### Лекция 16:

1. Создание сводных таблиц.
2. Агрегирование данных.
3. Группировка по наборам полей.
4. Дополнение: Dynamic SQL.

### Создание сводных таблиц:

Часто для всевозможных отчетов возникает необходимость отображать данные в виде «сводного отчета». Например, Ваши данные хранятся в таблице в виде: Месяц, Год, Кол-во (например, кол-во компьютеров, проданных по месяцам в каждом году). Вам требуется создать запрос, который будет отображать эти данные в виде:

По вертикали – Год, по горизонтали – Месяц (число от 1 до 12 или название месяца), на пересечении значения года и значения месяца – конкретное количество компьютеров, проданных в выбранном месяце в этом году.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Месяц | Год | Колво |  | Год | 1 | 2 | …. |
| 1 | 2020 | 15 |  | 2020 | 15 | 16 |  |
| 2 | 2020 | 16 | => | 2021 | 20 |  |  |
| … |  |  |  |  |  |  |  |
| 1 | 2021 | 20 |  |  |  |  |  |

Часто такой вид бывает более удобен для анализа, построения графиков, а так же восприятия информации.

В принципе, Вы это можете сделать только при использовании функций CASE (или IIF) и SUM.

Например, так:

В таблице @Данные содержатся Ваши данные по ежемесячным продажам.

DECLARE @Данные TABLE (Месяц int, Год int, Колво int)

INSERT INTO @Данные VALUES (1, 2020, 15), (1, 2021, 20), (2, 2020, 16), (2, 2021, 21), (3, 2020, 17), (3, 2021, 22), (4, 2020, 13), (4, 2021, 2), (5, 2020, 16), (6, 2020, 18), (7, 2020, 13), (8, 2020, 10), (9, 2020, 15), (10, 2020, 9), (11, 2020, 17), (12, 2020, 11), (1, 2019, 2), (2, 2019, 6), (3, 2019, 8), (4, 2019, 7), (5, 2019, 3), (6, 2019, 8), (7, 2019, 3), (8, 2019, 1), (9, 2019, 5), (10, 2019, 8), (11, 2019, 7), (12, 2019, 2)

SELECT Год,

SUM(CASE WHEN Месяц = 1 THEN Колво ELSE 0 END) Январь,

SUM(CASE WHEN Месяц = 2 THEN Колво ELSE 0 END) Февраль,

SUM(CASE WHEN Месяц = 3 THEN Колво ELSE 0 END) Март,

SUM(CASE WHEN Месяц = 4 THEN Колво ELSE 0 END) Апрель,

SUM(CASE WHEN Месяц = 5 THEN Колво ELSE 0 END) Май,

SUM(CASE WHEN Месяц = 6 THEN Колво ELSE 0 END) Июнь,

SUM(CASE WHEN Месяц = 7 THEN Колво ELSE 0 END) Июль,

SUM(CASE WHEN Месяц = 8 THEN Колво ELSE 0 END) Август,

SUM(CASE WHEN Месяц = 9 THEN Колво ELSE 0 END) Сентябрь,

SUM(CASE WHEN Месяц = 10 THEN Колво ELSE 0 END) Октябрь,

SUM(CASE WHEN Месяц = 11 THEN Колво ELSE 0 END) Ноябрь,

SUM(CASE WHEN Месяц = 12 THEN Колво ELSE 0 END) Декабрь

FROM @Данные

GROUP BY Год

Результат 1:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Год | Январь | Февраль | Март | Апрель | Май | Июнь | Июль | Август | Сентябрь | Октябрь | Ноябрь | Декабрь |
| 2019 | 2 | 6 | 8 | 7 | 3 | 8 | 3 | 1 | 5 | 8 | 7 | 2 |
| 2020 | 15 | 16 | 17 | 13 | 16 | 18 | 13 | 10 | 15 | 9 | 17 | 11 |
| 2021 | 20 | 21 | 22 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Но, в T-SQL есть дополнительные функции PIVOT и UNPIVOT, которые позволяют написать такой запрос в более коротком виде:

SELECT Год, [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12]

FROM **(SELECT Год, Месяц, Колво FROM @Данные)** AS D

PIVOT

(

SUM(Колво)

