

Глава 15. Оптимизация производительности

Это продолжение перевода книги Грегори Декер, Рик де Гроот, Мелисса де Кортс. Полное руководство по языку M Power Query. Хотя важно получать точные данные, скорость выполнения запросов существенно влияет на пользовательский опыт. Длительное обновление может привести к таймауту, а ожидание результатов преобразований в редакторе Power Query может быть раздражающим. В этой главе мы погрузимся в стратегии оптимизации производительности запросов. Сначала рассмотрим использование памяти при оценке запросов и как чрезмерное использование памяти замедляет их выполнение. Затем мы изучим различные стратегии, чтобы предотвратить это.

Мои комментарии набраны с отступом.

[Предыдущая глава](#) [Содержание](#) [Следующая глава](#)

Мы обсудим важность свертывания запросов (*query folding*) и как можно убедиться, что источник данных выполняет как можно больше преобразований. Затем мы рассмотрим фаервол (*formula firewall*), инструмент, разработанный для защиты запросов от раскрытия конфиденциальной информации. Мы объясним, что вызывает ошибки, связанные с ним. Мы исследуем, как хранение данных в памяти (буферизация) может ускорить выполнение запросов. Наконец, мы предложим советы по выбору источника данных с учетом производительности.

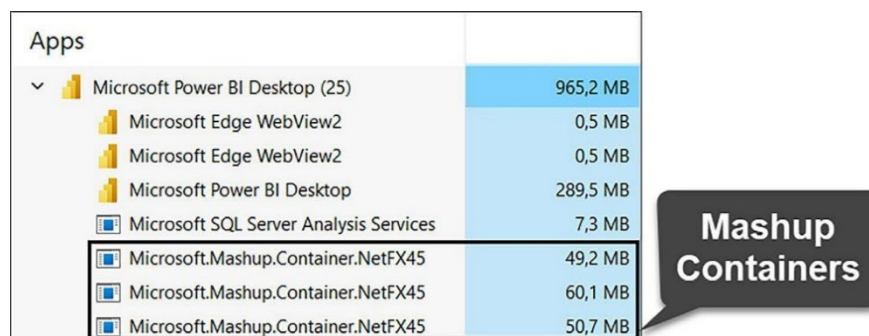
Мы рассмотрим следующие темы:

- Использование памяти при оценке запросов
- Свертывание запросов
- Фаервол
- Оптимизация производительности запросов
- Советы по улучшению производительности

Использование памяти при оценке запросов

Извлечение данных и выполнение преобразований требует ресурсов. Для выполнения запросов Power Query необходима память. Некоторые подходы требуют больше памяти, чем другие. Поэтому понимание того, как улучшить использование памяти, помогает при оптимизации запросов.

Запросы выполняются в контейнере запросов (*mashup container*), специальном процессе, ответственном за оценку запросов. Вы можете увидеть количество используемых контейнеров, открыв *Диспетчер задач* и просмотрев процессы для Power BI Desktop.



App	Memory Usage
Microsoft Power BI Desktop (25)	965,2 MB
Microsoft Edge WebView2	0,5 MB
Microsoft Edge WebView2	0,5 MB
Microsoft Power BI Desktop	289,5 MB
Microsoft SQL Server Analysis Services	7,3 MB
Microsoft.Mashup.Container.NetFX45	49,2 MB
Microsoft.Mashup.Container.NetFX45	60,1 MB
Microsoft.Mashup.Container.NetFX45	50,7 MB

Рис. 15.1. Контейнеры запросов отображаются в диспетчере задач

Объем памяти, предоставляемый каждому контейнеру, зависит от того, где выполняется Power Query. К распространенным средам относятся Power BI Desktop, служба Power BI и локальный шлюз данных. Так что же происходит, когда память контейнера заканчивается?

Когда потребности контейнера в памяти превышают выделенный лимит во время оценки запроса, Power Query начинает подкачку (передачу) данных запроса в дисковое хранилище. Это приводит к снижению производительности, так как доступ к диску значительно медленнее, чем доступ к памяти.

Вариации и настройки лимита памяти

Лимит памяти для каждого контейнера зависит от среды выполнения, и в некоторых случаях его можно настроить. Обычно используются следующие среды:

- *Машина шлюза*. Память на контейнер рассчитывается автоматически для машин, на которых работает локальный шлюз. Хотя вы не можете задать лимиты, увеличение объема памяти и скорости машины сократят время обновления запросов.
- *Сервис Power BI*. Вы не можете изменить лимит памяти. Если вы используете премиум-версию Power BI, вы получаете больше ресурсов, что улучшит производительность.
- *Power BI Desktop*. Вы можете настроить, сколько памяти использовать. В редакторе Power Query пройдите *Файл* → *Параметры и настройки* → *Параметры*. Перейдите на вкладку *Глобальные | Загрузка данных*:

Параметры



Рис. 15.2. Вы можете задать максимальный объем памяти для оценки запросов в Power BI Desktop

Помните, что максимальный объем памяти, который может быть выделен запросу, зависит от общего объема памяти, доступной на компьютере.

Для более быстрого выполнения запросов в Power Query, применяют две основные стратегии управления использованием памяти:

- минимизацию потребления памяти при импорте;
- минимизацию объема импортируемых данных.

Свертывание запросов

Свертывание запросов является одним из наиболее важных аспектов оптимизации производительности и полезно для любого разработчика, который подключается к внешним источникам данных. Это процесс, в котором оценка запроса переносится в источник данных.

Во время свертывания запросов Power Query преобразует запрос в язык источника данных. Этот подход объединяет несколько шагов на языке M в один эффективный запрос, который может выполнить источник данных. При этом вычислительная нагрузка этих шагов переносится из локальной среды, в которой работает Power Query, в источник данных, например в базу SQL. Иные шаги запроса M, которые не могут быть свернуты, выполняются локально. Основная цель свертывания запросов – сократить использование памяти Power Query за счет использования вычислительной мощности источника данных.

Предположим, вы подключаетесь к таблице в базе SQL и хотите получить данные только за последнюю неделю. Вместо того чтобы сначала импортировать все данные, а затем фильтровать их в Power Query, свертывание запросов отправляет точный запрос для извлечения только релевантных строк. Это дает два преимущества:

- *Оптимизация использования ресурсов*. Свертывание запросов использует сильные стороны источника данных, передавая нагрузку, связанную с преобразованиями, в источник. Когда преобразования выполняются на локальном компьютере, требуется извлечь все данные из источника, и затем обработать их в Power Query. Это может быть ресурсоемким и трудоемким процессом. Источник данных (сервер или база данных) выполняет эти задачи быстрее и эффективнее локальной системы.
- *Минимизация передачи данных по сети*. Важным побочным эффектом этого подхода является сокращение передачи данных по сети. Вместо того, чтобы извлекать большие объемы необработанных данных в локальную среду для преобразования, свертывание

запросов гарантирует, что будут извлечены только необходимые данные. Т.е., по сети отправляются только необходимые данные (после выбора конкретных строк/столбцов), что экономит пропускную способность и время, необходимое для передачи данных.

В дополнение к этим ключевым преимуществам, понимание свертывания запросов имеет решающее значение еще по двум причинам. Во-первых, свертывание запросов необходимо для использования таблиц с прямым запросом и двойным режимом хранения в Power BI. Эти режимы зависят от свертывания запросов для отправки запросов в базу данных.

Во-вторых, свертывание запросов играет важную роль в сценариях добавочного обновления. Добавочное обновление – это функция, которая позволяет загружать данные с определенного момента времени. Например, в таблице с миллионами строк, охватывающих 20 лет, можно обновить только недавнюю часть данных с помощью добавочного обновления. Однако это возможно только если источник поддерживает свертывание запросов, что позволяет запрашивать только необходимое подмножество данных. Без свертывания запросов весь набор данных должен быть извлечен и отфильтрован локально, что гораздо менее эффективно.

Как свертывание запросов работает на практике?

Свертывание запросов в действии

Чтобы понять, что такое свертывание запросов, давайте поразмышляем о скрипте M. Скрипт создает план обработки данных. Он инструктирует Power Query, какие шаги предпринять, в каком порядке и как эффективно объединить эти шаги. Всякий раз, когда мы хотим получить доступ к источнику данных, мы используем функции доступа к источнику данных. Допустим, мы подключаемся к базе данных SQL к таблице с именем *Track*:

```
1 let
2     Source = Sql.Database( Server, Database ),
3     dbo_Track = Source{[Schema="dbo", Item="Track"]}[Data]
4 in
5     dbo_Track
```

TrackId	Name	AlbumId	Composer
1	For Those About To Rock (We Salute You)	1	Angus Young, I
2	Balls to the Wall	2	
3	Fast As a Shark	3	F. Baltes, S. Kau
4	Restless and Wild	3	F. Baltes, R.A. S
5	Princess of the Dawn	3	Deaffy & R.A. S

Query settings

Properties

Name: Track

Applied steps

Source

dbo_Track

The query up to this step will be evaluated by the data source.

Рис. 15.3. Запрос, который подключается к базе данных SQL

Chat GPT считает, что перевод на русский язык термина *query folding* в контексте Power Query и баз данных еще не устоялся. Используются оба варианта – *сворачивание* и *свертывание*. При этом *сворачивание*, предпочтительнее, поскольку лучше передает смысл автоматической оптимизации процесса на уровне источника данных. На [сайте Microsoft](#) чаще используется *свертывание*. Я использую оба термина, как синонимы.

В новом интерфейсе Power Query вы можете найти индикаторы рядом с каждым шагом. Когда у шага зеленый значок базы данных с молнией, это указывает, что шаг будет свернут. Чтобы просмотреть, что Power Query делает *под капотом*, выберите любой сворачиваемый шаг, щелкните правой кнопкой мыши и выберите *Просмотреть запрос к источнику данных*.

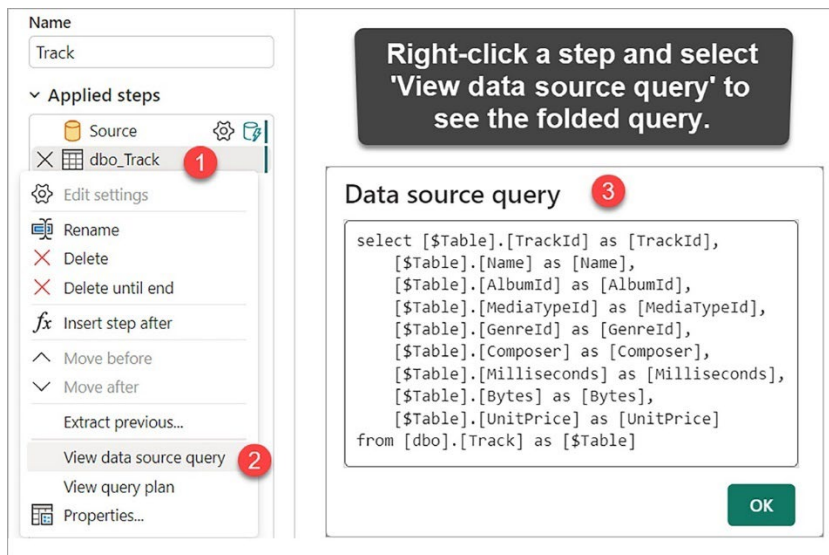


Рис. 15.4. Когда для шага включена опция *Просмотреть запрос к источнику данных*, вы можете увидеть свернутый запрос

В результате вы увидите экран с всплывающим окном, в котором отображен свернутый запрос – нативный SQL-запрос. Это означает, что данный запрос выполняется сервером базы данных.

Теперь давайте внесем некоторые изменения в запрос. Вместо того, чтобы показать все столбцы, я выбрал только пять столбцов. Проверив запрос источника данных на этом новом шаге *RemoveColumns*, мы получаем другой запрос, который выбирает только релевантные столбцы:

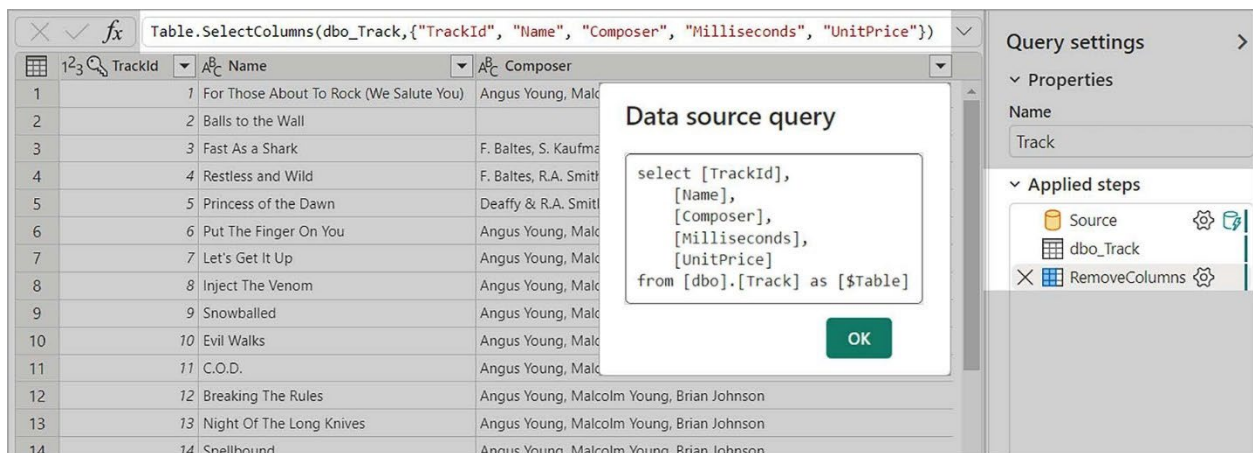


Рис. 15.5. Этот запрос к источнику данных отбирает некоторые столбцы

Без свертывания запросов Power Query пришлось бы импортировать все данные (включая ненужные столбцы). После импорта он отбросит ненужные столбцы. Другими словами, он будет использовать ресурсы для импорта и последующего удаления данных. Благодаря свертыванию запросов Power Query отправляет запрос, который извлекает только необходимые поля. Такой подход во-первых повышает производительность, так как в движок возвращаются только запрошенные данные. Во-вторых, пользователи, которые не знакомы с языком SQL-запросов, тем не менее используют автоматически сгенерированный запрос к источнику данных.

В старом интерфейсе Power Query нет таких индикаторов. Чтобы проверить, сворачивается ли шаг, щелкните по нему правой кнопкой мыши и выберите *Просмотреть машинный запрос*. Если параметр неактивен, то неясно, будет ли шаг сворачиваться. Это может означать, что свертывания не будет или коннектор не уверен, свернется ли этот шаг. Ниже мы рассмотрим это подробнее.

Как Power Query решает, какие операции могут быть свернуты. Другими словами, каков процесс оценки запроса в этом контексте?

