5.11. Секционирование таблиц

PostgreSQL поддерживает простое секционирование таблиц. В этом разделе описывается, как и почему бывает полезно применять секционирование при проектировании баз данных.

5.11.1. Обзор

Секционированием данных называется разбиение одной большой логической таблицы на несколько меньших физических секций. Секционирование может принести следующую пользу:

- В определённых ситуациях оно кардинально увеличивает быстродействие, особенно когда большой процент часто запрашиваемых строк таблицы относится к одной или лишь нескольким секциям. Секционирование по сути

заменяет верхние уровни деревьев индексов, что увеличивает вероятность нахождения наиболее востребованных частей индексов в памяти.

- Когда в выборке или изменении данных задействована большая часть одной секции, производительность может возрасти, если будет выполняться последовательное сканирование этой секции, а не поиск по индексу, сопровождаемый произвольным чтением данных, разбросанных по всей таблице.

- Массовую загрузку и удаление данных можно осуществлять, добавляя и удаляя секции, если такой вариант использования был предусмотрен при проектировании секций. Удаление отдельной секции командой `DROP TABLE` и действие `ALTER TABLE DETACH PARTITION` выполняются гораздо быстрее, чем аналогичная массовая операция. Эти команды полностью исключают накладные расходы, связанные с выполнением `VACUUM` после массовой операции `DELETE`.

- Редко используемые данные можно перенести на более дешёвые и медленные носители.

Всё это обычно полезно только для очень больших таблиц. Какие именно таблицы выиграют от секционирования, зависит от конкретного приложения, хотя, как правило, это следует применять для таблиц, размер которых превышает объём ОЗУ сервера.

PostgreSQL предлагает поддержку следующих видов секционирования:

`Секционирование по диапазонам`

Таблица секционируется по «диапазонам», определённым по ключевому столбцу или набору столбцов, и не пересекающимся друг с другом. Например, можно секционировать данные по диапазонам дат или по диапазонам идентификаторов определённых бизнес-объектов. Границы каждого диапазона считаются включающими нижнее значение и исключающими верхнее. Например, если для первой секции задан диапазон значений от `1` до `10`, а для второй — от `10` до `20`, значение `10` относится ко второй секции, а не к первой.

`Секционирование по списку`

Таблица секционируется с помощью списка, явно указывающего, какие значения ключа должны относиться к каждой секции.

`Секционирование по хешу`

Таблица секционируется по определённым модулям и остаткам, которые указываются для каждой секции. Каждая секция содержит строки, для которых хеш-значение ключа разбиения, делённое на модуль, равняется заданному остатку.

Если вашему приложению требуются другие формы секционирования, можно также прибегнуть к альтернативным реализациям, с использованием наследования и представлений с `UNION ALL`. Такие подходы дают гибкость, но не дают такого выигрыша в производительности, как встроенное декларативное секционирование.

5.11.2. Декларативное секционирование

PostgreSQL позволяет декларировать, что некоторая таблица разделяется на секции. Разделённая на секции таблица называется *секционированной таблицей*. Декларация секционирования состоит из описанного выше определения *метода разбиения* и списка столбцов или выражений, образующих *ключ разбиения*.

Сама секционированная таблица является «виртуальной» и как таковая не хранится. Хранилище используется её *секциями*, которые являются обычными таблицами, связанными с секционированной. В каждой секции хранится подмножество данных таблицы, определяемое её *границами секции*. Все строки, вставляемые в секционированную таблицу, перенаправляются в соответствующие секции в зависимости от значений столбцов ключа разбиения. Если при изменении значений ключа разбиения в строке она перестаёт удовлетворять ограничениям исходной секции, эта строка перемещается в другую секцию.

Сами секции могут представлять собой секционированные таблицы, таким образом реализуется *вложенное секционирование*. Хотя все секции должны иметь те же столбцы,

что и секционированная родительская таблица, в каждой секции независимо от других могут быть определены свои индексы, ограничения и значения по умолчанию. Подробнее о создании секционированных таблиц и секций рассказывается в описании CREATE TABLE.

Преобразовать обычную таблицу в секционированную и наоборот нельзя. Однако в секционированную таблицу можно добавить в качестве секции существующую обычную или секционированную таблицу, а также можно удалить секцию из секционированной таблицы и превратить её в отдельную таблицу; это может ускорить многие процессы обслуживания. Обратитесь к описанию ALTER TABLE, чтобы узнать больше о подкомандах `ATTACH PARTITION` и `DETACH PARTITION`.

Секции также могут быть сторонними таблицами, но учтите, что именно пользователь ответственен за соответствие содержимого сторонней таблицы правилу секционирования, это не контролируется автоматически. Также существуют и некоторые другие ограничения. За подробностями обратитесь к описанию CREATE FOREIGN TABLE.

5.11.2.1. Пример

Предположим, что мы создаём базу данных для большой компании, торгующей мороженым. Компания учитывает максимальную температуру и продажи мороженого каждый день в разрезе регионов. По сути нам нужна следующая таблица:

```
CREATE TABLE measurement (
    city_id         int not null,
    logdate         date not null,
    peaktemp        int,
    unitsales       int
);
```

Мы знаем, что большинство запросов будут работать только с данными за последнюю неделю, месяц или квартал, так как в основном эта таблица нужна для формирования текущих отчётов для руководства. Чтобы сократить объём хранящихся старых данных, мы решили оставлять данные только за 3 последних года. Ненужные данные мы будем

удалять в начале каждого месяца. В этой ситуации мы можем использовать секционирование для удовлетворения всех наших требований к таблице показателей.

