

Лекция 11

Представления и табличные объекты

Создание, редактирование, изменение и удаление представления.

Команды CREATE VIEW, ALTER VIEW, DROP VIEW.

Обновляемое представление.

Табличные переменные. Временные локальные и глобальные таблицы.

Производные таблицы. Обобщенные табличные выражения (СТЕ).

Представления, или просмотры (VIEW), представляют собой временные, производные (иначе - виртуальные) таблицы и являются объектами базы данных, информация в которых не хранится постоянно, как в базовых таблицах, а формируется динамически при обращении к ним.

Обычные таблицы относятся к базовым, т.е. содержащим данные и постоянно находящимся на устройстве хранения информации.

Представление не может существовать само по себе, а определяется только в терминах одной или нескольких таблиц.

Применение представлений позволяет разработчику базы данных обеспечить каждому пользователю или группе пользователей наиболее подходящие способы работы с данными, что решает проблему простоты их использования и безопасности.

Содержимое представлений выбирается из других таблиц с помощью выполнения запроса, причем при изменении значений в таблицах данные в представлении автоматически меняются.

Представление - это фактически тот же запрос, который выполняется всякий раз при участии в какой-либо команде. *Результат выполнения этого запроса в каждый момент времени становится содержанием представления.* У пользователя создается впечатление, что он работает с настоящей, реально существующей таблицей.

У СУБД есть две возможности реализации представлений.

Если его определение простое, то система формирует каждую запись представления по мере необходимости, постепенно считывая исходные данные из базовых таблиц.

В случае сложного определения СУБД приходится сначала выполнить такую операцию, как материализация представления, *т.е. сохранить информацию, из которой состоит представление, во временной таблице.* Затем система приступает к выполнению пользовательской команды и формированию ее результатов, *после чего временная таблица удаляется.*

Представление - это предопределенный запрос, хранящийся в базе данных, который выглядит подобно обычной таблице и не требует для своего хранения дисковой памяти. *Для хранения представления используется только оперативная память. В отличие от других объектов базы данных представление не занимает дисковой памяти за исключением памяти, необходимой для хранения определения самого представления.*

Создания и изменения представлений в стандарте языка и реализации в MS SQL Server совпадают и представлены следующей командой:

```
<определение_представления> ::=  
    { CREATE| ALTER} VIEW имя_представления  
    [(имя_столбца [...n])]  
    [WITH ENCRYPTION]  
    AS SELECT_оператор  
    [WITH CHECK OPTION]
```

Рассмотрим назначение основных параметров. **По умолчанию имена столбцов в представлении соответствуют именам столбцов в исходных таблицах.**

Явное указание имени столбца требуется для вычисляемых столбцов или при объединении нескольких таблиц, имеющих столбцы с одинаковыми именами. Имена столбцов перечисляются через запятую, в соответствии с порядком их следования в представлении.

Параметр **WITH ENCRYPTION** предписывает серверу шифровать SQL-код запроса, что гарантирует невозможность его несанкционированного просмотра и использования. *Если при определении представления необходимо скрыть имена исходных таблиц и столбцов, а также алгоритм объединения данных, необходимо применить этот аргумент.*

Параметр **WITH CHECK OPTION** предписывает серверу исполнять проверку изменений, производимых через представление, **на соответствие критериям, определенным в операторе SELECT.** *Это означает, что не допускается выполнение изменений, которые приведут к исчезновению строки из представления. Такое случается, если для представления установлен горизонтальный фильтр и изменение данных приводит к несоответствию строки установленным фильтрам.*

Использование аргумента **WITH CHECK OPTION** *гарантирует, что сделанные изменения будут отображены в представлении.*

Если пользователь пытается выполнить изменения, приводящие к исключению строки из представления, при заданном аргументе **WITH CHECK OPTION** сервер выдаст сообщение об ошибке и все изменения будут отклонены.

Обращение к представлению осуществляется с помощью оператора **SELECT** как к обычной таблице

SELECT * FROM view1

Представление можно использовать в команде так же, как и любую другую таблицу.

К представлению можно строить запрос, модифицировать его (если оно отвечает определенным требованиям), соединять с другими таблицами. Содержание представления не фиксировано и обновляется каждый раз, когда на него ссылаются в команде. Представления значительно расширяют возможности управления данными.

В частности, это прекрасный способ разрешить доступ к информации в таблице, скрыв часть данных.

Представления или Views представляют виртуальные таблицы. Но в отличие от обычных стандартных таблиц в базе данных представления содержат запросы, которые динамически извлекают используемые данные.

Представления дают нам ряд преимуществ. Они упрощают комплексные SQL-операции. Они защищают данные, так как представления могут дать доступ к части таблицы, а не ко всей таблице. Представления также позволяют возвращать отформатированные значения из таблиц в нужной и удобной форме.

