ARENADATA

Корпоративная платформа хранения
и обработки больших данных

Arenadata DB для Разработчиков

Часть 3

ADB – аналитическая СУБД для больших данных

Выполнение запросов:
Оптимизаторы.
Планы запросов.
Статистика оптимизатора.
Индексы в ADB.

Транзакции и блокировки:
ACID.
MVCC.
Ваккуумирование таблиц.

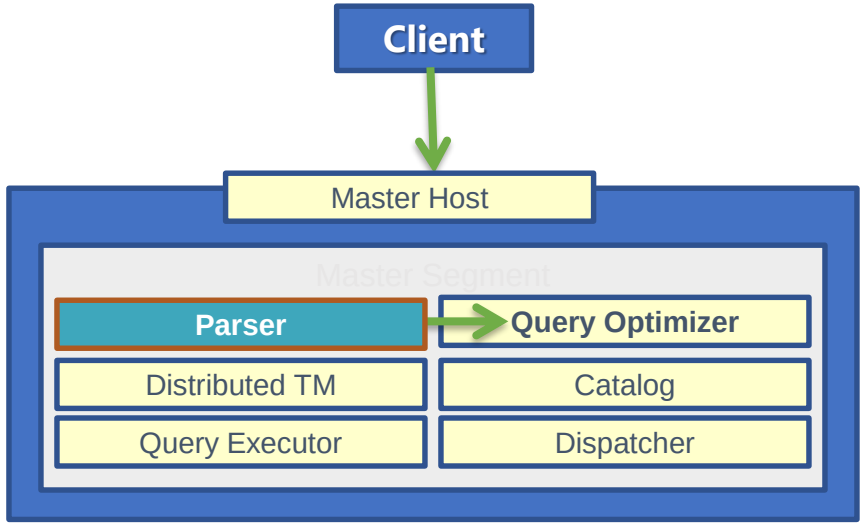
ARENADATA

Корпоративная платформа хранения
и обработки больших данных

Выполнение запроса

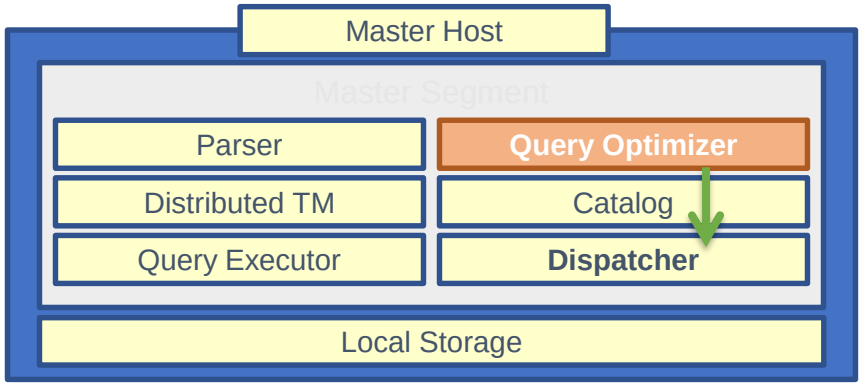
Начало выполнение запроса

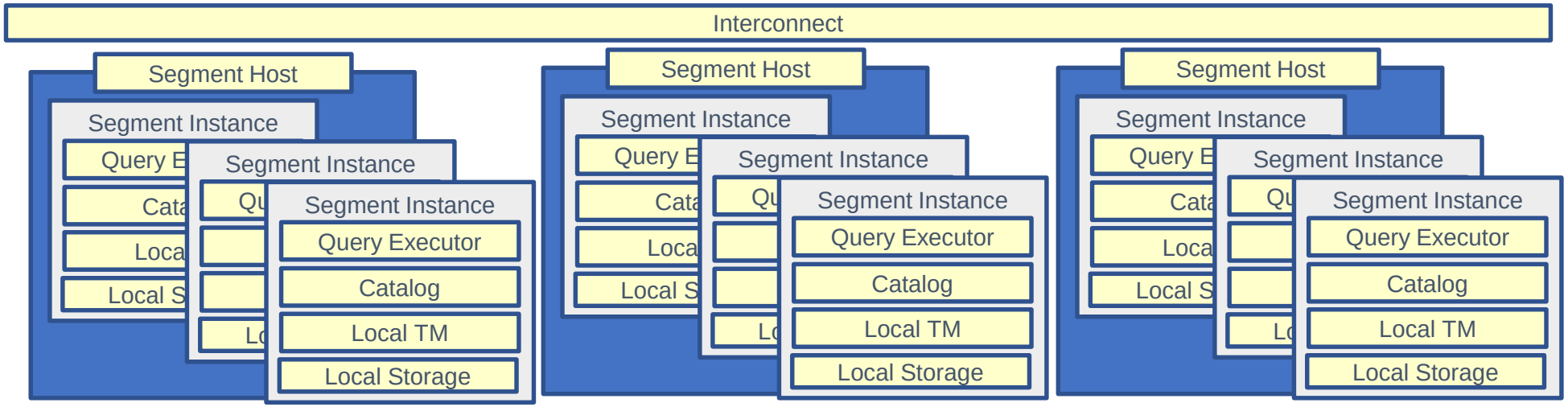
- Запросы направляются клиентом мастер-сегменту.
- PostgreSQL listener на мастере принимает входящие подключения (стандартный порт – 5432).
- Парсер (Query Parser) проверяет синтаксис, семантику и генерирует дерево запроса для оптимизатора (Query Optimizer).
- Мастер *может* выполнять некоторые финальные операции с данными – агрегации, сортировки и т.д., но основная работа происходит на праймари-сегментах сегментных серверов.
- Запросы пишутся на декларативном языке SQL, который предполагает, что пользователь описывает не процесс выполнения, а желаемый результат, поэтому после разбора запроса парсером управление передаётся оптимизатору на мастер-сервере.



Работа оптимизатора

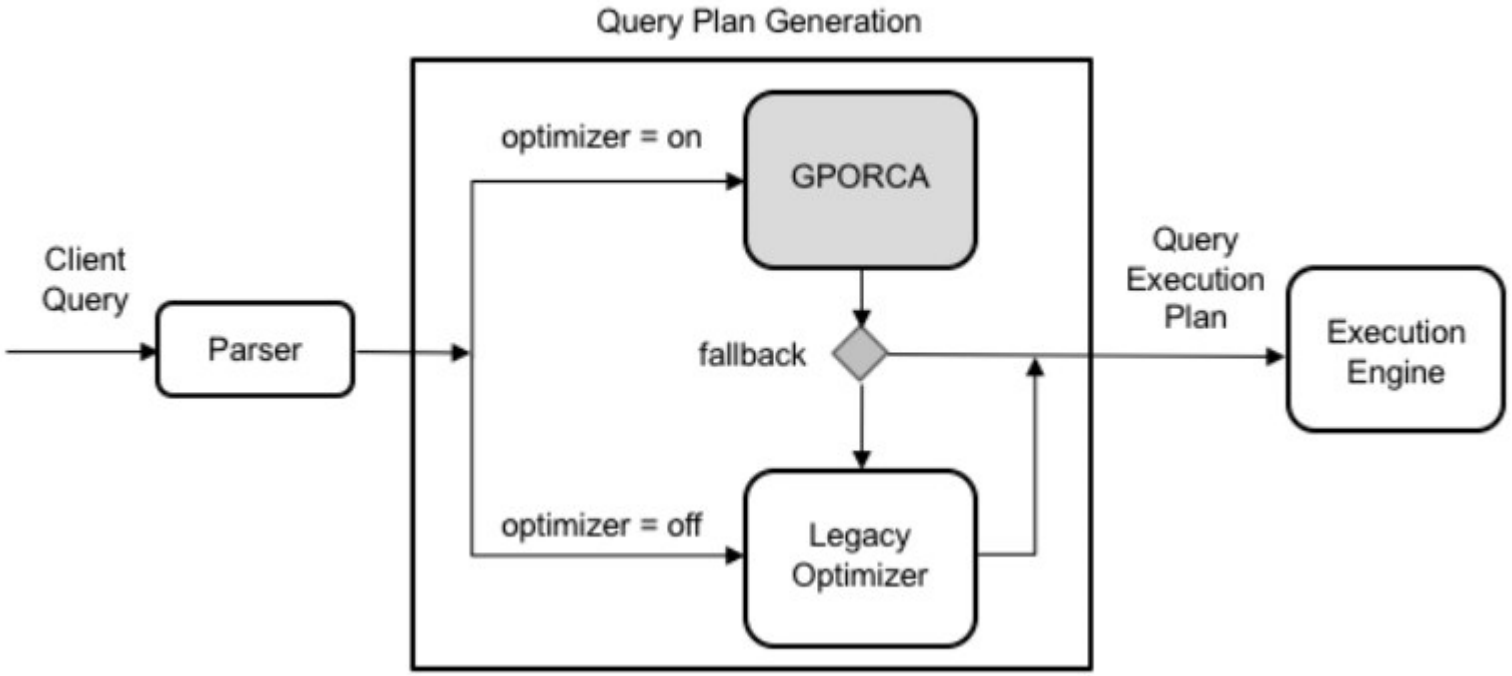
- В системе есть два оптимизатора: GPORCA и Postgres, они генерируют планы.
- Оптимизатор GPORCA был специально разработан для GP и формирует более эффективные планы запросов в большинстве случаев.
- PostgreSQL query optimizer умеет обрабатывать случаи, с которыми не справляется GPORCA, переключение при этом происходит автоматически.
- Каждый план имеет стоимость, план с меньшей стоимостью передаётся диспетчеру.





