

T-SQL.RU

Всё об MS SQL Server 7.0 - 2019 на русском

[Главная](#) [Архив](#) [Контакт](#) [Подписаться](#) [Filter by APML](#)[Войти](#)<< [SQL Validator \(on-line\)](#) | [Jabber клиент для SQL Server \(JabberCLR\)](#) >>

Join Hints

By Alexey Knyazev

4. декабря 2009 23:36



Join Hints (LOOP | HASH | MERGE | REMOTE) - Подсказки оптимизатору запросов на выбор определенной стратегии соединения двух таблиц (используется в SELECT, UPDATE и DELETE).

Оптимизатор запросов SQL Server обычно автоматически выбирает наилучший план выполнения запроса. Поэтому подсказки, в том числе <подсказки_по_соединению>, рекомендуются использовать только опытным пользователям и администраторам базы данных в случае крайней необходимости.

Без явного указания аргумента (LOOP | HASH | MERGE | REMOTE) оптимизатор выбирает, на его взгляд, самый оптимальный план. Но мы всегда можем повлиять на него, если явно укажем подсказку.

Ниже разберем каждый из аргументов подробнее.

Loop Join

Соединение **LOOP JOIN**, называемое также *nested iteration*, использует одну таблицу в качестве внешней (на графическом плане она является верхней), а второй в качестве внутренней (нижней). **LOOP JOIN** построчно сравнивает внешнюю таблицу с внутренней. В цикле для каждой внешней строки производится сканирование внутренней таблицы и выводятся совпадающие строки.

В простейшем случае во время поиска целиком сканируется таблица или индекс (*naive nested loops join*). Если при поиске используется индекс, то такой поиск называется *index nested loops join*. Если индекс создается в качестве части плана запроса (и уничтожается после завершения запроса), то он называется *temporary index nested loops join*. Оптимизатор сам выбирает один из этих поисков.

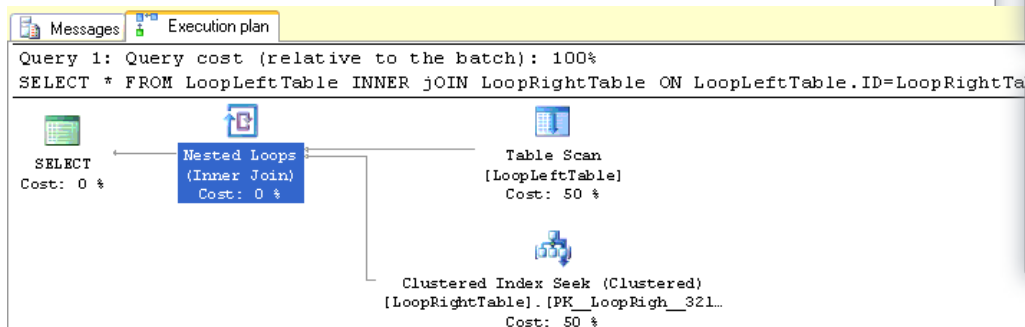
LOOP JOIN является особенно эффективным в случае, когда внешняя таблица сравнительно невелика, а внутренняя гораздо больше и для неё существуют индексы. В запросах с небольшим объемом строк, *index nested loops join* превосходит как **MERGE JOIN**, так и **HASH JOIN**. Однако в больших запросах **LOOP JOIN** часто являются не лучшим вариантом.