***CREATE VIEW название_представления [(столбец_1, столбец_2,)]
AS выражение_SELECT***

Рассмотрим пример, ранее созданную БД

```
3 CREATE TABLE Products
4 (
5     Id INT IDENTITY PRIMARY KEY,
6     ProductName NVARCHAR(30) NOT NULL,
7     Manufacturer NVARCHAR(20) NOT NULL,
8     ProductCount INT DEFAULT 0,
9     Price MONEY NOT NULL
10 );
11 CREATE TABLE Customers
12 (
13     Id INT IDENTITY PRIMARY KEY,
14     FirstName NVARCHAR(30) NOT NULL
15 );
16 CREATE TABLE Orders
17 (
18     Id INT IDENTITY PRIMARY KEY,
19     --ON DELETE CASCADE используется для автоматического удаления строк
20     --из дочерней таблицы при удалении строк из родительской таблицы
21     ProductId INT NOT NULL REFERENCES Products(Id) ON DELETE CASCADE,
22     CustomerId INT NOT NULL REFERENCES Customers(Id) ON DELETE CASCADE,
23     CreatedAt DATE NOT NULL,
24     ProductCount INT DEFAULT 1,
25     Price MONEY NOT NULL
26 );
```

```

1 /***** Скрипт для команды SelectTopNRows из среды SSMS *****/
2 SELECT TOP (1000) [Id]
3     , [ProductId]
4     , [CustomerId]
5     , [CreatedAt]
6     , [ProductCount]
7     , [Price]
8 FROM [testbasa].[dbo].[Orders]

```

Id	ProductId	CustomerId	CreatedAt	ProductCount	Price
1	4	1	2017-07-11	2	46000.00
2	2	1	2017-07-13	1	41000.00
3	2	2	2017-07-11	1	41000.00

```

1 /***** Скрипт для команды SelectTopNRows из среды SSMS *****/
2 SELECT TOP (1000) [Id]
3     , [ProductName]
4     , [Manufacturer]
5     , [ProductCount]
6     , [Price]
7 FROM [testbasa].[dbo].[Products]

```

Id	ProductName	Manufacturer	ProductCount	Price
1	iPhone 6	Apple	2	36000.00
2	iPhone 6S	Apple	2	41000.00
3	iPhone 7	Apple	5	52000.00
4	Galaxy S8	Samsung	2	46000.00
5	Galaxy S8 Plus	Samsung	1	56000.00
6	Mi 5X	Xiaomi	2	26000.00
7	OnePlus 5	OnePlus	6	38000.00

```

1 /***** Скрипт для команды SelectTopNRows из среды SSMS ***
2 SELECT TOP (1000) [Id]
3     , [FirstName]
4 FROM [testbasa].[dbo].[Customers]

```

Id	FirstName
1	Tom
2	Bob
3	Sam

При создании представлений следует учитывать, что представления, как и таблицы, должны иметь уникальные имена в рамках той же базы данных.

Представления могут иметь не более 1024 столбцов и могут обращаться не более чем к 256 таблицам.

Также можно создавать представления на основе других представлений. Такие представления еще называют вложенными (nested views). Однако уровень вложенности не может быть больше 32-х.

*Команда **SELECT**, используемая в представлении, не может включать выражения **INTO** или **ORDER BY** (за исключением тех случаев, когда также применяется выражение **TOP** или **OFFSET**).*

*Если же необходима сортировка данных в представлении, то выражение **ORDER BY** применяется в команде **SELECT**, которая извлекает данные из представления.*

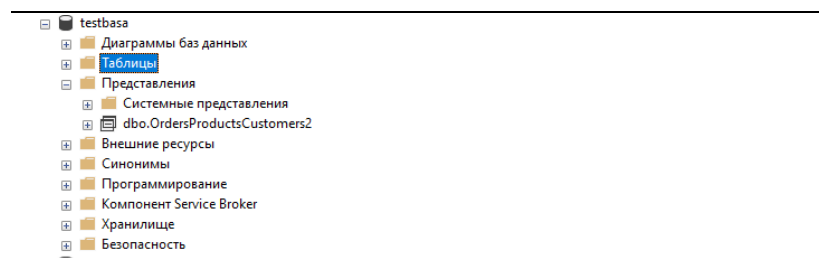
Также при создании представления можно определить набор его столбцов

```
1 USE testbasa;
2
3 CREATE VIEW OrdersProductsCustomers2 (OrderDate, Customer, Product)
4 AS SELECT Orders.CreatedAt,
5           Customers.FirstName,
6           Products.ProductName
7 FROM Orders INNER JOIN Products ON Orders.ProductId = Products.Id
8 INNER JOIN Customers ON Orders.CustomerId = Customers.Id
```

Сообщения

Выполнение команд успешно завершено.

Время выполнения: 2023-04-21T00:06:11.2778631+03:00



```
1 /***** Скрипт для команды SelectTopNRows из среды SSMS *****/
2 SELECT TOP (1000) [OrderDate]
3                , [Customer]
4                , [Product]
5 FROM [testbasa].[dbo].[OrdersProductsCustomers2]
```

152 %

Результаты

	OrderDate	Customer	Product
1	2017-07-11	Tom	Galaxy S5
2	2017-07-13	Tom	iPhone 6S
3	2017-07-11	Bob	iPhone 6S

Для изменения представления используется команда **ALTER VIEW**. Эта команда имеет практически тот же самый синтаксис, что и **CREATE VIEW**:

ALTER VIEW название представления [(столбец 1, столбец 2,)]
AS выражение **SELECT**

Например, изменим выше созданное представление **OrdersProductsCustomers**

```
12 |
13 Alter View OrdersProductsCustomers2
14 AS SELECT Orders.CreatedAt AS OrderDate,
15         Customers.FirstName AS Customer,
16         Products.ProductName AS Product,
17         Products.Manufacturer AS Manufacturer
18 FROM Orders INNER JOIN Products ON Orders.ProductId = Products.Id
19 INNER JOIN Customers ON Orders.CustomerId = Customers.Id
20
21
```

Сообщения

Выполнение команд успешно завершено.

