(ORDER BY val ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) l1

FROM @T

ORDER BY id

|  |  |  |
| --- | --- | --- |
| id | f1 | l1 |
| 1 | 1 | 6 |
| 1 | 1 | 6 |
| 2 | 1 | 6 |
| 2 | 1 | 6 |
| 4 | 1 | 6 |

Почему нужен FIRST и LAST, если в этом примере можно заменить на MIN и MAX? На этот вопрос можно ответить, посмотрев на следующий пример:

SELECT id,

FIRST\_VALUE(id) OVER(ORDER BY val ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) f1,

LAST\_VALUE (id) OVER(ORDER BY val ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) l1

FROM @T

ORDER BY id

|  |  |  |
| --- | --- | --- |
| id | f1 | l1 |
| 1 | 1 | 2 |
| 1 | 1 | 2 |
| 2 | 1 | 2 |
| 2 | 1 | 2 |
| 4 | 1 | 2 |

Сортировать мы можем по одной величине, а выводить другую, соответствующую максимальному или минимальному значению val.

Не забывайте, что PARTITION здесь так же можно использовать:

SELECT id,

FIRST\_VALUE(val) OVER(PARTITION BY id ORDER BY val ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS firstval,

LAST\_VALUE(val) OVER(PARTITION BY id ORDER BY val ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS lastval

FROM @T

|  |  |  |
| --- | --- | --- |
| id | firstval | lastval |
| 1 | 1 | 5 |
| 1 | 1 | 5 |
| 2 | 2 | 6 |
| 2 | 2 | 6 |
| 4 | 1 | 1 |

**LAG, LEAD**

Обе функции позволяют получить значение из N-й предыдущей или N-й следующей строки без использования оператора соединения. Синтаксис LAG(элемент, который планируется вернуть [, сдвиг, по умолчанию 1][, дефолтное значение, которое планируется вернуть, если нет строки по указанному сдвигу, по умолчанию NULL]). Аналогично для LEAD.

DECLARE @T TABLE (id int, val int)

INSERT INTO @T VALUES (1, 1), (1, 5), (2, 2), (2, 6), (4, 1)

SELECT id, val, LAG(val, 2, 10) OVER(ORDER BY id) lg, LEAD(val) OVER(ORDER BY id) ld

FROM @T

|  |  |  |  |
| --- | --- | --- | --- |
| id | val | lg | ld |
| 1 | 1 | 10 | 5 |
| 1 | 5 | 10 | 2 |
| 2 | 2 | 1 | 6 |
| 2 | 6 | 5 | 1 |
| 4 | 1 | 2 | NULL |

С использованием PARTITION:

SELECT id, val, LEAD(val) OVER (PARTITION BY id ORDER BY val) ld

FROM @T

|  |  |  |
| --- | --- | --- |
| id | val | ld |
| 1 | 1 | 5 |
| 1 | 5 | NULL |
| 2 | 2 | 6 |
| 2 | 6 | NULL |
| 4 | 1 | NULL |

Обратите внимание, что тот же результат Вы могли получить при помощи LEFT JOIN по определенному условию, но здесь Вы этот же результат получили более простым и коротким способом.

### Функции и процедуры.

Созданные функции и процедуры содержатся в дереве объектов в поддереве для каждой Вашей базы в пункте Programmability. Как и в прочих языках функции и процедуры используются, например, для инкапсуляции кода. Кроме того, использование этих объектов позволяет более правильно контролировать безопасность в БД, более просто отлаживать код.

### Процедуры:

**Пример процедуры без параметра:**

USE TEST

GO

CREATE PROCEDURE [dbo].[Товары\_по\_массе]

AS

SELECT Товар\_ID

FROM Товары

WHERE Масса > 100

GO

EXEC Товары\_по\_массе

**Пример процедуры с параметром:**

CREATE PROCEDURE [dbo].[Товары\_по\_массе\_параметр]

@param int

AS

SELECT Товар\_ID

FROM Товары

WHERE Масса > @param

GO

EXEC Товары\_по\_масса\_параметр 100

**Пример с дефолтными параметрами:**

CREATE PROC Колво\_товаров\_по\_массе\_деф\_параметр

@n AS INT,

@m AS float = 50,

@num AS INT OUTPUT

AS

SET @num =

(

SELECT COUNT(\*)

FROM Товары

WHERE Масса < @m AND Товар\_ID > @n

)

GO

Запуск:

DECLARE @count AS INT;

EXEC Колво\_товаров\_по\_массе\_деф\_параметр

@n = 1,

--@m = 10,

@num = @count OUTPUT;

SELECT @count AS numgoods;

Если оставить закомментированным значение параметра @m, то в процедуре будет использоваться дефолтное значение.