FOR Месяц IN([1],[2],[3],[4],[5],[6],[7],[8],[9],[10],[11],[12])

) AS P;

Результат 2:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Год | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 2019 | 2 | 6 | 8 | 7 | 3 | 8 | 3 | 1 | 5 | 8 | 7 | 2 |
| 2020 | 15 | 16 | 17 | 13 | 16 | 18 | 13 | 10 | 15 | 9 | 17 | 11 |
| 2021 | 20 | 21 | 22 | 2 | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

Заметьте, что в первом синтаксисе, если у нас нет данных за какие-то месяцы, то в сводном отчете отобразиться – 0 (при таком синтаксисе), а при втором способе с использованием PIVOT будет отображаться для таких случаев NULL.

Кроме того, обращаю внимание, что неудобство в виде того, что, если кол-во столбцов необходимо изменить, то придется переписывать запрос, есть в обоих синтаксисах. Позже мы посмотрим, как при помощи XML можно избежать этой проблемы.

Аналогично, достаточно часто встречается обратная ситуация, когда Ваши исходные данные (например, отчеты от Ваших региональных магазинов) приходят в виде таких сводных отчетов. Если Вам требуется преобразовать их к виду, удобному для автоматического расчета, то нужно использовать функцию UNPIVOT.

DECLARE @Сводная TABLE (Год int, Колво1 int, Колво2 int, Колво3 int, Колво4 int, Колво5 int, Колво6 int)

INSERT INTO @Сводная VALUES (2019,2,6,8,7,3,8),(2020,15,16,17,13,16,18),(2021,20,21,22,2,0,NULL)

--Unpivot the table.

SELECT Год,

LEFT(REVERSE(Месяц), PATINDEX('%[^0-9]%', REVERSE(Месяц)) - 1) Месяц,

Колво

FROM

(

SELECT Год, Колво1, Колво2, Колво3, Колво4, Колво5, Колво6

FROM @Сводная

) AS T

UNPIVOT

(

Колво

FOR Месяц IN (Колво1, Колво2, Колво3, Колво4, Колво5, Колво6)

) AS unpvt

Проверьте, что после выполнения запроса в результате не будет данных для 6-ого месяца 2021 года.

И в целом, намного проще пользоваться UNION.

Еще способ сделать Unpivot стандартными средствами SQL:

SELECT Год, Месяц,

CASE Месяц

WHEN 1 THEN Колво1

WHEN 2 THEN Колво2

WHEN 3 THEN Колво3

WHEN 4 THEN Колво4

WHEN 5 THEN Колво5

WHEN 6 THEN Колво6

END AS Колво

FROM @Сводная CROSS JOIN (VALUES(1),(2),(3),(4),(5),(6)) AS MNTH(Месяц);

### Агрегирование данных. Прочие функции

**Накопительная статистика:** данные агрегируются от первого элемента к текущему.

Пример: для той же таблицы @Данные Вы хотите посчитать на каждый месяц накопительно, сколько было продано с начала года.

Обратите внимание, что метод решения полностью повторяет задачу про расчет рейтинга:

SELECT T1.Год, T1.Месяц, T1.Колво, SUM(T2.Колво) СуммКолво

FROM @Данные T1 INNER JOIN @Данные T2 ON

T1.Год = T2.Год AND

T1.Месяц >= T2.Месяц

GROUP BY T1.Год, T1.Месяц, T1.Колво

ORDER BY 1, 2

Результат (только для 2019 года)

|  |  |  |  |
| --- | --- | --- | --- |
| Год | Месяц | Колво | СуммКолво |
| 2019 | 1 | 2 | 2 |
| 2019 | 2 | 6 | 8 |
| 2019 | 3 | 8 | 16 |
| 2019 | 4 | 7 | 23 |
| 2019 | 5 | 3 | 26 |
| 2019 | 6 | 8 | 34 |
| 2019 | 7 | 3 | 37 |
| 2019 | 8 | 1 | 38 |
| 2019 | 9 | 5 | 43 |
| 2019 | 10 | 8 | 51 |
| 2019 | 11 | 7 | 58 |
| 2019 | 12 | 2 | 60 |

