Написание запросов в Greenplum. Оптимизация

Запросы в Greenplum могут выполняться очень быстро, если уметь правильно пользоваться его особенностями. Если же их игнорировать, запросы будут падать или работать очень долго. Опишу здесь несколько технических (не всегда простых) моментов, на которые стоит обратить внимание, и которые помогут улучшить ваши запросы.

Все то же самое, чуть менее подробно, с картинками

Чек-лист написания запросов

Если запрос работает дольше, чем ожидалось, можно проверить его по следующим пунктам:

1. Выбираем только нужные колонки. Каждая ненужная колонка в *select ** занимает память, которую Greemplum мог бы потратить с пользой.
2. Фильтруем данные как можно раньше. Чем быстрее применим фильтры к таблицам, тем меньше строк будет обработано на всех последующих этапах, потребуется меньше памяти
3. Декомпозируем большие запросы. Чем больше запрос, тем сложнее для Greenplum составить оптимальный план выполнения запроса. Большие запросы лучше разделять на несколько меньших и сохранять промежуточные результаты во временных таблицах.
4. Стараемся читать каждую таблицу только один раз. Особенно это критично для больших таблиц.
5. Проверяем свои таблицы:
    a. Они должны быть правильно распределены.
    b. Со сжатием.
    c. Для широких таблиц обязательно колоночное хранение.
6. Что интересного можно посмотреть плане запроса (explain):
    a. Стараемся минимизировать количество слайсов.
    Если количество слайсов примерно равно количеству таблиц или превосходит количество таблиц - надо оптимизировать запрос. Почему большое количество слайсов нежелательно, написал в разделе "Слайсы и перемещения данных". Количество можно посмотреть в плане запроса, первая строчка будет выглядеть так:
    *Gather Motion 30:1 (**slice2**; segments: 30) (cost=0.00..136005.55 rows=1318948474 width=34)*

    b. Проверяем, какой оптимизатор использует Greenplum.
    Если в последней строчке плана вы видите *Optimizer: Pivotal Optimizer (GPORCA)* - значит Greenplum использует свой оптимизатор. Любое другое значение показывает, что Greenplum не смог спланировать ваш запрос и вызвал запасной оптимизатор (Postgres). Оптимизатор Postgres умеет больше, но не учитывает особенности Greenplum

    c. Выбираем только нужные партиции. Убедиться в этом можно только по плану запроса. Надо искать строки вида
    -> Partition Selector for s_order_item (dynamic scan id: 1) (cost=10.00..100.00 rows=4 width=4)
    Partitions selected: 1 (out of 113)
    Если выбраны только нужные партиции, запрос будет работать эффективнее. Если игнорировать выбор партиций, Greenplum начнет читать всю таблицу целиком - это займет больше времени.

    d. Стараемся уйти от nested loop join. Чаще используется при джойне в случае не равенства ключей. И логического ИЛИ - or. Соответственно, нужно оценить, можно ли уйти на строгое равенство. И на строгое логическое И - and.

    e. Минимизируем тяжелые операторы, генерирующие spil-файлы:
    HashJoin
    HashAgg
    Sort

Чек-лист для таблиц

1. Колонки не должны называться ключевыми словами SQL, т.е. избегаем *number, date, year, order* и пр.
2. Проверяем storage parameters. По умолчанию создаем таблицы с *appendonly=true, orientation=column, compresstype=zstd, compresslevel=2*
3. Проверяем выбор дистрибуции. Для небольших таблиц можно выбирать replicated. Для всех остальных надо выбирать колонку дистрибуции так, чтобы не было перекоса в данных. Если не удается найти такую колонку, выбираем дистрибуцию randomly, но это - крайний случай.
4. ...
5. PROFIT! (список будет дополняться)

Слайсы и перемещения данных

В больших запросах, где участвует много таблиц, полезно делать explain и оценивать количество **slice** и **motion** в запросе. Если запрос работает долго, вполне вероятно, что он переносит по между сегментами большие пачки данных. Такие переносы - узкое местов Greenplum. Каждый такой перенос отмечается в плане запроса как motion. Каждый motion разделяет запрос два слайса. Сделав explain на своем запросе, можно посмотреть кол-во слайсов (в самой первой строке). Количество motion-ов будет равно количеству слайсов.