Чтобы использовать декларативное секционирование в этом случае, выполните следующее:

1. Создайте таблицу `measurement` как секционированную таблицу с предложением `PARTITION BY`, указав метод разбиения (в нашем случае `RANGE`) и список столбцов, которые будут образовывать ключ разбиения.

   ```
   CREATE TABLE measurement (
       city_id         int not null,
       logdate         date not null,
       peaktemp        int,
       unitsales       int
   ) PARTITION BY RANGE (logdate);
   ```

2. Создайте секции. В определении каждой секции должны задаваться границы, соответствующие методу и ключу разбиения родительской таблицы. Заметьте, что указание границ, при котором множество значений новой секции пересекается со множеством значений в одной или нескольких существующих секциях, будет ошибочным.

   Секции, создаваемые таким образом, во всех отношениях являются обычными таблицами PostgreSQL (или, возможно, сторонними таблицами). В частности, для каждой секции можно независимо задать табличное пространство и параметры хранения.

   В нашем примере каждая секция должна содержать данные за один месяц, чтобы данные можно было удалять по месяцам согласно требованиям. Таким образом, нужные команды будут выглядеть так:

```
CREATE TABLE measurement_y2006m02 PARTITION OF measurement
    FOR VALUES FROM ('2006-02-01') TO ('2006-03-01');

CREATE TABLE measurement_y2006m03 PARTITION OF measurement
    FOR VALUES FROM ('2006-03-01') TO ('2006-04-01');

...
CREATE TABLE measurement_y2007m11 PARTITION OF measurement
    FOR VALUES FROM ('2007-11-01') TO ('2007-12-01');

CREATE TABLE measurement_y2007m12 PARTITION OF measurement
    FOR VALUES FROM ('2007-12-01') TO ('2008-01-01')
    TABLESPACE fasttablespace;

CREATE TABLE measurement_y2008m01 PARTITION OF measurement
    FOR VALUES FROM ('2008-01-01') TO ('2008-02-01')
    WITH (parallel_workers = 4)
    TABLESPACE fasttablespace;
```

(Как говорилось ранее, границы соседних секций могут определяться одинаковыми значениями, так как верхние границы не включаются в диапазон.)

Если вы хотите реализовать вложенное секционирование, дополнительно укажите предложение `PARTITION BY` в командах, создающих отдельные секции, например:

```
CREATE TABLE measurement_y2006m02 PARTITION OF measurement
    FOR VALUES FROM ('2006-02-01') TO ('2006-03-01')
    PARTITION BY RANGE (peaktemp);
```

Когда будут созданы секции `measurement_y2006m02`, данные, добавляемые в `measurement` и попадающие в `measurement_y2006m02` (или данные, которые могут добавляться непосредственно в `measurement_y2006m02` при условии соблюдения

ограничения данной секции) будут затем перенаправлены в одну из вложенных секций в зависимости от значения столбца `peaktemp`. Указанный ключ разбиения может пересекаться с ключом разбиения родителя, хотя определять границы вложенной секции нужно осмотрительно, чтобы множество данных, которое она принимает, входило во множество, допускаемое собственными границами секции; система не пытается контролировать это сама.

При добавлении в родительскую таблицу данных, которые не соответствуют ни одной из существующих секций, произойдёт ошибка; подходящую секцию нужно создавать вручную.

Создавать в таблицах ограничения с условиями, задающими границы секций, вручную не требуется. Такие ограничения будут созданы автоматически.

3. Создайте в секционируемой таблице индекс по ключевому столбцу (или столбцам), а также любые другие индексы, которые могут понадобиться. (Индекс по ключу, строго говоря, создавать не обязательно, но в большинстве случаев он будет полезен.) При этом автоматически будет создан соответствующий индекс в каждой секции и все секции, которые вы будете создавать или присоединять позднее, тоже будут содержать такой индекс. Индексы или ограничения уникальности, созданные в секционированной таблице, являются «виртуальными», как и сама секционированная таблица: фактически данные находятся в дочерних индексах отдельных таблиц-секций.

   ```
   CREATE INDEX ON measurement (logdate);
   ```

4. Убедитесь в том, что параметр конфигурации enable_partition_pruning не выключен в `postgresql.conf`. Иначе запросы не будут оптимизироваться должным образом.

В данном примере нам потребуется создавать секцию каждый месяц, так что было бы разумно написать скрипт, который бы формировал требуемый код DDL автоматически.

5.11.2.2. Обслуживание секций

Обычно набор секций, образованный изначально при создании таблиц, не предполагается сохранять неизменным. Чаще наоборот, планируется удалять секции со старыми данными и периодически добавлять секции с новыми. Одно из наиболее важных преимуществ секционирования состоит именно в том, что оно позволяет практически моментально выполнять трудоёмкие операции, изменяя структуру секций, а не физически перемещая большие объёмы данных.

Самый лёгкий способ удалить старые данные — просто удалить секцию, ставшую ненужной:

```
DROP TABLE measurement_y2006m02;
```

Так можно удалить миллионы записей гораздо быстрее, чем удалять их по одной. Заметьте, однако, что приведённая выше команда требует установления блокировки `ACCESS EXCLUSIVE`.