Другой вариант:

SELECT Год, Месяц, Колво,

SUM(Колво) OVER (

PARTITION BY Год ORDER BY Месяц

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

/\***ROWS** BETWEEN 2 PRECEDING AND 1 FOLLOWING \*/

) СуммКолво

FROM @Данные;

**Скользящая статистика:** скользящее среднее – хороший пример скользящего агрегирования.

Можно воспользоваться запросом выше, только ограничить нижнюю точку:

SELECT T1.Год, T1.Месяц, T1.Колво, SUM(T2.Колво) СуммКолво

FROM @Данные T1 INNER JOIN @Данные T2 ON

T1.Год = T2.Год AND

T1.Месяц >= T2.Месяц AND

T2.Месяц > T1.Месяц - 3

GROUP BY T1.Год, T1.Месяц, T1.Колво

ORDER BY 1, 2

Результат за 2019 год (суммирование данных за последние 3 месяца):

|  |  |  |  |
| --- | --- | --- | --- |
| Год | Месяц | Колво | СуммКолво |
| 2019 | 1 | 2 | 2 |
| 2019 | 2 | 6 | 8 |
| 2019 | 3 | 8 | 16 |
| 2019 | 4 | 7 | 21 |
| 2019 | 5 | 3 | 18 |
| 2019 | 6 | 8 | 18 |
| 2019 | 7 | 3 | 14 |
| 2019 | 8 | 1 | 12 |
| 2019 | 9 | 5 | 9 |
| 2019 | 10 | 8 | 14 |
| 2019 | 11 | 7 | 20 |
| 2019 | 12 | 2 | 17 |

Примеры выше можно было выполнить и, например, при помощи RANKING FUNCTION. Попробуйте сделать в качестве домашнего упражнения.

**Способы группировки:**

На SQL сервере есть возможность сгруппировать данные разным способом внутри одного запроса.

Зачем это нужно: часто для анализа информацию необходимо увидеть в разных разрезах. Например, в базе данных продаж сначала Вы смотрите продажи по годам, замечаете, что какой-то год выделяется на фоне других, для выяснения причины Вы начинаете разукрупнять группы, например, смотреть статистику по месяцам, потом только по покупателям или товарам, или совместно по покупателям и товарам и т.д. В случае если аналитик (или иной потребитель запросов) в Вашей компании не пишет запросы, а периодически изучает готовые справки, вычисляемые автоматически Вашими запросами, Вы должны заранее предусмотреть возможность выдавать данные в разных разрезах. Можно прописывать отдельные запросы для агрегирования в каждом разрезе, но это неоптимально с точки зрения обращения к таблицам. Например, у Вас в таблице N полей, по различным комбинациям которых Вы хотите вывести агрегированную статистику. В этом случае Вам нужно будет написать 2^N запросов (1 запрос суммарно, N запросов по 1 полю, N\*(N-1)/2 по 2-м полям и тд). Вместо этого нужно воспользоваться специальной возможностью SQL server для группировки. В этом случае таблица не будет сканироваться все эти 2^N раз, кол-во проходов будет оптимизировано.

**Коммент**: Обычно для анализа данных за прошлые периоды не нужны онлайн-данные, поэтому достаточно статистики на утро. В этом случае вычислять агрегированные показатели в нужных разрезах можно автоматически ночным роботом. При этом результат его работы предполагается складывать в отдельную таблицу, к которой на следующий день и будет обращаться пользователь. Концепция – OLAP-куб.

**Коммент** **2**: Первым шагом алгоритма нужно перебрать все группировки по одному полю, этим Вам может помочь как раз GROUPING SETS.