Для демонстрации создадим 2 тестовые таблицы:

```
CREATE TABLE LoopLeftTable (ID INT)
CREATE TABLE LoopRightTable (ID INT IDENTITY PRIMARY KEY)
```

И посмотрим план запроса:

```
SELECT * FROM
    LoopLeftTable
    INNER JOIN
    LoopRightTable
    ON LoopLeftTable.ID=LoopRightTable.ID
```



Как и описано выше оптимизатор выбрал **LOOP JOIN**. Но если мы вставим в таблицу достаточно большое кол-во строк, то оптимизатор откажется от соединения **LOOP JOIN**:

```
INSERT INTO LoopLeftTable
SELECT 1
GO 10000
```

MVP SQL Server



SQL PASS



Меню

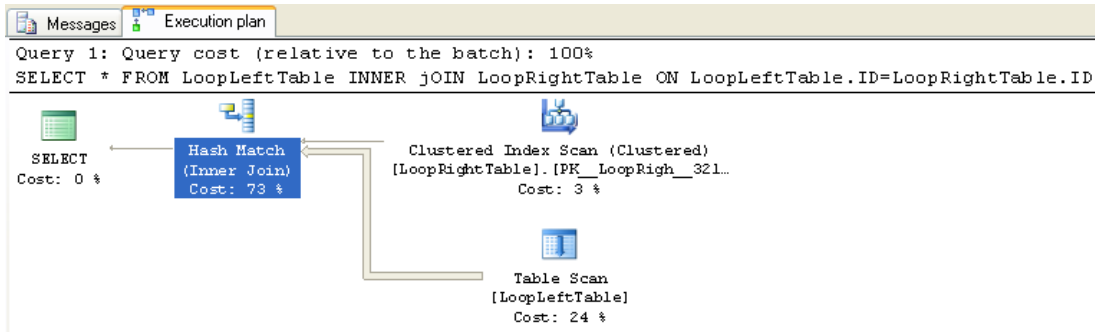
- [Версии SQL Server](#)
- [Книги](#)
- [Русские Блоггеры](#)
- [Стандарты SQL](#)
- [Утилиты](#)

История сообщений

- [2008](#)
- [2009](#)
- [2010](#)
- [2011](#)
- [2012](#)
- [2013](#)
- [2014](#)
- [2015](#)
- [2016](#)

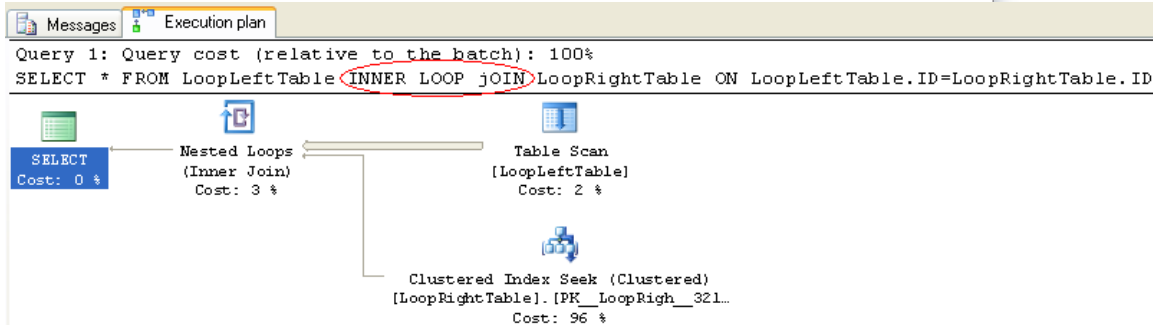
Облако поиска

CLR Denali Partition Poster PowerShell Scripts
SQL Server SQL Server 2000 SQL Server 2005 SQL Server 2008 SQL Server 2008 R2 SQL Server 2011 SQL Server 2012 SQL Server 2014 SQL Server 2016 SSMS Tools WebCast



Оптимизатор при выполнении запроса выбрал **HASH JOIN**, так как посчитал, что стоимость этого соединения будет ниже. Но если мы не доверяем оптимизатору то можем явно указать ему использовать **LOOP JOIN**:

```
SELECT * FROM
  LoopLeftTable
  INNER LOOP JOIN
  LoopRightTable
  ON LoopLeftTable.ID=LoopRightTable.ID
```



Кстати, если сравнить стоимость выполнения запроса выбранного оптимизатором и наш с подсказкой, то можно убедиться, что оптимизатор действительно выбрал верный план. (<http://msdn.microsoft.com/en-us/library/ms191318.aspx>)

Аргумент LOOP не может указываться вместе с параметрами RIGHT или FULL в качестве типа соединения.