Время выполнения: 2023-04-21T01:32:34.9854221+03:00

Представления

- Системные представления
- dbo.OrdersProductsCustomers2**
 - Столбцы
 - OrderDate (date, He NULL)
 - Customer (nvarchar(30), He NULL)
 - Product (nvarchar(30), He NULL)
 - Manufacturer (nvarchar(20), He NULL)
 - Триггеры
 - Индексы
 - Статистика

Скрипт для команды selecttopnrows из среды SSMS

```
1 /
2 SELECT TOP (1000) [OrderDate]
3         , [Customer]
4         , [Product]
5         , [Manufacturer]
6 FROM [testbasa].[dbo].[OrdersProductsCustomers2]
```

52 %

Результаты | Сообщения

	OrderDate	Customer	Product	Manufacturer
1	2017-07-11	Tom	Galaxy S8	Samsung
2	2017-07-13	Tom	iPhone 6S	Apple
3	2017-07-11	Bob	iPhone 6S	Apple

Удаление представления

Для удаления представления вызывается команда **DROP VIEW**:

DROP VIEW OrdersProductsCustomers

Также стоит отметить, что при удалении таблиц также следует удалить и представления, которые используют эти таблицы

Представления могут быть обновляемыми (updatable). В таких представлениях мы можем изменить или удалить строки или добавить в них новые строки.

При создании подобных представлений есть множество ограничений.

В частности, команда *SELECT* при создании обновляемого представления не может содержать:

TOP

DISTINCT

UNION

JOIN

агрегатные функции типа *COUNT* или *MAX*

GROUP BY и *HAVING*

подзапросы

производные столбцы или столбцы, которые вычисляются на основании нескольких значений

обращения одновременно к нескольким таблицам

Стоит отметить, что это касается именно обновляемого представления.

Например, для создания обычного представления мы можем использовать в команде *SELECT* оператор *JOIN*, однако такое представление не будет обновляемым.

При попытке его обновить, мы будем получать ошибку вида "*View or function название_представления is not updatable because the modification affects multiple base tables.*"

Допустим, у нас есть следующая таблица [Products]

	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	2	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	Mi 5X	Xiaomi	2	26000,00
7	7	OnePlus 5	OnePlus	6	38000,00

```
21 --обновляемое представление
22 CREATE VIEW ProductView
23 AS SELECT ProductName AS Product, Manufacturer, Price
24 FROM Products
```

Сообщения

Исполнение команд успешно завершено.

Время выполнения: 2023-04-21T06:41:55.8573495+03:00

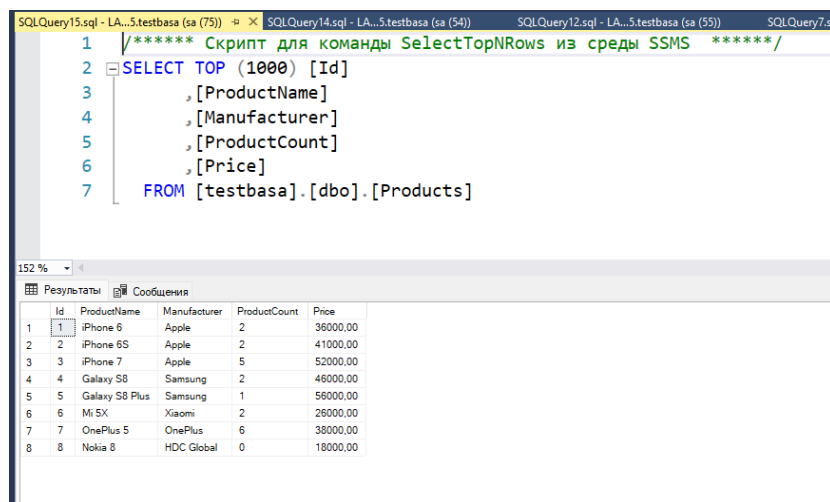
- dbo.table_name
- Представления
 - Системные представления
 - dbo.OrdersProductsCustomers2
 - dbo.ProductView
 - dbo.View_1
 - dbo.View_2
- Внешние ресурсы
- Синонимы

Добавим данные:

```
25
26 --добавим данные
27 INSERT INTO ProductView (Product, Manufacturer, Price)
28 VALUES('Nokia 8', 'HDC Global', 18000)
29
30 SELECT * FROM ProductView
```

	Product	Manufacturer	Price
1	iPhone 6	Apple	36000,00
2	iPhone 6S	Apple	41000,00
3	iPhone 7	Apple	52000,00
4	Galaxy S8	Samsung	46000,00
5	Galaxy S8 Plus	Samsung	56000,00
6	Mi 5X	Xiaomi	26000,00
7	OnePlus 5	OnePlus	38000,00
8	Nokia 8	HDC Global	18000,00

*Стоит отметить, что при добавлении фактически будет добавлен объект в таблицу **Products**, которую использует представление **ProductView**. И поэтому надо учитывать, что если в этой таблице есть какие-либо столбцы, в которые представление не добавляет данные, но которые не допускают значение **NULL** или не поддерживают значение по умолчанию, то добавление завершится с ошибкой.*