Почему так происходит

В Greenplum каждая таблица распределена по сегментам на основании ключа распределения (distribution). Изменение распределения таблицы вызывает перемещение данных между сегментами.
Наиболее частая причина таких перемещений - join. Физическое соединение двух таблиц может выполняться ТОЛЬКО в рамках одного сегменте = базы postgres.
Соответственно, если мы делаем join двух таблиц, распределенных по разным колонкам, данные одной из таблиц будут перераспределены по ключу join-а. Таким образом, на каждом сегменте окажутся нужные для join-а фрагменты данных обеих таблиц, и Greenplum сможет их соединить.

Виды motion

1. Gather motion. Присутствует в любом запросе, самая верхняя строчка плана. Также появится в сложной преаггрегации. (Например, distinct * join) Gather motion - фактически собирает данные с сегментов и передает их мастеру.
2. Redistribute motion. В этом случае между сегментами переносятся только нужные строки в соответствии с ключом распределения
3. Broadcast motion. Вся таблица копируется на каждый из сегментов. Может быть удобно для небольших таблиц.

Workaround

1. не писать большие запросы :)
2. следить, чтобы большие таблицы не переносились через broadcast motion
3. разделять большие запросы на несколько меньших так, чтобы внутри каждого из меньших запросов было минимум перераспределений (как минимум больших таблиц). Результаты сохраняем во временных таблицах с распределением, которое будет удобно использовать в дальнейшем.

Перекос в данных (data skew)

Перекос в данных - одна из причин долгого выполнения запросов. Перекос возникает из-за неправильного выбора ключа распределения (distribution).

При выборе ключа распределения надо руководствоваться следующими правилами:

1. будем часто делать join по этой колонке;
2. в колонке много уникальных значений;
3. в колонке нет пустых значений (NULL);
4. если ключ распределения совпадает с ключом партицирования, может сложиться ситуация, когда вся партиция целиком окажется только на одном сегменте.

При нарушении любого из пунктов 2, 3, 4, может получиться так, что на одном из сегментов образуется значительно больше строк чем на остальных (распространенная проблема с NULL). При этом Greenplum всегда работает со скоростью САМОГО МЕДЛЕННОГО сегмента. И соответственно, если на одном сегменте накопится слишком много данных, все остальные сегменты будут его ждать прежде чем вернуть результат запроса.

Случай из моей практики. Как-то раз я забыл указать distribution в скрипте, и Greenplum решил распределить данные по колонке с типом bool. Обнаружили это через несколько месяцев, когда таблица разрослась до нескольких млрд. строк, и все запросы с ее участием начали очень сильно тормозить.

Workaround

При выборе колонки распределения проверяем наличие перекоса запросом ниже:

```
SELECT max(c) AS "Max Seg Rows"
     , min(c) AS "Min Seg Rows"
     , (max(c)-min(c))*100.0/max(c) AS "Percentage Difference Between Max & Min"
FROM (SELECT count(*) c, gp_segment_id FROM dds.h_shipment GROUP BY 2) AS a;
```

Избыточные чтения

Каждую таблицу стараемся читать только один раз. Особенно это касается больших сущностей (shipment, order, order_item, line_items и др.). Например, в запросе ниже dds.s_order читается дважды.

```
explain
with order_user_b2b as (
    select o.order_id
         , u.registration_b2b_flg
    from dds.s_order as o
        left join
            dds.s_user as u on u.user_id = o.user_id
)
select s.shipment_id
     , b.registration_b2b_flg
from dds.s_shipment as s
    inner join
        dds.s_order as o on o.order_id = s.shipment_id
    left join
        order_user_b2b as b on b.order_id = o.order_id;
```