Merge Join

Merge Join требует сортировки обоих наборов входных данных по столбцам слияния, которые определены предложениями равенства (ON) предиката соединения. (т.е. если мы имеем предикат соединения "T1.a = T2.b", таблица T1 должна быть отсортирована по T1.a, а таблица T2 должна быть отсортирована по T2.b).

Так как каждый набор входных данных сортируется, оператор **Merge Join** получает строку из каждого набора входных данных и сравнивает их. Например, для операций INNER JOIN строки возвращаются в том случае, если они равны. Если они не равны, строка с меньшим значением не учитывается, и из этого набора входных данных берется другая строка. Этот процесс повторяется, пока не будет выполнена обработка всех строк.

MERGE JOIN может поддерживать слияние "многие ко многим". В этом случае, при каждом соединении двух строк нужно сохранять копию каждой строки второго входного потока. Это позволяет, при последующем обнаружении в первом входном потоке дубликатов строк, воспроизвести сохраненные строки. С другой стороны, если будет ясно, что следующая строка первого входного потока не является дубликатом, от сохраненных строк можно отказаться. Такие строки сохраняются во временно таблице базы tempdb. Размер дискового пространства, который для этого необходим, зависит от числа дубликатов во втором входном потоке.

MERGE JOIN "один ко многим" всегда будет эффективнее слияния "многие ко многим", поскольку для него не требуется временная таблица. Для того, что бы задействовать слиянием "один ко многим", оптимизатор должен иметь возможность определить, что один из входных потоков состоит из уникальных строк. Как правило, это означает, что у такого входного потока существует уникальный индекс или в плане запроса присутствует явный оператор (например, сортировка при DISTINCT или группировка), который гарантирует, что строки на входе будут уникальны.

Merge Join — очень быстрая операция, но она может оказаться ресурсоемкой, если требуется выполнение операций сортировки. Однако на больших объемах при наличии индексов и предварительной сортировке, соединение слиянием является самым быстрым из доступных алгоритмов соединения.

Для демонстрации создадим 2 таблицы очень похожие на те, что были созданы в примере с **LOOP JOIN**:

```
CREATE TABLE MergeLeftTable (ID INT IDENTITY PRIMARY KEY)
CREATE TABLE MergeRightTable (ID INT IDENTITY PRIMARY KEY)
```

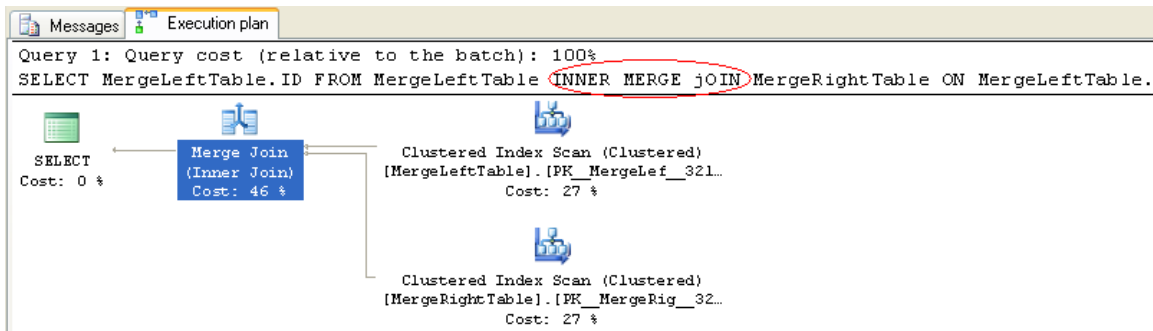
Если посмотреть какой план выбрал оптимизатор на пустых таблицах, если в качестве запроса указать:

```
SELECT MergeLeftTable.ID FROM
  MergeLeftTable
  INNER JOIN
  MergeRightTable
  ON MergeLeftTable.ID=MergeRightTable.ID
```

то окажется, что опять используется LOOP JOIN.