Оценка запроса

Большинство запросов состоят из нескольких шагов. Оценка запроса проходит следующие этапы:

1. *Первичное получение данных.* Движок Power Query собирает M-код запроса, и получает учетные данные источника данных и уровни конфиденциальности источника данных.
2. *Чтение метаданных источника данных.* Движок запрашивает метаданные у источника. Этот шаг определяет возможности источника данных и возможность свертывания.
3. *Генерация запроса к источнику данных.* На основе полученной информации движок определяет (с помощью механизма свертывания запросов), какую информацию можно извлечь из источника данных, а какие преобразования должны быть выполнены внутри движка Power Query. Исходя из этого, он генерирует запрос к источнику данных.
4. *Получение данных.* Движок Power Query получает данные, как было указано в запросе.
5. *Выполнение оставшихся операций.* Любые преобразования, которые не удалось свернуть, далее выполняются движком M.
6. *Загрузка данных.* Обработанные данные загружаются в Power BI или Excel.

Основная функция свертывания – это преобразование шагов запроса на языке M в запрос к источнику данных, который может быть выполнен самим источником данных. Соединение с источником данных изначально устанавливается с помощью функции из семейства функций доступа к данным. Свертывание запросов в первую очередь нацелено на шаги преобразований, которые сразу следуют за шагом подключения к источнику. Эффективность свертывания в переводе этих преобразований в запрос к источнику данных зависит от возможностей используемого коннектора источника данных.

Свертывание, частичное свертывание и отсутствие свертывания

К сожалению, коннекторы Power Query не достаточно умны, чтобы переводить весь M-код в запрос к источнику данных. Некоторые операции приводят к нарушению свертывания запросов. В дальнейшем в этой главе мы рассмотрим стратегии для сохранения свертывания запросов.

Возможны следующие исходы:

- *Полное свертывание запроса.* Все преобразования происходят на стороне источника данных, а движок Power Query получает результат.
- *Частичное свертывание запроса.* Подмножество преобразований сворачивается и отправляется в источник данных. Оставшиеся шаги, которые не удалось свернуть, обрабатываются движком Power Query.
- *Отсутствие свертывания запроса.* Когда ни один из примененных шагов не может быть преобразован в язык источника данных, все преобразования выполняются движком Power Query. Это может произойти, если коннектор не поддерживает свертывание запросов или конкретное преобразование не поддерживается.

Если ваш источник данных работает с языком запросов, скорее всего, он поддерживает свертывание запросов. Это касается таких источников, как базы данных, потоки OData и Active Directory. В то же время файлы Excel, CSV или web не поддерживают свертывание запросов.

Понимание того, как работает свертывание запросов в Power BI, важно для уменьшения использования памяти. Однако это было бы сложно, если бы вы не знали, какие шаги свертываются, а какие нет. Для решения этой задачи Power Query предоставляет ряд инструментов, которые показывают, свертывается ли преобразование.

Инструменты для определения свертывания

В Power Query есть три основных инструмента, которые позволяют определить, сворачивается ли шаг. К таким инструментам относятся:

- Просмотр запроса к источнику данных
- Индикаторы свертывания шагов
- План запроса

Просмотр запроса к источнику данных

Если щелкнуть шаг правой кнопкой мыши, можно выбрать *Просмотреть запрос к источнику данных*:

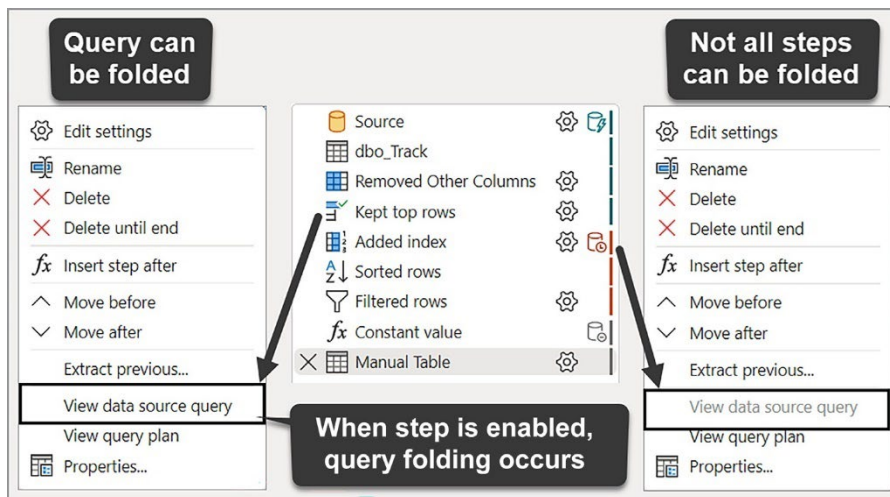


Рис. 15.6. Просмотр запроса к источнику данных

Когда функция активна, можно увидеть запрос к источнику данных. Если функция неактивна, шаг не сворачивается. Однако не обязательно весь запрос не сворачивается. Он может быть частично сворачиваемым, или коннектор не поддерживает конкретную функцию. Просмотреть запрос к источнику данных можно для коннектора базы данных SQL, но не для коннектора OData.

Преимущество этого подхода в том, что он работает как в старом, так и в новом интерфейсе Power Query. Недостаток в том, что нужно пройти каждый шаг, чтобы увидеть, сворачивается ли он.

Индикаторы свертывания запросов

Индикаторы доступны в новом интерфейсе Power Query и предоставляют визуальные подсказки в редакторе запросов Power BI. Они предоставляют информацию о том, сворачивается ли каждый шаг в запросе или какой-то конкретный шаг останавливает свертывание. Это полезно, поскольку сигнализирует, какие операции будут обрабатываться вне источника данных.

На каждом шаге запроса будет отображаться один из следующих индикаторов:

Индикатор	Иконка	Описание
Сворачивается		Указывает на то, что эта часть запроса будет обработана источником данных.
Не сворачивается		Указывает, что этот шаг будет обработан вне источника данных.
Возможно будет свернут		Неизвестно, будет ли шаг запроса обработан источником данных. Это определяется во время выполнения запроса. Чаще всего происходит при ODBC или OData соединениях.
Не определен		Указывает на неопределенный план запроса, часто из-за использования вручную введенных таблиц или неподдерживаемых преобразований/коннекторов.
Неизвестно		Указывает, что план запроса недоступен, что может быть связано с ошибкой или тем, что запрос включает форматы данных, отличные от таблиц.

Рис. 15.7. Обзор индикаторов свертывания запросов

Когда на шаге запроса отображается некий индикатор, все последующие шаги, имеющие вертикальную линию того же цвета, имеют тот же статус свертывания запроса. Эти индикаторы предоставляют сведения о том, как Power Query взаимодействует с источником данных и использует ли он возможности обработки источника данных.

Рассмотрим примеры. Предположим, мы используем функцию *Sql.Database* для подключения к базе данных SQL. Коннектор поддерживает свертывание и пытается преобразовать код M в запрос источника данных. Мы можем использовать выражение:

```
Sql.Database( ServerAddress, DatabaseName )
```

Шаги запроса разбиваются на три области:



Рис. 15.8. Индикаторы сигнализируют о состоянии сворачивания каждого шага

Первые четыре шага запроса свернуты, т.е., Power Query преобразовал их в запрос, понятный источнику данных. На первом шаге отображается индикатор сворачивания, а следующие три шага отмечены вертикальной линией того же цвета, указывающей на их свернутое состояние.

На шаге *Added index* отражается индикатор *Not Folding* (не сворачивается). Это поворотный момент для оптимизации запроса, так как он указывает, что дальнейшие шаги не будут сворачиваться и будут обрабатываться подсистемой Power Query.

Важно отметить, что наличие индикатора *Not Folding* не означает, что весь запрос не сворачивается. Просто часть запроса, начиная с этой точки, не сворачивается, хотя другие части все еще могут быть свернуты.

Также следует помнить, что текущий план запроса оценки сворачивания поддерживает только таблицы. Таким образом, начиная с шага *Constant value*, на котором добавляется постоянное значение, плану запроса присваивается индикатор *Не определен*, отражающий это ограничение.

В новом интерфейсе Power Query наряду с индикаторами можно использовать еще один метод – план запроса.

План запроса

План запроса описывает, как движок выполняет запрос к данным. Это полезный инструмент для анализа порядка, в котором движок выполняет свои операции. Новый интерфейс Power Query позволяет изучать план запроса, созданный ядром M. Чтобы получить к нему доступ, щелкните правой кнопкой мыши любой шаг и выберите *View query plan*:

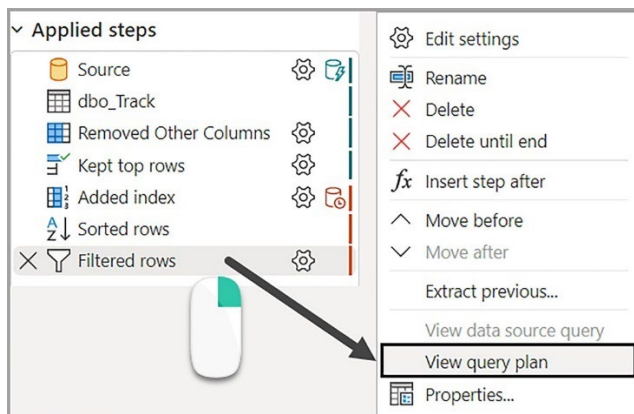


Рис. 15.9. Просмотр плана запроса

После этого появится окно, в котором отображаются шаги плана запроса:

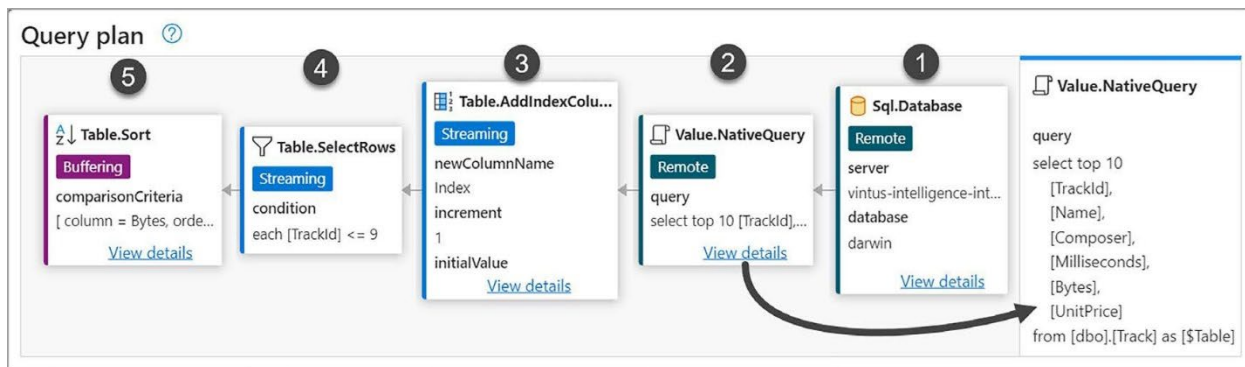


Рис. 15.10. План запроса, сгенерированный движком Power Query

План запроса представляет собой все шаги, ведущие к выбранному шагу. Он читается справа налево, и видно, как он связан с шагами на рисунке 15.9:

1. Подключение к базе данных SQL с помощью свертывания запросов.
2. Извлечение первых 10 строк с помощью собственного запроса, в том числе с помощью свертывания запросов.
3. Добавление индексного столбца, который прерывает свертывание запроса. На это изменение указывает переход от метки *Remote* – внешняя обработка [в источнике данных] к *Streaming* – потоковая обработка [в Power Query].
4. Фильтрация строк.
5. Сортировка строк.

Последовательность в плане запроса не всегда соответствует порядку применяемых шагов. Это может сбить с толку. Движок Power Query оптимизирует запрос, потенциально выполняя следующие действия:

- *Объединение шагов.* Движок может объединить шаги, применив нескольких фильтров.
- *Удаление шагов.* Если будущий шаг делает предыдущий нерелевантным, план запроса может удалить его.
- *Изменение порядка шагов.* Движок может изменить порядок шагов, чтобы увеличить число сворачиваемых шагов.

В приведенном на рис. 15.10 плане запроса движок фильтрует строки перед их сортировкой, хотя порядок примененных шагов обратный. План запроса более эффективен, так как сортировка сокращенного набора данных требует меньше ресурсов для получения того же результата.

Операции, которые успешно сворачиваются в плане запроса, помечаются метками *Remote*. Щелкнув *Подробности (View details)* на шаге 2, можно просмотреть запрос к источнику данных. Эти визуальные подсказки можно использовать, чтобы узнать, успешно ли сворачиваются шаги запроса.

Если вы заметите, что критические шаги не сворачиваются, рассмотрите возможность реструктуризации запроса, чтобы улучшить сворачиваемость и тем самым повысить производительность. Можно разбить сложные операции на более простые, которые будут сворачиваться или переставить шаги, чтобы передать больше операций в источник данных.

Вы узнали о трех средствах проверки сворачиваемости запроса (шага). Вы также можете проверить, сворачивается ли шаг запроса, с помощью функции диагностики запроса. Тем не менее, мы не рекомендуем использовать ее, так как информацию трудно интерпретировать.

Узнав, как определить сворачиваемость, давайте подробнее изучим операции, которые обычно сворачиваются, и те, которые этого не делают.

Операции и их влияние на свертывание

В этом разделе мы подробно рассмотрим, какие операции сворачиваются, а какие нет. Хотя механизм свертывания запросов не поддерживает одни и те же операции для каждого источника данных, можно выявить общие принципы. Реляционные базы данных (SQL Server, Oracle и т. д.), кубы (SSAS, SAP Hana/BW и Google Analytics), Azure Data Explorer, OData и потоки данных уровня *Премиум*, поддерживают свертывание запросов. Поскольку базы данных SQL относятся к наиболее распространенным источникам данных, следующие примеры связаны с SQL.

Сворачиваемые операции

Единственные операции, которые способны сворачиваться, – это преобразования. Как правило, если преобразование помещается в инструкцию SELECT, оно, скорее всего, будет свернуто. К таким операторам относятся WHERE, GROUP BY и JOIN. Вы также можете создавать столбцы с простыми выражениями, совместимыми с базами данных SQL. Следующие типы операций обычно хорошо сворачиваются:

- Модификация столбцов: выбор, удаление, переименование или изменение порядка столбцов, что соответствует оператору SELECT в SQL.
- Пользовательские вычисления: поддерживаются базовые логические операции, текстовые преобразования, математические вычисления и изменения типов данных, которые вписываются в SELECT-запрос.
- Сортировка столбцов: соответствует оператору ORDER BY в SQL.
- Фильтрация строк: соответствует оператору WHERE в SQL.
- Группировка данных: связана с оператором GROUP BY.
- Соединение таблиц: обычные (не размытые) соединения попадают под оператор JOIN.
- Объединение запросов: соответствует оператору UNION ALL в SQL.
- Поворот и разворот данных: операции, аналогичные операторам PIVOT/UNPIVOT в SQL, позволяют изменить структуру данных для более удобного анализа.