ARENADATA

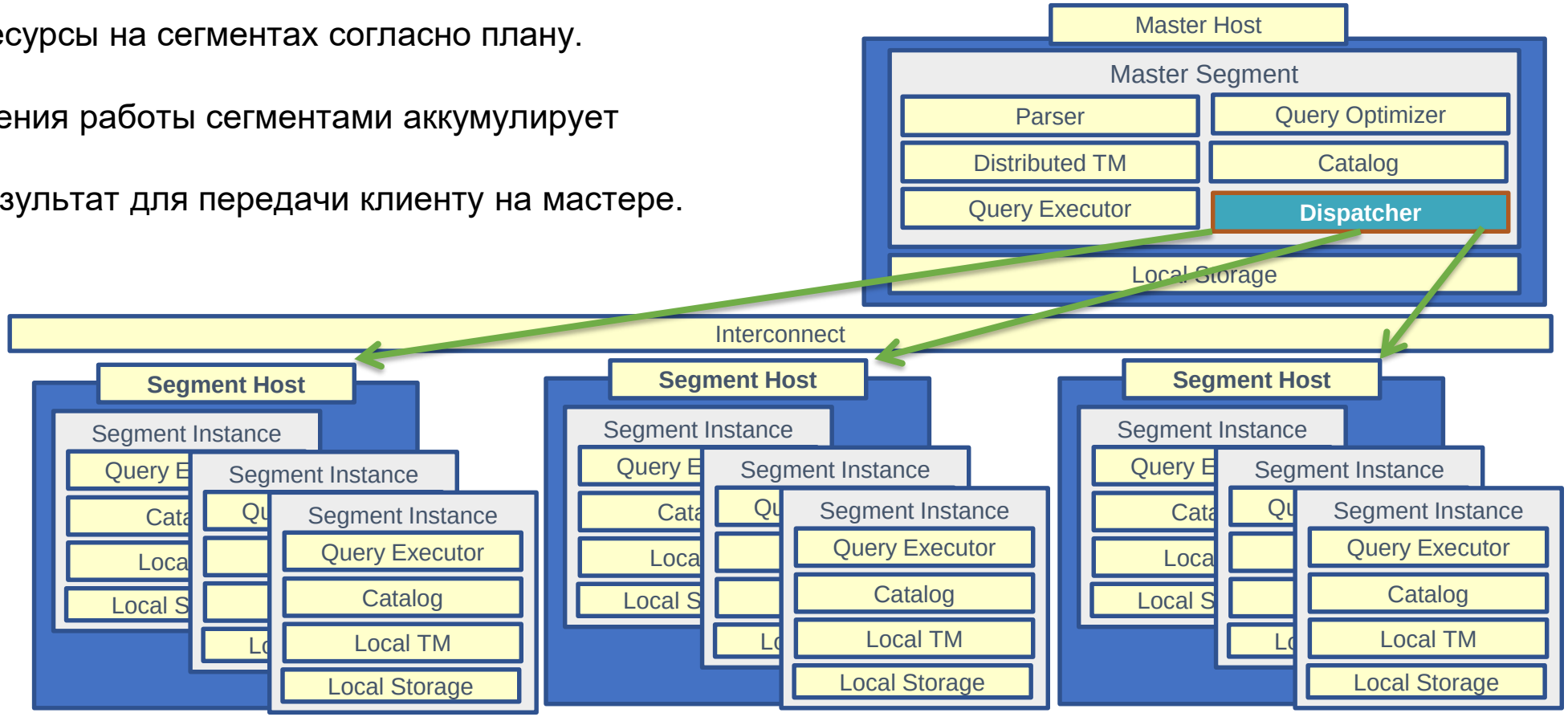
Выбор оптимизатора системой



Copyright arenadata.io

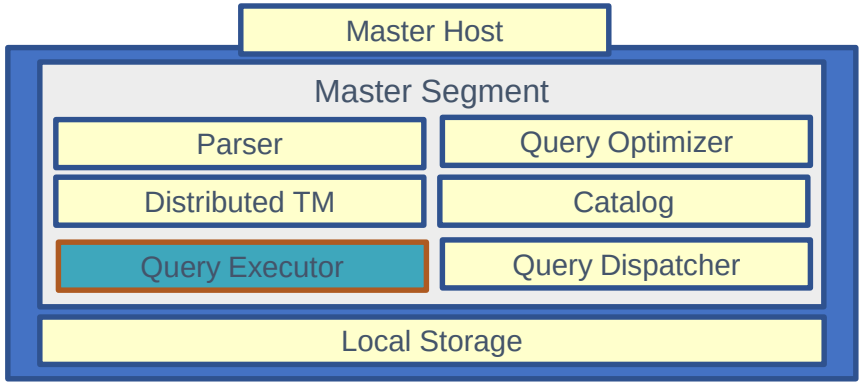
Передача плана сегментам

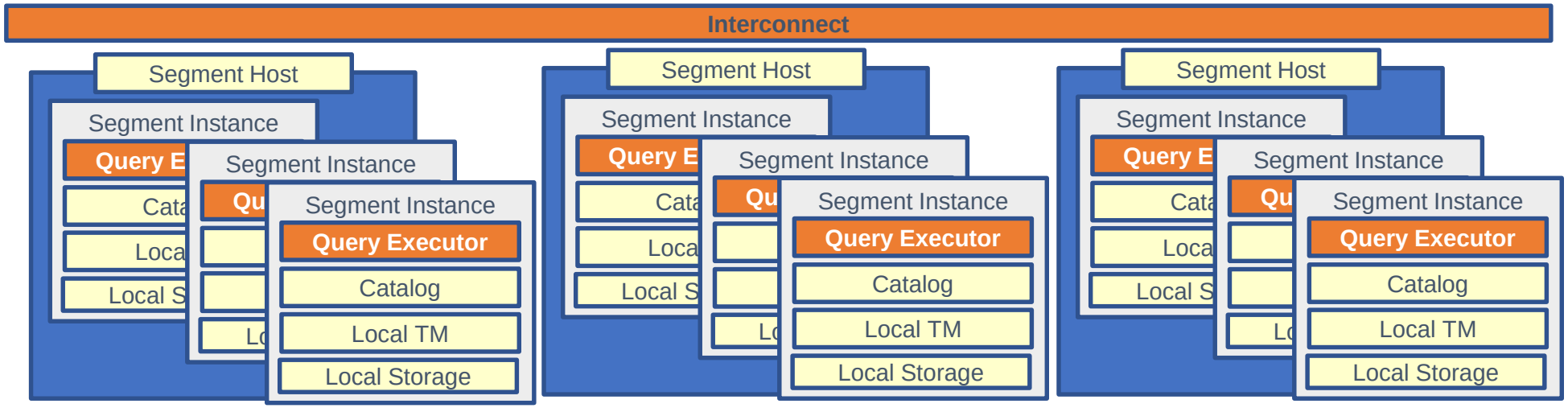
- Диспетчер передаёт план запроса сегментам.
- Аллоцирует ресурсы на сегментах согласно плану.
- После выполнения работы сегментами аккумулирует финальный результат для передачи клиенту на мастере.



Работа на сегментах

- Шаги плана на сегментах выполняют исполнители запросов (Query Executor).
- В ходе выполнения исполнители могут обмениваться промежуточными результатами с исполнителями на других сегментах.
- Каждый исполнитель запускает для запроса несколько процессов (Worker).
- Результаты выполнения запросов на сегментах передаются на мастер, где происходит финальная сборка и выдача результата.





Спилл-файлы

- В случае, если при выполнении запроса СУБД не хватает памяти для хранения временных данных (хеши джойнов, перераспределённые таблицы и т.д.), СУБД скидывает часть данных в спилл-файлы;
- Спилл-файлы располагаются в директории pgsql_tmp;
- Локацию спилл-файлов можно переносить в другие директории с помощью переменной temp_tablespaces;
- Используйте представления:
  - **gp_workfile_entries**
  - **gp_workfile_usage_per_query**
  - **gp_workfile_usage_per_segment**
- Также существуют несколько параметров для управления спилл-файлами:
  - **gp_workfile_checksumming** – считать ли контрольную сумму временных файлов;
  - **gp_workfile_compress_algorithm** – сжимать ли временные файлы;
  - **gp_workfile_limit_files_per_query** – ограничить количество временных файлов на запрос;
  - **gp_workfile_limit_per_query** – ограничить объём временных данных на запрос;
  - **gp_workfile_limit_per_segment** – ограничить объём временных данных на сегмент. Обязательно установите этот параметр в разумное значение!