Ещё один часто более предпочтительный вариант — убрать секцию из главной таблицы, но сохранить возможность обращаться к ней как к самостоятельной таблице:

```
ALTER TABLE measurement DETACH PARTITION measurement_y2006m02;
ALTER TABLE measurement DETACH PARTITION measurement_y2006m02 CONCURRENTLY;
```

При этом можно будет продолжать работать с данными, пока таблица не будет удалена. Например, в этом состоянии очень кстати будет сделать резервную копию данных, используя `COPY`, pg_dump или подобные средства. Возможно, эти данные также можно будет агрегировать, перевести в компактный формат, выполнить другую обработку или построить отчёты. Первая форма команды требует блокировки `ACCESS EXCLUSIVE` родительской таблицы. Благодаря указанию `CONCURRENTLY`, как во второй форме, для операции отсоединения будет требоваться только блокировка `SHARE UPDATE EXCLUSIVE` родительской таблицы, но при этом действуют ограничения, описанные в `ALTER TABLE ... DETACH PARTITION`.

Аналогичным образом можно добавлять новую секцию с данными. Мы можем создать пустую секцию в главной таблице так же, как мы создавали секции в исходном состоянии до этого:

```
CREATE TABLE measurement_y2008m02 PARTITION OF measurement
    FOR VALUES FROM ('2008-02-01') TO ('2008-03-01')
    TABLESPACE fasttablespace;
```

А иногда удобнее создать новую таблицу вне структуры секций и сделать её полноценной секцией позже. При таком подходе новые данные можно будет загрузить, проверить и преобразовать до того, как они появятся в секционированной таблице. Обойтись без кропотливого воспроизведения определения родительской таблицы можно, воспользовавшись функциональностью `CREATE TABLE ... LIKE`:

```
CREATE TABLE measurement_y2008m02
  (LIKE measurement INCLUDING DEFAULTS INCLUDING CONSTRAINTS)
  TABLESPACE fasttablespace;

ALTER TABLE measurement_y2008m02 ADD CONSTRAINT y2008m02
   CHECK ( logdate >= DATE '2008-02-01' AND logdate < DATE '2008-03-01' );

\copy measurement_y2008m02 from 'measurement_y2008m02'
-- possibly some other data preparation work

ALTER TABLE measurement ATTACH PARTITION measurement_y2008m02
    FOR VALUES FROM ('2008-02-01') TO ('2008-03-01' );
```

Команда `ATTACH PARTITION` требует блокировки `SHARE UPDATE EXCLUSIVE` для секционированной таблицы.

Прежде чем выполнять команду `ATTACH PARTITION`, рекомендуется создать ограничение `CHECK` в присоединяемой таблице, соответствующее ожидаемому ограничению секции, как показано выше. Благодаря этому система сможет обойтись без сканирования, необходимого для проверки неявного ограничения секции. Без этого ограничения `CHECK`

нужно будет просканировать и убедиться в выполнении ограничения секции, удерживая блокировку ACCESS EXCLUSIVE в этой секции. После выполнения команды ATTACH PARTITION рекомендуется удалить ограничение CHECK, поскольку оно больше не нужно. Если присоединяемая таблица также является секционированной таблицей, то каждая из её секций будет рекурсивно блокироваться и сканироваться до тех пор, пока не встретится подходящее ограничение CHECK или не будут достигнуты конечные разделы.

Точно так же, если в секционированной таблице есть секция DEFAULT, рекомендуется создать ограничение CHECK, которое исключает ограничение раздела, подлежащего присоединению. Если этого не сделать, то раздел DEFAULT будет просканирован, чтобы убедиться, что он не содержит записей, которые должны быть расположены в подключаемом разделе. Эта операция будет выполняться при удержании блокировки ACCESS EXCLUSIVE на разделе DEFAULT. Если раздел DEFAULT сам является секционированной таблицей, то каждая из её секций будет рекурсивно проверяться таким же образом, как и присоединяемая таблица, как упоминалось выше.

Как говорилось выше, в секционированных таблицах можно создавать индексы так, чтобы они применялись автоматически ко всей иерархии. Это очень удобно, так как индексироваться будут не только все существующие секции, но и любые секции, создаваемые в будущем. Но есть одно ограничение — такой секционированный индекс нельзя создать в неблокирующем режиме (с указанием CONCURRENTLY). Чтобы избежать блокировки на долгое время, для создания индекса в самой секционированной таблице можно использовать команду CREATE INDEX ON ONLY; такой индекс будет помечен как нерабочий, и он не будет автоматически применён к секциям. Индексы собственно в секциях можно создать в индивидуальном порядке с указанием CONCURRENTLY, а затем *присоединить* их к индексу родителя, используя команду ALTER INDEX .. ATTACH PARTITION. После того как индексы всех секций будут присоединены к родительскому, последний автоматически перейдёт в рабочее состояние. Например:

```
CREATE INDEX measurement_usls_idx ON ONLY measurement (unitsales);

CREATE INDEX measurement_usls_200602_idx
    ON measurement_y2006m02 (unitsales);
```

```
ALTER INDEX measurement_usls_idx
    ATTACH PARTITION measurement_usls_200602_idx;
...
```

Этот приём можно применять и с ограничениями `UNIQUE` и `PRIMARY KEY`; для них индексы создаются неявно при создании ограничения. Например:

```
ALTER TABLE ONLY measurement ADD UNIQUE (city_id, logdate);

ALTER TABLE measurement_y2006m02 ADD UNIQUE (city_id, logdate);
ALTER INDEX measurement_city_id_logdate_key
    ATTACH PARTITION measurement_y2006m02_city_id_logdate_key;
...
```

5.11.2.3. Ограничения

С секционированными таблицами связаны следующие ограничения:

- Ограничения уникальности (а значит и первичные ключи) в секционированных таблицах должны включать все столбцы ключа разбиения. Это требование объясняется тем, что отдельные индексы, образующие ограничение, могут непосредственно обеспечивать уникальность только в своих секциях. Поэтому сама структура секционирования должна гарантировать отсутствие дубликатов в разных секциях.