Важно отметить, что результаты некоторых преобразований в Power Query могут отличаться в зависимости от того, выполняются ли они на стороне Power Query или на стороне источника данных, что связано с различиями в обработке. Например:

- Обработка значений *null* при объединениях: в Power Query два значения *null* считаются равными, что может привести к возвращению значений при соединении, тогда как в большинстве баз данных значения *null* не равны.
- Чувствительность к регистру при фильтрации: допустим, вы фильтруете столбец по названию города *Mexico* в Power Query. По умолчанию операция чувствительна к регистру. Однако, когда этот запрос сворачивается и выполняется на стороне источника данных, регистр может игнорироваться, так как некоторые базы данных не учитывают регистр при фильтрации текста. Это означает, что одна и та же операция может возвращать разные результаты в зависимости от того, где она выполняется.

При переводе кода M в код источника данных, необходимо учитывать эти различия. Если вы хотите большего контроля над запросами, создавайте пользовательские запросы на основе функции *Value.NativeQuery*. Мы подробно рассмотрим ее позже, когда будем изучать стратегии поддержания свертывания запросов.

Несворачиваемые операции

Рассмотрим, какие операции не сворачиваются.

Преобразования

Преобразования – одна из наиболее важных категорий операций, которые не сворачиваются. Несмотря на то, что многие коннекторы эффективно преобразуют код M в запросы к источнику данных, часть преобразований все еще не сворачивается. Обычно это происходит, когда коннектор не поддерживает некоторые преобразования или источник данных не имеет эквивалентных операций. К распространенным несворачиваемым преобразованиям относятся:

- Добавление индексных столбцов или столбцов ранжирования.
- Объединение или добавление различных источников данных.
- Объединение запросов с разными уровнями конфиденциальности.
- Операции, для которых нет эквивалентных в источнике данных: транспонирование, сохранение нижних *n* строк и каждое слово с заглавной буквы.

Нет никакой гарантии, свернется ли операция с типами данных. Например, изменение в тип *text* может свернуться, а преобразование в тип времени – нет. Коннекторы постоянно обновляются, так что некоторые операции, которые не сворачиваются сегодня, могут быть свернуты в будущем. Помимо преобразований на свертывание влияют уровни конфиденциальности.

Уровни конфиденциальности источника данных

При работе с источниками данных с несовместимыми уровнями конфиденциальности может срабатывать фаервол (*formula firewall*), который предотвращает свертывание запросов. Подробнее см. ниже в этой главе.

Нативные запросы к базам данных

Использование нативных запросов в коде M обычно негативно влияет на свертывание запросов. Такие запросы используют функцию *Value.NativeQuery* или пользовательские операторы в функциях доступа к данным. Тем не менее, существуют случаи, когда пользовательские запросы все еще могут сворачиваться, и мы рассмотрим их далее.

Функции, предназначенные для предотвращения свертывания запросов

Некоторые функции предназначены для предотвращения свертывания запросов. Например, *Table.StopFolding* гарантирует, что последующие операции будут выполняться в Power Query. Такие функции можно использовать, когда источник данных, несмотря на поддержку свертывания, работает очень медленно.

Аналогичным образом, функции *List.Buffer*, *Table.Buffer* и *Binary.Buffer* также прекращают свертывание при загрузке данных в память. Буферные функции и их роль в оптимизации будут рассмотрены далее в этой главе.

Учитывая эти факторы, какие стратегии можно использовать, чтобы сохранить свертывание запросов как можно дольше?

Стратегии поддержки свертывания запросов

В этом разделе мы рассмотрим различные сценарии, которые нарушают свертывание запросов, и методы противодействия. Чтобы экспериментировать с кодом потребуется доступ к источнику данных, поддерживающему свертывание запросов, например к базе данных. Поскольку в этой книге основное внимание уделяется коду M, следующие примеры, показывающие способы сохранения свертывания запросов, предназначены только для чтения.

Перестановка шагов

Порядок выполнения операций может повлиять на свертывание запросов. Может потребоваться переставить шаги, чтобы источник данных выполнял как можно больше операций. Предположим, у нас есть набор данных с музыкальными треками, и мы хотим выполнить преобразования:

- *RemoveColumns*: выбрать столбцы *TrackId*, *Name*, *Composer* и *Milliseconds* из исходной таблицы.
- *KeepTopRows*: сохранить первые 10 строк таблицы.
- *Filter_TrackID*: отфильтровать *TrackId* со значением ≥ 5 .

На следующем рисунке показаны индикаторы свертывания запросов для каждого шага:

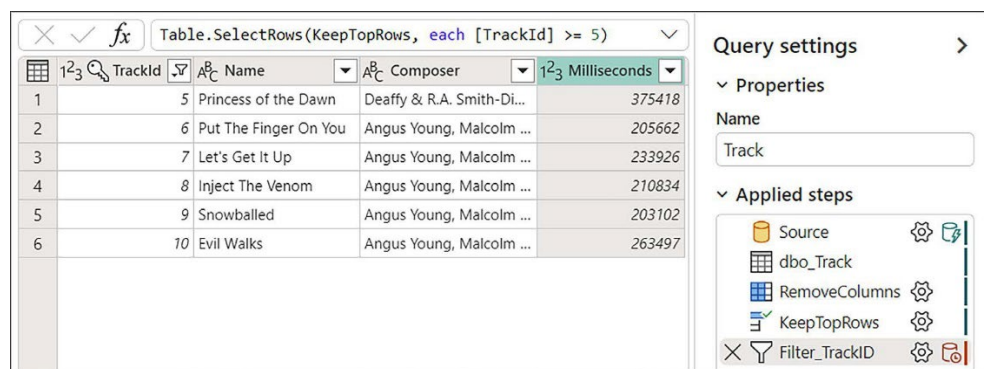


Рис. 15.11. Индикаторы свертывания запросов показывают, будет ли свернут шаг

Первые четыре шага сворачиваются, то есть их выполнит источник данных. Шаг *Filter_TrackID* не сворачивается. Power Query выполнит его после загрузки результатов с предыдущих шагов. Это удивительно, так как фильтрация обычно поддерживается источником данных.

В таких ситуациях стоит поэкспериментировать с последовательностью шагов. Изменение порядка может улучшить способность коннектора поддерживать свертывание запроса на протяжении большего количества шагов. Попробуем переместить шаг *Filter_TrackID* ближе к началу:

Table.FirstN(Filter_TrackID, 10)				
	TrackId	Name	Composer	Milliseconds
1	5	Princess of the Dawn	Deaffy & R.A. Smith-Di...	375418
2	6	Put The Finger On You	Angus Young, Malcolm ...	205662
3	7	Let's Get It Up	Angus Young, Malcolm ...	233926
4	8	Inject The Venom	Angus Young, Malcolm ...	210834
5	9	Snowballed	Angus Young, Malcolm ...	203102
6	10	Evil Walks	Angus Young, Malcolm ...	263497
7	11	C.O.D.	Angus Young, Malcolm ...	199836
8	12	Breaking The Rules	Angus Young, Malcolm ...	263288
9	13	Night Of The Long Kn...	Angus Young, Malcolm ...	205688
10	14	Spellbound	Angus Young, Malcolm ...	270863

Query settings

Properties

Name: Track

Applied steps

- Source
- dbo_Track
- RemoveColumns
- Filter_TrackID
- KeepTopRows

Рис. 15.12. Теперь сворачиваются все шаги запроса

В приведенном сценарии перемещение шага привело к получению иной таблицы, так что перемещение шага может быть нежелательным. Мы просто показали, что порядок шагов может изменить объем сворачиваемых операций.

Вы можете проверить запрос, который Power Query создает для источника данных, щелкнув правой кнопкой мыши шаг *KeepTopRows* и выбрав *Просмотреть запрос источника данных*: Выходной запрос выглядит следующим образом:

```

Data source query

select top 10
  [].[TrackId],
  [].[Name],
  [].[Composer],
  [].[Milliseconds]
from
(
  select [TrackId],
    [Name],
    [Composer],
    [Milliseconds]
  from [dbo].[Track] as [Table]
) as [.]
where [].[TrackId] >= 5

```

Рис. 15.13. Запрос к источнику данных генерируется путем свертывания запроса

Этот запрос может быть не таким оптимальным, как SQL-запрос, написанный вручную, но он намного лучше, чем загружать все данные и затем применять преобразования в Power Query. Однако ничто не мешает вам передавать собственные SQL-запросы.

Работа с нативными запросами

При использовании Power Query специалисты с опытом работы с SQL могут подготовить оптимизированный запрос. Тем не менее, важно помнить о нескольких соображениях, чтобы обеспечить успешное свертывание запросов с помощью пользовательских SQL-запросов.

Основным методом подключения к базе данных SQL в Power Query является функция *Sql.Database*. Синтаксис:

```

Sql.Database(
  server as text,
  database as text,
  optional options as nullable record,
) as table

```

Рассмотрим сценарий, в котором SQL-запрос хранится на шаге с именем *myQuery*:

```

myQuery =
"SELECT TOP (10)
  [TrackId],[Name], [Composer], [Milliseconds]
FROM [dbo].[Track]
WHERE [TrackId] >= 5"

```

Вы можете включить этот запрос в функцию *Sql.Database*:

```
Sql.Database( ServerAddress,  
  DatabaseName,  
  [ query = myQuery ]  
)
```

Пользовательский SQL-запрос вернет:

TrackId	Name	Composer	Milliseconds
1	5 Princess of the D...	Deaffy & R.A. Smith-Diesel	375418
2	6 Put The Finger O...	Angus Young, Malcolm Young, Brian Johns...	205662
3	7 Let's Get It Up	Angus Young, Malcolm Young, Brian Johns...	233926
4	8 Inject The Venom	Angus Young, Malcolm Young, Brian Johns...	210834
5	9 Snowballed	Angus Young, Malcolm Young, Brian Johns...	203102
6	10 Evil Walks	Angus Young, Malcolm Young, Brian Johns...	263497
7	11 C.O.D.	Angus Young, Malcolm Young, Brian Johns...	199836

Рис. 15.14. Свертывание пользовательского SQL-запроса

Индикаторы подтверждают, что запрос обрабатывается в источнике данных. Рассмотрим, что произойдет, если мы добавим фильтрацию строк – операцию, которая обычно не мешает свертыванию запроса. Удивительно, но индикатор шага *Filter_Name* показывает, что свертывание запроса не происходит:

TrackId	Name	Composer	Milliseconds
1	8 Inject The Venom	Angus Young, Malcolm Young, Brian Johns...	210834

Рис. 15.15. Свертывание запроса не происходит для шагов, следующих за пользовательским SQL-запросом

Хотя мы можем передать пользовательский SQL-запрос в функцию *Sql.Database*, любые последующие шаги к сожалению не будут свернуты. Хорошая новость заключается в том, что есть решение для работы с нативными SQL-запросами. Мы можем использовать функцию *Value.NativeQuery* для свертывания запросов с пользовательскими SQL-запросами. Как это работает? Вместо того чтобы вводить SQL-инструкцию непосредственно в функцию *Sql.Database*, мы предоставим ее функции *Value.NativeQuery*:

```
1 let  
2   myQuery =  
3     "SELECT TOP (10)  
4       [TrackId], [Name], [Composer] ,[Milliseconds]  
5       FROM [dbo].[Track]  
6       WHERE [TrackId] >= 5",  
7   Source = Sql.Database( ServerAddress, DatabaseName ),  
8   NativeQuery = Value.NativeQuery( Source, myQuery, null, [EnableFolding = true] )  
9 in  
10  NativeQuery
```

Рис. 15.16. Пользовательский SQL-запрос, переданный в функцию *Value.NativeQuery*

Вот как выглядит этот процесс. Сначала установите соединение с базой данных с помощью функции *Sql.Database*. Далее настройте функцию *Value.NativeQuery*:

- Передайте подключение к базе данных в качестве первого аргумента функции *Value.NativeQuery*.
- Включите пользовательский SQL-запрос в виде строки во второй аргумент.
- Поместите *null* в третьем аргументе – необязательные параметры не требуются.

- Для записи параметров укажите аргумент *EnableFolding* со значением *true*, чтобы обеспечить свертывание запроса.

При такой настройке, если мы попытаемся отфильтровать столбец *Name* по значению *Inject the Venom*, мы увидим, что свертывание запроса эффективно применяется ко всем шагам запроса:

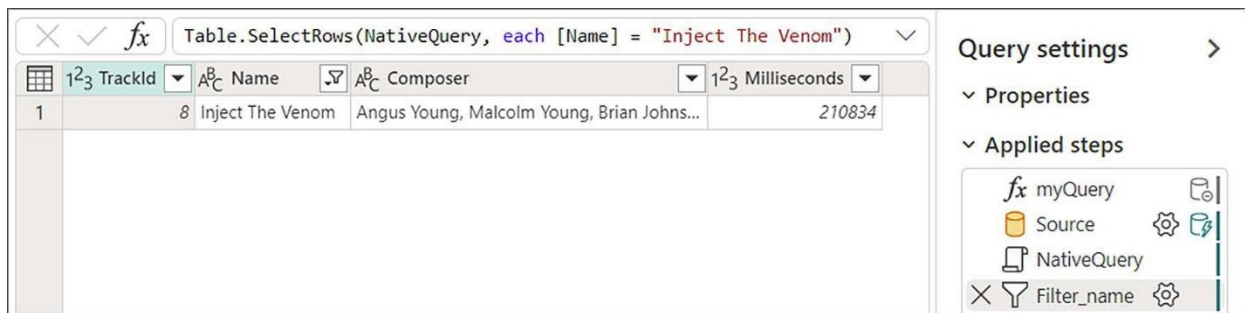


Рис. 15.17. Функция *Value.NativeQuery* позволяет сворачивать запросы, используя пользовательский SQL-запрос

В запросе к источнику данных мы видим как предоставленный нами пользовательский SQL-запрос, так и дополнительный код, использующий механизм свертывания запросов:

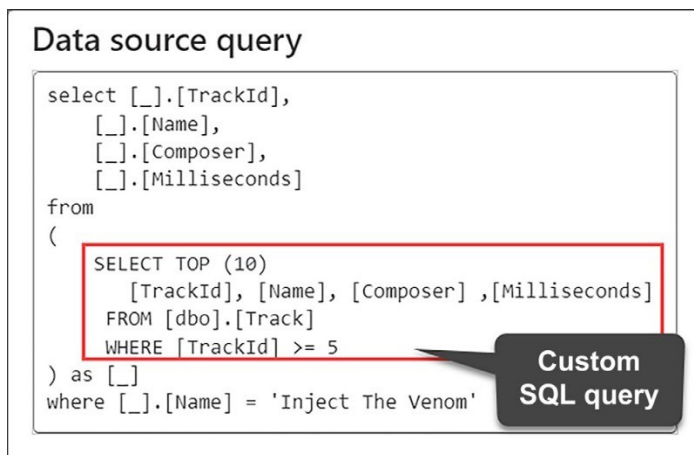


Рис. 15.18. Запрос к источнику данных, сгенерированный механизмом свертывания запросов

Переписывание кода

Иногда вы можете столкнуться со сценарием, в котором некая операция не поддерживает свертывание запросов, хотя вы ожидаете обратного. Если шаг не сворачивается, переписывание выражения может привести к созданию версии, поддерживающей свертывание запросов. В следующей таблице столбец *TrackID* имеет значение *null* в строке 5:

	TrackId	Name	Composer	Milliseconds
1	1	For Those About ...	Angus Young, Malc...	343719
2	2	Balls to the Wall	<i>null</i>	342562
3	3	Fast As a Shark	F. Baltes, S. Kaufma...	230619
4	4	Restless and Wild	F. Baltes, R.A. Smit...	252051
5	<i>null</i>	Princess of the D...	Deaffy & R.A. Smit...	375418
6	6	Put The Finger O...	Angus Young, Malc...	205662
7	7	Let's Get It Up	Angus Young, Malc...	233926
8	8	Inject The Venom	Angus Young, Malc...	210834

Рис. 15.19. Исходные данные

Предположим, вы хотите выбрать идентификаторы 3, 6 и 8. Этого можно добиться с помощью функции *List.Contains*, которая преобразуется в оператор *IN* в SQL и поддерживает свертывание запросов. Выглядит это следующим образом:

```
Table.SelectRows( Source,
  each List.Contains( {3, 6, 8}, [TrackId] ) )
```

TrackId	Name	Composer	Milliseconds
1	6 Put The Finger On You	Angus Young, Malcolm Yo...	205662
2	8 Inject The Venom	Angus Young, Malcolm Yo...	210834
3	3 Fast As a Shark	F. Baltes, S. Kaufman, U. Di...	230619

Рис. 15.20. *List.Contains* – преобразование, которое может сворачиваться

Проблема возникает, если необходимо включить в выборку нулевые значения. В SQL значения NULL не допускаются в предложении IN. Следующее выражение не сворачивается:

```
Table.SelectRows( Source,
  each List.Contains( {3, 6, 8, null}, [TrackId] ) )
```

TrackId	Name	Composer	Milliseconds
1	3 Fast As a Shark	F. Baltes, S. Kaufman, U. Di...	230619
2	null Princess of the Dawn	Deaffy & R.A. Smith-Diesel	375418
3	6 Put The Finger On You	Angus Young, Malcolm Yo...	205662
4	8 Inject The Venom	Angus Young, Malcolm Yo...	210834

Рис. 15.21. Свертывание не происходит при включении значений NULL в *List.Contains*

Чтобы решить эту проблему, можно поэкспериментировать, включив условие *null* отдельно от функции *List.Contains*:

```
Table.SelectRows( Source,
  each List.Contains( {3, 6, 8 }, [TrackId] )
  or [TrackId] = null )
```

TrackId	Name	Composer	Milliseconds
1	6 Put The Finger On You	Angus Young, Malcolm ...	205662
2	8 Inject The Venom	Angus Young, Malcolm ...	210834
3	3 Fast As a Shark	F. Baltes, S. Kaufman, U...	230619
4	null Princess of the Dawn	Deaffy & R.A. Smith-Di...	375418

Рис. 15.22. Свертывание работает при фильтрации нулевых значений отдельно

Использование свертывания с несколькими базами данных

Еще одна проблема, связанная со свертыванием запросов, возникает при работе с данными в разных базах. Power Query не предназначен для свертывания запросов, которые по умолчанию охватывают несколько баз данных. Вы можете столкнуться с этим ограничением при выполнении операции слияния. Но, когда эти базы размещены на одном сервере, существует обходной путь.

Рассмотрим сценарий, в котором мы хотим сравнить адреса электронной почты клиентов в двух базах данных, чтобы выявить сходства. Базы AW (AdventureWorks) и CT (Contoso) содержат таблицы с аналогичной структурой:

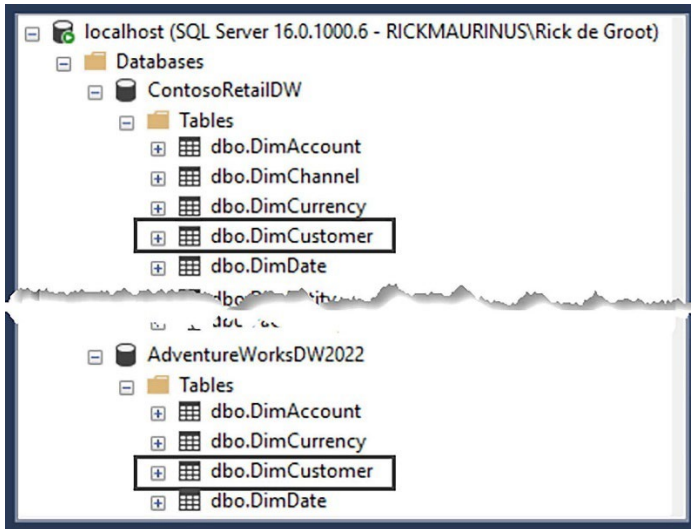


Рис. 15.23. Два идентичных названия таблиц в разных базах данных на одном сервере

Мы используем два запроса. Каждый запрос подключается к одной базе данных и извлекает информацию из таблицы клиентов. Мы сосредоточимся на полях *BirthDate* и *EmailAddress*:

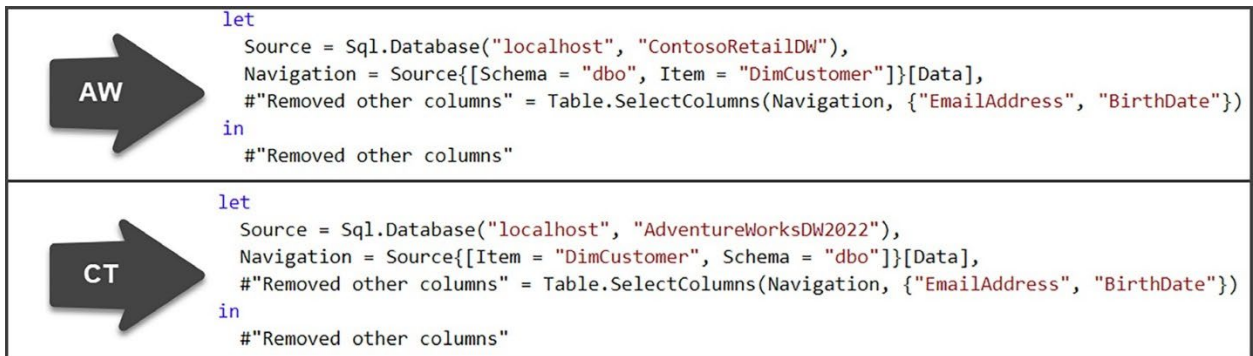


Рис. 15.24. Запросы, которые извлекают столбцы из таблиц в двух базах данных

Индикаторы свертывания запросов для обоих запросов показывают, что все шаги в этих запросах были успешно свернуты:

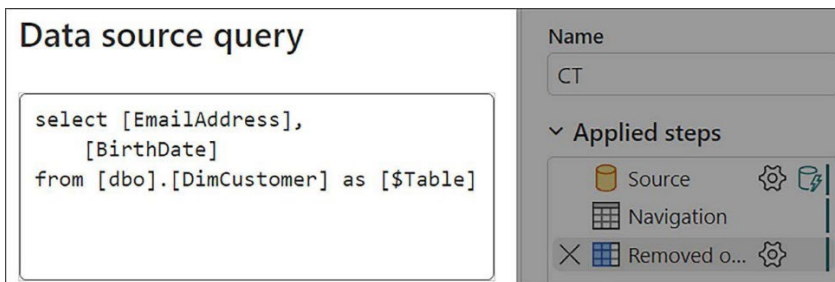


Рис. 15.25. Запрос к базе данных CT, сгенерированный механизмом свертывания запросов

На следующем шаге соединим данных из этих двух запросов. Для этого перейдите *Главная* → *Объединить запросы* → *Объединить запросы в новый*. В появившемся диалоговом окне выберите *EmailAddress* в качестве поля для объединения и выберите *Внешнее соединение слева*:

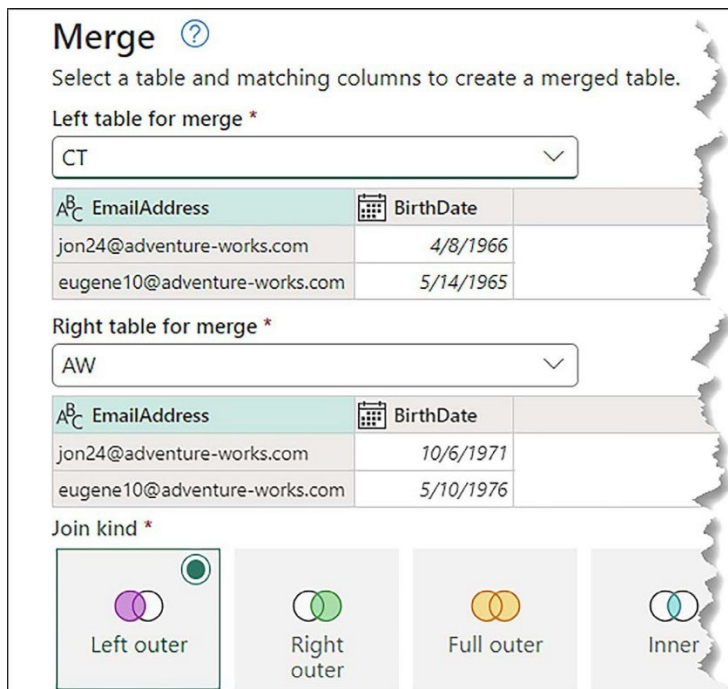


Рис. 15.26. Окно настройки объединения запросов

После слияния мы разворачиваем столбец *BirthDate*, чтобы обеспечить прямое сравнение в запросе:

Table.ExpandTableColumn(Source, "AW", {"BirthDate"}, {"AW BirthDate"})

	EmailAddress	BirthDate	AW BirthDate
1	jon24@adventure-works.com	4/8/1966	10/6/1971
2	eugene10@adventure-works.com	5/14/1965	5/10/1976
3	ruben35@adventure-works.com	8/12/1965	2/9/1971
4	christy12@adventure-works.com	2/15/1968	8/14/1973
5	elizabeth5@adventure-works.com	8/8/1968	8/5/1979

Advanced editor

```

1 let
2   Source = Table.NestedJoin(CT, {"EmailAddress"}, AW, {"EmailAddress"}, "AW", JoinKind.LeftOuter),
3   #"Expanded AW" = Table.ExpandTableColumn(Source, "AW", {"BirthDate"}, {"AW BirthDate"})
4 in
5   #"Expanded AW"

```

Performs a left outer join on the 'EmailAddress' Field

The join on multiple databases does not fold

Рис. 15.27. Объединение запросов не сворачивается при использовании нескольких баз данных

Несмотря на то, что объединение таблиц из баз данных AW (AdventureWorks) и CT (Contoso) прошло успешно, операция *Expanded* прервала свертывание запросов. Хотя отдельные запросы можно сворачивать, последующее слияние – нет. Можно ли это исправить?

Полезным оказывается параметр *EnableCrossDatabaseFolding* функции *Sql.Database*. Этот параметр предназначен для упрощения свертывания запросов между несколькими базами данных, если они расположены на одном сервере. Изначально мы подключились к базе данных CT с помощью выражения:

```
Sql.Database("localhost", "ContosoRetailDW")
```

Чтобы использовать свертывание запросов между базами данных, это выражение следует изменить:

```
Sql.Database(
  "localhost",
  "ContosoRetailDW",
  [EnableCrossDatabaseFolding = true])
```

Чтобы свертывание запросов работало, мы должны применить такую же модификацию к подключению к базе данных AW. Пересмотренный подход приводит к сценарию, в котором операция соединения между двумя базами данных эффективно обрабатывается путем свертывания запросов:

The screenshot shows a Power Query interface with a data table and a query editor. The data table has columns: EmailAddress, BirthDate, and AW BirthDate. The query editor shows a SQL query that joins data from two databases: ContosoRetail and AdventureWorks. The query uses a left outer join to combine columns from both databases based on email addresses.

```

select [$Outer].[EmailAddress] as [EmailAddress],
       [$Outer].[BirthDate] as [BirthDate],
       [$Inner].[BirthDate2] as [AW BirthDate]
from
(
  select [BirthDate],
         [EmailAddress]
  from [ContosoRetailDW].[dbo].[DimCustomer] as [$Table]
) as [$Outer]
left outer join
(
  select [EmailAddress] as [EmailAddress2],
         [BirthDate] as [BirthDate2]
  from [AdventureWorksDW2022].[dbo].[DimCustomer] as [$Table]
) as [$Inner] on ([$Outer].[EmailAddress] = [$Inner].[EmailAddress2] or [$Outer].[EmailAddress] is null and [$Inner].[EmailAddress2] is null)
  
```

Рис. 15.28. Запрос к источнику данных, объединяющий таблицы из нескольких баз данных на одном сервере

Свертывание запросов между базами данных позволяет источнику данных самому обрабатывать преобразования, а не полагаться на движок Power Query. Это работает даже тогда, когда операции ссылаются на несколько баз данных на одном сервере.

Итак, мы рекомендуем поручить источнику данных выполнять всю тяжелую работу по преобразованиям. Благодаря предоставленным стратегиям вы теперь знаете несколько способов поддержания свертывания запросов.

Далее мы изучим еще одну важную тему – фаервол формул. Это функция, которая может повлиять на свертывание запросов, но также является распространенным источником разочарования, когда она блокирует выполнение запросов.

Фаервол формул (formula firewall)

При работе с запросами распространено комбинирование разных источников данных. Вполне вероятно, что вы столкнетесь с ошибкой, которая не позволит вам объединить данные. Эта ошибка является результатом работы фаервола формул, также известного как брандмауэр конфиденциальности данных.

Что такое фаервол формул?

Фаервол формул – функция Power Query, которая предотвращает случайную передачу данных между источниками. Эта функция особенно важна при работе с конфиденциальной информацией. Фаервол был разработан из-за свертывания запросов. Свертывание запросов позволяет преобразовывать запрос Power Query в запрос к источнику данных.

Представьте ситуацию, в которой вы работаете с двумя запросами в Power Query. *Query1* содержит конфиденциальную информацию, например номера социального страхования. Во внутреннем соединении вы используете эти данные для извлечения сведений из *Query2*. Однако для повышения эффективности следует избегать импорта всей таблицы SQL, связанной с *Query2*, в Power Query. Импортировать всю таблицу только для того, чтобы потом отфильтровать ненужные строки, не оптимально. В конце концов, использование свертывания запросов является более эффективным подходом, так как при свертывании импортируются только необходимые данные.

Однако, несмотря на то, что свертывание запросов эффективно, оно также сопряжено с потенциальным риском. В процессе свертывания запроса конфиденциальная информация о номерах социального страхования из *Query1* может быть непреднамеренно внедрена в запрос *Query2* к источнику данных. Если бы кто-то, обладающий знаниями в области баз данных, или кто-либо, кто следит за вашей сетью, перехватил этот запрос, он потенциально мог бы получить доступ к конфиденциальной информации, встроенной в него.

Именно здесь в игру вступает фаервол формул. Его цель – предотвратить отправку во внешние источники конфиденциальных данных. Чтобы определить, в чем заключается потенциальный риск, фаервол формул использует разделы.