Использование **GROUPING SETS** и **GROUPING\_ID**:

Пример: Данные по продажам хранятся в таблице «Данные»:

SET dateformat dmy

DECLARE @Документы TABLE (ндок int, Дата datetime, Пок\_ID int, Товар\_ID int, Колво int, primary key(ндок, Товар\_ID))

INSERT INTO @Документы VALUES

(1, '20210115', 1, 1, 15),

(1, '20210115', 1, 2, 10),

(2, '20210215', 2, 1, 5)

Запрос, который считает агрегированную статистику в разных разрезах:

SELECT GROUPING\_ID(Пок\_ID, Товар\_ID, YEAR(Дата), MONTH(Дата)) Грп,

YEAR(Дата) Год, MONTH(Дата) Месяц, Пок\_ID, Товар\_ID, SUM(Колво) СуммКолво

FROM @Документы

GROUP BY GROUPING SETS

(

( Пок\_ID, Товар\_ID, YEAR(Дата), MONTH(Дата) ),

( Пок\_ID, YEAR(Дата), MONTH(Дата) ),

( Товар\_ID, YEAR(Дата), MONTH(Дата) ),

( Товар\_ID, YEAR(Дата) ),

()

)

ORDER BY 1

Результат:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Грп | Год | Месяц | Пок\_ID | Товар\_ID | СуммКолво |
| 0 | 2021 | 1 | 1 | 1 | 15 |
| 0 | 2021 | 2 | 2 | 1 | 5 |
| 0 | 2021 | 1 | 1 | 2 | 10 |
| 4 | 2021 | 1 | 1 | NULL | 25 |
| 4 | 2021 | 2 | 2 | NULL | 5 |
| 8 | 2021 | 1 | NULL | 2 | 10 |
| 8 | 2021 | 2 | NULL | 1 | 5 |
| 8 | 2021 | 1 | NULL | 1 | 15 |
| 9 | 2021 | NULL | NULL | 1 | 20 |
| 9 | 2021 | NULL | NULL | 2 | 10 |
| 15 | NULL | NULL | NULL | NULL | 30 |

Видно, что первый результат (ГРП = 0) – это результат выполнения запроса, если бы мы написали GROUP BY Пок\_ID, Товар\_ID, YEAR(Дата), MONTH(Дата). Последний результат, это результат запроса SELECT 15, NULL, NULL, NULL, NULL, SUM(Колво) FROM @Данные.

Следующий запрос является полным аналогом в плане выдаваемого результата:

SELECT 0 Грп, YEAR(Дата) Год, MONTH(Дата) Месяц, Пок\_ID, Товар\_ID, SUM(Колво) СуммКолво

FROM @Документы

GROUP BY YEAR(Дата), MONTH(Дата), Пок\_ID, Товар\_ID

UNION

SELECT 4, YEAR(Дата) Год, MONTH(Дата) Месяц, Пок\_ID, NULL, SUM(Колво) СуммКолво

FROM @Документы

GROUP BY YEAR(Дата), MONTH(Дата), Пок\_ID

UNION

SELECT 8, YEAR(Дата) Год, MONTH(Дата) Месяц, NULL, Товар\_ID, SUM(Колво) СуммКолво

FROM @Документы

GROUP BY YEAR(Дата), MONTH(Дата), Товар\_ID

UNION

SELECT 9, YEAR(Дата) Год, NULL, NULL, Товар\_ID, SUM(Колво) СуммКолво

FROM @Документы

GROUP BY YEAR(Дата), Товар\_ID

UNION

SELECT 15, NULL, NULL, NULL, NULL, SUM(Колво) СуммКолво

FROM @Документы

Как работает функция **GROUPING\_ID**:

В принципе, если бы мы точно знали, что все поля, по которым мы группируем, всегда принимают not NULL значение, то мы бы могли бы самостоятельно разбить разные группировки на отдельные группы, но так как в данных могут быть NULL-значения, такая задача становится более трудной. Один из вариантов – разбивать на группы при помощи GROUPING\_ID.

GROUPING\_ID() – функция, которая вычисляет уровень группировки. Её аргумент – это выражение в GROUP BY.

GROUPING\_ID (a, b) – что выдает, если группировка происходит по указанному множеству.

|  |  |  |
| --- | --- | --- |
|  | 11 | 3 |
| a | 01 | 1 |
| b | 10 | 2 |
| ab | 00 | 0 |

То есть если атрибут не входит в группировку, то его значение для функции – 1, иначе 0.

Соответственно, на примере выше: GROUPING\_ID(Пок\_ID, Товар\_ID, YEAR(Дата), MONTH(Дата)) и в GROUP BY ( Товар\_ID, YEAR(Дата) ), то это равносильно 1001, что равно 1\*2^3 + 1\*2^0 = 9.