The screenshot shows a SQL Server Enterprise Manager window with a query editor and a results pane. The query editor contains a SQL script for selecting the top 1000 rows from the Products table. The results pane displays a table with 8 rows of data.

```
1 /***** Скрипт для команды SelectTopNRows из среды SSMS *****/
2 SELECT TOP (1000) [Id]
3     ,[ProductName]
4     ,[Manufacturer]
5     ,[ProductCount]
6     ,[Price]
7 FROM [testbasa].[dbo].[Products]
```

	Id	ProductName	Manufacturer	ProductCount	Price
1	1	iPhone 6	Apple	2	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	Mi 5X	Xiaomi	2	26000.00
7	7	OnePlus 5	OnePlus	6	38000.00
8	8	Nokia 8	HDC Global	0	18000.00

Обновление строки представления:

```
31
32 --обновление данных строки
33 UPDATE ProductView
34 SET Price= 15000 WHERE Product='Nokia 8'
35 SELECT * FROM ProductView
```

Результаты

Product	Manufacturer	Price
iPhone 6	Apple	36000,00
iPhone 6S	Apple	41000,00
iPhone 7	Apple	52000,00
Galaxy S8	Samsung	46000,00
Galaxy S8 Plus	Samsung	56000,00
Mi 5X	Xiaomi	26000,00
OnePlus 5	OnePlus	38000,00
Nokia 8	HDC Global	15000,00

Удаление строки в представлении

```
37 --удаление
38 DELETE FROM ProductView
39 WHERE Product='Nokia 8'
40 SELECT * FROM ProductView
```

Результаты

Product	Manufacturer	Price
iPhone 6	Apple	36000,00
iPhone 6S	Apple	41000,00
iPhone 7	Apple	52000,00
Galaxy S8	Samsung	46000,00
Galaxy S8 Plus	Samsung	56000,00
Mi 5X	Xiaomi	26000,00
OnePlus 5	OnePlus	38000,00

Изменения также происходят и в табл. Products

```
1 /***** Скрипт для команды SelectTopNRows из среды SSMS *****/
2 SELECT TOP (1000) [Id]
3     , [ProductName]
4     , [Manufacturer]
5     , [ProductCount]
6     , [Price]
7 FROM [testbasa].[dbo].[Products]
```

Результаты

Id	ProductName	Manufacturer	ProductCount	Price
1	iPhone 6	Apple	2	36000,00
2	iPhone 6S	Apple	2	41000,00
3	iPhone 7	Apple	5	52000,00
4	Galaxy S8	Samsung	2	46000,00
5	Galaxy S8 Plus	Samsung	1	56000,00
6	Mi 5X	Xiaomi	2	26000,00
7	OnePlus 5	OnePlus	6	38000,00

ВЫВОДЫ

Обновление данных в представлениях

Не все представления в SQL могут быть модифицированы.

Модифицируемое представление определяется следующими критериями:

- основывается только на одной базовой таблице;
- содержит первичный ключ этой таблицы;
- не содержит **DISTINCT** в своем определении;
- не использует **GROUP BY** или **HAVING** в своем определении;
- по возможности не применяет в своем определении подзапросы;
- не использует константы или выражения значений среди выбранных полей вывода;
- в просмотр должен быть включен каждый столбец таблицы, имеющий атрибут **NOT NULL** ;
- оператор **SELECT** просмотра не использует агрегирующие (итоговые) функции, соединения таблиц, хранимые процедуры и функции, определенные пользователем;
- основывается на одиночном запросе, поэтому объединение **UNION** не разрешено.

Если просмотр удовлетворяет этим условиям, к нему могут применяться операторы **INSERT, UPDATE, DELETE**.

Различия между модифицируемыми представлениями и представлениями, предназначенными только для чтения, не случайны.

Цели, для которых их используют, различны.

С модифицируемыми представлениями в основном обходятся точно так же, как и с базовыми таблицами.

Фактически, пользователи не могут даже осознать, является ли объект, который они запрашивают, базовой таблицей или представлением, т.е. прежде всего это средство защиты для сокрытия конфиденциальных или не относящихся к потребностям данного пользователя частей таблицы.

Представления в режиме <только для чтения> позволяют получать и форматировать данные более рационально. Они создают целый арсенал сложных запросов, которые можно выполнить и повторить снова, сохраняя полученную информацию. Результаты этих запросов могут затем использоваться в других запросах, что позволит избежать сложных предикатов и снизить вероятность ошибочных действий.

Преимущества и недостатки представлений

Механизм представления - мощное средство СУБД, позволяющее скрыть реальную структуру БД от некоторых пользователей за счет определения представлений.

Любая реализация представления должна гарантировать, что состояние представляемого отношения точно соответствует состоянию данных, на которых определено это представление.

Обычно вычисление представления производится каждый раз при его использовании.

Когда представление создается, информация о нем записывается в каталог БД под собственным именем.

Любые изменения в данных адекватно отобразятся в представлении - в этом его отличие от очень похожего на него запроса к БД.

В то же время запрос представляет собой как бы <мгновенную фотографию> данных и при изменении последних запрос к БД необходимо повторить.

Наличие представлений в БД необходимо для обеспечения логической независимости данных.