Общие сведения о разделах

При оценке запроса Power Query разбивает сам запрос и связанные с ним запросы на разделы. Эти разделы представляют собой группы из одного или нескольких шагов. Всякий раз, когда раздел ссылается на другой раздел, фаервол формул заменяет ссылку на вызов функции *Value.Firewall*. Эта функция гарантирует, что разделы не будут обращаться друг к другу напрямую. Вместо этого все ссылки должны быть сначала оценены фаерволом. Данные допускаются в текущий раздел только после того, как фаервол авторизует взаимодействие между разделами.

Создание разделов невероятно важно. Включение меньшего или большего количества шагов в один раздел может определить, будет ли задействован фаервол. Однако процесс разделения запросов на разделы сложен и выходит за рамки данной книги. Для тех, кто заинтересован в более глубоком понимании этого процесса, дополнительная информация доступна по [адресу \(русская версия\)](#). Мы будем ссылаться на эту информацию далее.

Основной принцип фаервола формул

Фаервол формул работает на основе основного принципа, связанного с разделами. Этот принцип состоит из двух ключевых пунктов:

- *Ссылка на другие разделы*: раздел может использовать данные или результаты другого раздела; этот раздел может быть другим запросом или разделом того же запроса.
- *Доступ к совместимым источникам данных*: раздел может получать данные из внешних источников, если у них совместимый уровень конфиденциальности.

Возможен любой из двух вариантов, но не оба одновременно.

Нарушение этого правила приведет к одному из двух сообщений фаервола:

- Ошибка при ссылке *Formula.Firewall*: Запрос "X" (шаг "Y") ссылается на другие запросы или шаги, поэтому он не может напрямую обращаться к источнику данных. Пожалуйста, восстановите эту комбинацию данных.
- Ошибка при обращении к несовместимым источникам данных *Formula.Firewall*: Запрос 'X' (шаг 'Y') – это доступ к источникам данных с несовместимыми уровнями конфиденциальности. Пожалуйста, восстановите эту комбинацию данных.

Прочитав коды ошибок, трудно понять, что происходит, но подождите. В следующем разделе мы рассмотрим причины этих ошибок и обсудим стратегии их устранения.

Ошибка фаервола: ссылка на другие разделы

Эта ошибка блокирует выдачу результатов запроса. К сожалению, ее устранение без знания механики фаервола может быть сложной задачей. Рассмотрим следующий пример:

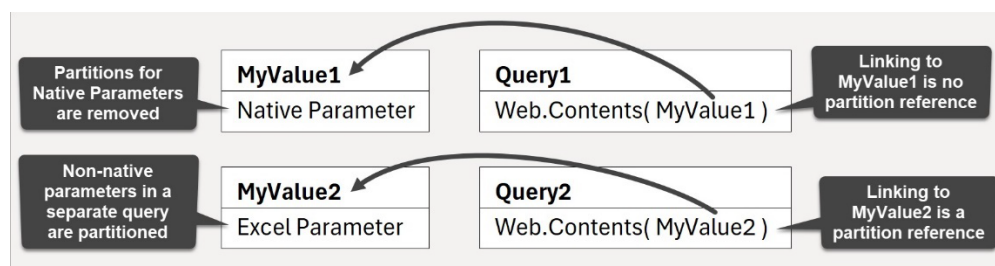


Рис. 15.29. Два похожих запроса, один из которых сталкивается с ошибкой *Formula Firewall*

Здесь два похожих запроса, но только один из них обнаруживает ошибку *Formula Firewall*. В *Query1* есть ссылка на параметр с именем *MyValue1*, созданный через пользовательский

интерфейс. Процесс секционирования автоматически удаляет секции из таких собственных параметров, что обеспечивает бесперебойную работу *Query1*.

Однако, если вы вручную настроите параметр, который извлекает значение из файла Excel, как это видно на примере *MyValue2*, процесс секционирования будет работать иначе. Он не удаляет раздел, поэтому любая ссылка на *MyValue2* рассматривается как ссылка на раздел. Поэтому, когда *Query2* пытается использовать *MyValue2* с функцией *Web.Contents* для доступа к другому источнику данных и в то же время ссылается на другой раздел, он запускает фаервол.

Давайте рассмотрим пример, чтобы увидеть эту ошибку в действии. Чтобы следовать приведенным ниже примерам, вы можете скачать файлы упражнений из репозитория на [GitHub](#).

Подключение к URL с помощью нативных параметров

Распространенной операцией для извлечения данных является подключение по URL-адресу. Например, получение оценок фильмов с IMDb, получение данных из таблицы в Википедии или возврат последней информации о прогнозе погоды. Независимо от того, какая информация вам нужна, вы можете подключиться к сайту с помощью веб-коннектора Power Query, что позволяет проиллюстрировать работу фаервола формул.

Предположим, вы хотите подключиться к веб-сайту [PowerQuery.How](#) и получить HTML-код. Сначала создадим параметр. Для этого в редакторе Power Query пройдите *Главная* → *Управление параметрами* → *Создать параметр*. Назовите параметр *myURLParameter*. Установите *Тип* = *Текст*, введите *Текущее значение* = *https://powerquery.how/*:

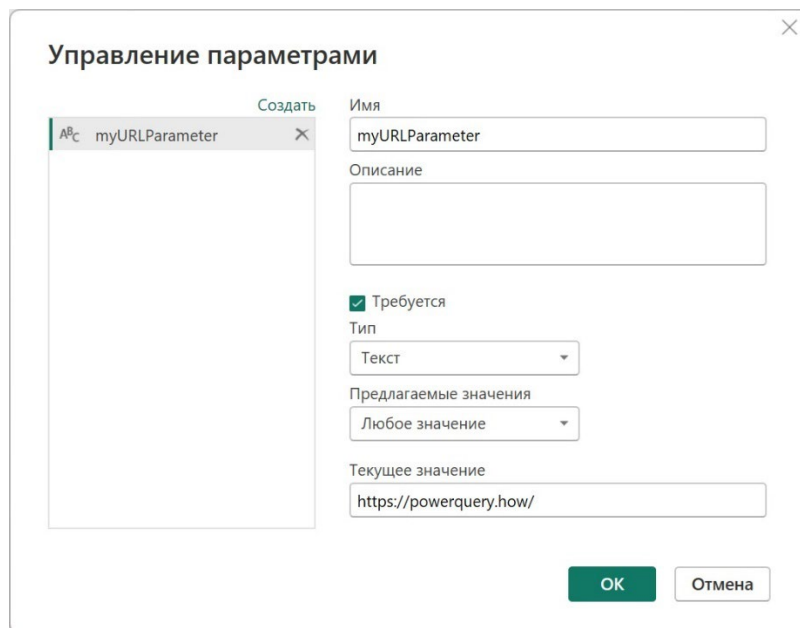


Рис. 15.30. Параметр, содержащий URL-адрес для подключения

Этот параметр включает статичный URL, и не имеет доступа к источникам данных. Далее настроим запрос для подключения к сайту и получения HTML-кода. В области *Запросы* щелкните правой кнопкой мыши, выберите *Новый запрос* → *Интернет*. В диалоговом окне *Из Интернета* в левой части выберите *Параметр*. Поскольку он один, *myURLParameter* подставится автоматически.

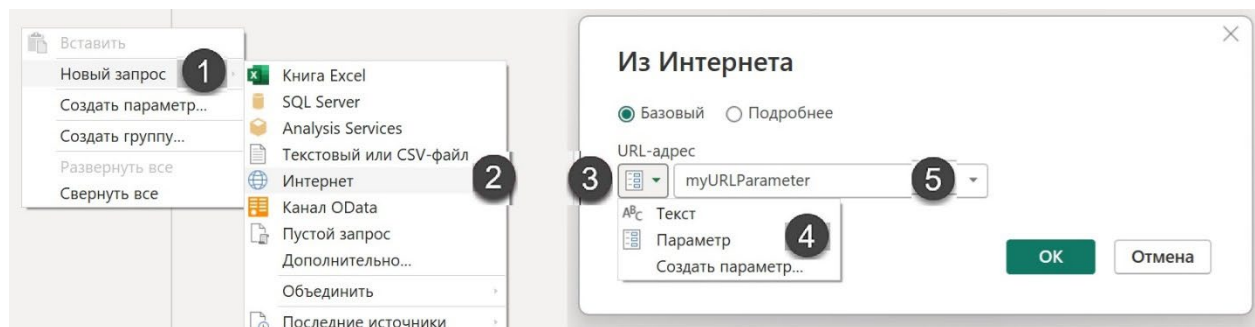


Рис. 15.31. Создание веб-запроса, использующего параметр в качестве URL-адреса

При первом подключении к веб-источнику вам будет предложено выбрать способ подключения. Выберите *Анонимный*, нажмите *Подключение*. На следующем экране установите флажок напротив поля *HTML-код*, чтобы вернуть содержимое HTML. Нажмите ОК:

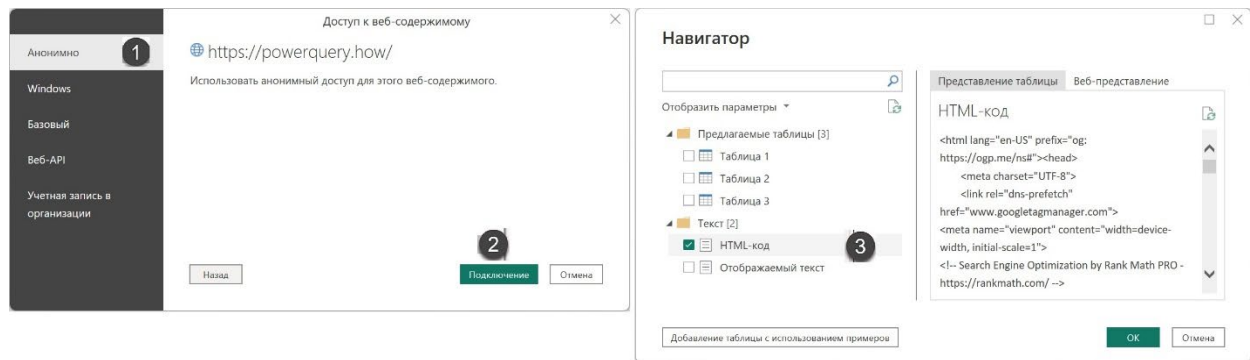


Рис. 15.32. Выбор вариантов доступа к веб-контенту и извлечения данных

Выполнив эти шаги, вы извлекаете содержимое HTML с веб-сайта с помощью жестко заданного параметра. Ваша среда содержит два запроса.

Первый – параметр *myURLParameter*, с кодом:

```
"https://powerquery.how/"  
  meta [IsParameterQuery=true, Type="Text", IsParameterQueryRequired=true]
```

Второй запрос называется *HTML-код* и задается одной строкой:

```
Web.BrowserContents(myURLParameter)
```

До этого момента фаервол не отмечал никаких проблем и успешно возвращал HTML-код. Такая плавная работа обусловлена процессом разбиения на разделы, как подробно описано на веб-сайте Microsoft. Этот процесс удаляет секции из собственных параметров, установленных с помощью пользовательского интерфейса Power Query.

В нашем случае запрос HTML-кода ссылается на *myURLParameter*. Однако, поскольку этот параметр не включен в раздел и не связан с какими-либо источниками данных, фаервол формул не активирует никаких предупреждений. Теперь перейдем к ситуации, когда небольшое изменение в настройках приводит к ошибке *Formula Firewall*.

Подключение к URL с помощью параметра Excel

Изменим наш подход. Вместо собственного параметра мы получим URL-адрес для подключения из таблицы на листе Excel. Такая настройка позволяет легко изменять URL-адрес в файле Excel без необходимости возиться с Power Query. Это обеспечивает удобный для пользователя способ ввода данных, но возникает риск столкнуться с фаерволом.

Чтобы продолжить работу, убедитесь, что вы скачали файл *ExcelParameterFile* из репозитория. Он содержит таблицу *ParameterTable*, состоящую из одного столбца с URL-адресом в виде текстового значения <https://powerquery.how/>:

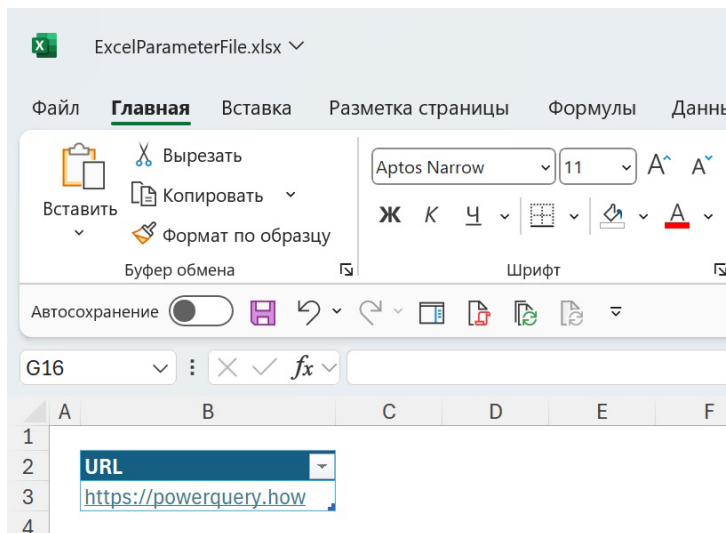


Рис. 15.33. Лист Excel, содержащий URL-адрес для запроса

Сохраните этот файл. Чтобы подключиться к нему, в редакторе Power Query пройдите *Создать источник* → *Книга Excel*, выберите файл, нажмите *Открыть*. В окне *Навигатор* выберите *ParameterTable* и нажмите *ОК*. После импорта данных появится таблица с одной ячейкой, содержащей URL. Чтобы использовать этот URL в своих запросах, вы должны извлечь значение из этой ячейки. Кликните правой кнопкой мыши по ячейке и выберите *Детализация*. Этот шаг вернет вам значение ячейки: `https://powerquery.how`.

Код запроса для извлечения значения из файла Excel:

```
let
    Источник = Excel.Workbook(File.Contents("...\Guide\15\ExcelParameterFile.xlsx"), null, true),
    ParameterTable_Table = Источник{[Item="ParameterTable",Kind="Table"]}[Data],
    URL = ParameterTable_Table{0}[URL]
in
    URL
```

Переименуйте запрос в *myExcelURL*. Выше мы использовали *myURLParameter*. Что, если мы заменим его на *myExcelURL*? Попробуйте использовать это в запросе:

```
Web.BrowserContents( myExcelURL )
```

Вы столкнетесь с ошибкой:

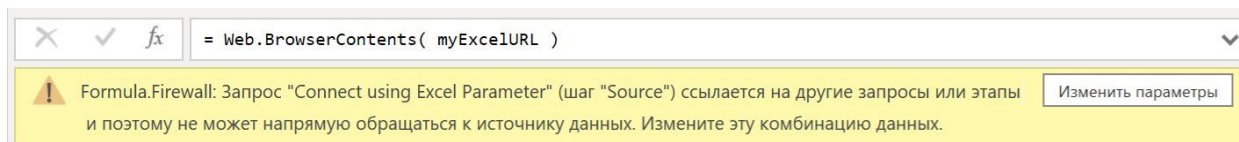


Рис. 15.36. Ошибка *Formula Firewall* при ссылке на запросы, подключающиеся к источнику данных

Почему возникает эта ошибка? При том, что мы успешно использовали параметр в предыдущем примере! Чтобы понять это, давайте сосредоточимся на фундаментальном принципе фаервола: «Раздел может либо ссылаться на другие разделы, либо получать доступ к данным из источников, которые имеют совместимый уровень конфиденциальности. Он не может делать и то, и другое одновременно».