- Создать ограничение-исключение, охватывающее всю секционированную таблицу, нельзя; можно только поместить такое ограничение в каждую отдельную секцию с данными. И это также является следствием того, что установить ограничения, действующие между секциями, невозможно.

- Триггеры `BEFORE ROW` для `INSERT` не могут менять секцию, в которую в итоге попадёт новая строка.

- Смешивание временных и постоянных отношений в одном дереве секционирования не допускается. Таким образом, если секционированная

таблица постоянная, такими же должны быть её секции; с временными таблицами аналогично. В случае с временными отношениями все таблицы дерева секционирования должны быть из одного сеанса.

На уровне реализации отдельные секции связываются с секционированной таблицей средствами наследования. Однако с декларативно секционированными таблицами или их секциями нельзя использовать некоторую общую функциональность наследования, как описано ниже. А именно, секция не может иметь никаких других родителей, кроме секционированной таблицы, к которой она относится, равно как и любая таблица не может наследоваться и от секционированной, и от обычной таблицы. Это означает, что секционированные таблицы и их секции не совмещаются в иерархии наследования с обычными таблицами.

Так как иерархия секционирования, образованная секционированной таблицей и её секциями, является одновременно и иерархией наследования, она содержит `tableoid` и на неё распространяются все обычные правила наследования, описанные в Разделе 5.10, с некоторыми исключениями:

- В секциях не может быть столбцов, отсутствующих в родительской таблице. Такие столбцы невозможно определить ни при создании секций командой `CREATE TABLE`, ни путём последующего добавления в секции командой `ALTER TABLE`. Таблицы могут быть подключены в качестве секций командой `ALTER TABLE ... ATTACH PARTITION`, только если их столбцы в точности соответствуют родительской таблице.

- Ограничения `CHECK` вместе с `NOT NULL`, определённые в секционированной таблице, всегда наследуются всеми её секциями. Ограничения `CHECK` с характеристикой `NO INHERIT` в секционированных таблицах создавать нельзя. Также нельзя удалить ограничение `NOT NULL`, заданное для столбца секции, если такое же ограничение существует в родительской таблице.

- Использование указания `ONLY` при добавлении или удалении ограничения только в секционированной таблице поддерживается лишь когда в ней нет секций. Если секции существуют, при попытке использования `ONLY` возникнет

ошибка. С другой стороны, ограничения можно добавлять или удалять непосредственно в секциях (если они отсутствуют в родительской таблице).

- Так как секционированная таблица сама по себе не содержит данные, использование `TRUNCATE ONLY` для секционированной таблицы всегда будет считаться ошибкой.

5.11.3. Секционирование с использованием наследования

Хотя встроенное декларативное секционирование полезно во многих часто возникающих ситуациях, бывают обстоятельства, требующие более гибкого подхода. В этом случае секционирование можно реализовать, применив механизм наследования таблиц, что даст ряд возможностей, неподдерживаемых при декларативном секционировании, например:

- При декларативном секционировании все секции должны иметь в точности тот же набор столбцов, что и секционируемая таблица, тогда как обычное наследование таблиц допускает наличие в дочерних таблицах дополнительных столбцов, отсутствующих в родителе.
- Механизм наследования таблиц поддерживает множественное наследование.
- С декларативным секционированием поддерживается только разбиение по спискам, по диапазонам и по хешу, тогда как с наследованием таблиц данные можно разделять по любому критерию, выбранному пользователем. (Однако заметьте, что если исключение по ограничению не позволяет эффективно устранять дочерние таблицы из планов запросов, производительность запросов будет очень низкой.)

5.11.3.1. Пример

В этом примере будет создана структура секционирования, равнозначная структуре из примера с декларативным секционированием. Выполните следующие действия:

1. Создайте «главную» таблицу, от которой будут наследоваться все «дочерние» таблицы. Главная таблица не будет содержать данные. Не определяйте в ней никакие

ограничения-проверки, если только вы не намерены применить их во всех дочерних таблицах. Также не имеет смысла определять в ней какие-либо индексы или ограничения уникальности. В нашем примере главной таблицей будет `measurement` со своим изначальным определением:

```
CREATE TABLE measurement (
    city_id         int not null,
    logdate         date not null,
    peaktemp        int,
    unitsales       int
);
```

2. Создайте несколько «дочерних» таблиц, унаследовав их все от главной. Обычно в таких таблицах не будет никаких дополнительных столбцов, кроме унаследованных. Как и с декларативным секционированием, эти таблицы во всех отношениях будут обычными таблицами PostgreSQL (или сторонними таблицами).

```
CREATE TABLE measurement_y2006m02 () INHERITS (measurement);
CREATE TABLE measurement_y2006m03 () INHERITS (measurement);
...
CREATE TABLE measurement_y2007m11 () INHERITS (measurement);
CREATE TABLE measurement_y2007m12 () INHERITS (measurement);
CREATE TABLE measurement_y2008m01 () INHERITS (measurement);
```

3. Добавьте в дочерние таблицы неперекрывающиеся ограничения, определяющие допустимые значения ключей для каждой из них.