Если система **обеспечивает физическую независимость** данных, то изменения в физической структуре БД не влияют на работу пользовательских программ.

Логическая независимость подразумевает тот факт, что при изменении логической структуры данных влияние на пользовательские программы также не оказывается, а значит, система должна уметь решать проблемы, связанные с ростом и реструктуризацией БД. *Очевидно, что с увеличением количества данных, хранимых в БД, возникает необходимость ее расширения за счет добавления новых атрибутов или отношений - это называется ростом БД.*

Реструктуризация данных подразумевает сохранение той же самой информации, но изменяется ее расположение, например, за счет перегруппировки атрибутов в отношениях.

Предположим, некоторое отношение в силу каких-либо причин необходимо разделить на два. Соединение полученных отношений в представлении воссоздает исходное отношение, а у пользователя складывается впечатление, что никакой реструктуризации не производилось. **Помимо решения проблемы реструктуризации представление можно применять для просмотра одних и тех же данных разными пользователями и в различных вариантах. С помощью представлений пользователь имеет возможность ограничить объем данных для удобства работы.** Наконец, механизм представлений позволяет скрыть служебные данные, не интересные пользователям.

В случае выполнения СУБД на отдельно стоящем персональном компьютере использование представлений обычно имеет целью лишь упрощение структуры запросов к базе данных. Однако в случае многопользовательской сетевой СУБД представления играют ключевую роль

в определении структуры базы данных и организации защиты информации. Рассмотрим основные преимущества применения представлений в подобной среде.

Независимость от данных

С помощью представлений можно создать согласованную, неизменную картину структуры базы данных, которая будет оставаться стабильной даже в случае изменения формата исходных таблиц (например, добавления или удаления столбцов, изменения связей, разделения таблиц, их реструктуризации или переименования).

Если в таблицу добавляются или из нее удаляются не используемые в представлении столбцы, то изменять определение этого представления не потребуется.

Если структура исходной таблицы переупорядочивается или таблица разделяется, можно создать представление, позволяющее работать с виртуальной таблицей прежнего формата. В случае разделения исходной таблицы, прежний формат может быть виртуально воссоздан с помощью представления, построенного на основе соединения вновь созданных таблиц - конечно, если это окажется возможным. Последнее условие можно обеспечить с помощью помещения во все вновь созданные таблицы первичного ключа прежней таблицы.

Актуальность

Изменения данных в любой из таблиц базы данных, указанных в определяющем запросе, немедленно отображается на содержимом представления.

Повышение защищенности данных

Права доступа к данным могут быть предоставлены исключительно через ограниченный набор представлений, содержащих только то подмножество данных, которое необходимо пользователю. Подобный подход позволяет существенно ужесточить контроль за доступом отдельных категорий пользователей к информации в базе данных.

Снижение стоимости

Представления позволяют упростить структуру запросов за счет объединения данных из нескольких таблиц в единственную виртуальную таблицу. В результате многотабличные запросы сводятся к простым запросам к одному представлению.

Дополнительные удобства

Создание представлений может обеспечивать пользователей дополнительными удобствами - например, возможностью работы только с действительно нужной частью данных. В результате можно добиться максимального упрощения той модели данных, которая понадобится каждому конечному пользователю.

Возможность настройки

Представления являются удобным средством настройки индивидуального образа базы данных. В результате одни и те же таблицы могут быть предъявлены пользователям в совершенно разном виде.

Обеспечение целостности данных

Если в операторе **CREATE VIEW** будет указана фраза **WITH CHECK OPTION**, то СУБД станет осуществлять контроль за тем, чтобы в исходные таблицы базы данных не была введена ни одна из строк, не удовлетворяющих предложению **WHERE** в определяющем запросе. Этот механизм гарантирует целостность данных в представлении.

Практика ограничения доступа некоторых пользователей к данным посредством создания специализированных представлений, безусловно, имеет значительные преимущества перед предоставлением им прямого доступа к таблицам базы данных.

Однако использование представлений в среде SQL не лишено недостатков.

Ограниченные возможности обновления

В некоторых случаях представления не позволяют вносить изменения в содержащиеся в них данные.

Структурные ограничения

Структура представления устанавливается в момент его создания.

Если определяющий запрос представлен в форме SELECT * FROM_, то символ * ссылается на все столбцы, существующие в исходной таблице на момент создания представления. Если впоследствии в исходную таблицу базы данных добавятся новые столбцы, то они не появятся в данном представлении до тех пор, пока это представление не будет удалено и вновь создано.

Снижение производительности

Использование представлений связано с определенным снижением производительности. В одних случаях влияние этого фактора совершенно незначительно, тогда как в других оно может послужить источником существенных проблем. Например, представление, определенное с помощью сложного многотабличного запроса, может потребовать значительных затрат времени на обработку, поскольку при его разрешении потребуется выполнять соединение таблиц всякий раз, когда понадобится доступ к данному представлению. Выполнение разрешения представлений связано с использованием дополнительных *вычислительных ресурсов*.

Временные таблицы

Обычные таблицы не всегда подходят для хранения временных данных. Представьте, что у вас есть информация, которая должна быть доступна только для текущей сессии (или даже для текущего пакета). Это могут быть данные, которые нужны во время вычислительных операций.