Второй вариант – это функция GROUPING, которая принимает на вход название столбца, и выдает 1 – если это поле не участвует в группировке в этой строке и 0 – если это поле участвует в группировке в этой строке.

SELECT GROUPING\_ID(Пок\_ID, Товар\_ID, YEAR(Дата), MONTH(Дата)) Грп,

YEAR(Дата) Год, MONTH(Дата) Месяц, Пок\_ID, Товар\_ID, SUM(Колво) СуммКолво,

GROUPING(Пок\_ID)

FROM @Документы

GROUP BY GROUPING SETS

(

( Пок\_ID, Товар\_ID, YEAR(Дата), MONTH(Дата) ),

( Пок\_ID, YEAR(Дата), MONTH(Дата) ),

( Товар\_ID, YEAR(Дата), MONTH(Дата) ),

( Товар\_ID, YEAR(Дата) ),

()

)

ORDER BY 1

**CUBE**

Способ сгенерировать подгруппы для группировки, не перечисляя их явно как в GROUPING SETS.

Для примера выше запрос:

SELECT GROUPING\_ID(Пок\_ID, Товар\_ID, YEAR(Дата), MONTH(Дата)) Грп,

YEAR(Дата) Год, MONTH(Дата) Месяц, Пок\_ID, Товар\_ID, SUM(Колво) СуммКолво

FROM @Документы

GROUP BY CUBE (Пок\_ID, Товар\_ID, YEAR(Дата), MONTH(Дата))

ORDER BY 1

сгенерирует 36 строк, которые будут содержать в себе группировки по всем возможным комбинациям из 4-х полей. В том числе результат будет содержать и строки выше (группировка по части полей).

Обратите внимание, что если Вы напишете CUBE( (a,b), c), то сгенерируется не 8 подмножеств, а 4, так как здесь 2 элемента: (a, b) и c. Группы будут следующими: (a, b, c), (a, b), (c), ().

Аналогичный результат выдает (реализация до 2008 SQL, работает и в новых):

SELECT GROUPING\_ID(Пок\_ID, Товар\_ID, YEAR(Дата), MONTH(Дата)) Грп, YEAR(Дата) Год, MONTH(Дата) Месяц, Пок\_ID, Товар\_ID, SUM(Колво) AS СуммКолво

FROM @Документы

GROUP BY YEAR(Дата), MONTH(Дата), Пок\_ID, Товар\_ID

WITH CUBE

ORDER BY 1;

Недостаток: не позволяет определить группировочный набор отличным от SELECT способом.

**ROLLUP**

В отличие от Cube генерирует только подмножество всех наборов. ROLLUP от n элементов производит n+1 набор. На следующем примере покажем, когда это может понадобиться и что за наборы генерируются.

DECLARE @Сводная TABLE (Год int, Месяц int, Колво int)

INSERT INTO @Сводная

VALUES (2019, 1, 50), (2019, 2, 100), (2020, 1, 25), (2020, 2, 30), (2021, 1, 35), (2021, 2, 80)

SELECT GROUPING\_ID(Год, Месяц) Грп,

Год, Месяц, SUM(Колво) СуммКолво

FROM @Сводная

GROUP BY CUBE(Год, Месяц)

ORDER BY 1

|  |  |  |  |
| --- | --- | --- | --- |
| **Грп** | **Год** | **Месяц** | **СуммКолво** |
| 0 | 2019 | 1 | 50 |
| 0 | 2020 | 1 | 25 |
| 0 | 2021 | 1 | 35 |
| 0 | 2019 | 2 | 100 |
| 0 | 2020 | 2 | 30 |
| 0 | 2021 | 2 | 80 |
| 1 | 2019 | NULL | 150 |
| 1 | 2020 | NULL | 55 |
| 1 | 2021 | NULL | 115 |
| 2 | NULL | 2 | 210 |
| 2 | NULL | 1 | 110 |
| 3 | NULL | NULL | 320 |

Если мы, как и раньше, воспользуемся CUBE, то получим 4 группы, одна из которых будет группировкой только по месяцу. В этом случае просуммируются результаты для всех лет в один месяц. Возможно, для некоторых задач это и имеет смысл, но если Вы хотите этого избежать, нужно пользоваться ROLLUP.