   Типичные примеры таких ограничений:

```
CHECK ( x = 1 )
CHECK ( county IN ( 'Oxfordshire', 'Buckinghamshire', 'Warwickshire' ))
CHECK ( outletID >= 100 AND outletID < 200 )
```

Убедитесь в том, что ограничения не пересекаются, то есть никакие значения ключа не относятся сразу к нескольким дочерним таблицам. Например, часто допускают такую ошибку в определении диапазонов:

```
CHECK ( outletID BETWEEN 100 AND 200 )
CHECK ( outletID BETWEEN 200 AND 300 )
```

Это не будет работать, так как неясно, к какой дочерней таблице должно относиться значение 200. Поэтому диапазоны должны определяться следующим образом:

```
CREATE TABLE measurement_y2006m02 (
    CHECK ( logdate >= DATE '2006-02-01' AND logdate < DATE '2006-03-01' )
) INHERITS (measurement);

CREATE TABLE measurement_y2006m03 (
    CHECK ( logdate >= DATE '2006-03-01' AND logdate < DATE '2006-04-01' )
) INHERITS (measurement);

...
CREATE TABLE measurement_y2007m11 (
    CHECK ( logdate >= DATE '2007-11-01' AND logdate < DATE '2007-12-01' )
) INHERITS (measurement);

CREATE TABLE measurement_y2007m12 (
    CHECK ( logdate >= DATE '2007-12-01' AND logdate < DATE '2008-01-01' )
) INHERITS (measurement);

CREATE TABLE measurement_y2008m01 (
    CHECK ( logdate >= DATE '2008-01-01' AND logdate < DATE '2008-02-01' )
) INHERITS (measurement);
```

4. Для каждой дочерней таблицы создайте индекс по ключевому столбцу (или столбцам), а также любые другие индексы по своему усмотрению.

```
CREATE INDEX measurement_y2006m02_logdate ON measurement_y2006m02 (logdate
CREATE INDEX measurement_y2006m03_logdate ON measurement_y2006m03 (logdate
CREATE INDEX measurement_y2007m11_logdate ON measurement_y2007m11 (logdate
CREATE INDEX measurement_y2007m12_logdate ON measurement_y2007m12 (logdate
CREATE INDEX measurement_y2008m01_logdate ON measurement_y2008m01 (logdate
```

5. Мы хотим, чтобы наше приложение могло сказать `INSERT INTO measurement ...` и данные оказались в соответствующей дочерней таблице. Мы можем добиться этого, добавив подходящую триггерную функцию в главную таблицу. Если данные всегда будут добавляться только в последнюю дочернюю таблицу, нам будет достаточно очень простой функции:

```
CREATE OR REPLACE FUNCTION measurement_insert_trigger()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO measurement_y2008m01 VALUES (NEW.*);
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

После функции мы создадим вызывающий её триггер:

```
CREATE TRIGGER insert_measurement_trigger
    BEFORE INSERT ON measurement
    FOR EACH ROW EXECUTE FUNCTION measurement_insert_trigger();
```

Мы должны менять определение триггерной функции каждый месяц, чтобы она всегда вставляла данные в текущую дочернюю таблицу. Определение самого

триггера, однако, менять не требуется.

Но мы можем также сделать, чтобы сервер автоматически находил дочернюю таблицу, в которую нужно направить добавляемую строку. Для этого нам потребуется более сложная триггерная функция:

```
CREATE OR REPLACE FUNCTION measurement_insert_trigger()
RETURNS TRIGGER AS $$
BEGIN
    IF ( NEW.logdate >= DATE '2006-02-01' AND
         NEW.logdate < DATE '2006-03-01' ) THEN
        INSERT INTO measurement_y2006m02 VALUES (NEW.*);
    ELSIF ( NEW.logdate >= DATE '2006-03-01' AND
            NEW.logdate < DATE '2006-04-01' ) THEN
        INSERT INTO measurement_y2006m03 VALUES (NEW.*);
    ...
    ELSIF ( NEW.logdate >= DATE '2008-01-01' AND
            NEW.logdate < DATE '2008-02-01' ) THEN
        INSERT INTO measurement_y2008m01 VALUES (NEW.*);
    ELSE
        RAISE EXCEPTION
  'Date out of range.  Fix the measurement_insert_trigger() function!';
    END IF;
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

Определение триггера остаётся прежним. Заметьте, что все условия `IF` должны в точности отражать ограничения `CHECK` соответствующих дочерних таблиц.

Хотя эта функция сложнее, чем вариант с одним текущим месяцем, её не придётся так часто модифицировать, так как ветви условий можно добавить заранее.

Примечание

На практике будет лучше сначала проверять условие для последней дочерней таблицы, если строки добавляются в неё чаще всего, но для простоты мы расположили проверки триггера в том же порядке, что и в других фрагментах кода для этого примера.

Другой способ перенаправления добавляемых строк в соответствующую дочернюю таблицу можно реализовать, определив для главной таблицы не триггер, а правила. Например:

```
CREATE RULE measurement_insert_y2006m02 AS
ON INSERT TO measurement WHERE
    ( logdate >= DATE '2006-02-01' AND logdate < DATE '2006-03-01' )
DO INSTEAD
    INSERT INTO measurement_y2006m02 VALUES (NEW.*);
...
CREATE RULE measurement_insert_y2008m01 AS
ON INSERT TO measurement WHERE
    ( logdate >= DATE '2008-01-01' AND logdate < DATE '2008-02-01' )
DO INSTEAD
    INSERT INTO measurement_y2008m01 VALUES (NEW.*);
```

С правилами связано гораздо больше накладных расходов, чем с триггером, но они относятся к запросу в целом, а не к каждой строке. Поэтому этот способ может быть более выигрышным при массовом добавлении данных. Однако в большинстве случаев триггеры будут работать быстрее.

Учтите, что команда `COPY` игнорирует правила. Если вы хотите вставить данные с помощью `COPY`, вам придётся копировать их сразу в нужную дочернюю таблицу, а не

в главную таблицу. С другой стороны, `COPY` не отменяет триггеры, так что с триггерами вы сможете использовать её обычным образом.

Ещё один недостаток подхода с правилами связан с невозможностью выдать ошибку, если добавляемая строка не подпадает ни под одно из правил; в этом случае данные просто попадут в главную таблицу.

6. Убедитесь в том, что параметр конфигурации constraint_exclusion не выключен в `postgresql.conf`. В противном случае дочерние таблицы могут сканироваться, когда это не требуется.

Как уже можно понять, для реализации сложной иерархии таблиц может потребоваться DDL-код значительного объёма. В данном примере нам потребуется создавать дочернюю таблицу каждый месяц, так что было бы разумно написать скрипт, формирующий требуемый код DDL автоматически.

5.11.3.2. Обслуживание таблиц, секционированных через наследование

Чтобы быстро удалить старые данные, просто удалите ставшую ненужной дочернюю таблицу:

```
DROP TABLE measurement_y2006m02;
```

Чтобы удалить дочернюю таблицу из иерархии наследования, но сохранить к ней доступ как к самостоятельной таблице:

```
ALTER TABLE measurement_y2006m02 NO INHERIT measurement;
```

Чтобы добавить новую дочернюю таблицу для новых данных, создайте пустую дочернюю таблицу так же, как до этого создавали начальные:

```
CREATE TABLE measurement_y2008m02 (
    CHECK ( logdate >= DATE '2008-02-01' AND logdate < DATE '2008-03-01' )
) INHERITS (measurement);
```

Можно также создать новую таблицу и наполнить её данными до добавления в иерархию таблиц. Это позволит загрузить, проверить и при необходимости преобразовать данные до того, как запросы к главной таблице смогут их увидеть.

```
CREATE TABLE measurement_y2008m02
  (LIKE measurement INCLUDING DEFAULTS INCLUDING CONSTRAINTS);
ALTER TABLE measurement_y2008m02 ADD CONSTRAINT y2008m02
   CHECK ( logdate >= DATE '2008-02-01' AND logdate < DATE '2008-03-01' );
\copy measurement_y2008m02 from 'measurement_y2008m02'
-- возможна дополнительная подготовка данных
ALTER TABLE measurement_y2008m02 INHERIT measurement;
```

5.11.3.3. Ограничения

С реализацией секционирования через наследование связаны следующие ограничения:

- Система не может проверить автоматически, являются ли все ограничения CHECK взаимно исключающими. Поэтому безопаснее будет написать и отладить код для формирования дочерних таблиц и создания и/или изменения связанных объектов, чем делать это вручную.
- Индексы и внешние ключи относятся к определённой таблице, но не к её иерархии наследования, поэтому с ними связаны дополнительные ограничения.
- Показанные здесь схемы подразумевают, что ключевой столбец (или столбцы) в строке никогда не меняется, или меняется не настолько, чтобы строку потребовалось перенести в другую секцию. Если же попытаться выполнить такой оператор UPDATE, произойдёт ошибка из-за нарушения ограничения CHECK. Если вам нужно обработать и такие случаи, вы можете установить подходящие триггеры на обновление в дочерних таблицах, но это ещё больше усложнит управление всей конструкцией.
- Если вы выполняете команды VACUUM или ANALYZE вручную, не забывайте, что их нужно запускать для каждой дочерней таблицы в отдельности. Команда

```
ANALYZE measurement;
```

обработает только главную таблицу.

- Операторы `INSERT` с предложениями `ON CONFLICT` скорее всего не будут работать ожидаемым образом, так как действие `ON CONFLICT` предпринимается только в случае нарушений уникальности в указанном целевом отношении, а не его дочерних отношениях.

- Для направления строк в нужные дочерние таблицы потребуются триггеры или правила, если только приложение не знает непосредственно о схеме секционирования. Разработать триггеры может быть довольно сложно, и они будут работать гораздо медленнее, чем внутреннее распределение кортежей при декларативном секционировании.

5.11.4. Устранение секций

Устранение секций — это приём оптимизации запросов, который ускоряет работу с декларативно секционированными таблицами. Например:

```
SET enable_partition_pruning = on;                 -- по умолчанию
SELECT count(*) FROM measurement WHERE logdate >= DATE '2008-01-01';
```

Без устранения секций показанный запрос должен будет просканировать все секции таблицы `measurement`. Когда устранение секций включено, планировщик рассматривает определение каждой секции и может заключить, что какую-либо секцию сканировать не нужно, так как в ней не может быть строк, удовлетворяющих предложению `WHERE` в запросе. Когда планировщик может сделать такой вывод, он исключает (*устраняет*) секцию из плана запроса.