**Как воспользоваться результатами процедуры:**

-- как вставить результат выполнения процедуры во временную таблицу?

-- в обычную вставлять точно так же

CREATE TABLE #T(Товар\_ID int)

INSERT INTO #T

EXEC Товары\_по\_масса\_параметр 100

SELECT \*

FROM #T

DROP TABLE #T

ms-help://MS.SQLCC.v10/MS.SQLSVR.v10.en/s10de\_6tsql/html/afe3d86d-c9ab-44e4-b74d-4e3dbd9cc58c.htm

### Функции:

**Скалярная функция:** возвращает одно число (Scalar Functions)

Как создать:

-- функция-значение

CREATE FUNCTION dbo.f\_Товары\_по\_массе\_число()

RETURNS FLOAT AS

BEGIN

RETURN

(

SELECT MAX(Масса)

FROM Товары

)

END

GO

Как обратиться за результатом:

SELECT dbo.f\_Товары\_по\_массе\_число()

**Табличная функция:** результат – таблица, возможно из одного числа. (Multistatement Table-valued Functions)

-- табличная функция

CREATE FUNCTION dbo.f\_Товары\_по\_массе()

RETURNS @Товары TABLE (Товар\_ID int)

AS

BEGIN

INSERT INTO @Товары

SELECT Товар\_ID

FROM Товары

WHERE Масса > 100

RETURN

END

GO

--Получение значений

SELECT \*

FROM dbo.f\_Товары\_по\_массе()

В предложении FROM результат функции можно соединять с другими таблицами (функциями), достаточно задать псевдоним для функции.

**Функция с параметром** (параметров может быть несколько):

CREATE FUNCTION dbo.f\_Товары\_по\_массе\_параметр(@m int)

RETURNS @Товары TABLE (Товар\_ID int)

AS

BEGIN

INSERT INTO @Товары

SELECT Товар\_ID

FROM Товары

WHERE Масса > @m

RETURN

END

GO

Синтаксис проще: подходит, если то, что нужно вернуть формируется одним SELECT.

(Inline Table-Valued Functions)

CREATE FUNCTION dbo.f\_Товары\_по\_массе\_параметр(@m int)

RETURNS TABLE

AS

RETURN

SELECT Товар\_ID

FROM Товары

WHERE Масса > @m

GO

Обращаться:

SELECT \*

FROM f\_Товары\_по\_массе\_параметр (10)

Функция с параметром может быть и скалярной, а не только табличной.

Подробнее прочитать можно тут:

ms-help://MS.SQLCC.v10/MS.SQLSVR.v10.en/s10de\_6tsql/html/864b393f-225f-4895-8c8d-4db59ea60032.htm

### Views (вьюшки):

Views – так же пример табличного выражения, которое можно использовать многократно. После создания Вьюшка – это постоянная часть базы данных, и удалена она может быть только явно.

Пример создания:

CREATE VIEW dbo.v\_Товары\_по\_массе

AS

SELECT Товар\_ID

FROM Товары

WHERE Масса > 100

GO

Проверьте, что этот объект появился у Вас в разделе View (в базе, которая указана, как активная в текущий момент времени). При создании view нужно перечислять все поля, если Вы во view используете \*, то у Вас каждый раз при запуске view будут выдаваться поля, которые изначально были в таблице, на которую смотрит view. (изменить это можно запуском системной процедуры: sp\_refreshview).

Как обратиться к view:

SELECT \*

FROM dbo.v\_Товары\_по\_массе

Так как вьюшка – это реальный объект базы данных, то Вы можете выдать права только на неё, не выдавая права на объекты, которые используются в этой вьюшке.

Более подробно можно прочитать в хелпе:

ms-help://MS.SQLCC.v10/MS.SQLSVR.v10.en/s10de\_6tsql/html/aecc2f73-2ab5-4db9-b1e6-2f9e3c601fb9.htm

**Комментарий:**

Как проверять, существует ли объект (кроме триггеров)?