В нашем первоначальном примере было две причины отсутствия ошибки. Во-первых, *myURLParameter* был нативным, жестко заданным параметром, который не имел доступа к каким-либо внешним источникам данных. Во-вторых, в процессе секционирования нативный параметр был исключен из его области, то есть ссылка на него не рассматривалась как ссылка на другой раздел.

Ситуация меняется, когда мы используем параметр, полученный из файла Excel. Такой параметр обращается к данным из другого источника (файла Excel) и, таким образом, включается в процесс секционирования. Когда функция *Web.BrowserContents* пытается получить доступ к веб-сайту

(PowerQuery.How) и одновременно ссылается на раздел (параметр Excel), она нарушает основное правило фаервола.

Секционирование в Power Query может быть довольно сложным. Даже чтение документации Microsoft по нему оставит у вас множество вопросов. Тем не менее, главное, что нужно помнить: если вы ссылаетесь на один запрос для получения данных из источника, например, из файла Excel, а затем пытаетесь использовать эти данные в другом запросе для доступа к источнику, это нарушает правила фаервола. Итак, как лучше всего решить эту проблему?

Устранение ошибки фаервола

Наша цель – извлечь URL-адрес из внешнего источника, и найти способ сохранить логику обращения к сайту в том же разделе. Рассмотрим два метода.

Способ 1: Использование функции

Первый метод предполагает хранение логики параметра *myExcelURL* в функции. Функция сохраняет логику в повторно используемом коде, но не выполняет его. Когда мы затем вызываем функцию в запросе, доступ к данным происходит в пределах раздела, в котором вызывается функция. Это отличается от ссылки на запрос, при которой доступ к данным происходит в другом запросе, и он рассматривается как отдельный раздел, что приведет к срабатыванию фаервола.

Чтобы создать функцию, вы можете добавить простое определение функции, которое ссылается на предыдущий шаг в конце вашего запроса:

```
let
    Источник = Excel.Workbook(File.Contents("...\Guide\15\ExcelParameterFile.xlsx"), null, true),
    ParameterTable_Table = Источник[{Item="ParameterTable",Kind="Table"}][Data],
    URL = ParameterTable_Table{0}[URL],
    Custom1 = () => URL
in
    Custom1
```

Этот код создает функцию. Назовите ее *fxMyExcelURL*. Используйте ее в другом запросе для обращения к сайту:

```
Web.BrowserContents( fxMyExcelURL() )
```

Это позволит успешно выполнить запрос, не сталкиваясь с фаерволом.

Способ 2: Интеграция логики в один запрос

Второй подход заключается в интеграции всей логики в одном запросе. Это гарантирует, что все связанные шаги находятся в одном разделе, что позволяет использовать параметр для доступа к внешнему источнику данных в соответствии с правилами фаервола. Чтобы объединить шаги, можно взять логику запроса книги Excel и добавить шаг в конце запроса. На этом шаге вы сошлетесь на URL-адрес из книги Excel:

```
let
    Источник = Excel.Workbook(File.Contents("...\Guide\15\ExcelParameterFile.xlsx"), null, true),
    ParameterTable_Table = Источник[{Item="ParameterTable",Kind="Table"}][Data],
    URL = ParameterTable_Table{0}[URL],
    HTMLCode = Web.BrowserContents( URL )
in
    HTMLCode
```

Объединение этих шагов в один запрос позволяет процессу секционирования сохранять шаги по доступу к данным в пределах одной секции. Этот подход успешно устраняет первую ошибку *Formula Firewall*. Чтобы убедиться, поэкспериментируйте с файлом упражнений этой главы.

Применив вышеприведенный код вы окажитесь в следующей ситуации:

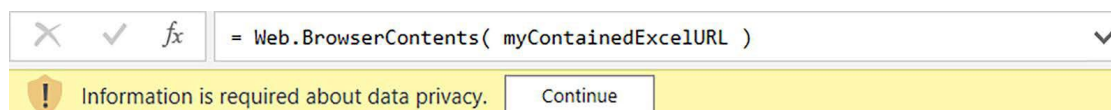


Рис. 15.37. Запрос на установку уровней конфиденциальности

Требуется дополнительная информация о конфиденциальности данных. Если вы нажмете *Продолжить* и выберете *Общий*, ваш запрос вернет результат. Однако, если вы выберете *Частный* или *Организационный*, запрос может привести ко второму типу ошибки *Formula Firewall*, о котором мы поговорим далее.

Ошибка фаервола: доступ к совместимым источникам данных

Вторая ошибка фаервола может произойти при объединении источников данных с несовместимыми уровнями конфиденциальности. Фаервол предназначен для предотвращения непреднамеренной утечки данных. Сообщение об этой ошибке выглядит следующим образом: *Formula.Firewall: Запрос 'X' (шаг 'Y') – это доступ к источникам данных с несовместимыми уровнями конфиденциальности. Пожалуйста, восстановите эту комбинацию данных.*

Рассмотрим сценарии, изображенные на рисунке:

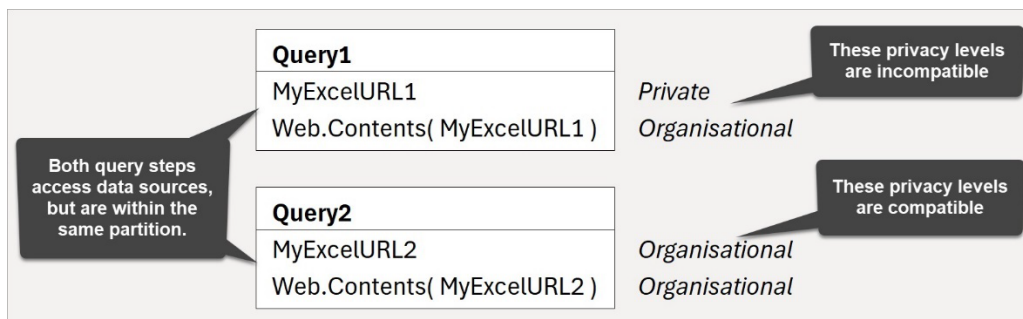


Рис. 15.38. Два запроса, один из которых содержит несовместимые источники данных

Query1 получает доступ к данным из двух внешних источников. Если бы шаг *Web.Contents* сослался на другой раздел, он вызывал бы ошибку «ссылка на другие разделы» из-за одновременного доступа к источнику данных и ссылки на другой раздел. Однако проблема не в этом. Данные находятся в одном запросе. Проблема *Query1* в несоответствии уровней конфиденциальности: источник данных *MyExcelURL1* помечен как *Частный*, в то время как для шага *Web.Contents* установлен уровень конфиденциальности *Организация*. Так как эти уровни не совпадают, Power Query сообщает о втором типе ошибки – доступе к несовместимым источникам данных.

Query2 не обнаруживает эту ошибку, так как оба шага запроса связаны с источниками данных с одинаковыми уровнями конфиденциальности. Таким образом, фаервол позволит объединить оба источника данных.

Понимание этого может показаться сложным, но не волнуйтесь, мы разберем пример. Чтобы понять сообщение об ошибке, важно сначала узнать об уровнях конфиденциальности.

Понимание уровней конфиденциальности

В Power Query каждому источнику данных можно назначить уровень конфиденциальности. Эти уровни важны для определения того, как могут быть объединены данные из разных источников.

Уровни конфиденциальности и их влияние на свертывание запросов:

- *Общий*. Данные из общего источника можно свободно комбинировать с другими источниками данных.
- *Организационный*. Этот уровень позволяет комбинировать данные только с другими организационными источниками.
- *Частный*. Самый строгий уровень. Частные данные не комбинируются с другими источниками, поддерживая строгую изоляцию.

Думайте об уровнях конфиденциальности следующим образом: если вы пометите источник данных как *частный*, он не будет смешиваться с другими данными в запросах. Это обеспечивает безопасность и разделение конфиденциальной информации. Данные, помеченные как *организационные*, могут быть смешаны только с другими данными с тем же уровнем конфиденциальности. *Общие* данные обеспечивают наибольшую гибкость, поскольку их можно легко объединить с любым другим источником данных.

Настройка уровней конфиденциальности

При попытке объединить источник данных без заданного уровня конфиденциальности Power Query предлагает выбрать один из них. Чтобы установить уровни конфиденциальности вручную в редакторе Power BI Desktop пройдите *Главная* → *Настройки источника данных*. Выберите источник данных и для изменения уровня конфиденциальности нажмите *Править разрешения*:

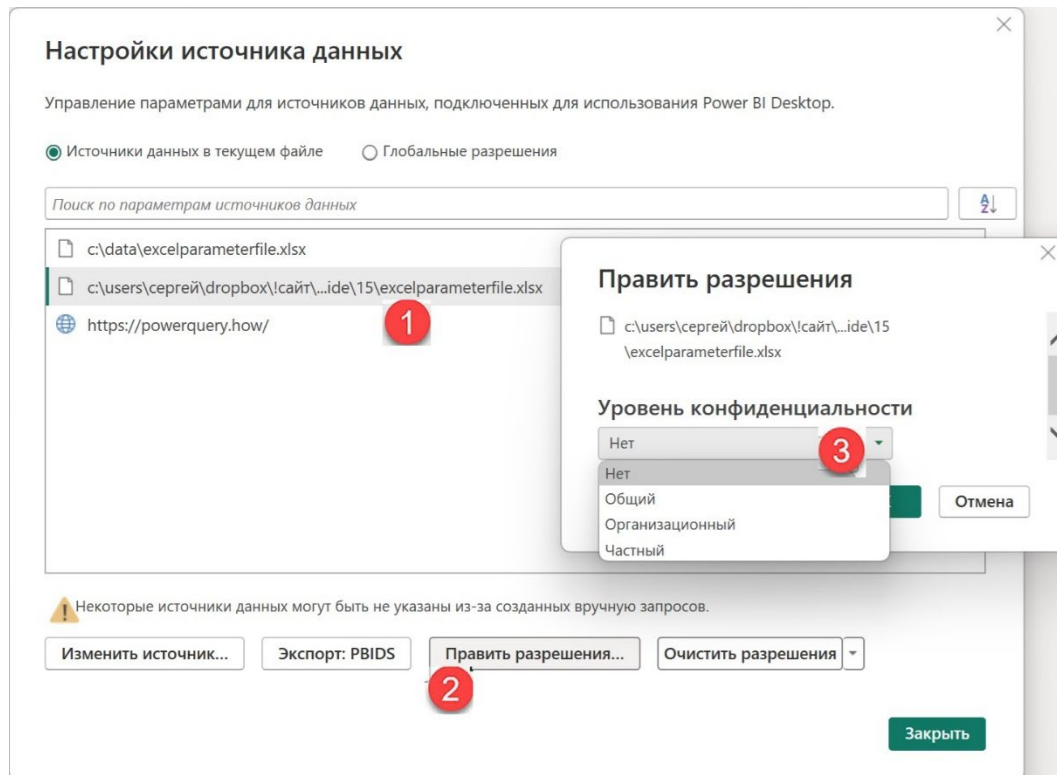


Рис. 15.39. Установка уровней конфиденциальности в настройках источника данных

Чтобы установить уровни конфиденциальности в службе Power BI откройте настройки с помощью значка шестеренки в правом верхнем углу, затем выберите *Управление подключениями и шлюзами*. Найдите источник данных и установите уровень конфиденциальности в нижней части экрана.

Какое отношение все это имеет к фаерволу формул?

Устранение ошибки фаервола

Вернемся к примеру, где мы получили URL-адрес из файла Excel. Это действие привело к запросу информации об уровне конфиденциальности для источников данных (см. рис. 15.37). Чтобы устранить ошибку, можно поступить двояко.

Способ 1: Установка совместимых уровней конфиденциальности

Получим ли мы ошибку фаервола формул, зависит от настроек уровня конфиденциальности web-запроса:

- *Общий*: запрос выполнится сразу, если только уровень конфиденциальности Excel-файла, к которому мы подключаемся, не установлен на *Частный* или *Организационный*. В последнем случае возникнет ошибка фаервола. Уровни конфиденциальности *Нет* и *Общий* совместимы.
- *Организационный*: будет проверен уровень конфиденциальности Excel-файла. Запрос выполнится без проблем, если файл, с которым мы комбинируем данные, также имеет *Организационный* уровень. Любой другой уровень конфиденциальности вызовет ошибку фаервола формул.
- *Частный*: такой выбор неизбежно приведет к ошибке фаервола.

Допустим, мы хотим установить для web-запроса уровень конфиденциальности *Организационный*. При нажатии *Продолжить* и назначении параметра *Организационный* возникает ошибка фаервола. Это происходит потому, что, хотя мы установили для веб-запроса

значение *Организационный*, уровень конфиденциальности для подключения к файлу Excel все еще не определен.

Чтобы решить эту проблему, перейдите *Главная* → *Настройки источника данных*. В списке *Источники данных в текущем файле* (см. переключатель в верхней части рис. 15.39) найдите файл Excel. Кликните *Править разрешения* и установите уровень конфиденциальности файла Excel *Организационный*. Подтвердите свой выбор, и вернитесь к своему запросу. Нажмите *Обновить предварительный просмотр*. После выполнения этих действий запросы должны работать без ошибок, связанных с уровнем конфиденциальности.

Способ 2: Игнорирование уровней конфиденциальности

Еще один способ предотвратить ошибку фаервола – игнорировать настройки уровня конфиденциальности. Роль фаервола в качестве привратника между источниками данных может замедлить выполнение запросов. Эта функция важна для предотвращения утечек данных, особенно в сценариях свертывания запросов. Однако у вас есть альтернатива, если вы имеете дело с неконфиденциальными данными.