Copyright arenadata.io

Запросы в ADB: оптимизаторы

По умолчанию используется оптимизатор GPORCA, можно переключить на Postgres, например, на время сессии: `set optimizer = off;`

Как правило, GPORCA показывает лучшие результаты в следующих случаях:

- Запросы, которые содержат CTE, в т.ч. вложенные.
- Запросы, которые содержат подзапросы:
  - CSQ в инструкции SELECT.
  - CSQ, которые могут обращаться к таблице из внешнего запроса, пропуская промежуточный подзапрос (skip-level correlation references).
  - CSQ содержащие логическое условие OR.
  - CSQ с агрегатами и неравенством.
- Запросы к партиционированным таблицам.

Пример CSQ и план запроса

```
SELECT *,
      (SELECT min(price) FROM products p2 WHERE p1.group_name = p2.group_name) AS min_price
FROM products p1 ORDER BY group_name;
```

```
Gather Motion 4:1  (slice3; segments: 4)  (cost=0.00..862.00 rows=12 width=36) (actual time=3.201..3.546 rows=12 loops=1)
  ->  Result  (cost=0.00..862.00 rows=3 width=36) (actual time=2.435..2.763 rows=8 loops=1)
        ->  Hash Left Join  (cost=0.00..862.00 rows=4 width=36) (actual time=2.428..2.755 rows=8 loops=1)
              Hash Cond: ((products.group_name)::text = (products_1.group_name)::text)
              Extra Text: (seg0)   Hash chain length 1.0 avg, 1 max, using 2 of 262144 buckets.Hash chain length 1.0 avg, 1 max, using 2 of 32 buckets; total 0 expansions.

              ->  Redistribute Motion 4:4  (slice1; segments: 4)  (cost=0.00..431.00 rows=3 width=28) (actual time=0.013..0.016 rows=8 loops=1)
                    Hash Key: products.group_name
                    ->  Seq Scan on products  (cost=0.00..431.00 rows=3 width=28) (actual time=0.008..0.008 rows=4 loops=1)
              ->  Hash  (cost=431.00..431.00 rows=2 width=16) (actual time=0.830..0.830 rows=2 loops=1)
                    ->  HashAggregate  (cost=0.00..431.00 rows=2 width=16) (actual time=0.814..0.825 rows=2 loops=1)
                          Group Key: products_1.group_name
                          Extra Text: (seg0)   Hash chain length 1.0 avg, 1 max, using 2 of 32 buckets; total 0 expansions.

                          ->  Redistribute Motion 4:4  (slice2; segments: 4)  (cost=0.00..431.00 rows=3 width=13) (actual time=0.012..0.751 rows=8 loops=1)
                                Hash Key: products_1.group_name
                                ->  Seq Scan on products products_1  (cost=0.00..431.00 rows=3 width=13) (actual time=0.009..0.010 rows=4 loops=1)
Planning time: 6.571 ms
  (slice0)    Executor memory: 135K bytes.
  (slice1)    Executor memory: 60K bytes avg x 4 workers, 60K bytes max (seg0).
  (slice2)    Executor memory: 60K bytes avg x 4 workers, 60K bytes max (seg0).
  (slice3)    Executor memory: 2188K bytes avg x 4 workers, 2232K bytes max (seg0).  Work_mem: 1K bytes max.
Memory used:  128000kB
Optimizer: Pivotal Optimizer (GPORCA) version 3.88.0
Execution time: 4.762 ms
```

Copyright arenadata.io

ARENADATA

Корпоративная платформа хранения
и обработки больших данных

План запроса

План запроса

- План запроса – набор отдельных операций, необходимых для получения описанного пользователем в запросе результата.
- План не является линейным списком, а представляется в виде дерева, узлы которого являются операциями.
- Есть две команды для вывода планов: EXPLAIN и EXPLAIN ANALYZE. Они указываются перед запросом, например:

```
EXPLAIN SELECT * FROM TABLE table2 WHERE id < 101;
```

- EXPLAIN возвращает план запроса, построенный на основе статистики об объектах в запросе, *не выполняя сам запрос*.
- EXPLAIN ANALYZE возвращает аналогичный план запроса, добавляя статистику его выполнения, *выполняя запрос*.
- Каждый запрос и его шаг имеют стоимость (cost). В ADB это условная величина, которая не позволяет напрямую связать её со временем выполнения. Связь обеспечивается с помощью тонких настроек (для каждого оптимизатора они свои):

Postgres Planner

- **seq_page_cost** cost of a sequentially fetched disk page **= 1 (time a sequential 8kb block read takes)**.
- **random_page_cost** cost of a nonsequentially fetched disk page.
- **cpu_tuple_cost** cost of processing each tuple (row).
- **cpu_operator_cost** cost of processing each operator or function call.
- **cpu_index_tuple_cost** cost of processing each index entry during an index scan.

GPORCA

- **optimizer_sort_factor** controls the cost factor to apply to sorting operations

Обычно их изменять не потребуется.

- Разные планировщики генерируют разные планы для одних и тех же запросов.

Вывод команды EXPLAIN

План – это дерево в классическом математическом представлении, поэтому чтение плана выполняется снизу вверх.

Пример плана, который выводится командой EXPLAIN:

```
adb=# EXPLAIN SELECT * FROM table2 where id < 101;


                                  QUERY PLAN
-------------------------------------------------------------------------------
 Gather Motion 4:1  (slice1; segments: 4)  (cost=0.00..455.07 rows=200000 width=30)
   ->  Seq Scan on table2  (cost=0.00..432.21 rows=50000 width=30)
         Filter: (id1 < 101)
Optimizer: Pivotal Optimizer (GPORCA) version 3.80.0
```

Вывод команды EXPLAIN

```
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Gather Motion 4:1  (slice1; segments: 4)  (cost=0.00..455.07 rows=200000 width=30)
   ->  Seq Scan on table2  (cost=0.00..432.21 rows=50000 width=30)
         Filter: (id1 < 101)
Optimizer: Pivotal Optimizer (GPORCA) version 3.80.0
```

- Первая строка снизу указывает на то, какой оптимизатор использовался при построении плана.
- В данном случае использовался родной оптимизатор GPORCA.

Вывод команды EXPLAIN

```
Gather Motion 4:1  (slice1; segments: 4)  (cost=0.00..455.07 rows=200000 width=30)
   ->  Seq Scan on table2  (cost=0.00..432.21 rows=50000 width=30)
         Filter: (id1 < 101)
Optimizer: Pivotal Optimizer (GPORCA) version 3.80.0
```

- Вторая строка снизу – узел операции простого сканирования файла таблицы (выборка данных).
- Для узлов приводятся:

1. Название операции.
2. Объект.
3. Условия (в данном примере filter для условия WHERE).
4. Параметры узла в виде набора значений в круглых скобках.

Уровень узла в дереве определяется отступом (табуляцией), а не порядком следования в выводе плана.

Вывод команды EXPLAIN

```
->  Seq Scan on table2  (cost=0.00..432.21 rows=50000 width=30)
```

Параметры узла содержат три значения:

1. **Cost** – два числа, первое из которых показывает *оценочную* стоимость запуска операции, а второе – полную стоимость её выполнения.
2. **Rows** – *расчётное* число строк, которое будет обработано в запросе на каждом сегменте (исключение – операция Gather Motion, в которой указывается общее расчётное число на весь кластер).
3. **Width** – *расчётная* ширина обработанных строк в байтах.

Вывод команды EXPLAIN

```
Gather Motion 4:1  (slice1; segments: 4)  (cost=0.00..455.07 rows=200000 width=30)
   ->  Seq Scan on table2  (cost=0.00..432.21 rows=50000 width=30)
         Filter: (id1 < 101)
Optimizer: Pivotal Optimizer (GPORCA) version 3.80.0
```

- Следующим узлом является операция Gather Motion. В подавляющем большинстве случаев это означает сбор данных со всех сегментов на мастер-сервер.
- Блок параметров у данной операции идентичен, за исключением того, что в параметре rows указано общее расчётное число строк, которое будет собрано со всех сегментов. В данном случае 50000 строк с четырёх сегментов дают нам в сумме 200000 строк, отправленных на мастер.

Вывод команды EXPLAIN

```
Gather Motion 4:1  (slice1; segments: 4)  (cost=0.00..455.07 rows=200000 width=30)
   ->  Seq Scan on table2  (cost=0.00..432.21 rows=50000 width=30)
         Filter: (id1 < 101)
Optimizer: Pivotal Optimizer (GPORCA) version 3.80.0
```

- Дополнительный блок в круглых скобках – это не параметр операции Gather Motion.
- Слайс (slice) – это отдельный путь выполнения в дереве плана.
- Для слайса указывается число сегментов, на которых выполняются узлы, входящие в него.
- К слайсу относятся операции, которые указаны после него (необходимо смотреть на вложенность).
- В данном случае в слайсе 1 находится операция Seq Scan on table2.
- Существует слайс 0, который выполняется на мастер-сервере. В выводе Explain при отсутствии дополнительных работ на мастере он не приводится, но указывается в выводе EXPLAIN ANALYZE.