При этом если переписать запрос, количество чтений сократится, а логика не поменяется.

```
explain
with order_user_b2b as (
    select o.order_id
         , u.registration_b2b_flg
    from dds.s_order as o
        left join
            dds.s_user as u on u.user_id = o.user_id
)
select s.shipment_id
     , o.registration_b2b_flg
from dds.s_shipment as s
    inner join
        order_user_b2b as o on o.order_id = s.shipment_id;

--

explain
select s.shipment_id
     , u.registration_b2b_flg
from dds.s_shipment as s
    inner join
        dds.s_order as o on o.order_id = s.shipment_id
    left join
        dds.s_user as u on u.user_id = o.user_id;
```

Использование партиций

Если таблица разбита на партиции, то одним из лучших способов оптимизации является выбор только нужных партиций в запросе. В этом случае будет просканирована не вся таблица целиком, а только нужный ее фрагмент: в результате потребуется меньше читать диск, меньше памяти для хранения промежуточных результатов запроса.

Информацию о том, есть у таблицы партиции, и как они определены, проще всего получить из DDL таблицы или из системного представления *pg_partitions*.

Чтобы выбрать только нужные партиции, необходимо указать фильтр по колонке партицирования при обращении к таблице. Например, так:

```
select *
from dm.fct_order_item
where created_dttm >= '2022-09-01'
  and created_dttm < '2023-01-01';
```

Или так, если таблица указана в джоине:

```
select *
from dm.fct_shipment as s
    left join
        dm.fct_order_item as i on s.shipment_id = i.shipment_id
                              and i.created_dttm >= '2022-09-01'
                              and i.created_dttm < '2023-01-01';
```

Убедиться в том, что будут выбраны только нужные партиции можно по плану запроса. Если для запроса к таблице с партициями вызвать *explain*, то в плане будут выведены строки о выбранных партициях:
-> Partition Selector for fct_order_item (dynamic scan id: 1) (cost=10.00..100.00 rows=4 width=4)
Partitions selected: 5 (out of 113)

Подробнее об использовании партиций и их видах можно почитать в документации.

Оптимизатор

Для построения плана запросов в Greenplum есть два оптимизатора:

1. Pivotal Optimizer (GPORCA) - оптимизатор Greenplum
2. Postgres query optimizer - оптимизатор Postgres

Желательно, чтобы всегда работал оптимизатор Greenplum - GPORCA. Но некоторые запросы он обработать не может, и в этом случае вызывается оптимизатор Postgres. Оптимизатор Postgres может больше, но он ничего не знает об особенностях Greenplum, таких как распределение таблицы и др. И в результате, если в планировании запроса участвует postgres-оптимизатор, запрос может выполняться неэффективно (но зато он выполнится).

Посмотреть, какой оптимизатор применяется в запросе можно, вызвав план запроса *explain*. В последней строке будет указан оптимизатор: *Optimizer: Pivotal Optimizer (GPORCA)*.

Надо стараться писать запросы так, чтобы их мог спланировать родной оптимизатор Greenplum - это наиболее эффективно. При этом надо помнить, что не всегда это возможно. И также, если в результате переписывания запроса страдает его читабельность, или запрос становится сложнее (появляются избыточные чтения таблиц, избыточные джоины), то лучше остаться с postgres-оптимизатором.

Статистика

От статистики зависит, например, оценивает ли ГП таблицу как большую или как маленькую.

Поэтому важно на таблицы делать **analyze**. Делаем после каждого наполнения таблицы. Иногда полезно перед селектом сделать analyze используемых таблиц.

Не используем

- **distinct ***. Особенно в середине запроса.
- **union** - это неявный distinct. Используем union all.
- любые операций в where над **партицированной** колонкой. Никаких where part_column*2 > 100. Строго part_column > 50. upper(part_column) = 'ABC' также недопустимо.

Plan errors:

Вместо заключения. На что в первую очередь смотреть при оптимизации запроса.

- перекос строк (распределение)
- бродкаст большой таблицы
- замножение записей после джойна
- хэш на большую таблицу
- плохой выбор движения данных
- вложенный цикл больших таблиц
- сортировка большого объема данных
- подозрение на неактуальную статистику