В Power BI Desktop вы можете игнорировать уровни конфиденциальности. Это также может помочь повысить производительность запросов, так как любые проверки совместимости могут быть пропущены. В предыдущем примере, когда вас просят установить уровни конфиденциальности, если вы продолжите их настройку, появится всплывающее сообщение:

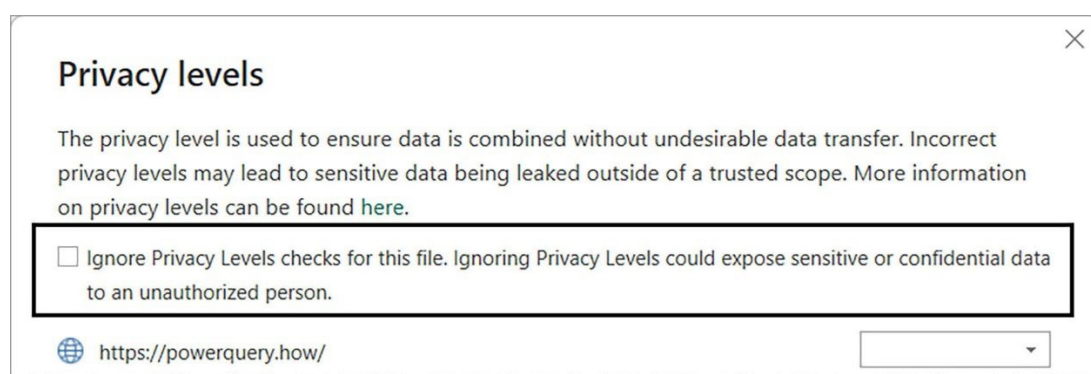


Рис. 15.41. Уведомление об уровнях конфиденциальности – опция игнорирования уровней конфиденциальности

Уровень конфиденциальности используется для обеспечения объединения данных без нежелательной передачи данных. Неправильные уровни конфиденциальности могут привести к утечке конфиденциальных данных за пределы доверенной области. Более подробную информацию об уровнях конфиденциальности можно найти [здесь](#).

Игнорировать проверки уровней конфиденциальности для этого файла. Игнорирование уровней конфиденциальности может привести к раскрытию чувствительных или конфиденциальных данных неуполномоченному лицу.

У вас есть возможность выбрать опцию *Игнорировать уровни конфиденциальности*. Если вы выберете этот вариант, ваш запрос будет выполнен без возникновения ошибок фаервола, связанных с конфиденциальностью.

Вы можете установить эту настройку в редакторе Power Query, пройдя *Файл* → *Параметры и настройки* → *Параметры*. В разделе *Конфиденциальность* выберите *Всегда игнорировать настройки уровня конфиденциальности*:

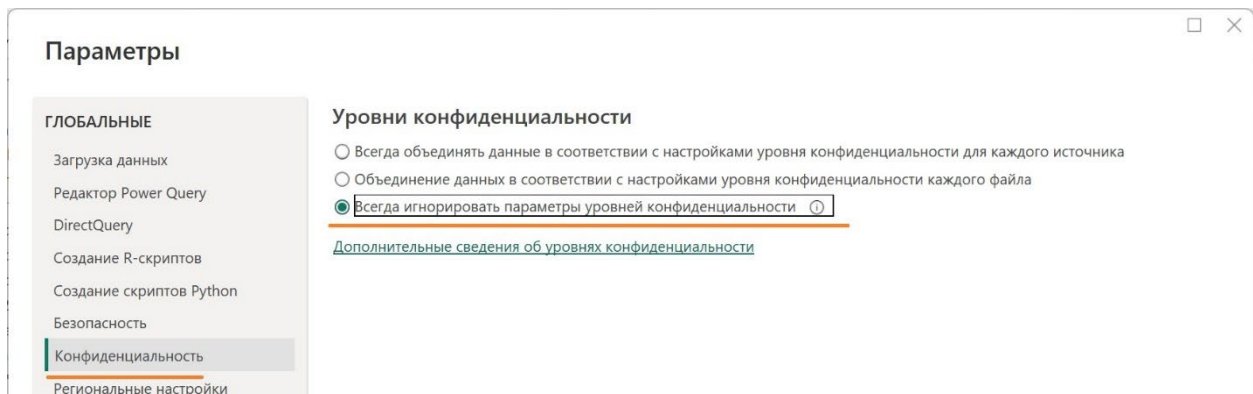


Рис. 15.42. Окно *Параметры* для настройки уровней конфиденциальности

Такая настройка параметра может привести к раскрытию конфиденциальных данных. Кроме того, этот параметр применим только в Power BI Desktop. Служба Power BI не учитывает этот параметр, поэтому вам потребуется настроить соответствующие уровни конфиденциальности.

В заключение хочу сказать, что в этой главе было продемонстрировано, как устранять различные ошибки фаервола формул с помощью нескольких стратегий. Мы решили проблему с привязкой к секциям, объединив логику в рамках одного запроса и создав пользовательскую функцию. Из-за ошибки конфиденциальности данных мы либо выравнивали уровни конфиденциальности, либо полностью игнорировали настройки конфиденциальности.

Однако важно отметить, что примеры, приведенные в этой главе, не поддерживают обновление в службе Power BI. Это ограничение возникает из-за того, что в примере используется URL-адрес в одном запросе и его использование в другом. Подсистема Power Query требует, чтобы базовый URL-адрес веб-запроса был явно виден подсистеме M. Наша динамическая ссылка на URL-адрес скрывает это, что приводит к ошибке динамических запросов.

Несмотря на это специфическое ограничение для веб-запроса, методы, описанные в этой главе, остаются эффективными для разрешения ситуаций, связанных с фаерволом формул. Например, если эти методы применяются для фильтрации запроса к базе данных, запрос будет успешно обновлен в службе Power BI.

К настоящему моменту вы узнали, как работать с фаерволом формул и что можно сделать, чтобы обеспечить свертывание запросов. В следующем разделе мы рассмотрим другие стратегии повышения производительности запросов.

Оптимизация производительности запросов

Power Query работает в ограниченной среде. В частности, каждый контейнер имеет ограниченный объем ресурсов. Это ограничение является важным фактором, который следует учитывать при создании запросов. Важно уменьшить объем данных в самом начале запроса. Так вы не превысите лимит ресурсов, то есть не вызовете разбиение на страницы. Последнее приведет к снижению производительности или сбоям.

Использование механизма свертывания запросов является одной из наиболее эффективных стратегий оптимизации производительности. Однако существуют сценарии, в которых источник данных не поддерживает свертывание запросов или необходимое преобразование прерывает процесс свертывания. В таких случаях стратегия смещается в сторону минимизации объема памяти, занимаемой запросом. Итак, какие методы позволят достигнуть этой цели?

Приоритет фильтрации строк и удалению столбцов

С первых шагов запроса сосредоточьтесь на уменьшении набора данных, чтобы сохранить только то, что необходимо:

- *Фильтрация строк*: уменьшая количество строк, вы снижаете объем данных, которые Power Query должен хранить в памяти, что ускоряет процесс и уменьшает использование памяти.
- *Удаление ненужных столбцов*: как и фильтрация строк, удаление ненужных столбцов также важно. Эта практика помогает сократить данные.

Обе операции важны с вычислительной точки зрения. Но есть еще один тип операций, которые следует выполнять в первую очередь.

Буферизация и потоковая передача

Операции в источнике данных часто выполняются быстрее, чем локально. А среди локальных можно встретить два типа операций: буферизация и потоковая передача. Что же это за операции и как они связаны с производительностью?

Операции буферизации

... требуют чтения всего набора данных для получения результата. Эти операции являются ресурсоемкими и требуют полного сканирования данных. Это означает, что объем используемой памяти пропорционален размеру входных данных.

Например, мы сортируем данные, а затем извлекаем первые 1000 строк. Для определения первых 1000 строк необходим весь набор данных. С другой стороны, если вы сначала выберете первые 1000 строк, а затем отсортируете их, процесс сортировки должен сохранить в памяти только эти 1000 строк. Он может сделать это, просто получив первые 1000 строк, не требуя полного набора данных. В этом сценарии сортировка считается операцией буферизации.

Операции буферизации также включают в себя некоторые типы соединений, группирование данных и изменение структуры таблицы, например сведение или транспонирование. Аналогичным образом, операции, возвращающие уникальные значения или добавляющие столбец ранжирования, требуют сканирования всей таблицы. Это те типы операций, которые используют много ресурсов и должны быть ограничены при оптимизации производительности.

Операции потоковой передачи

В отличие от операций буферизации, операции потоковой передачи не требуют сканирования всего набора данных для получения результата. Например фильтрация. Здесь список или таблица данных обрабатываются в потоковом режиме. По мере прохождения данных Power Query оценивает и возвращает результаты постепенно. Это означает, что для экрана предварительного просмотра в редакторе Power Query требуется только часть данных, чтобы заполнить первые 1000 строк. Это также напрямую приводит к более быстрому отклику редактора Power Query, поскольку он может быстрее отображать предварительный просмотр данных. К типичным операциям потоковой передачи относятся добавление столбцов, фильтрация строк и выбор столбцов. К объектам, поддерживающим потоковую передачу, относятся объекты табличного типа, списки и двоичные данные.

Максимальное использование преобразований потоковой передачи помогает сократить потребление памяти и избежать проблем с производительностью при выполнении запроса. Зная это, вы можете задаться вопросом, как определить, работаете ли вы с операцией потоковой передачи или буферизации. Ответ заключается в том, что вы можете определить это в новом интерфейсе Power Query с помощью плана запроса.

Использование плана запроса

План запроса – это функция в новом интерфейсе Power Query, предназначенная для предоставления дополнительных сведений об оценке запроса. Если вы хотите узнать подробности об оценке шага, щелкните правой кнопкой мыши на шаге и выберите *План запроса*.

Рассмотрим запрос, который подключается к базе данных SQL, выбирает несколько столбцов, добавляет столбец индекса, фильтрует первые 20 строк и сортирует таблицу:

```
1 let
2     Source = Sql.Database("localhost", "ContosoRetailDW"),
3     Navigation = Source[[Schema = "dbo", Item = "DimCustomer"]][Data],
4     RemoveOtherColumns = Table.SelectColumns(Navigation, {"EmailAddress", "BirthDate"}),
5     AddIndex = Table.AddIndexColumn(RemoveOtherColumns, "Index", 0, 1, Int64.Type),
6     FilterTop20 = Table.SelectRows(AddIndex, each [Index] <= 20),
7     SortRows = Table.Sort(FilterTop20, {"BirthDate", Order.Ascending})
8 in
9     SortRows
```

Рис. 15.43. Запрос к базе данных и последующие преобразования

Чтобы проанализировать план запроса для шага *SortRows*, щелкните шаг правой кнопкой мыши и выберите *Просмотреть план запроса*:

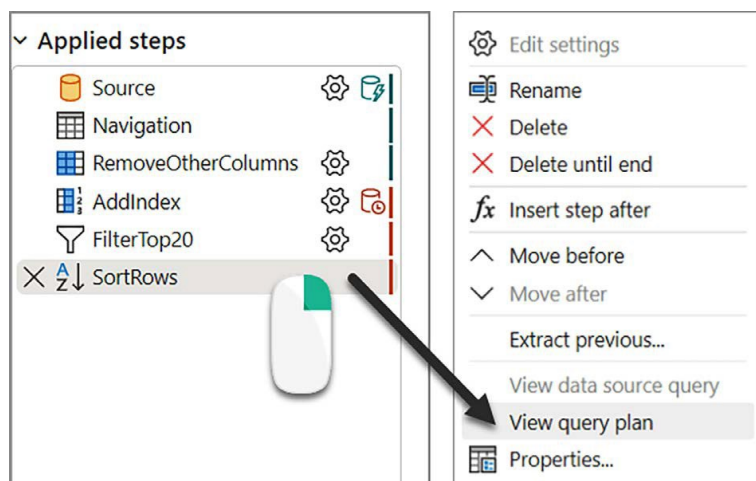


Рис. 15.44. Щелкните правой кнопкой мыши шаг, чтобы выбрать *Просмотреть план запроса*

Откроется окно с отображением плана запроса:

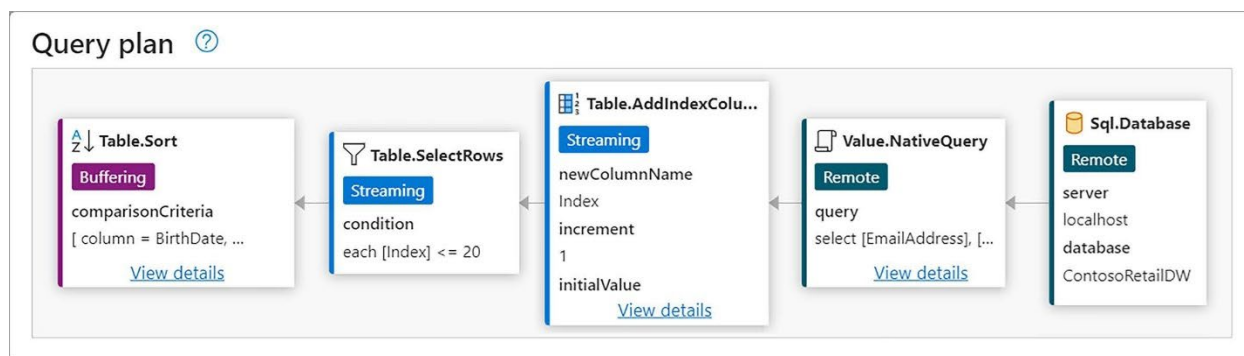


Рис. 15.45. План запроса указывает, как оценивается шаг

План запроса визуализирует путь оценки запроса. Он начинается справа, показывая подключение к базе данных SQL, и продвигается влево, ведя к последнему шагу сортировки таблицы.

В плане вы можете встретить три типа индикаторов для каждого шага:

- **Удаленный:** такие шаги выполняются в источнике данных. Когда шаг сворачивается, вы можете просмотреть исходный запрос, выбрав *Просмотреть детали*.
- **Потоковый:** эти шаги не требуют сканирования всех данных для получения результата. Они обрабатывают данные в процессе их потока, извлекая только те строки, которые необходимы для заполнения предварительного просмотра.
- **Буферизация:** такие шаги требуют полного сканирования данных из предыдущего шага.

План запроса показывает, является ли операция буферизацией или потоковой передачей, а также показывает, какие шаги свернуты, и какой код имеют запросы к источнику данных.

Совет по оптимизации: приоритизируйте сворачивание запросов, где это возможно. После того как сворачивание больше невозможно, используйте потоковые операции перед буферизацией. Такой подход обеспечивает быстрое заполнение предварительных данных, сокращая время ожидания при разработке. Используйте буферизацию только в случае необходимости для завершения запросов. Стратегическое последовательное использование этих операций может значительно сократить время загрузки предварительного просмотра при изменении запроса.

Еще одна стратегия повышения производительности запросов заключается в сохранении частей запроса в памяти перед применением дорогостоящей операции. Это можно сделать с помощью буферных функций, о которых мы поговорим далее.

Использование функций буфера

Буферные функции в языке M, определяемые суффиксом *.Buffer*, играют важную роль в повышении производительности операций Power Query. Они временно сохраняют данные в

памяти компьютера. Этот процесс, называемый буферизацией, и может ускорить выполнение запросов, но также имеет некоторые ограничения.

Функции буферизации наиболее полезны в ситуациях, когда Power Query может выполнять несколько запросов для получения одних и тех же данных из внешнего источника. Это может происходить при выполнении сложных операций, требующих многократного доступа к данным, например, при детализированных преобразованиях построчно. Такое также можно наблюдать при использовании таблицы замены для выполнения нескольких замен.

Поскольку повторный запрос одних и тех же данных может быть медленным, особенно из внешнего источника, можно рассмотреть возможность буферизации данных в памяти. Таким образом, данные извлекаются только один раз и могут быть повторно использованы из памяти. Еще один сценарий, в котором функция буфера может быть полезна, – это вычисление промежуточного итога. Производительность значительно повышается при буферизации значений для промежуточного итога.