Можно явно указать оптимизатору использовать MERGE JOIN:

```
SELECT MergeLeftTable.ID FROM
  MergeLeftTable
  INNER MERGE JOIN
  MergeRightTable
  ON MergeLeftTable.ID=MergeRightTable.ID
```

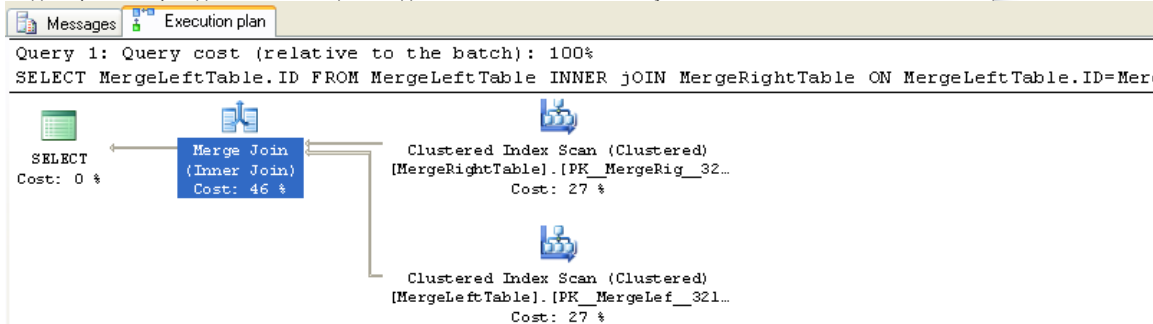


Но оптимизатор сам выберет для этих таблиц MERGE JOIN, если наполнить их (таблицы) хотя бы небольшим кол-ом данных:

```
INSERT INTO MergeLeftTable
DEFAULT VALUES
GO 40
```

```
INSERT INTO MergeRightTable
DEFAULT VALUES
GO 40
```

Тогда первый запрос для этих таблиц без подсказки выполнится по плану с MERGE JOIN:



(<http://msdn.microsoft.com/en-us/library/ms190967.aspx>)

Hash Join

Hash Join - более эффективен при работе с большими наборами данных и даже тогда, когда таблицы не отсортированы по столбцам, по которым производится соединение. **Hash Join** распараллеливается и масштабируется лучше любого другого соединения и сильно выигрывает при большой производительности информационных хранилищ.

Соединение происходит с использованием хеширования, вычисляя хеш записей из меньшей таблицы (Build-таблица) и вставляя их в хеш-таблицу, затем обрабатывается большая таблица (Probe-таблица) по одной записи, сканируя хеш-таблицу для поиска совпадений.

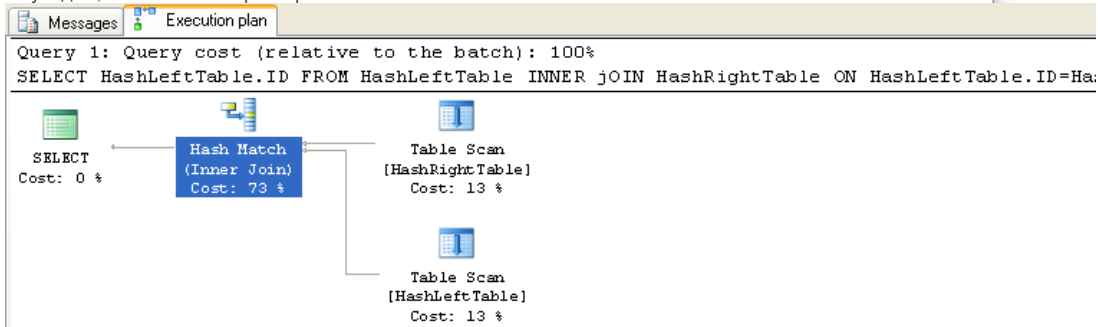
Создадим две таблицы для демонстрации:

```
CREATE TABLE HashLeftTable (ID INT)
CREATE TABLE HashRightTable (ID INT)
```