IF OBJECT\_ID (N'dbo.f\_test', N'FN') IS NOT NULL

DROP FUNCTION dbo.f\_test;

GO

Функция Object\_ID смотрит на системную View sys.objects, которую так же можно найти в поддереве каждой Вашей базы в разделе View. Указывать тип объекта необязательно.

Если Вы хотите проверить, существует ли временная таблица #, то нужно дополнительно указывать конкретную временную базу, где находятся временные процедуры.

SELECT OBJECT\_ID('tempdb..#mytemptable')

Базу tempdb каждый из Вас может найти в дереве объектов в разделе System Databases.

Более подробная информация про функцию Object\_ID (откуда она берет информацию и тд) содержится так же в хелпе.

*Совет:*

1. Описание создания процедур и функций приведено в лекции по минимуму, читайте хелп для MS SQL, там есть много примеров для всех случаев жизни.
2. Информация про префикс N так же есть в хелпе. Смотрите по поисковой строке “N prefix”.

### Apply. Cross Apply.

Apply Table Operator применяет выражение справа от оператора к каждой строке выражения слева от оператора. При логической интерпретации Apply нужно сначала оценивать выражение слева от оператора.

Пример: Создадим функцию, похожую на функцию в предыдущем разделе, только с параметром:

Пусть есть следующие таблицы:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | Товар\_ID | Масса | Цена | | 1 | 10 | 50 | | 2 | 15 | 100 | | 3 | 5 | 200 | | 4 | 1 | 20 | | 5 | 2 | 250 | | 6 | 50 | 10 | | 7 | 30 | 90 | | |  |  |  | | --- | --- | --- | | Магазин\_ID | Полок | МаксМасса | | 1 | 3 | 50 | | 2 | 4 | 20 | |

Необходимо для каждого магазина вывести набор подходящих товаров, при этом у магазина есть ограничение на колво уникальных артикулов, которое определяется колвом полок (то есть каждый товар должен лежать на своей полке: адресное хранение товара). Ваш алгоритм предполагает, что в этом случае нужно завозить самые дешевые товары (может быть это Ваша специфика, что в магазине более востребованы именно самые дешевые товары). Нужно реализовать набор товаров для каждого магазина.

CREATE FUNCTION dbo.f\_Товары\_по\_массе(@mass as INT, @n INT)

RETURNS TABLE

AS

RETURN

SELECT TOP (@n) WITH TIES Товар\_ID, Цена, Масса

FROM Товары

WHERE Масса <= @mass

ORDER BY Цена asc

GO

Теперь выполним следующий запрос:

SELECT \*

FROM Склады CROSS APPLY f\_Товары\_по\_массе(Склады.МаксМасса, Склады.Полок) Товары

ORDER BY Склады.Магазин\_ID, Товары.Цена desc

Результат будет следующим:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Магазин\_ID | Полок | МаксМасса | Товар\_ID | Цена | Масса |
| 1 | 3 | 50 | 1 | 50 | 10 |
| 1 | 3 | 50 | 4 | 20 | 1 |
| 1 | 3 | 50 | 6 | 10 | 50 |
| 2 | 4 | 20 | 3 | 200 | 5 |
| 2 | 4 | 20 | 2 | 100 | 15 |
| 2 | 4 | 20 | 1 | 50 | 10 |
| 2 | 4 | 20 | 4 | 20 | 1 |

В принципе для этой задачи не нужно было пользоваться APPLY, можно было то же самое реализовать при помощи подзапроса. Но насколько Вы повысите читаемость запроса, в случае если функция, выдающая нужный результат, будет достаточно большой.

Коммент: здесь есть небольшое несоответствие описанию, WITH TIES выдаст больше товаров, чем требуется по параметру @n, так как он выведет все товары из «хвоста» с одинаковой ценой. Но как правильнее – на Ваше усмотрение, либо Вы в этом случае должны ввести доп параметр для автоматического отбора, либо включить ручную работу по отбору более подходящего товара, либо сказать, что нам не нужна определенность, уберем WITH TIES.

Кроме CROSS APPLY есть еще OUTER APPLY, отличие их друг от друга такое же, как у INNER JOIN и LEFT JOIN. Если выражение справа для строки слева выдает пустое значение, то такая левая строка не появится в выводе. Чтобы она появилась нужно использовать OUTER APPLY.

Комментарий:

Запросы:

SELECT \*

FROM Товары CROSS JOIN Склады

SELECT \*

FROM Товары CROSS APPLY Склады

Выдадут один и тот же результат.