С другой стороны, если вы буферизируете большой объем данных в памяти и превышаете лимит памяти контейнера, система начинает перемещать данные между оперативной памятью и хранилищем (пейджинг или подкачка). Это замедляет выполнение запросов из-за более низкой скорости доступа по сравнению с оперативной памятью. Чтобы запросы выполнялись быстро, важно учитывать объем памяти, который они занимают. В следующем разделе мы рассмотрим различные подходы к созданию накопительного итога и то, как функции буферизации помогают улучшить производительность запросов.

Влияние буферизации на скорость расчета промежуточных итогов

В этом разделе мы будем использовать пример для измерения производительности и потребления памяти при различных подходах. К сожалению, Power Query не имеет простого встроенного способа измерения производительности. Несмотря на то, что диагностика запросов доступна в Power BI Desktop, эта функция затрудняет интерпретацию метрик.

Рекомендую надстройку [Excel](#) от Михаила Музыкина, предназначенную для оценки скорости выполнения запросов PQ и формул на листах.

Для более точного измерения скорости запросов мы использовали инструмент под названием SQL-profiler и подключили его к нашему набору данных Power BI. Мы рассчитали время, обновив каждый запрос три раза и взяв среднее. Давайте посмотрим, как выглядит наша настройка.

Настройка

Рассмотрим набор данных, содержащий транзакции по парковке, в том числе сумму платежа:

	Transaction ID	Meter Code	Transaction DateTime	Amount Paid
1	1250162207	12028002	11.07.2023 7:30:00	26
2	1250162278	19232002	11.07.2023 7:31:00	152
3	1250131414	19127010	11.07.2023 4:20:00	967
4	1250134465	5073002	11.07.2023 4:48:00	4
5	1250134860	19161010	11.07.2023 4:52:00	673
6	1250134876	5019002	11.07.2023 9:51:00	3
7	1250134941	19126010	11.07.2023 4:52:00	877

Рис. 15.46. Часть набора данных о транзакциях парковки

Вы можете следовать примеру, используя файл *Paid_Parking_Transaction_DataSmall.csv*.

Наша цель – рассчитать сумму платежей накопительным итогом. Для вычислений мы используем две функции. *List.FirstN* выбирает значения до определенной строки, а *List.Sum* складывает эти значения, чтобы получить накопительный итог. А начнем мы с добавления индексного столбца:

```
let
Source = Csv.Document(
    File.Contents(
        "C:\...\Guide\15\Paid_Parking_Transaction_DataSmall.csv"
    ),
    [Delimiter = ";", Encoding = 1252]
```

```

),
PromotedHeaders = Table.PromoteHeaders(Source, [PromoteAllScalars = true]),
ChangedType = Table.TransformColumnTypes(
    PromotedHeaders,
    {{"Amount Paid", Int64.Type}, {"Transaction DateTime", type datetime}}
),
AddedIndex = Table.AddIndexColumn(ChangedType, "Index", 1, 1, Int64.Type)
in
    AddedIndex

```

Далее мы рассмотрим четыре метода вычисления промежуточных итогов:

- Из обычной таблицы
- Из буферизованной таблицы
- Из буферизованного столбца
- Из буферизованного столбца с помощью *List.Generate*

Способ 1: Промежуточный итог из обычной таблицы

Для первого способа мы добавим новый столбец:

```
List.Sum( List.FirstN( AddedIndex[Amount Paid], [Index] ))
```

Эта формула вычисляет промежуточный итог для каждой транзакции:

Transaction ID	Meter Code	Transaction DateTime	Amount Paid	Index	Running Total
1	12028002	11.07.2023 7:30:00	26	1	26
2	19232002	11.07.2023 7:31:00	152	2	178
3	19127010	11.07.2023 4:20:00	967	3	1145
4	5073002	11.07.2023 4:48:00	4	4	1149
5	19161010	11.07.2023 4:52:00	673	5	1822
6	5019002	11.07.2023 9:51:00	3	6	1825
7	19126010	11.07.2023 4:52:00	877	7	2702
8	19129010	11.07.2023 4:53:00	877	8	3579
9	1143002	11.07.2023 5:01:00	1	9	3580
10	1012002	11.07.2023 5:27:00	3	10	3583

Рис. 15.47. Добавление промежуточного итога без буферизации значений

В Power BI время обработки в среднем занимает 128 секунд. Метод извлекает 1,88 ГБ данных. Учитывая, что исходный файл имеет размер всего 393 КБ, значительный объем обработанных данных позволяет предположить, что Power Query считывает файл множество раз. Эта неэффективность является существенным недостатком использования обычной таблицы для расчетов промежуточных итогов.

Способ 2: Промежуточный итог из буферизованной таблицы

Вместо обычной таблицы этот метод буферизирует таблицу в памяти:

```

let
    Source = myCSV,
    AddedIndex = Table.AddIndexColumn(Source, "Index", 1, 1, Int64.Type),
    BufferTable = Table.Buffer(AddedIndex),
    AddRunningTotal = Table.AddColumn(
        BufferTable,
        "Custom",
        each List.Sum(List.FirstN(BufferTable[Amount Paid], [Index]))
    )
in
    AddRunningTotal

```

На шаге *BufferTable* данные загружаются в память. Теперь не нужны множественные запросы к исходному файлу. Данные хранятся в памяти. Последующие операции с таблицей могут

использовать данные в памяти. Для этого метода среднее время обработки 61 секунды. Получено только 394 КБ данных. Теперь общий объем полученных данных соответствует размеру исходного файла CSV. Это также показывает, насколько эффективна функция *Table.Buffer* для оптимизации производительности.

Способ 3: Промежуточный итог из буферизованного столбца

Ранее в этой главе мы подчеркивали важность минимизации использования памяти движка. Следуя этому принципу, метод 3 буферизует только один столбец, используемый при вычислении промежуточного итога. Такой подход уменьшает объем памяти и, как мы надеемся, сокращает время обновления запроса.

```
let
  Source = myCSV,
  AddedIndex = Table.AddIndexColumn(Source, "Index", 1, 1, Int64.Type),
  BufferColumn = List.Buffer(AddedIndex[Amount Paid]),
  AddRunningTotal = Table.AddColumn(
    AddedIndex,
    "Running Total",
    each List.Sum(List.FirstN(BufferColumn, [Index]))
  )
in
  AddRunningTotal
```

В этом запросе шаг *BufferColumn* буферизирует только столбец *Amount Paid*. Среднее время обработки 1,2 секунды. Объем извлекаемых данных 786 КБ. Метод обеспечивает ошеломляющее сокращение времени запроса! Метод обращается к файлу CSV дважды – скорее всего, один раз для исходной таблицы и один раз для буферизации столбца, используемого при вычислении промежуточного итога. Это показывает, что объем полученных данных не всегда является показателем скорости выполнения запроса. Этот сценарий указывает на стратегический баланс между эффективностью загрузки памяти и скоростью обработки.

Означает ли это, что улучшить больше нечего? Не совсем так. Метод *List.FirstN* для создания промежуточного итога не является самым эффективным. Чтобы еще больше улучшить ситуацию, мы можем использовать функцию *List.Generate*.

Способ 4: Промежуточный итог из буферизованного столбца с помощью *List.Generate*

Метод *List.FirstN*, хотя и является функциональным, не самый эффективный для суммирования значений при вычислении промежуточного итога. Для этого необходимо получить и добавить диапазон значений для каждой строки. Альтернативным и более эффективным методом является использование функции *List.Generate*. Этот метод вычисляет промежуточный итог путем добавления каждого нового значения к ранее вычисленному итогу, а не пересчитывает сумму по постоянно увеличивающемуся диапазону значений каждый раз. Более подробное обсуждение метода *List.Generate* см. [в главе 14](#).

```
let
  Source = myCSV,
  BufferedColumn = List.Buffer(Source[Amount Paid]),
  RunningTotal = List.Generate(
    () => [RT = BufferedColumn{0}, RowIndex = 0],
    each [RowIndex] < List.Count(BufferedColumn),
    each [RT = List.Sum({[RT], BufferedColumn{[RowIndex] + 1}}), RowIndex = [RowIndex] + 1],
    each [RT]
  ),
  CombineTables = Table.FromColumns(
    Table.ToColumns(Source) & {RunningTotal},
    Table.ColumnNames(Source) & {"Running Total"}
  )
in
  CombineTables
```

Этот метод по-прежнему буферизует только один столбец. Затем функция *List.Generate* эффективно вычисляет промежуточный итог. Наконец, исходные столбцы таблицы объединяются с вновь созданным столбцом промежуточных итогов. Среднее время обработки 201 миллисекунды. Объем извлекаемых данных: 786 КБ.

Производительность зависит не только от структуры запроса. Важным фактором является скорость работы источников данных, о которой мы расскажем далее.

Рекомендации по источникам данных

При работе с медленными запросами важно изучить на что ваши запросы тратят свое время. Даже если вам удастся сохранить небольшой объем памяти, занимаемый запросами, если ваш источник данных работает медленно, вы будете ждать поступления данных. Другими словами, выбор правильного источника данных может значительно повлиять на производительность запросов.

Источники данных и скорость

Хотя не всегда возможно перейти на другой источник данных, полезно знать некоторые общие правила относительно источников данных и их производительности. Вот несколько рекомендаций, о которых стоит подумать:

- *Реляционные базы данных*, как правило, работают быстрее, чем файлы. Дополнительным преимуществом является то, что многие коннекторы баз данных поддерживают сворачивание запросов и позволяют загружать данные поэтапно.
- *Файлы*: храните данные в формате CSV или текстовых файлах. Они работают гораздо быстрее, чем файлы JSON, XML и Excel.
- *Сетевое расположение*: получение файлов из сетевого расположения, такого как OneDrive или SharePoint, обычно медленнее, чем работа с локальными файлами. Во время разработки имеет смысл рассмотреть возможность хранения файлов локально.
- *Интернет*: подключение к веб-службам или API может быть медленным. Аналогично, импорт больших объемов данных через интернет может занять много времени.

Природа источников данных замедляет выполнение запросов, особенно при работе с устаревшими источниками данных или данными из сетевых расположений. Если вы окажетесь в такой ситуации, рассмотрите возможность использования потоков данных, о которых мы расскажем далее.

Использование потоков данных

Некоторые источники данных могут быть сложными для оптимизации. Они могут не поддерживать сворачивание запросов, работать медленно или испытывать нехватку ресурсов для обработки больших запросов. У вас может не быть инструментов для ускорения их работы. Если вы работаете с данными из медленного источника, рассмотрите возможность использования потока данных, также известного как аналитический поток данных.

Поток данных подключается к источнику данных с помощью Power Query. Вы можете настроить его для регулярного сбора данных из медленного источника. После сбора данных поток данных сохраняет их в Azure Data Lake Storage Power BI. Здесь данные хранятся в формате, известном как папки CDM (Common Data Model). Эти папки содержат CSV-файлы для каждой части данных и файл метаданных JSON, описывающий содержимое и структуру данных.

Поток данных можно использовать для хранения данных. Вместо того чтобы подключаться напрямую к медленному источнику, запросы могут подключаться к потоку данных, что значительно ускорит обработку запросов.

Для компаний с Power Premium улучшенный вычислительный движок является отличной функцией для повышения эффективности потоков данных. Этот движок ускоряет время отклика потоков данных на ваши запросы. Поток данных способен использовать сворачивание запросов. Запросы, отправленные для получения данных из потока данных, выполняются против кэша в SQL, а не в папке CDM для потоков данных, для которых эта функция отключена.

Другим преимуществом потоков данных является их способность хранить выходные данные ваших запросов. Когда у вас много запросов в редакторе запросов, это может замедлить работу, скорее всего, из-за дополнительных затрат на загрузку запросов в фоновом режиме. Чтобы ускорить пользовательский интерфейс, полезно сократить количество запросов. Хороший способ

сделать это – сохранить часть логики ваших запросов в потоке данных. Затем вы можете создать один запрос, который подключается к этому потоку данных, чтобы получить результаты ваших оригинальных запросов. Этот подход не только ускоряет пользовательский интерфейс, но и позволяет вашим запросам обновляться быстрее, так как поток данных просто возвращает выходные данные ваших запросов вместо их вычисления во время обновления.

Чтобы узнать больше о потоках данных, ознакомьтесь с этой [статьей](#) ([русскоязычный вариант](#)).

Советы по повышению производительности

Оптимизация производительности – сложная тема, даже после прочтения этой главы. Вам придется протестировать различные подходы. Тем не менее, у нас есть несколько общих советов по производительности.

Подключите запрос к быстрому источнику данных. Если исходная система работает медленно, рассмотрите возможность перемещения данных в более быстрый источник или сохранения данных в потоке данных.

Увеличьте количество шагов, которые могут быть свернуты. Если вы хорошо разбираетесь в SQL, рассмотрите возможность создания эффективного SQL-запроса и примените параметр *EnableFolding*. Если SQL не является вашей сильной стороной, отдайте приоритет размещению сворачиваемых шагов в начале запроса. Такой подход гарантирует, что большая часть преобразований будет обработана в источнике данных.

Если шаги запроса больше не сворачиваются, следующей задачей должно быть сокращение использования памяти. Разместите фильтрацию строк и удаление столбцов как можно раньше. Максимально уменьшите размер запроса перед выполнением ресурсоемких операций буферизации, таких как группировка, сведение или объединение. На последующих шагах их логика должна быть применена к меньшему набору данных.

Используйте функции буфера для шагов, которые постоянно запрашивают данные. При этом убедитесь, что буферизуются только те данные, которые необходимы. Например, только один столбец.

У нас также есть несколько рекомендаций, которые улучшают скорость запросов при разработке в редакторе запросов:

- Рассмотрите возможность работы с подмножеством данных, чтобы ускорить разработку. Фильтруйте строки как можно раньше насколько это возможно, чтобы оставшийся запрос имел меньше потребностей в обработке.
- Попробуйте использовать операции потоковой передачи как можно чаще в начале запроса. Это обеспечивает быстрое заполнение экрана предварительного просмотра и улучшает процесс разработки. После ввода операций буферизации запрос должен будет оценить все строки в наборе данных.
- По мере увеличения количества преобразований и запросов Power Query имеет тенденцию к замедлению.
- Рассмотрите возможность хранения (части) логики в потоке данных, чтобы уменьшить объем ресурсов, используемых при разработке.

Сводка

В этой главе мы узнали о нескольких подходах к оптимизации производительности в Power Query. Мы изучили свертывание запросов и стратегии навигации по фаерволу формул. Разницу между операциями буферизации и потоковой передачи, а также использование функций буфера и важность сокращения использования памяти в запросах.

Ключевым моментом этой главы является понимание того, что управление аспектами памяти помогает повысить производительность запросов. Благодаря интеграции этих концепций и подходов вы теперь обладаете всесторонним пониманием различных элементов, которые способствуют оптимизации производительности операций Power Query.

В следующей главе мы изучим расширения. Вы узнаете, как создавать пользовательские коннекторы и какие инструменты используются в этом процессе.