Если посмотреть план запроса:

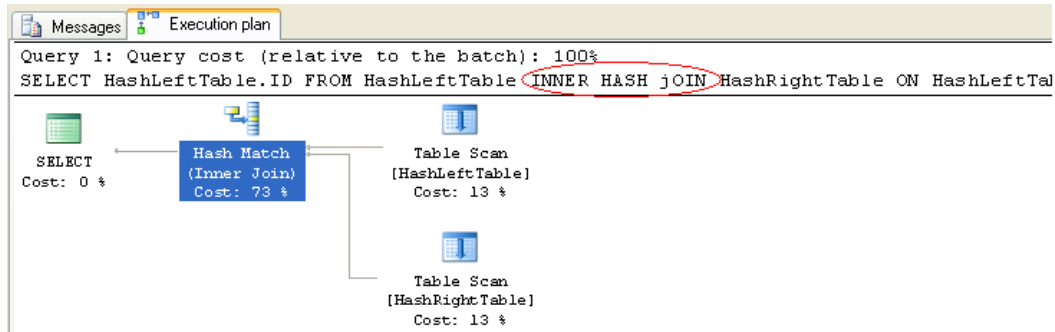
```
SELECT HashLeftTable.ID FROM
HashLeftTable
INNER JOIN
HashRightTable
ON HashLeftTable.ID=HashRightTable.ID
```

То увидим, что оптимизатор выбрал Hash Join:



Но если оптимизатор выбрал другой план, но мы явно желаем использовать хеш-объединение, то мы так же можем "подсказать" это оптимизатору:

```
SELECT HashLeftTable.ID FROM
HashLeftTable
INNER HASH JOIN
HashRightTable
ON HashLeftTable.ID=HashRightTable.ID
```



Hash Join бывают 3х видов:

- *In-Memory Hash Join* Когда таблицы небольшого размера и могут полностью быть помещены в память
- *Grace Hash Join* Если размер таблиц превышает максимально допустимый объем памяти, то хэш-соединение проводится в несколько шагов.
- *Recursive Hash Join* Этот вид объединения используется для сложных таблиц и для таблиц, которые являются очень большими и требуют многоуровневое соединение в несколько шагов.

(<http://msdn.microsoft.com/en-us/library/ms189313.aspx>)

Remote Join

Remote Join может быть использован только при операциях INNER JOIN.

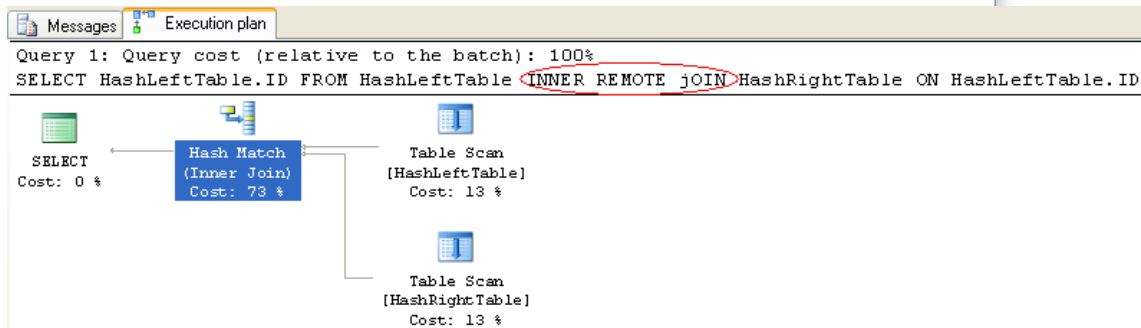
Remote Join задает, что операция соединения проводится на странице таблицы, расположенной справа. Данный аргумент удобно использовать в случае, когда таблица, расположенная слева, является локальной, а справа располагается удаленная таблица (Linked Server). Аргумент REMOTE может использоваться в случае, когда в таблице слева содержится большее количество строк, чем в таблице справа. Если таблица, расположенная справа, является локальной, то операция соединения также проводится локально. Если обе таблицы являются удаленными, но