Вывод команды EXPLAIN ANALYZE

EXPLAIN ANALYZE выводит тот же план, но запрос выполняется, поэтому в плане добавляются данные о том, как прошел процесс его выполнения.

```
adb=# EXPLAIN ANALYZE SELECT * FROM table2 where id<101;

                                                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
 Gather Motion 4:1  (slice1; segments: 4)  (cost=0.00..455.07 rows=200000 width=30) (actual time=3.634..74.391
rows=200000 loops=1)
   -> Seq Scan on table2(cost=0.00..432.21 rows=50000 width=30)(actual time=0.814..13.545 rows=50093 loops=1)
         Filter: (id1 < 101)
Planning time: 1.637 ms
   (slice0)    Executor memory: 520K bytes.
   (slice1)    Executor memory: 644K bytes avg x 4 workers, 644K bytes max (seg0).
 Memory used:  128000kB
 Optimizer: Pivotal Optimizer (GPORCA) version 3.80.0
 Execution time: 100.478 ms
```

Вывод команды EXPLAIN ANALYZE

```
Gather Motion 4:1  (slice1; segments: 4)  (cost=0.00..455.07 rows=200000 width=30) (actual time=3.634..74.391
rows=200000 loops=1)
   -> Seq Scan on table2(cost=0.00..432.21 rows=50000 width=30)(actual time=0.814..13.545 rows=50093 loops=1)
         Filter: (id1 < 101)
Planning time: 1.637 ms
   (slice0)    Executor memory: 520K bytes.
   (slice1)    Executor memory: 644K bytes avg x 4 workers, 644K bytes max (seg0).
 Memory used:  128000kB
 Optimizer: Pivotal Optimizer (GPORCA) version 3.80.0
 Execution time: 100.478 ms
```

- Для выполненного запроса приводится время его полного выполнения (Execution time) и время, за которое оптимизатор построил план (Planning time) в миллисекундах.

Вывод команды EXPLAIN ANALYZE

```
Gather Motion 4:1  (slice1; segments: 4)  (cost=0.00..455.07 rows=200000 width=30) (actual time=3.634..74.391
rows=200000 loops=1)
   -> Seq Scan on table2(cost=0.00..432.21 rows=50000 width=30)(actual time=0.814..13.545 rows=50093 loops=1)
         Filter: (id1 < 101)
Planning time: 1.637 ms
   (slice0)    Executor memory: 520K bytes.
   (slice1)    Executor memory: 644K bytes avg x 4 workers, 644K bytes max (seg0).
 Memory used:  128000kB
 Optimizer: Pivotal Optimizer (GPORCA) version 3.80.0
 Execution time: 100.478 ms
```

- Параметр Memory Used показывает, сколько памяти было выделено запросу на основании настроек системы и параметров ресурсной группы пользователя, запустившего запрос.
- Это не потребленная запросом память. Запрос может потребить как меньше памяти, так и больше.
- При превышении данного параметра рядом появится запись Memory Wanted.
- Если памяти потребовалось больше, может быть выделена дополнительная память или же запрос начнёт писать свои данные на диск во временные файлы (спилл-файлы).

Вывод команды EXPLAIN ANALYZE

```
Gather Motion 4:1  (slice1; segments: 4)  (cost=0.00..455.07 rows=200000 width=30) (actual time=3.634..74.391
rows=200000 loops=1)
   -> Seq Scan on table2(cost=0.00..432.21 rows=50000 width=30)(actual time=0.814..13.545 rows=50093 loops=1)
         Filter: (id1 < 101)
Planning time: 1.637 ms
   (slice0)    Executor memory: 520K bytes.
   (slice1)    Executor memory: 644K bytes avg x 4 workers, 644K bytes max (seg0).
 Memory used:  128000kB
 Optimizer: Pivotal Optimizer (GPORCA) version 3.80.0
 Execution time: 100.478 ms
```

- Также приводятся параметры потребления памяти для каждого слайса в запросе.
- Для слайсов с номерами 1 и более, выполняющихся на сегментных серверах, указывается два параметра: среднее потребление и максимальное (с указанием сегмента, который потратил памяти больше всего).
- Для слайса 0 приводятся данные о потреблении памяти на мастере.

Вывод команды EXPLAIN ANALYZE

```
Gather Motion 4:1  (slice1; segments: 4)  (cost=0.00..455.07 rows=200000 width=30) (actual time=3.634..74.391
rows=200000 loops=1)
   -> Seq Scan on table2(cost=0.00..432.21 rows=50000 width=30)(actual time=0.814..13.545 rows=50093 loops=1)
         Filter: (id1 < 101)
Planning time: 1.637 ms
   (slice0)    Executor memory: 520K bytes.
   (slice1)    Executor memory: 644K bytes avg x 4 workers, 644K bytes max (seg0).
 Memory used:  128000kB
 Optimizer: Pivotal Optimizer (GPORCA) version 3.80.0
 Execution time: 100.478 ms
```

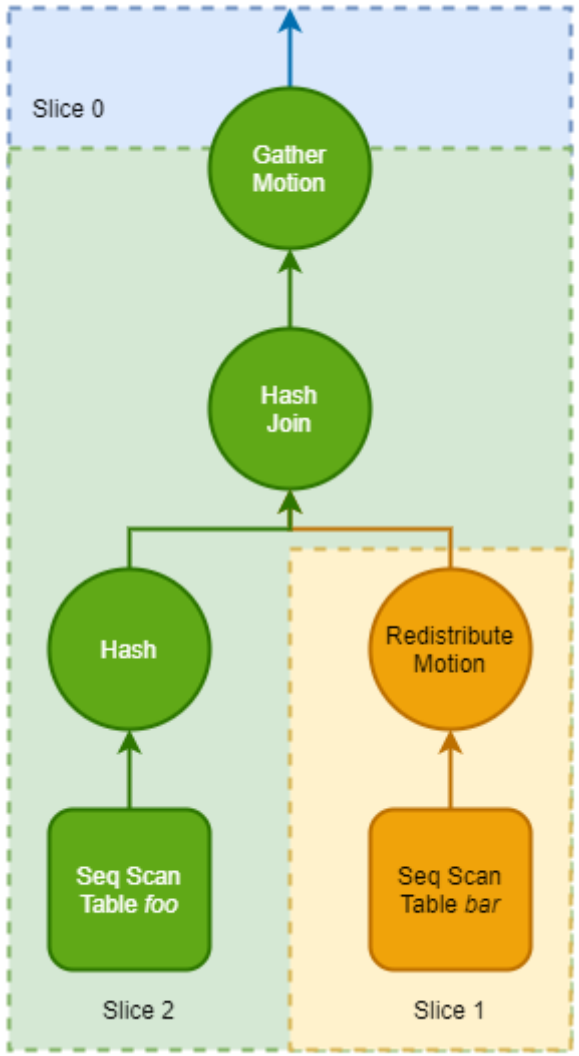
- У каждого узла в дереве плана появляется дополнительный блок с тремя параметрами, которые отражают реальную статистику выполнения узла *для самого загруженного сегмента*:

1. **Actual time** – приводится время начала выполнения и полного выполнения узла в миллисекундах.
2. **Rows** – количество получившихся при работе узла строк.
3. **Loops** – число повторений узла (иногда узел выполняется более одного раза).

Слайсы

```
adb=# EXPLAIN ANALYZE SELECT * FROM foo f join bar b on f.a=b.a;

                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
 Gather Motion 4:1  (slice2; segments: 4)  (cost=0.00..1921.12 rows=1512926 width=74) (actual time=84.825..2006.945 rows=1202000 loops=1)
   ->  Hash Join  (cost=0.00..1546.62 rows=378232 width=74) (actual time=89.113..1436.769 rows=301256 loops=1)
         Hash Cond: (bar.a = foo.a)
         Extra Text: (seg2)   Hash chain length 1.6 avg, 8 max, using 161453 of 262144 buckets.
         ->  Redistribute Motion 4:4  (slice1; segments: 4)  (cost=0.00..601.14 rows=800000 width=37) (actual time=0.065..967.172 rows=801937)
               Hash Key: bar.a
               ->  Seq Scan on bar  (cost=0.00..453.44 rows=800000 width=37) (actual time=0.342..352.239 rows=3000000 loops=1)
         ->  Hash  (cost=438.02..438.02 rows=250250 width=37) (actual time=85.309..85.309 rows=250911 loops=1)
               ->  Seq Scan on foo  (cost=0.00..438.02 rows=250250 width=37) (actual time=0.158..25.031 rows=250911 loops=1)
 Planning time: 7.773 ms
   (slice0)    Executor memory: 119K bytes.
   (slice1)    Executor memory: 196K bytes avg x 4 workers, 220K bytes max (seg0).
   (slice2)    Executor memory: 26888K bytes avg x 4 workers, 26888K bytes max (seg0).  Work_mem: 15682K bytes max.
 Memory used:  128000kB
 Optimizer: Pivotal Optimizer (GPORCA) version 3.86.0
 Execution time: 2163.160 ms
```

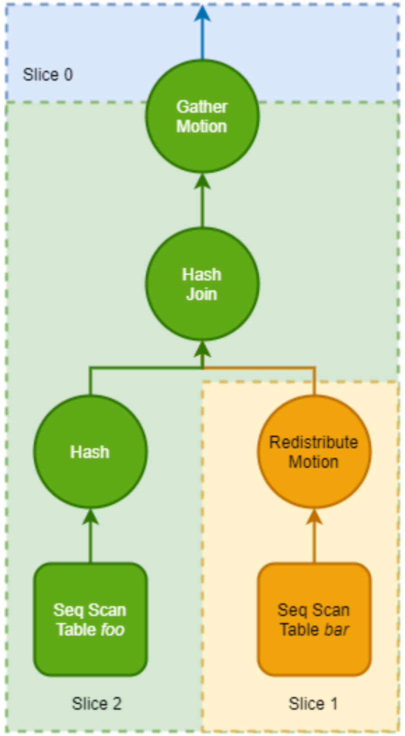
Слайсы



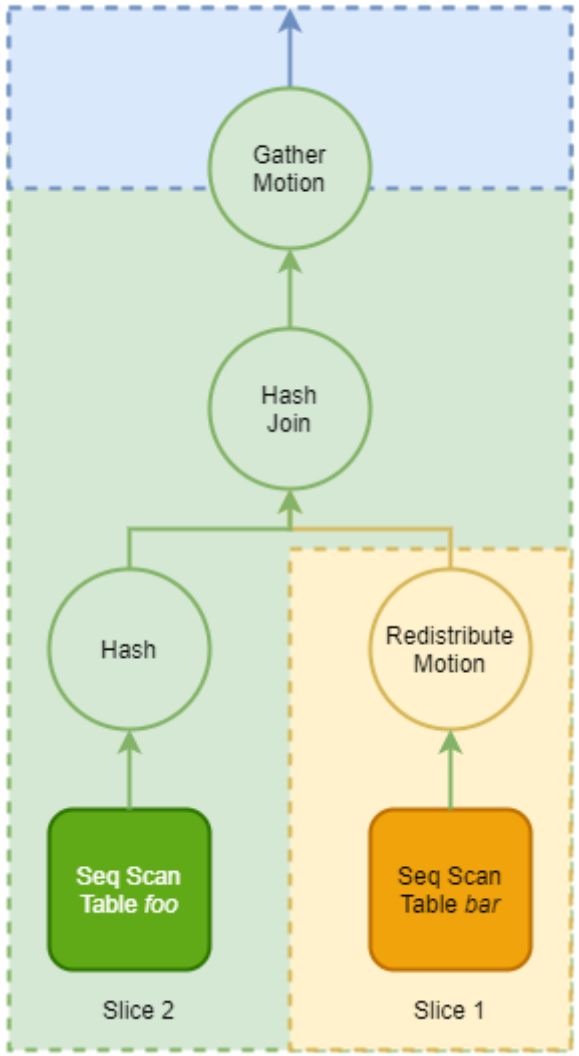
- В графическом виде план данного запроса можно представить в виде дерева.
- В дереве присутствуют все шесть операций плана.
- Некоторые операции могут выполняться независимо, поэтому они помещаются в разные ветви дерева.
- Отдельный параллельный путь выполнения в дереве называется слайсом.
- Существуют операции, которым нужны данные из разных слайсов. В таких узлах происходит слияние, остаётся один из слайсов, в рамках которого продолжается выполнение.
- В нашем случае независимыми операциями являются выборки данных из двух разных таблиц, а также построение хешей по одной и перераспределение другой.
- Слияние происходит в операции соединения таблиц.

Слайсы

```
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Gather Motion 4:1  (slice2; segments: 4)  (cost=0.00..1921.12 rows=1512926 width=74) (actual time=84.825..2006.945 rows=1202000 loops=1)
   ->  Hash Join  (cost=0.00..1546.62 rows=378232 width=74) (actual time=89.113..1436.769 rows=301256 loops=1)
         Hash Cond: (bar.a = foo.a)
         Extra Text: (seg2)   Hash chain length 1.6 avg, 8 max, using 161453 of 262144 buckets.
         ->  Redistribute Motion 4:4  (slice1; segments: 4)  (cost=0.00..601.14 rows=800000 width=37) (actual time=0.065..967.172 rows=801937)
               Hash Key: bar.a
               ->  Seq Scan on bar  (cost=0.00..453.44 rows=800000 width=37) (actual time=0.342..352.239 rows=3000000 lo
         ->  Hash  (cost=438.02..438.02 rows=250250 width=37) (actual time=85.309..85.309 rows=250911 loops=1)
               ->  Seq Scan on foo  (cost=0.00..438.02 rows=250250 width=37) (actual time=0.158..25.031 rows=250911 loo
 Planning time: 7.773 ms
   (slice0)    Executor memory: 119K bytes.
   (slice1)    Executor memory: 196K bytes avg x 4 workers, 220K bytes max (seg0).
   (slice2)    Executor memory: 26888K bytes avg x 4 workers, 26888K bytes max (seg0).  Work_mem: 15682K bytes max.
 Memory used:  128000kB
 Optimizer: Pivotal Optimizer (GPORCA) version 3.86.0
 Execution time: 2163.160 ms
```

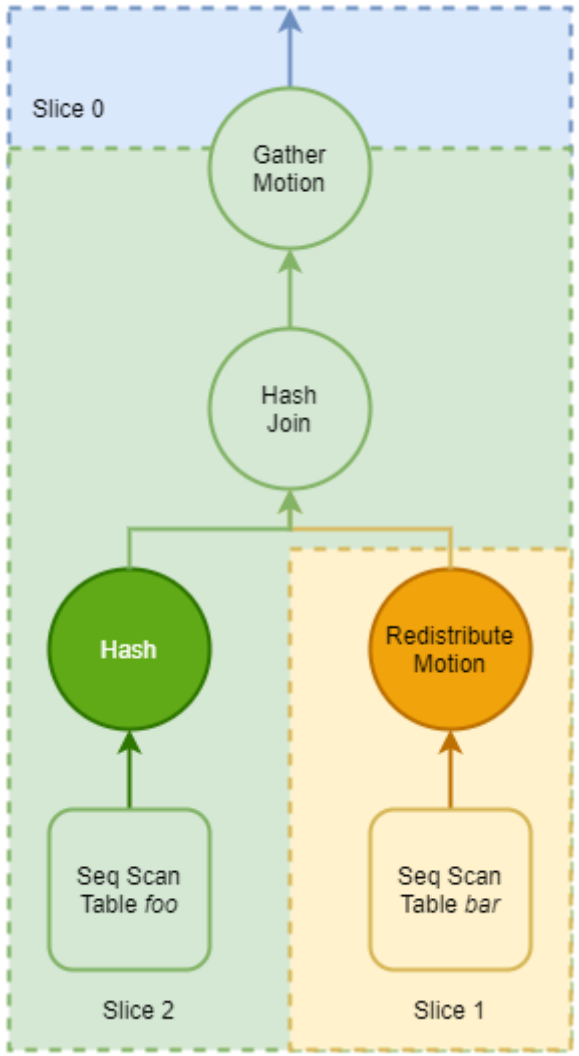


Слайсы



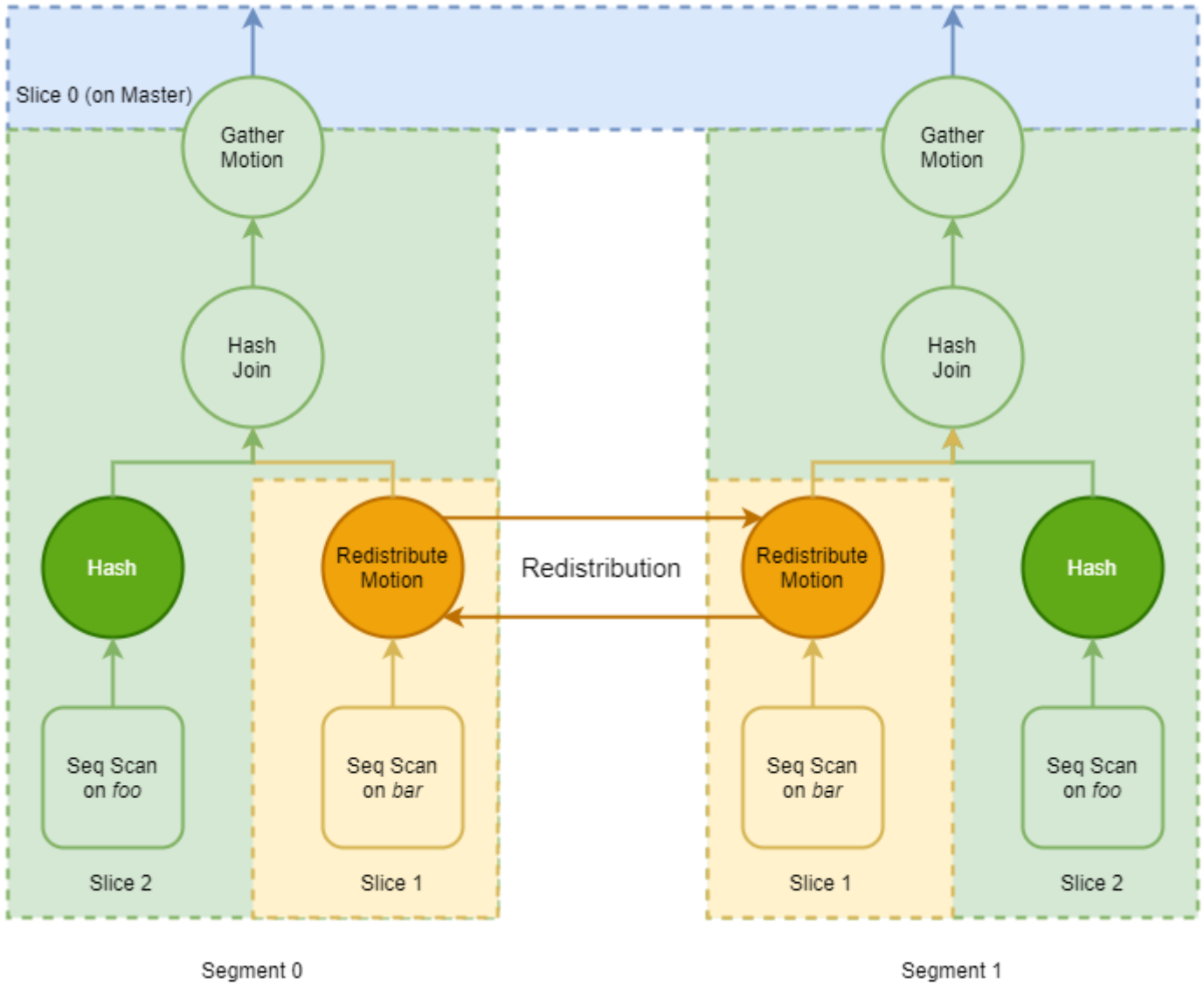
- Сканирование таблиц выполняется параллельно, полученные данные далее передаются в узлы выше по дереву. Работа происходит в разных слайсах.

Слайсы



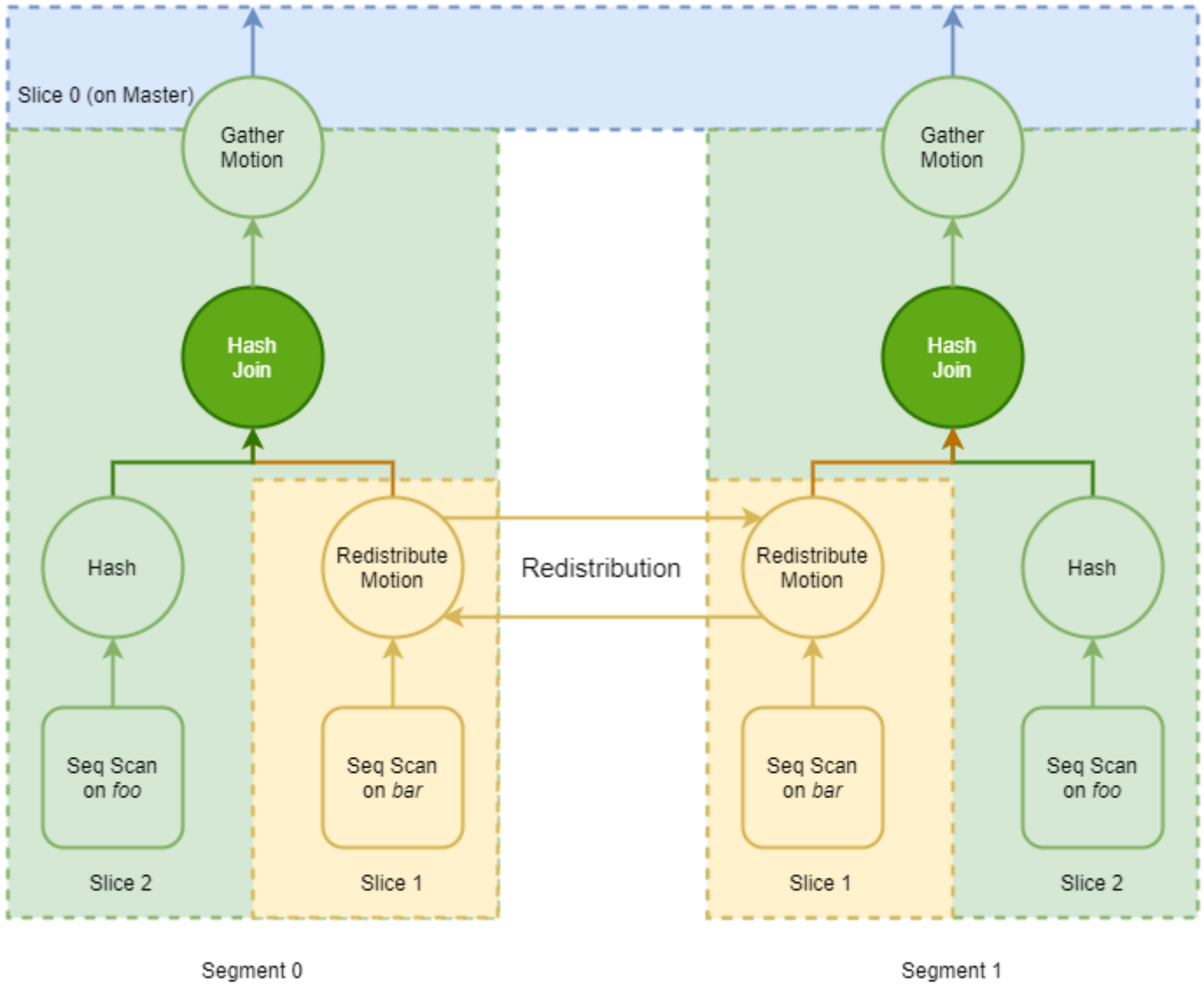
- Для следующих операций достаточно тех данных, которые присутствуют в их слайсах, поэтому они тоже выполняются параллельно.

Слайсы



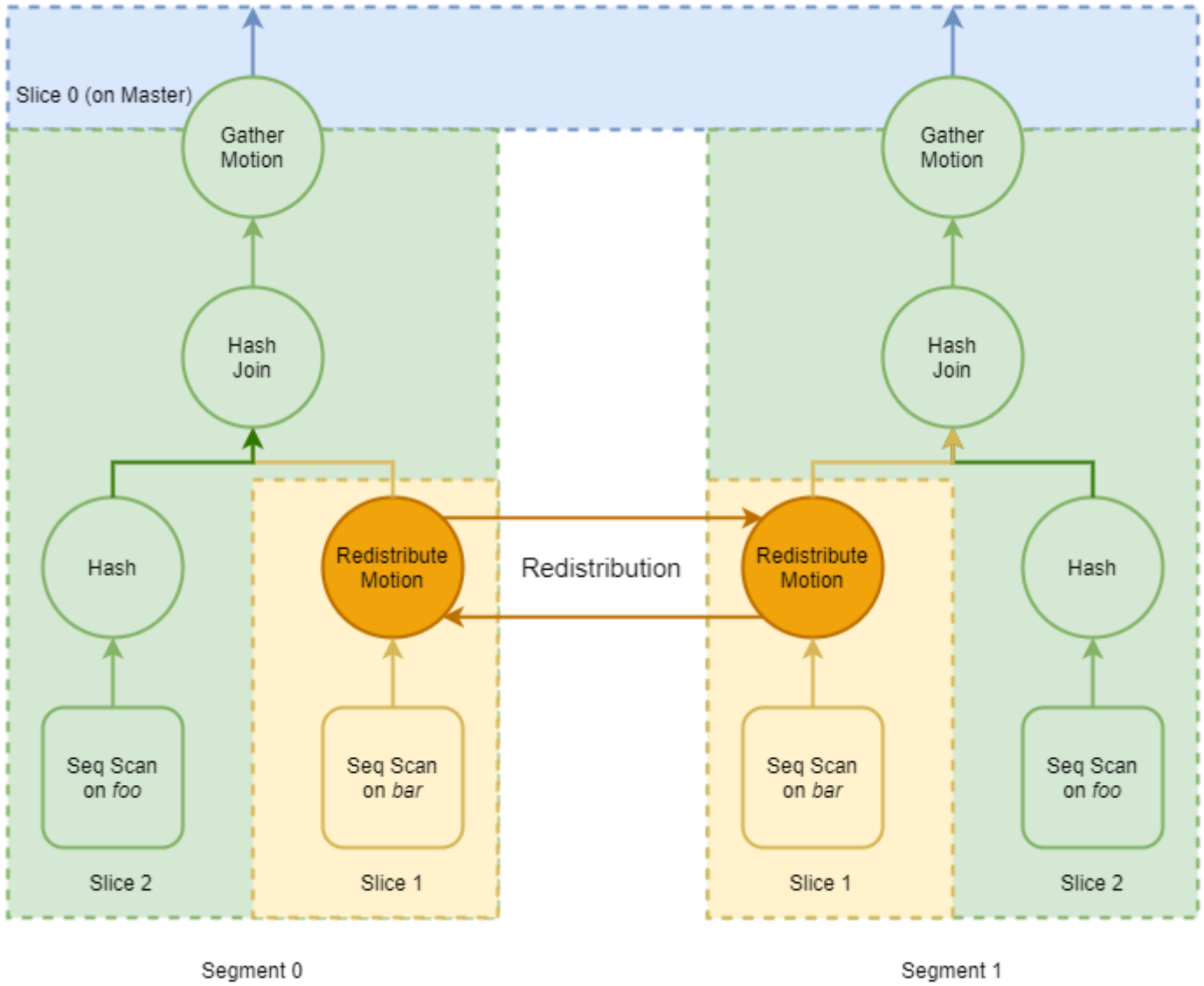
Стоит помнить, что план выполняется не на одном сегменте, и сегменты могут общаться между собой, обмениваясь данными.

Слайсы



Как только операции, данные которых нужны для вышестоящей, будут выполнены, один из слайсов завершится, работа будет происходить в оставшемся.

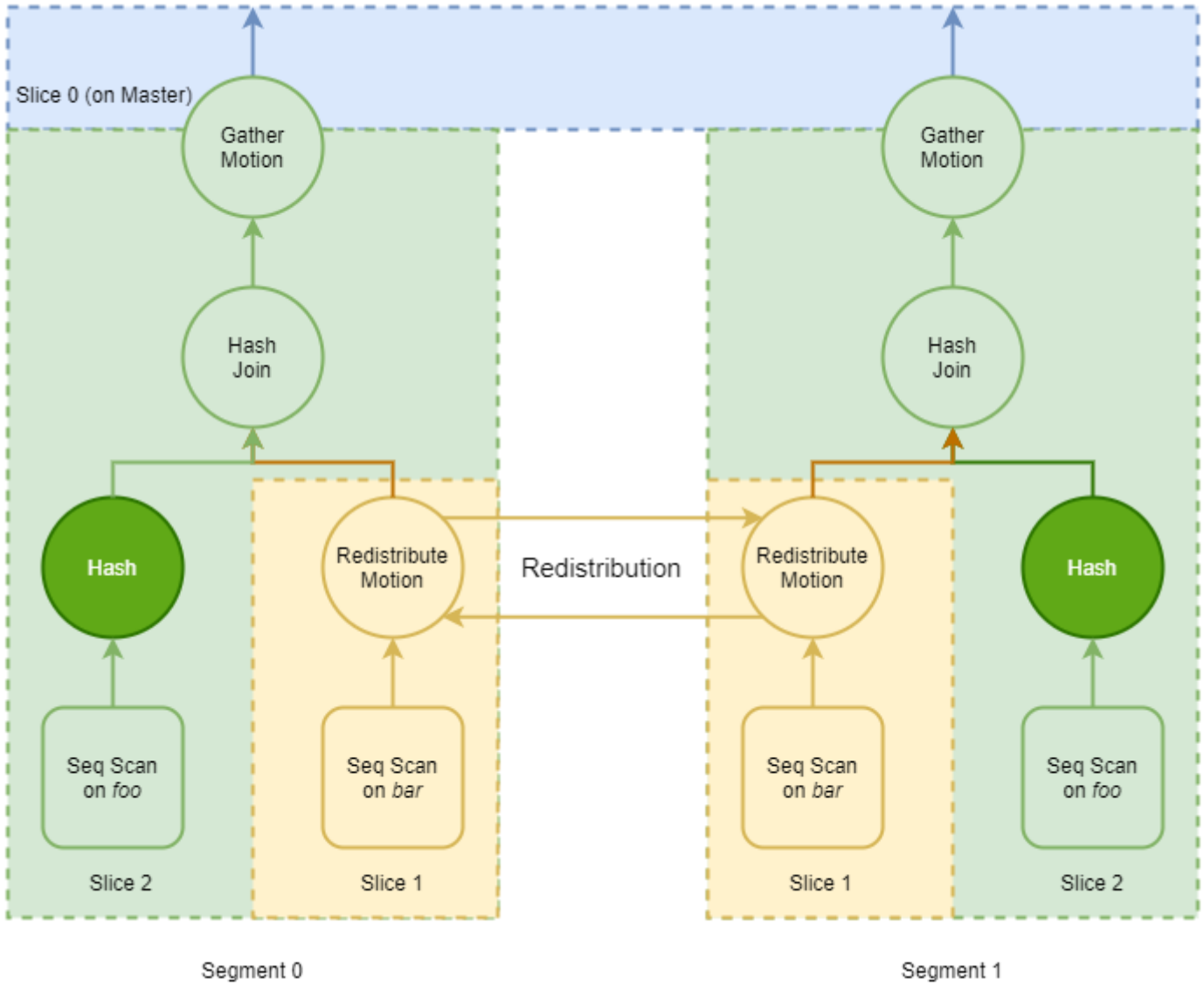
Слайсы



Операции могут выполняться разное время.

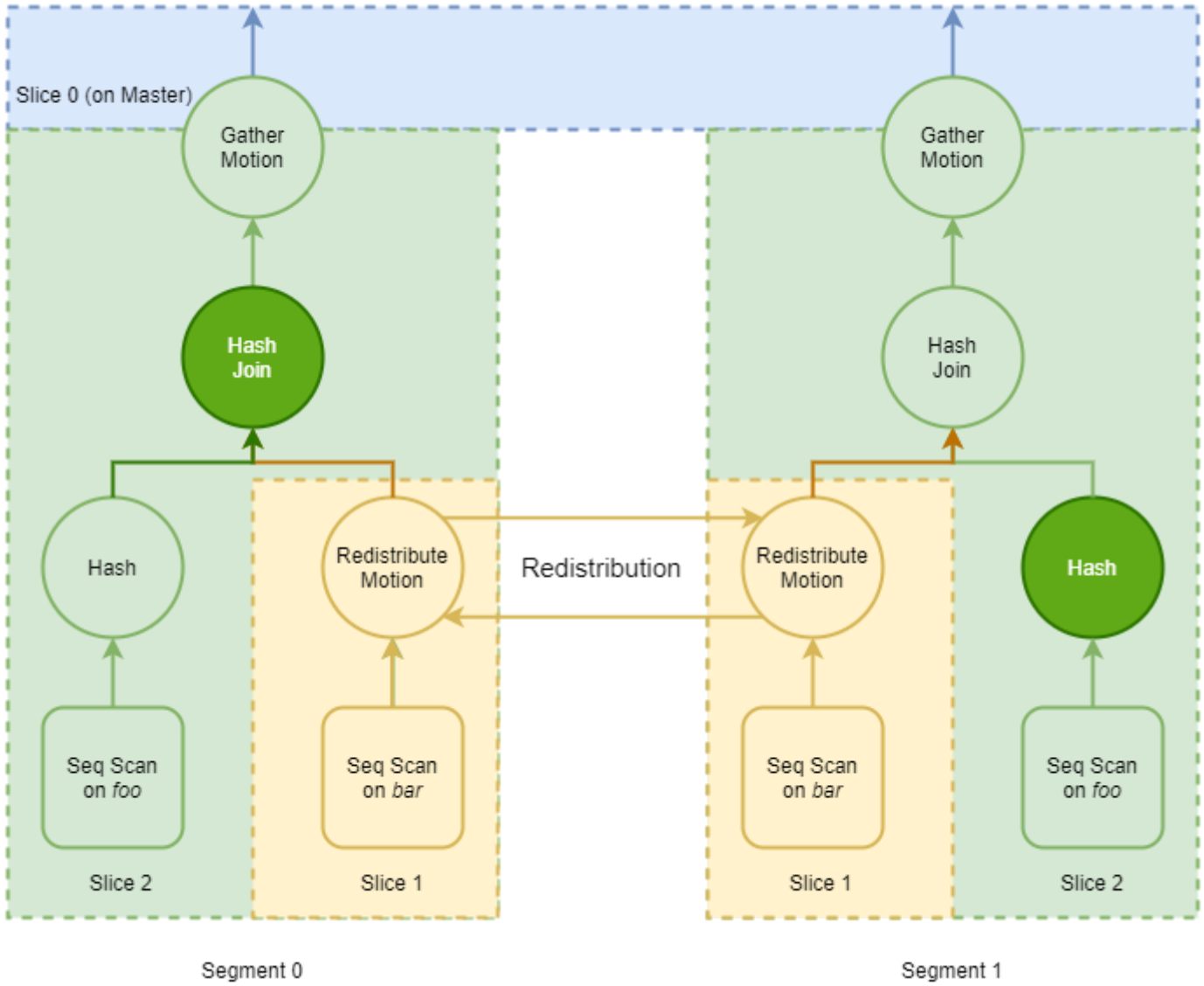
Какая операция будет быстрее можно оценить по параметру cost.

Слайсы



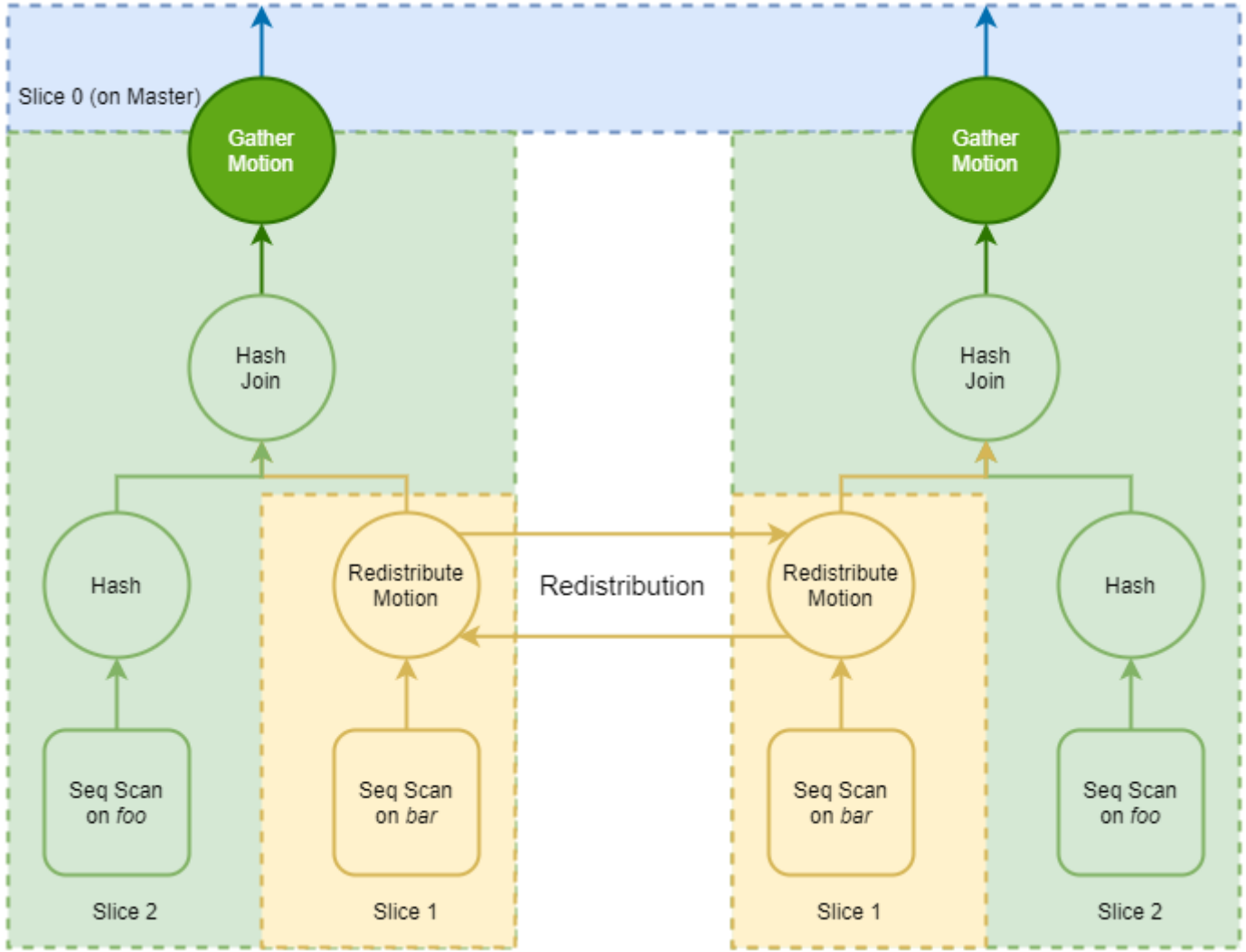
Но пока одна из операций не завершена, выполнение этой части дерева не может быть продолжено.

Слайсы



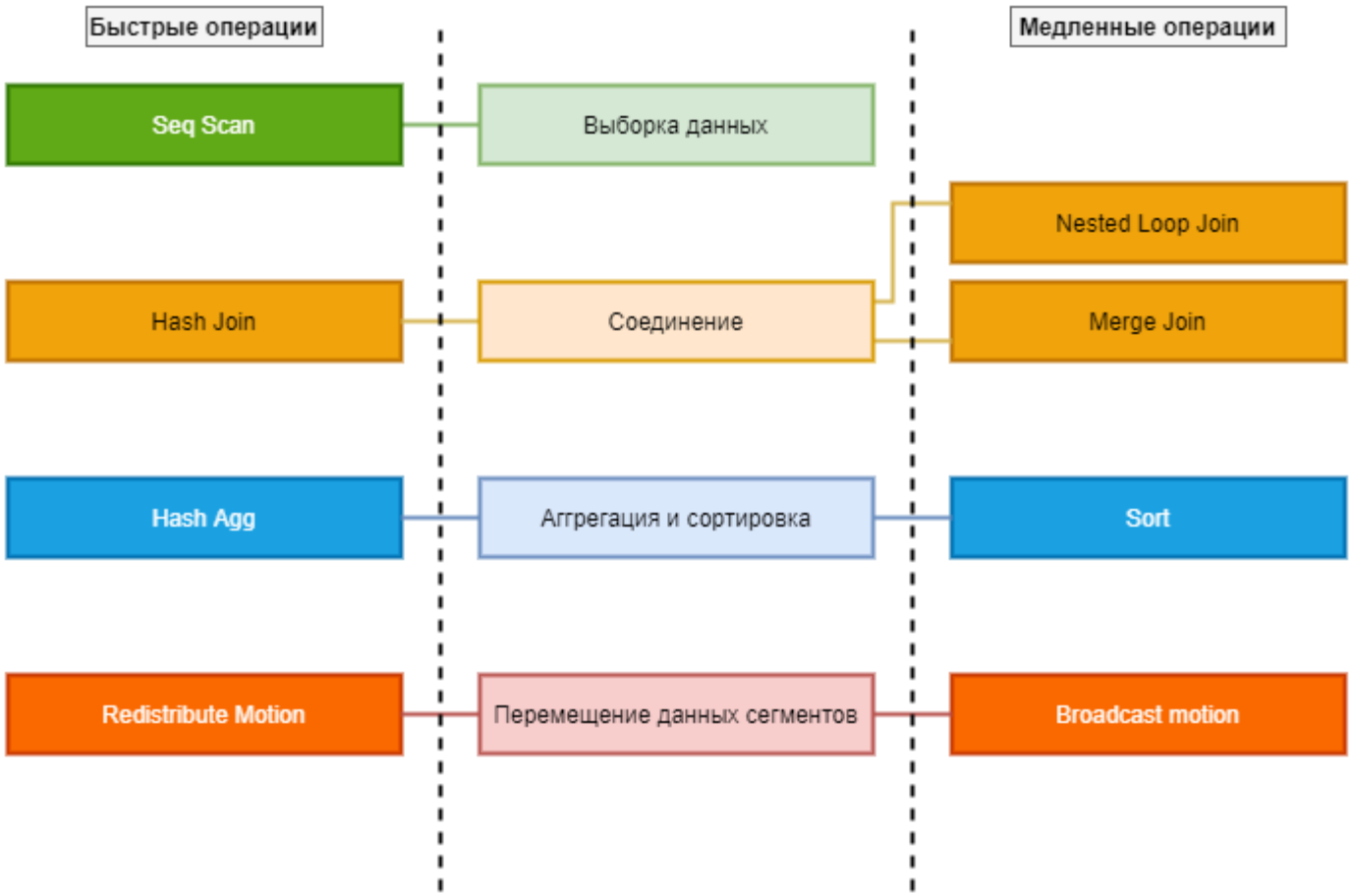
При этом стоит помнить, что работа на сегментах может происходить с разной скоростью (чего, конечно, стоит избегать, максимально равномерно распределяя данные как при хранении, так и при обработке).

Слайсы



Выполнение запроса завершится тогда, когда все сегменты отдадут свои данные мастеру и там пройдет и слияние в итоговую выборку.

Операции в плане



Операции в плане. Выборка данных

- Seq Scan on <table>:
  - Последовательное сканирование – быстрая операция в ADB.
- Index Scan using <index> on <table>:
  - Часто не дает преимуществ перед обычным Seq Scan и не выбирается планировщиком.
- Dynamic Table Scan:
  - Выборка данных из партиций при использовании оптимизатора GPORCA.

Операции в плане. Соединение

- Hash Join:
  - Строит хеши по меньшей из таблиц и сканирует большую.
  - Самый быстрый общий способ соединить две большие таблицы.
  - Не работает для количественного сравнения и при преобразовании типов.
- Nested Loop:
  - Для каждой строки в большей таблице сканирует меньшую.
  - Самый быстрый способ для очень маленьких объёмов.
  - