Основные конструкции SQL

Раздаточный материал ко 2-му уроку, посвященному дистрибуции и созданию таблиц.

|  |  |
| --- | --- |
| СОДЕРЖАНИЕ | |
| 1 | DML операции |
| 2 | Транзакции |
| 3 | Уровни изоляции |
| 4 | Очистка |
| 5 | Блокировки |
| 6 | Мониторинг текущих запросов |
| 7 | План запроса |
| 8 | Оптимизаторы |
| 9 | Сбор статистики |

DML операции

DML – (Data Manipulation Language) это язык манипулирования данными, который используется для извлечения, удаления, обновления и вставки новых данных.

Greenplum поддерживает большинство функций стандарта SQL 2003 в том числе такие команды как:

* SELECT – извлечение.
* DELETE – удаление.
* INSERT – вставка.
* UPDATE - обновление

Команда MERGE на текущий момент не поддерживается.

INSERT добавляет строки в таблицу. Эта команда может добавить одну или несколько строк, сформированных выражениями значений, либо ноль или более строк, выданных дополнительным запросом.

Имена целевых столбцов могут перечисляться в любом порядке. Если список с именами столбцов отсутствует, по умолчанию целевыми столбцами становятся все столбцы заданной таблицы; либо первые N из них, если только N столбцов поступает от предложения VALUES или запроса. Значения, получаемые от предложения VALUES или запроса, связываются с явно или неявно определённым списком столбцов слева направо.

Все столбцы, не представленные в явном или неявном списке столбцов, получат значения по умолчанию, если для них заданы эти значения, либо NULL в противном случае.

Если выражение для любого столбца выдаёт другой тип данных, система попытается автоматически привести его к нужному.

Операция INSERT с таблицами без уникальных индексов не блокируется параллельно выполняемыми операциями. В таблицах с уникальными индексами эта операция может блокироваться, если в параллельных сеансах выполняются действия, которые блокируют или изменяют строки, совпадающие с вставляемыми значениями в уникальном индексе

UPDATE изменяет значения указанных столбцов во всех строках, удовлетворяющих условию. В предложении SET должны указываться только те столбцы, которые будут изменены; столбцы, не изменяемые явно, сохраняют свои предыдущие значения.

Изменить строки в таблице, используя информацию из других таблиц в базе данных, можно двумя способами: применяя вложенные запросы или указав дополнительные таблицы в предложении FROM. Выбор предпочитаемого варианта зависит от конкретных обстоятельств.

Команда DELETE используется для удаления строк из таблиц. В команде необходима указать имя таблицы и условие удаления. Если не будет выставлено условие удаления, то в таком случае удалятся все строки из таблицы. Следует учесть, что для очистки всех строк из таблицы рекомендуется использовать команду TRUNCATE, которая отрабатывает за меньшее количество времени

Транзакции

Суть транзакций заключается в том, что они объединяют последовательность DML операций в одну операцию по принципу «Все или ничего». Если не выполняется хотя бы одна операция, то изменения, сделанные предыдущими операциями, также откатываются

Промежуточное состояние внутри этой последовательности не видны другим транзакциям.

Чтобы операции выполнялись в рамках одной транзакции, необходимо объединить их в один блок, который начинается с ключевого поля BEGIN и заканчивается ключевым словом COMMIT.

В Greenplum по умолчанию включен режим AUTOCOMMIT, в этом режиме любая DML операция, которая не содержит ошибок, будет зафиксирована в базе данных без необходимости явно прописывать COMMIT

Уровни изоляции

В СУБД многие транзакции могут выполняться одновременно, соответственно такие транзакции могут пересекаться по запросам, читать и модифицировать данные в одних и тех же таблицах. Именно от уровня изоляции зависит будет ли транзакция видеть те изменения, которые в это же время производятся в других параллельных транзакциях. В стандарте SQL выделяют 4 основные уровня изоляции

1. READ UNCOMMITED - Чтение незафиксированных данных.
2. READ COMMITTED - Чтение зафиксированных данных.
3. REPEATABLE READ - Повторное чтение
4. SERIALIZABLE – Сериализация

При выполнении параллельных транзакции на каждом уровне изоляции могут проявляться следующие аномалии.

1. DIRTY READ - Грязное чтение. Другие транзакции видят изменения текущей транзакции даже если эти изменения не были зафиксированы
2. NON-REPEATABLE READ - Неповторяемое чтение. При повторном чтении в рамках одной транзакции данные оказываются измененными
3. PHANTOM READ - Фантомное чтение. При повторном чтении в рамках одной транзакции изменяется количество строк

Greenplum реализует только два уровня изоляции транзакций, хотя разработчик приложения может запросить любой возможных. Уровень READ UNCOMMITTED в Greenplum реализуется READ COMMITTED, а уровень SERIALIZABLE – как REPEATABLE READ. Причем реализация REPEATABLE READ в Greenplum не допускает фантомного чтения.

По умолчанию Greenplum использует уровень изоляции READ COMMITTED,но это значение может быть изменено в параметре конфигурации сервера. Поэтому для надежности рекомендуется устанавливать уровень изоляции с помощью SQL команды SET TRANSACTION,

READ COMMITTED отличается от других уровней изоляции тем, что SQL-запрос на выборку данных с оператором SELECT без предложения FOR UPDATE/SHARE видит только данные, зафиксированные до начала запроса. По сути, этот запрос SELECT видит моментальный снимок базы данных в момент начала выполнения запроса. Но запрос SELECT также видит результаты предыдущих обновлений, выполненных в рамках его собственной транзакции, даже если они еще не зафиксированы. Две последовательные команды SELECT могут видеть разные данные, даже если они находятся в одной транзакции, если другие транзакции фиксируют изменения после запуска первой команды SELECT и до начала второй команды SELECT.

Такое поведение делает режим READ COMMITTED непригодным для команд со сложными условиями поиска, но подходит для более простых случаев. Приложениям, выполняющим сложные запросы и обновления, может потребоваться более строго согласованное представление базы данных, чем обеспечивает уровень изоляции READ COMMITTED.

Уровень изоляции REPEATABLE READ видит только данные, зафиксированные до начала транзакции и последствия предыдущих обновлений, выполненных в рамках его собственной транзакции, даже если они еще не зафиксированы. Этот уровень изоляции отличается от READ COMMITTED тем, что запрос в транзакции REPEATABLE READ видит моментальный снимок в момент начала первого оператора, а не текущего оператора транзакции. Последовательные команды SELECT в рамках одной транзакции видят одни и те же данные, без изменений, сделанных другими транзакциями, зафиксированными после запуска их собственной.

Приложения с таким уровнем изоляции должны быть готовы к повторным попыткам транзакций из-за сбоев сериализации. Транзакция REPEATABLE READ не может изменять или блокировать строки, измененные другими транзакциями после начала транзакции. Когда приложение получает это сообщение об ошибке, оно должно прервать текущую транзакцию и повторить ее с самого начала. Во второй раз транзакция увидит ранее зафиксированное изменение как часть своего первоначального представления базы данных, поэтому нет логического конфликта при использовании новой версии строки в качестве отправной точки для обновления новой транзакции. Обычно повторная попытка требуется только для транзакций обновления данных, а чтение проходит без конфликтов сериализации.

Таким образом, уровень REPEATABLE READ обеспечивает строгую гарантию того, что каждая транзакция видит полностью стабильное представление базы данных. Но это представление не обязательно всегда будет соответствовать последовательному выполнению параллельных транзакций одного уровня. Поэтому с таким уровнем изоляции часто применяются явные блокировки конфликтующих транзакций.

Уровень SERIALIZABLE не поддерживается Greenplum и аналогичен уровню REPEATABLE READ. В Greenplum модель многоверсионности предотвращает грязное чтение, неповторяемое чтение и фантомное чтение без дорогостоящей блокировки, но не обеспечивает полную сериализуемость транзакций. Такие аномалии часто случаются из-за того, что Greenplum не выполняет блокировку предикатов, т. е. запись в одной транзакции может повлиять на результат предыдущего чтения в другой параллельной транзакции.

Для того, чтобы перейти на уровень изоляции уровню REPEATABLE READ, используется команда SET TRANSACTION ISOLATION LEVEL REPEATABLE READ READ WRITE;

Многоверсионность

Записи в таблицах Greenplum хранятся в соответствии с внутренней системой версионности. Это означает, что строки могут иметь несколько версий. Так, например, при удалении строки из таблицы, она физически остается в ней, однако становится невидима для будущих запросов.

При выполнении операции UPDATE, фактически выполняются две операции, удаление стоки из таблицы и вставка новой версии строки с обновленным значением. Таким образом старая версия физически остается в таблице, но закрывается по периоду действия и появляется новая актуальная версия.

Для определения периодов актуальности строк используются не время изменения строк, а номера транзакций в рамках который вносились эти изменения

Номер транзакции – это уникальный идентификатор, который выдается транзакции в момент ее старта. Этот идентификатор хранится в технических полях xmin и xmax

Очистка

В связи с тем, что строки в результате изменений в таблице физически не удаляется спустя некоторое время таких строк в таблице может накопиться очень много. Особенно если в таблице происходит частое удаление данных. Такие строки не только занимают место на диске, но и увеличивают время чтения данных. Для того чтобы очистить таблицу от неактуальных записей в Greenplum существует 2 операции VACUUM и VACUUM FULL.

Команда VACUUM помечает место, которое занимали удаленные строки, как потенциально свободное. Физически эти строки не удаляются. при следующей вставки новых строк в таблицу они будут вставлены на место неактуальных строк. Если запустить команду VACUUM без параметров, то она будет выявлять неактуальные строки во всех таблицах в базе данных.

Команда VACUUM может выполняться параллельно с командами чтения и записи в таблицу, так как она не требует блокировки на таблицу

Команда VACUUM FULL делает то же самое, что и команда VACUUM, но дополнительно выполняет сжатие таблицы. Таким образом на место строк, которые были помечены как неактуальные физически перемещаются активные строки этой таблицы. В связи с этими при выполнении этой команды размер таблицы физически становится меньше.

Во время выполнения команды VACUUM FULL данные физически перемещаются внутри таблицы, поэтому на время ее выполнения на таблицу накладывается эксклюзивная блокировка. Другие операции с таблицей невозможны пока идет VACUUM FULL.

Примечания:

* Иногда создать копию таблицы быстрее чем проводить VACUUM FULL.
* VACUUM для AO-таблиц ведёт себя как VACUUM FULL, если bloat (раздутие)>10%;
* Bloat - соотношение актуальных строк в таблице к неактуальным.
* Текущий bloat приведён в gp\_toolkit.gp\_bloat\_diag (нужна актуальная статистика)

Блокировки

В СУБД Greenplum мы стремимся увеличить производительность системы за счет максимальной параллелизации процессов и экономии на блокировках. модель многоверсионности позволяет распараллеливать процессы чтения и редактирования данных. Таким образом читающая транзакция никогда не заблокирует пишущую транзакцию и наоборот транзакция, которая меняет строчку, никогда не заблокирует читающую транзакцию. Также читающая транзакция не блокирует другую читающую транзакцию.  
 На первый взгляд единственная ситуация в которых требуется блокировка строк, это ситуация, в которой две транзакции редактируют одну и ту же строку. В этом случае вторая транзакция блокируется и будет ожидать окончания первой. В действительности в базе данных существуют не только блокировки на уровне строк, но блокировки на уровень выше, блокировки таблиц.

Такие блокировки необходимы, когда одна транзакция читает записи в таблице, а другая транзакция пытается удалить эту таблицу. Поэтому даже селект накладывает определенную блокировку на таблицу, для того чтобы не удалили саму таблицу или столбцы в ней в момент считывания данных.

Существую различные уровни и типы блокировок. В таблице на слайде вы видите типы блокировок, при каких операциях они возникают и с какими типами конфликтуют

Перечислим особенности, которые надо учитывать при работе с блокировками Greenplum:

* Выносите операции требующий ACCESS EXCLUSIVE из длительных транзакций/функций в отдельные транзакции. Это связано с тем, что блокировка, вызванная в процессе транзакции сохраняется до её завершения.
* Ожидающие в очереди блокировки тоже конфликтуют с новыми запросами.

При использовании блокировок возможна ситуация взаимоблокировки (или тупика) . Она возникает, когда одна транзакция пытается заблокировать таблицу, которая, уже заблокирована другой транзакцией, в то время как другая транзакция пытается заблокировать таблицу, заблокированную первой. Если возникает такая ситуация, то вторая транзакция принудительно откатывается

Мониторинг текущих запросов

Вся необходимая информация для мониторинга текущих запросов содержится в таблице pg\_stat\_activity.  
Разберем основные поля этой таблицы:

* pid содержит Идентификатор серверного процесса
* state - Состояние серверного процесса  
  в данном поле могут быть следующие значения
* active – выполняет запрос
* idle – ожидает новой команды от пользователя
* idle in transaction – процесс удерживает соединение в рамках начатой транзакции, но уже ничего не делает, возможно в ожидании коммита или роллбэка

План запроса

При написании запроса на языке SQL мы используем декларативный подход. Т. е. мы не указываем способ получения данных, а просто сообщаем какие данные хотим извлечь и что хотим получить. Структура базы данных определяет, где эта информация находится. А план запроса отвечает, как извлечь информацию из базы данных

В Greenplum существует планировщик, который также называют оптимизатором, который перед каждым выполнением запроса строит план, по которому этот запрос будет выполняться.

План запроса представляет собой последовательность физических операций с данными для получения результата запроса

При построении плана оптимизатор опирается на статистику. Статистика — это предварительно собранная информация о кол-ве строк в таблице и характере распределения данных. Подробнее о статистике и методах ее сбора поговорим похоже  
На основе статистики оптимизатор рассчитывает стоимость выполнения каждого шага запроса. Стоимость — это количественная характеристика, связанная с временем выполнения запроса. Так как планов выполнения может быть множество, оптимизатор выбирает план с наименьшей стоимостью

Для построения плана запроса используют команды EXPLAIN и EXPLAIN ANALYZE.  
Разница между ними заключается в том, что EXPLAIN строит план запрос, но не выполняет его. EXPLAIN ANALYZE строит план запроса и выполняет его выводя вместе с планом запроса фактические результаты его исполнения.

Следует учитывать, что EXPLAIN ANALYZE для операций вставки, удаления и изменения данных приведет к изменению таблиц. чтобы откатить изменения сделанные операцией используйте команду rollback пример синтаксиса вы видите на экране.

План запроса нужно читать снизу вверх. На самой нижней строке вы видите какой оптимизатор использовался для построения запроса. В нашем случае это GROPCA.   
Далее следуют шаги выполнения плана. Напротив каждой операции есть метрики которые означают следующее:

* cost- Оценочная стоимость извлечения строк
* rows - ожидаемое количество строк на один сегмент
* width - Размер строки в байтах

Если для этого запроса построить план с помощью команды EXPLAIN ANALYZE, то запрос сначала выполняется, поэтому в плане запроса мы увидим более детальную информацию с фактическими значениями метрик.

в разделе execution time видно сколько по времени выполнялся запрос.  
memory used - показывает сколько памяти потребовалось для его исполнения  
Появилась информация о выполнении частей запроса, slice. О них расскажем чуть позже.  
Также Напротив каждого шага добавились скобки, в которых показываются фактические время извлечения строк, количество строк, количество циклов.

Слайс — это порция плана, которая выполняется на одном сегменте независимо от других сегментов. Таким образом обеспечивая параллельную обработку запросов и повышая производительность.

Процесс выполнения запроса в Greenplum выглядит следующим образом.  
После того, как пользователь вводит запрос, он отправляется на мастер сервер, где происходит синтаксическая проверка запроса и построение плана его выполнения. После этого для того, чтобы выполнить запрос оптимизатор создает специальные процессы на каждом узле кластера. На мастере этот процесс называется Query Dispatcher. Сначала этот процесс доставляет план на сегменты, а после того, как план выполниться на сегментах аккумулирует результат на мастере для последующей передачи клиенту.  
Процесс передачи данных от сегментов мастеру называется gather motion.

На сегментах за выполнение плана отвечают процессы, которые называются query executor. В тексте плана запроса они называются workers.  
В зависимости от плана запроса на сегменте может работать один или несколько процессов query executor.

Каждый шаг плана будет обрабатываться своим query executor. Преимущество этого подхода заключается в том, что такие порции плана выполняются на каждом сегменте параллельно таким образом ускоряя выполнение всего запроса в целом

Операторы соединения.  
В Greenplum существует три метода соединения.

1. Соединение хэширование
2. Соединение вложенными циклами
3. Соединение слиянием

Соединение хэшированием — это метод соединения, в котором для меньшей по объёму таблице сначала строится хэш таблица. В качестве хэш ключа используется поле по которому идет соединение.

Далее выполняется сканирование полей по объёму таблицы с последовательным вычислением хэш ключа соединения, если хэш ключ совпадает, то строки объединяются  
HASH JOIN является наиболее быстрым и часто используемым типом соединения.  
Для того, чтобы он использовался в условие соединения должно стоять равенство и отсутствовать какие-либо преобразования полей. Оптимизатор в первую очередь стремится использовать именно этот метод соединения.

Если не удаётся применить HASH JOIN, то применяется NESTED LOOP.

Этот тип соединения выполняется с помощью вложенных циклов. Для каждой записи из большой таблицы последовательно сканируется меньшая таблица и ищутся совпадающие по ключу строки. Данный тип соединения является самым быстрым для очень маленьких объёмов. Обычно возникает, когда в соединении есть условие или когда условие отсутствует

Третий способ соединения — это соединение слиянием. Сначала в обоих таблицах сортируются все строки, а затем соединяются.  
Сортировка очень дорогая операция в Greenplum. Поэтому этот способ соединения хорошо работает, только для отсортированных данных. Но на практике такого тяжело добиться так как данные в Greenplum хранятся в несортированном виде

Разберем операторы перемещения данных.

Необходимость перемещения обусловлена распределенной архитектурой Greenplum. Данные располагаются на разных сегментах и в некоторых случаях их необходимо перемещать между сегментами или между мастером и сегментами. Например, для того, чтобы соединить таблицы, распределенные по разным ключам дистрибуции или для выполнения промежуточных расчетов и агрегаций  
Существует 3 типа перемещения данных

1. BROADCAST MOTION
2. REDISTRIBUTE MOTION
3. GATHER MOTION

BROADCAST MOTION — это тип перемещения в результате которому каждый сегмент отправляет всем остальным свои данные в результате такого перемещения каждый сегмент будет иметь свою копию таблицы. Обычно используется когда необходимо соединить небольшой справочник с большой таблицей фактов.

Таким образом маленькая таблица сначала целиком копируется на все сегменты, а затем локально соединяется со второй таблицей.

Оптимизатор может по ошибке применить этот тип перемещения для двух больших таблиц из-за неактуальной статистики. В этом случае запрос будет или очень долго выполняться или не выполниться вовсе из-за нехватки ресурсов.

REDISTRIBUTED MOTION — Это тип перемещения, в котором строки динамически перераспределяются по сегментам в соответствии с новым ключом дистрибуции. Возникает, когда мы соединяем 2 таблицы с разным ключом дистрибуции. Для того чтобы соединить строки оптимизатор сначала выполняет распределение строк первой таблицы согласно ключу дистрибуции второй таблицы, а затем выполняет соединение строк локально на сегментах.

Также перераспределение возникает, когда ключ соединения отличается от ключа дистрибуции таблицы или выполняется агрегация по полю отличному от ключа дистрибуции.

При таком типе перемещения есть риск возникновения перекоса вычисления skew. Когда в ходе выполнения запроса на каком-либо сегменте оказывается намного больше данных чем на других сегментах. Это может замедлить работу не только запроса, но и всей системы в целом.

Последний тип перемещения это GATHER MOTION. Он объединяет результаты выполнения запроса со всех сегментов на мастере. Является финальной операцией для большинства запросов

На какие моменты в плане запроса необходимо обращать внимание. Так называемые узкие места.

* Проверяйте насколько корректно оценил кол-во строк в таблице оптимизатор. В случае большого расхождения с действительностью необходимо произвести сбор статистики.
* Проверьте отсутствие перекосов вычисления. Для этого сравните максимальное и среднее значение используемой памяти. Если максимальное значение существенно выше среднего, то это означает наличие перекоса вычислений
* Убедитесь в использовании выборочного сканирования партиций. Если таблица партиционирована и в фильтре запроса стоит ключ партиционирования, то в плане должны быть просканированы только нужные таблицы.

Заключительный момент, про который надо помнить при анализе плана запроса - это использования spil файлов. В Greenplum каждому запросу в момент его старта выделяется определенный объем оперативной памяти, который не меняется в момент его выполнения. Если в ходе выполнения запроса на каком-либо сегменте не хватает оперативной памяти, то данные размещаются на диске в виде spil файлов. Обработка данных на диске выполняется медленнее чем в оперативной памяти, крооме того может быть превышен заданный лимит для spil файлов, что приведет к падению запроса

Для того чтобы избежать превышения оперативной памяти используют следующие методы.

* Изменение запроса или разбиение его на маленькие шаги.
* Изменяют способ хранения таблиц.
* Перераспределяют ключ дистрибуции или устанавливают колоночный формат хранение данных. При нем будут считываться только нужные колонки, которые без проблем поместятся в памяти.

Оптимизаторы

Greenplum может использовать один из двух оптимизаторов

GPORCA - Является более продвинутым оптимизатором. Используется по умолчанию,  
Создан специально для Greenplum и постоянно развивается  
Имеет ряд улучшений при работе с:

* Партиционированными таблицами
* С CTE
* С подзапросами
* С DML операциями

LEGACY оптимизатор является наследием СУБД PostgreSQL. То есть это планировщик Postgres, но с определенными доработками, учитывающими MPP архитектуру.  
Используется при явно заданной конфигурации или в случае fallback. В отличие от GROPCA поддерживает все SQL конструкции и функции  
Иногда строит более оптимальный план чем GROPCA

После того как запрос пользователя прошел синтаксический разбор выполняется проверка конфигурационного параметра optimizer. По умолчанию он равен on. Это означает, что для построения плана запроса будет использоваться GPORCA.  
Если в optimizer установлено значение off, то для построения плана будет использован legacy оптимизатор.

Также легаси оптимизатор будет использован если при установленном значении on в запросе пользоватея будут использоваться функцию, которые не поддерживает gporca. В таком случае произойдет принудительное переключение на легаси планировщик так называемый fallback.

Для определения текущего параметра optimizer используется команда SHOW, а для установки значения команда SET

Статистика

Детальная информация о данных в таблице. Хранится в системном каталоге. Может быть собрана вручную и автоматически. Должна быть актуальной для построения оптимального плана запроса  
  
Необходимо собирать после:

* Загрузки данных
* Операции CREATE INDEX
* Массовых операций INSERT, UPDATE, DELETE

Статистика собирается с помощью команды ANALYZE.

1. Если применить ее без параметров, то статистика собирается для всех таблиц в Базе данных.
2. Если указать имя таблицы, то статистика будет собрана только по этой таблице.
3. Также можно собрать статистику для определенных колонок.

В Greenplum есть возможность сбора статистики по конкретным партициям таблицы. Для этого после команды ANALYZE указывается имя партиции.

Поэтому следует учитывать, что для партиционированных таблиц существует два вида статистики. Статистика собранная на уровне родительской таблицы и статистика собранная на уровне партиции самого нижнего уровня

* GROPCA использует статистику родительской таблицы
* LEGACY использует статистику партиций нижнего уровня

Поэтому важно иметь актуальную статистику для построения плана тем или иным оптимизатором.

В Greenplum за автоматический сбор статистики отвечают 2 конфигурационных параметра gp\_autostats\_ mode и gp\_autostats\_threshold.  
Если параметр gp\_autostats\_mode выставлено значение on\_no\_stats , что является значением по умолчанию. то сбор статистики будет автоматически выполняться после операций CREATE TABLE AS SELECT, INSERT или COPY для таблиц у которых раньше не было статистики.

Источники информации (ссылки)

1. [Управление распределенными транзакциями](https://www.bigdataschool.ru/blog/distributed-transactions-and-isolation-levels-in-greenplum.html)
2. [Анализируй и оптимизируй: статистика таблиц и планы выполнения SQL-запросов в Greenplum](https://www.bigdataschool.ru/blog/operator-analyze-for-sql-queries-in-greenplum.html)
3. [Документация: SELECT](https://postgrespro.ru/docs/postgrespro/13/sql-select)
4. [Документация: INSERT](https://postgrespro.ru/docs/postgrespro/13/sql-insert)
5. [Документация: DELETE](https://postgrespro.ru/docs/postgrespro/13/sql-delete)
6. [Документация: UPDATE](https://postgrespro.ru/docs/postgrespro/13/sql-update)