Используя команду EXPLAIN и параметр конфигурации enable_partition_pruning, можно увидеть отличие плана, из которого были устранены секции, от плана без устранения.

Типичный неоптимизированный план для такой конфигурации таблицы будет выглядеть так:

```
SET enable_partition_pruning = off;
EXPLAIN SELECT count(*) FROM measurement WHERE logdate >= DATE '2008-01-01';
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=188.76..188.77 rows=1 width=8)
   ->  Append  (cost=0.00..181.05 rows=3085 width=0)
         ->  Seq Scan on measurement_y2006m02  (cost=0.00..33.12 rows=617 width
               Filter: (logdate >= '2008-01-01'::date)
         ->  Seq Scan on measurement_y2006m03  (cost=0.00..33.12 rows=617 width
               Filter: (logdate >= '2008-01-01'::date)
...
         ->  Seq Scan on measurement_y2007m11  (cost=0.00..33.12 rows=617 width
               Filter: (logdate >= '2008-01-01'::date)
         ->  Seq Scan on measurement_y2007m12  (cost=0.00..33.12 rows=617 width
               Filter: (logdate >= '2008-01-01'::date)
         ->  Seq Scan on measurement_y2008m01  (cost=0.00..33.12 rows=617 width
               Filter: (logdate >= '2008-01-01'::date)
```

В некоторых или всех секциях может применяться не полное последовательное сканирование, а сканирование по индексу, но основная идея примера в том, что для удовлетворения запроса не нужно сканировать старые секции. И когда мы включаем устранение секций, мы получаем значительно более эффективный план, дающий тот же результат:

```
SET enable_partition_pruning = on;
EXPLAIN SELECT count(*) FROM measurement WHERE logdate >= DATE '2008-01-01';
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=37.75..37.76 rows=1 width=8)
   ->  Seq Scan on measurement_y2008m01  (cost=0.00..33.12 rows=617 width=0)
         Filter: (logdate >= '2008-01-01'::date)
```

Заметьте, что механизм устранения секций учитывает только ограничения, определённые неявно ключами разбиения, но не наличие индексов. Поэтому определять индексы для столбцов ключа не обязательно. Нужно ли создавать индекс для определённой секции, зависит от того, какую часть секции (меньшую или большую), по вашим представлениям, будут сканировать запросы, обращающиеся к этой секции. Индекс будет полезен в первом случае, но не во втором.

Устранение секций может производиться не только при планировании конкретного запроса, но и в процессе его выполнения. Благодаря этому может быть устранено больше секций, когда условные выражения содержат значения, неизвестные во время планирования, например параметры, определённые оператором `PREPARE`, значения, получаемые из подзапросов, или параметризованные значения во внутренней стороне соединения с вложенным циклом. Устранение секций в процессе выполнения запроса возможно в следующие моменты времени:

- Во время подготовки плана запроса. В этот момент можно устранить секции, учитывая значения параметров, известные при подготовке выполнения запроса. Секции, устранённые на этом этапе, не будут видны в выводе `EXPLAIN` или `EXPLAIN ANALYZE`. Их общее количество можно определить по свойству «Subplans Removed» в выводе `EXPLAIN`.

- В процессе собственно выполнения плана запроса. Устранение секций также может выполняться на этом этапе и позволяет отфильтровать секции, используя значения, которые становятся известны, когда запрос выполняется фактически. В частности это могут быть значения из подзапросов и значения параметров времени выполнения, например из параметризованных соединений с вложенными циклами. Так как значения параметров могут меняться многократно при выполнении запроса, устранение секций выполняется при изменении любого из параметров, анализируемых механизмом устранения. Чтобы определить, были ли секции устранены на данном этапе, нужно внимательно изучить свойство `loops` в выводе `EXPLAIN ANALYZE`. Подпланы, соответствующие разным секциям, могут иметь разные значения, в зависимости от того, сколько раз они устранялись во время выполнения.

Некоторые из них могут даже иметь значение `(never executed)` (никогда не выполнялись), если они устранялись всегда.

Устранение секций можно отключить, воспользовавшись параметром enable_partition_pruning.

5.11.5. Секционирование и исключение по ограничению

Исключение по ограничению — приём оптимизации запросов, подобный устранению секций. Прежде всего он применяется, когда секционирование осуществляется с использованием старого метода наследования, но он может быть полезен и для других целей, включая декларативное секционирование.

Исключение по ограничению работает во многом так же, как и устранение секций; отличие состоит в том, что оно использует ограничения `CHECK` всех таблиц (поэтому оно так и называется), тогда как для устранения секций используются границы секции, которые существуют только в случае декларативного секционирования. Ещё одно различие состоит в том, что исключение по ограничению применяется только во время планирования; во время выполнения секции из плана удаляться не будут.

То, что исключение по ограничению использует ограничения `CHECK` (вследствие чего оно работает медленнее устранения секций), иногда может быть и преимуществом. Ограничения могут быть определены даже для декларативно секционированных таблиц, в дополнение к внутренним границам секций, и тогда исключение по ограничению сможет дополнительно убрать некоторые секции из плана запроса.

По умолчанию параметр constraint_exclusion имеет значение не `on` и не `off`, а промежуточное (и рекомендуемое) значение `partition`, при котором этот приём будет применяться только к запросам, где предположительно будут задействованы таблицы, секционированные с использованием наследования. Значение `on` обязывает планировщик просматривать ограничения `CHECK` во всех запросах, даже в самых простых, где выигрыш от исключения по ограничению маловероятен.

Применяя исключения по ограничению, необходимо учитывать следующее:

- Исключение по ограничению применяется только при планировании запроса, в отличие от устранения секций, которое может осуществляться и при выполнении запроса.

- Исключение по ограничению работает только когда предложение `WHERE` в запросе содержит константы (или получаемые извне параметры). Например, сравнение с функцией переменной природы, такой как `CURRENT_TIMESTAMP`, нельзя оптимизировать, так как планировщик не может знать, в какую дочернюю таблицу попадёт значение функции во время выполнения.

- Ограничения секций должны быть простыми, иначе планировщик не сможет вычислить, какие дочерние таблицы не нужно обрабатывать. Для секционирования по спискам используйте простые условия на равенства, а для секционирования по диапазонам — простые проверки диапазонов, подобные показанным в примерах. Рекомендуется создавать ограничения секций, содержащие только такие сравнения секционирующих столбцов с константами, в которых используются операторы, поддерживающие B-деревья. Это объясняется тем, что в ключе разбиения допускаются только такие столбцы, которые могут быть проиндексированы в B-дереве.

- При анализе для исключения по ограничению исследуются все ограничения всех дочерних таблиц, относящихся к главной, так что при большом их количестве время планирования запросов может значительно увеличиться. Поэтому устаревший вариант секционирования, основанный на наследовании, будет работать хорошо, пока количество дочерних таблиц не превышает примерно ста; не пытайтесь применять его с тысячами дочерних таблиц.

5.11.6. Рекомендации по декларативному секционированию

К секционированию таблицы следует подходить продуманно, так как неудачное решение может отрицательно повлиять на скорость планирования и выполнения запросов.

Одним из самых важных факторов является выбор столбца или столбцов, по которым будут секционироваться ваши данные. Часто оптимальным будет секционирование по

столбцу или набору столбцов, которые практически всегда присутствуют в предложении WHERE в запросах, обращающихся к секционируемой таблице. Предложения WHERE, совместимые с ограничениями границ секции, могут применяться для устранения ненужных для выполнения запроса секций. Однако наличие ограничений PRIMARY KEY или UNIQUE может подтолкнуть к выбору и других столбцов в качестве секционирующих. Также при планировании секционирования следует продумать, как будут удаляться данные. Секцию целиком можно отсоединить очень быстро, поэтому может иметь смысл разработать стратегию секционирования так, чтобы массово удаляемые данные оказывались в одной секции.

Также важно правильно выбрать число секций, на которые будет разбиваться таблица. Если их будет недостаточно много, индексы останутся большими, не улучшится и локальность данных, вследствие чего процент попаданий в кеш окажется низким. Однако и при слишком большом количестве секций возможны проблемы. С большим количеством секций увеличивается время планирования и потребление памяти как при планировании, так и при выполнении запросов, о чём рассказывается далее. Выбирая стратегию секционирования таблицы, также важно учитывать, какие изменения могут произойти в будущем. Например если вы решите создавать отдельные секции для каждого клиента, и в данный момент у вас всего несколько больших клиентов, подумайте, что будет, если через несколько лет у вас будет много мелких клиентов. В этом случае может быть лучше произвести секционирование по хешу (HASH) и выбрать разумное количество секций, но не создавать секции по списку (LIST) в надежде, что количество клиентов не увеличится до такой степени, что секционирование данных окажется непрактичным.

Вложенное секционирование позволяет дополнительно разделить те секции, которые предположительно окажутся больше других. Также можно использовать разбиение по диапазонам с ключом, состоящим из нескольких столбцов. Но при любом из подходов легко может получиться слишком много секций, так что рекомендуется использовать их осмотрительно.

Важно учитывать издержки секционирования, которые отражаются на планировании и выполнении запросов. Планировщик запросов обычно довольно неплохо справляется с

иерархиями, включающими до нескольких тысяч секций, если при выполнении типичных запросов ему удаётся устранить почти все секции. Однако когда после устранения остаётся большое количество секций, возрастает и время планирования запросов, и объём потребляемой памяти. Наличие большого количества секций нежелательно ещё и потому, что потребление памяти сервером может значительно возрастать со временем, особенно когда множество сеансов обращаются ко множеству секций. Это объясняется тем, что в локальную память каждого сеанса, который обращается к секциям, должны быть загружены метаданные всех этих секций.

С нагрузкой, присущей информационным хранилищам, может иметь смысл создавать больше секций, чем с нагрузкой OLTP. Как правило, в информационных хранилищах время планирования запроса второстепенно, так как гораздо больше времени тратится на выполнение запроса. Однако при любом типе нагрузки важно принимать правильные решения на ранней стадии реализации, так как процесс переразбиения таблиц большого объёма может оказаться чрезвычайно длительным. Для оптимизации стратегии секционирования часто бывает полезно предварительно эмулировать ожидаемую нагрузку. Но ни в коем случае нельзя полагать, что чем больше секций, тем лучше, равно как и наоборот.

← Пред.
5.10. Наследование

Наверх ↑
Начало

След. →
5.12. Сторонние данные