Где еще полезно использовать APPLY?

Пример: Есть уже знакомая Вам таблица «Продажи»:

|  |
| --- |
| **Продажи** |
| **ндок** |
| Дата |
| Покупатель\_ID |
| **Товар\_ID** |
| Колво |
| Цена |

Вы можете легко вывести ТОП-1 товар, дающий наибольший оборот фирме за один раз, например, вот так:

SELECT TOP 1 WITH TIES Товар\_ID

FROM Продажи

ORDER BY Колво\*Цена desc

Но если нужно вывести ТОП-1 товар для каждого покупателя, то запрос будет более сложным. Варианты могут быть следующими:

SELECT distinct Продажи.Покупатель\_ID, Продажи.Товар\_ID

FROM Продажи INNER JOIN

(

SELECT Покупатель\_ID, MAX(Колво\*Цена) МаксСтоим

FROM Продажи

GROUP BY Покупатель\_ID

) МаксТовар ON

Продажи.Покупатель\_ID = МаксТовар.Покупатель\_ID AND

Продажи.Колво\*Продажи.Цена = МаксСтоим

Или

SELECT distinct Продажи.Покупатель\_ID, Продажи.Товар\_ID

FROM Продажи

WHERE Товар\_ID in

(

SELECT TOP 1 WITH TIES Товар\_ID

FROM Продажи T

WHERE T.Покупатель\_ID = **Продажи.Покупатель\_ID**

ORDER BY Колво\*Цена desc

)

Где последний запрос будет работать медленнее, но легко модифицируется на случай, если нужно вывести 3 или 10 лучших товаров для каждого покупателя. Если модифицировать запрос в джойне, то запрос станет более сложным и более медленным.

Решение при помощи CROSS APPLY работает быстрее, но в этом случае лучше воспользоваться таблицей Покупатели, либо использовать distinct, чтобы каждый покупатель не вывелся столько раз, сколько покупок он совершил.

SELECT distinct Покупатели.Покупатель\_ID, Товары.Товар\_ID

FROM Покупатели CROSS APPLY

(

SELECT TOP 1 T.Товар\_ID

FROM Продажи T

WHERE T.Покупатель\_ID = **Покупатели.Покупатель\_ID**

ORDER BY Колво\*Цена desc

) AS Товары

Коммент: кроме того, сейчас Вы еще знаете и оконные функции. Функция ROW\_NUMBER тоже упростила бы решение подобной задачи.

**Использование CROSS APPLY и VALUES**. При помощи VALUES можно формировать список значений, которые можно использовать в WHERE или SELECT.

Пример:

DECLARE @T TABLE (id int, val int)

INSERT INTO @T

VALUES (1, 10), (2, 20), (3, 30), (4, 40)

SELECT \*

FROM @T T CROSS APPLY (VALUES (1), (2), (3), (4)) as Nums(Num)

В результате такого подзапроса каждая строка из таблицы T повториться 4 раза с разным значением Num.

Возможно и такое применение VALUES:

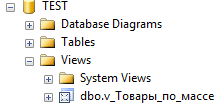
SELECT \*

FROM @T T CROSS APPLY (VALUES(val - 10, val\*5)) as Val(Sub10, Mult5)

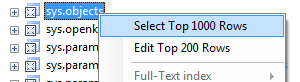
WHERE Sub10 > 10

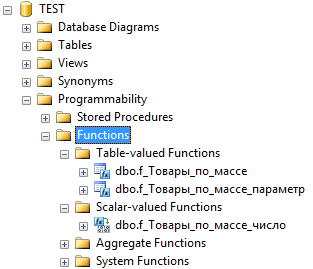
Здесь в Values мы формируем поле, к которому потом обращаемся в предложении WHERE. Это удобно, если мы не хотим писать подзапросы или использовать сложные конструкции с вычисляемыми полями.

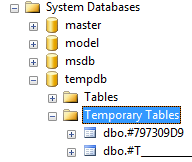
Где в Managements Studio найти вьюшки, процедуры и функции:

View: 

System view (sys.objects) – разверните дерево System Views (поддерево у узла Views)



Procedure, Function:

Временные таблицы (#): 

**Литература:**

1. Itzik Ben-Gan «Inside Microsoft SQL Server 2008: T-SQL Querying»
2. MSDN