расположены в различных источниках данных, то при задании аргумента REMOTE операция соединения проводится на странице таблицы, расположенной справа. Если обе таблицы являются удаленными таблицами в одном источнике данных, то аргумент REMOTE не требуется.

Как и в предыдущих случаях, мы можем явно указать оптимизатору использовать необходимое объединение:

```

SELECT HashLeftTable.ID FROM
    HashLeftTable
    INNER REMOTE JOIN
    HashRightTable
    ON HashLeftTable.ID=HashRightTable.ID
  
```



По теме:

- Loop Join
- Merge Join
- Hash Join


Tags: SQL Server, Scripts

SQL Server

E-Mail | Kick it! | DZone it! | del.icio.us

Permalink | Комментарии (7)

Комментарии (7)


zont
02.04.2010 21:00:23 #

Спасибо, очень приятно тебя читать. Очень понятно пишешь. Удачи!

Reply



Игорь

06.05.2011 10:53:55 #

Спасибо за статью, долго не мог найти толкового объяснения, данных операторов.

[Reply](#)

vadimman

24.05.2012 15:09:10 #

Спасибо, подробно и понятно все написано!

[Reply](#)

Stan

30.10.2012 17:39:34 #

Спасибо за статью. Странно, что подобного нет в BOL.

[Reply](#)

DNA

14.09.2013 0:23:03 #

Действительно странно, что не описано в боле, а только хинт remote. Ещё бы добавил, что при указании любого из этих хинтов в джоине приведет к тому, что все джоины будут выполняться этим способом, так что если вы джоинете более одной таблицы - надо быть аккуратным. В противном случае, если необходимо эту таблицу заджоинить именно так, а другие необязательно, то стоит использовать этот хинт как запросный, после условия ON. SELECT *
FROM sales.SalesOrderheader h JOIN sales.SalesOrderDetail d
ON h.SalesOrderID = d.SalesOrderDetailID OPTION (LOOP JOIN)
sqlmag.com/.../do-you-need-sql-server-query-hint тут описана эта ситуация.
<http://sql.pingvin4ik.info/>

[Reply](#)

Eugene

08.01.2014 13:15:01 #

DNA,

>>Ещё бы добавил, что при указании любого из этих хинтов в джоине приведет к тому, что все джоины будут выполняться этим способом

Ничего подобного. Хинт действует только на конкретное соединение. OPTION(LOOP|MERGE|HASH JOIN) действует на весь запрос, и указание вместе с ним другого хинта в джоине приведёт к ошибке на шаге построения плана.

Спасибо за статью!!

Ещё добавил бы, что указание хинтов явно указывает оптимизатору порядок соединения таблиц. Если писать без хинтов оптимизатор сам выберет нужную последовательность обработки.
И ещё, если в запросе(не в отдельном предложении FROM, а именно в запросе) фигурирует более 10ти таблиц - то оптимизатор выберет не лучший на его взгляд план, а лучший из тех, что успел проанализировать за определенное время... Так что в некоторых случаях хинты необходимы....\

[Reply](#)

Илана

27.07.2014 15:38:23 #

Спасибо за статью. Очень толковая!

[Reply](#)

Добавить комментарий

Имя*

E-mail*

Страна

5+5*0 =

[Комментарий](#) [Предпросмотр](#)

[b](#) [i](#) [u](#) [quote](#)

☐ Уведомлять о новых комментариях

Сохранить комментарий