*Специально для таких случаев в SQL Server предусмотрено **три вида временных таблиц**: локальные, глобальные и табличные переменные.*

Локальные таблицы

Имена локальных временных таблиц должны начинаться со знака решетки, например #T1. Их содержимое (как и содержимое остальных двух видов временных таблиц) хранится в БД *tempdb*.

Локальные временные таблицы видны только в контексте сессии, в которой они были созданы; доступ к ним можно получить на том уровне, где находится их определение, и далее по стеку вызовов (в том числе внутри вложенных процедур, функций, триггеров и динамических пакетов). Они автоматически уничтожаются, когда текущий уровень выходит за рамки пространства имен.

Представьте, что у нас есть цепочка вызовов хранимых процедур, начиная с Proc1 и заканчивая Proc4.

В процедуре Proc2 перед вызовом Proc3 создается локальная временная таблица под названием #T1. Она будет доступна процедурам Proc2, Proc3 и Proc4, но не Proc1;

SQL Server удалит ее сразу же, как только процедура Proc2 завершит свою работу.

Таблица, которая создается в произвольном пакете на самом верхнем уровне вложенности (то есть когда функция @@NESTLEVEL возвращает 0), доступна для всех последующих пакетов; ее удаление будет выполнено только после отключения сессии, в которой она была создана.

Вам, наверное, интересно, каким образом SQL Server предотвращает конфликт имен, когда две сессии создают локальные временные таблицы с одинаковыми названиями. Все очень просто: к имени каждой таблицы автоматически добавляется суффикс, который обеспечивает ее уникальность в рамках БД *tempdb*. Одним из очевидных способов применения временных таблиц является хранение промежуточных результатов (например, во время выполнения цикла) с возможностью последующего доступа к ним.

Иногда возникает необходимость многократного обращения к данным, которые представляют собой результаты ресурсоемких вычислений.

Допустим, вам нужно соединить таблицы Sales.Orders и Sales.OrderDetails, вычислить, какое количество товара было заказано в каждом году, затем попарно объединить полученные результаты для сравнения годовых показателей. В нашей БД таблицы Orders и OrderDetails имеют очень маленький объем, но в реальных условиях количество записей может измеряться миллионами. Мы могли бы воспользоваться табличными выражениями, но, как вы помните, они являются сугубо виртуальными объектами; следовательно, вся работа по сканированию данных, соединению таблиц и агрегированию будет выполнена два раза. Чтобы не повторять все эти ресурсоемкие операции, результаты проще сохранить в локальной временной таблице; после этого достаточно взять два экземпляра имеющихся у нас данных и выполнить соединение.

Глобальные таблицы

Глобальная временная таблица доступна для всех сессий и обозначается с помощью двойного знака решетки, например ##T1. Ее удаление происходит в момент отключения сессии, в которой она была создана (при условии отсутствия внешних ссылок).

Глобальные временные таблицы используются для хранения общедоступных промежуточных данных. Для обращения к ним (в том числе с помощью элементов DDL и DML) не требуется никаких специальных полномочий. Конечно, это означает, что любой пользователь может изменять или даже удалять их содержимое, поэтому прежде чем использовать их в своих запросах, следует взвесить все за и против.

Следующий код создает глобальную временную таблицу под названием ##Globals, которая содержит столбцы id и val.

```
CREATE TABLE dbo.##Globals (  
id sysname NOT NULL PRIMARY KEY, val SQL_VARIANT NOT NULL  
);
```

Эта таблица выполняет те же функции, что и глобальные переменные. Столбцы id и val имеют типы данных SYSNAME (используется для внутреннего представления идентификаторов) и SQL_VARIANT (универсальный тип, совместимый практически с любыми данными).

Доступ к данной таблице открыт для всех. Например, следующий код добавляет в нее строку, которая представляет переменную i, и присваивает ей целочисленное значение 10.

```
INSERT INTO dbo.##Globals(id, val) VALUES(N'i', CAST(10 AS INT));
```

Вы можете также изменять и извлекать данные из этой таблицы. Запустите следующий код в любой сессии, чтобы получить текущее значение переменной i.

Не стоит забывать, что глобальная временная таблица автоматически уничтожается сразу после отключения сессии, в которой она была создана (при условии отсутствия внешних ссылок).

Если вы хотите, чтобы глобальная временная таблица создавалась при каждом запуске SQL Server и была доступна после завершения работы, вам необходимо использовать хранимую процедуру, помеченную параметром startup

Чтобы вручную удалить глобальную временную таблицу, откройте любую сессию и выполните следующий код.

```
DROP TABLE dbo.##Globals
```


Табличные переменные

Табличные переменные, которые объявляются с помощью уже знакомой вам команды **DECLARE**, имеют много общего с локальными временными таблицами.

Как и другие виды временных таблиц, табличные переменные физически хранятся в БД **tempdb**. Их область видимости ограничена не только сессией, в которой они были созданы, но и текущим пакетом. При этом доступ к ним закрыт даже для вложенных и всех последующих пакетов, объявленных в рамках сессии.

При откате транзакции, которая была объявлена вручную, отменяются все изменения, внесенные ею во временные таблицы; однако в случае с табличными переменными отменяется только действие последней команды, которая завершилась ошибкой или не успела выполняться до конца.

Временные таблицы и табличные переменные отличаются в плане оптимизации. Достаточно сказать, что табличные переменные обычно работают быстрее с минимальными объемами данных (в несколько строк); во всех остальных случаях временные таблицы показывают более высокую производительность.

Табличные переменные (table variable) позволяют сохранить содержимое целой таблицы. Формальный синтаксис определения подобной переменной во многом похож на создание таблицы:

DECLARE @табличная_переменная TABLE
(столбец_1 тип_данных [атрибуты_столбца],
столбец_2 тип_данных [атрибуты_столбца])
[атрибуты_таблицы]

Например:

DECLARE @ABrends TABLE (ProductId INT, ProductName NVARCHAR(20))

В данном случае переменная @ABrends будет содержать два столбца.

В дальнейшем мы сможем работать с этой переменной как с обычной таблицей, то есть добавлять в нее данные, изменять, удалять и извлекать их:

```
42  
43 --Табличные переменные  
44  
45 DECLARE @ABrends TABLE (ProductId INT, ProductName NVARCHAR(20))  
46  
47 INSERT INTO @ABrends  
48 VALUES(1, 'iPhone 8'),  
49 (2, 'Samsung Galaxy S8')
```

152 %

Результаты		Сообщения	
	ProductId	ProductName	
1	1	iPhone 8	
2	2	Samsung Galaxy S8	

Временные локальные и глобальные таблицы

В дополнение к табличным переменным можно определять временные таблицы. Такие таблицы могут быть полезны для хранения табличных данных внутри сложного комплексного скрипта.

Временные таблицы существуют на протяжении сессии базы данных. Если такая таблица создается в редакторе запросов (Query Editor) в SQL Server Management Studio, то таблица будет существовать пока открыт редактор запросов. Таким образом, к временной таблице можно обращаться из разных скриптов внутри редактора запросов.

После создания все временные таблицы сохраняются в таблице *tempdb*, которая имеется по умолчанию в MS SQL Server.

Если необходимо удалить таблицу до завершения сессии базы данных, то для этой таблицы следует выполнить команду DROP TABLE.

Название временной таблицы начинается со знака решетки #. Если используется один знак #, то создается локальная таблица, которая доступна в течение текущей сессии. Если используются два знака ##, то создается глобальная временная таблица. В отличие от локальной глобальная временная таблица доступна всем открытым сессиям базы данных.

```
53
54 --Например, создадим локальную временную таблицу
55 CREATE TABLE #ProductSummary
56 (ProdId INT IDENTITY,
57  ProdName NVARCHAR(20),
58  Price MONEY)
59
60 INSERT INTO #ProductSummary
61 VALUES ('Nokia 8', 18000),
62         ('iPhone 8', 56000)
63
64 SELECT * FROM #ProductSummary
65
66
```

% ▾

Результаты Сообщения

ProdId	ProdName	Price
1	Nokia 8	18000,00
2	iPhone 8	56000,00

*И с этой таблицей можно работать в большей степени как и с обычной таблицей - получать данные, добавлять, изменять и удалять их. **Только после закрытия редактора запросов эта таблица перестанет существовать.***

Подобные таблицы удобны для каких-то временных промежуточных данных.

Например, пусть у нас есть три таблицы, рассмотренные ранее в лекции

```

1 /***** Скрипт для команды SelectTopNRows из среды SSMS *****/
2 SELECT TOP (1000) [Id]
3     , [ProductId]
4     , [CustomerId]
5     , [CreatedAt]
6     , [ProductCount]
7     , [Price]
8 FROM [testbasa].[dbo].[Orders]

```

Id	Products	CustomerId	CreatedAt	ProductCount	Price
1	4	1	2017-07-11	2	46000.00
2	2	1	2017-07-13	1	41000.00
3	2	2	2017-07-11	1	41000.00

```

1 /***** Скрипт для команды SelectTopNRows из среды SSMS *****/
2 SELECT TOP (1000) [Id]
3     , [ProductName]
4     , [Manufacturer]
5     , [ProductCount]
6     , [Price]
7 FROM [testbasa].[dbo].[Products]

```

Id	ProductName	Manufacturer	ProductCount	Price
1	iPhone 6	Apple	2	36000.00
2	iPhone 6S	Apple	2	41000.00
3	iPhone 7	Apple	5	52000.00
4	Galaxy S8	Samsung	2	46000.00
5	Galaxy S8 Plus	Samsung	1	56000.00
6	Mi 5X	Xiaomi	2	26000.00
7	OnePlus 5	OnePlus	6	38000.00

```

1 /***** Скрипт для команды SelectTopNRows из среды SSMS ***
2 SELECT TOP (1000) [Id]
3     , [FirstName]
4 FROM [testbasa].[dbo].[Customers]

```

Id	FirstName
1	Tom
2	Bob
3	Sam

```

65 --Выведен во временную таблицу промежуточные данные из таблицы Orders
66 SELECT ProductId,
67     SUM(ProductCount) AS TotalCount,
68     SUM(ProductCount * Price) AS TotalSum
69 INTO #OrdersSummary
70 FROM Orders
71 GROUP BY ProductId
72
73
74 SELECT Products.ProductName, #OrdersSummary.TotalCount, #OrdersSummary.TotalSum
75 FROM Products
76 JOIN #OrdersSummary ON Products.Id = #OrdersSummary.ProductId

```

ProductName	TotalCount	TotalSum
iPhone 6S	2	82000.00
Galaxy S8	2	92000.00

Здесь вначале извлекаются данные во временную таблицу **#OrdersSummary**. Причем так как данные в нее извлекаются с помощью выражения **SELECT INTO**, то предварительно таблицу не надо создавать. И эта таблица будет содержать id товара, общее количество проданного товара и на какую сумму был продан товар.

Затем эта таблица может использоваться в выражениях **INNER JOIN**.

Подобным образом определяются глобальные временные таблицы, единственное, что их имя начинается с двух знаков ##:

```
//
78  --глобальные временные таблицы
79  CREATE TABLE ##OrderDetails
80  (ProductId INT, TotalCount INT, TotalSum MONEY)
81
82  INSERT INTO ##OrderDetails
83  SELECT ProductId, SUM(ProductCount), SUM(ProductCount * Price)
84  FROM Orders
85  GROUP BY ProductId
86
87  SELECT * FROM ##OrderDetails
```

152 %

Результаты

Сообщения

	ProductId	TotalCount	TotalSum
1	2	2	82000,00
2	4	2	92000,00

Обобщенные табличные выражения

Кроме временных таблиц MS SQL Server позволяет создавать **обобщенные табличные выражения (common table expression или CTE)**, которые являются производными от обычного запроса и в плане производительности являются более эффективным решением, чем временные.

Обобщенное табличное выражение задается с помощью ключевого слова WITH

Обобщенные табличные выражения (ОТВ)

ОТВ очень похожи на производные таблицы, но имеют несколько важных преимуществ.

```
WITH <имя_выражения>[(<список_задействованных_столбцов>)]
AS(
    <внутренний_запро_определяющий_выражение>
)
<запрос_к_выражению>;
```

Внутренний запрос должен соответствовать всем требованиям, которые предъявляются к любому табличному выражению и которые мы уже перечислили ранее.

В отличие от временных таблиц табличные выполнения хранятся в оперативной памяти и существуют только во время первого выполнения запроса, который представляет это табличное выражение.

Как и в случае с производными таблицами, результат ОТВ становится недоступным сразу после выполнения внешнего кода.

Псевдонимы столбцов

ОТВ тоже поддерживают две формы синтаксиса для назначения псевдонимов.

Первая выглядит как <выражение> AS <псевдоним_столбца>;
второй список столбцов указывается в скобках сразу после названия выражения

Ниже показан пример первой формы.

```
WITH C AS (
    SELECT YEAR(orderdate) AS orderyear, custid FROM Sales.Orders
)
SELECT orderyear, COUNT(DISTINCT custid) AS numcusts FROM C
GROUP BY orderyear;
```

А вот как выглядит вторая форма.

```
WITH C(orderyear, custid) AS (
    SELECT YEAR(orderdate), custid FROM Sales.Orders
)
SELECT orderyear, COUNT(DISTINCT custid) AS numcusts FROM C
GROUP BY orderyear;
```

Использование аргументов

Как и в случае с производными таблицами, при определении ОТВ можно использовать аргументы.

Ниже показан пример.

```
DECLARE @empid AS INT = 3; WITH C AS  
(  
    SELECT YEAR(orderdate) AS orderyear, custid FROM Sales.Orders  
    WHERE empid = @empid  
)  
SELECT orderyear, COUNT(DISTINCT custid) AS numcusts FROM C  
GROUP BY orderyear;
```

Определение нескольких ОТВ

Но тот факт, что ОТВ используются уже после своего определения, дает им несколько важных преимуществ. Прежде всего, чтобы вызвать одно выражение из другого, необязательно делать их вложенными.

Вместо этого в инструкции *WITH* следует объявить список разных ОТВ, разделяя их запятыми.

Каждое ОТВ может ссылаться на предыдущее, и все они доступны во внешнем коде.

Например,

```
WITH C1 AS (  
    SELECT YEAR(orderdate) AS orderyear, custid FROM Sales.Orders  
)  
C2 AS (  
    SELECT orderyear, COUNT(DISTINCT custid) AS numcusts FROM C1  
    GROUP BY orderyear  
)  
SELECT orderyear, numcusts FROM C2  
WHERE numcusts > 70;
```

Поскольку вы определили ОТВ перед его непосредственным использованием, **вам не пришлось прибегать к вложенности**. Каждое ОТВ размещается в запросе в виде отдельного модуля, что позволяет сделать код более легким для восприятия и сопровождения.

Формально вы не можете сделать ОТВ вложенными или объявить их внутри скобок, которые считаются частью определения производной таблицы. Но, как вы уже знаете, вложенность влечет множество проблем, поэтому можете думать об этом не как о препятствии, а как о средстве, которое позволяет сделать код более прозрачным.

```

88
89
90 WITH OrdersInfo AS
91 (
92     SELECT ProductId,
93            SUM(ProductCount) AS TotalCount,
94            SUM(ProductCount * Price) AS TotalSum
95     FROM Orders
96     GROUP BY ProductId
97 )
98
99 SELECT * FROM OrdersInfo -- здесь нормально
100 SELECT * FROM OrdersInfo -- здесь ошибка
101 SELECT * FROM OrdersInfo -- здесь ошибка

```

152 %

Результаты Сообщения

	ProductId	TotalCount	TotalSum
1	2	2	82000,00
2	4	2	92000,00