SELECT GROUPING\_ID(Год, Месяц) Грп,

Год, Месяц, SUM(Колво) СуммКолво

FROM @Сводная

GROUP BY ROLLUP(Год, Месяц)

ORDER BY 1

|  |  |  |  |
| --- | --- | --- | --- |
| **Грп** | **Год** | **Месяц** | **СуммКолво** |
| 0 | 2019 | 1 | 50 |
| 0 | 2019 | 2 | 100 |
| 0 | 2020 | 1 | 25 |
| 0 | 2020 | 2 | 30 |
| 0 | 2021 | 1 | 35 |
| 0 | 2021 | 2 | 80 |
| 1 | 2021 | NULL | 115 |
| 1 | 2020 | NULL | 55 |
| 1 | 2019 | NULL | 150 |
| 3 | NULL | NULL | 320 |

В этом случае ROLLUP эквивалентен следующему набору GROUPING SETS:

SELECT GROUPING\_ID(Год, Месяц) Грп,

Год, Месяц, SUM(Колво) СуммКолво

FROM @Сводная

GROUP BY GROUPING SETS

(

(Год),

(Год, Месяц),

()

)

ORDER BY 1

Аналог в прежних версиях SQL Server:

SELECT GROUPING\_ID(Год, Месяц) Грп,

Год, Месяц, SUM(Колво) СуммКолво

FROM @Сводная

GROUP BY Год, Месяц

WITH ROLLUP

ORDER BY 1

Для Grouping sets, Cube и Rollup указан минимальный синтаксис. Более подробно читайте в Хелпе.

**Динамический SQL**

Если у Вас возникает необходимость изменять текст промежуточного запроса (выводимые поля, например) в процессе выполнения задачи, то Вам может быть полезен динамический SQL (Dynamic SQL). Например, Вам нужно создавать сводную таблицу, но Вы не знаете заранее, сколько столбцов Вам нужно, и количество столбцов динамически зависит от внешних условий или результатов промежуточных запросов.

Есть 2 способа запуска:

**Exec** (не поддерживает автоматическую подстановку параметров в строку):

Простой вариант:

DECLARE @sql1 AS NVARCHAR(100);

SET @sql1 = 'SELECT \* FROM Товары WHERE Товар\_ID = 1;';

EXEC (@sql1)

Вариант с подстановкой параметра:

DECLARE @sql2 AS NVARCHAR(100), @n as nvarchar(2);

SET @n = 1

SET @sql2 = 'SELECT \* FROM Товары WHERE Товар\_ID = CAST(' + @n + ' as int)';

EXEC (@sql2)

Вариант с SQL-инъекцией:

DECLARE @sql3 AS NVARCHAR(100), @k as nvarchar(100);

SET @k = '1; DROP TABLE Товары'

SET @sql3 = 'SELECT \* FROM Товары WHERE Товар\_ID = ' + @k + '';

EXEC (@sql3)

Обратите внимание, что в этом случае таблица Товары дропнется.

**sp\_executesql:** более гибкий вариант, принимает входные и генерирует выходные параметры.

DECLARE @sql4 AS NVARCHAR(100);

SET @sql4 = 'SELECT TOP 1 \* FROM Товары WHERE Товар\_ID = @good;';

EXEC sp\_executesql

@stmt = @sql4,

@params = N'@good AS INT',

@good = 1;

DECLARE @sql5 AS NVARCHAR(200);

SET @sql5 = 'SELECT \* FROM Товары WHERE Товар = @good;';

EXEC sp\_executesql

@stmt = @sql5,

@params = N'@good AS NVARCHAR(200)',

@good = '1; DROP TABLE Товары';

А здесь уже в качестве значения параметра подставляется вся строка полностью (то есть не формируется новый запрос, а проверяется, равен ли товар указанному значению). Поэтому dropa не будет!

Дополнительное чтение: **SQL injection**

(ms-help://MS.SQLCC.v10/MS.SQLSVR.v10.en/s10de\_4deptrbl/html/c04f1f33-d607-461b-9c75-799c55087a63.htm)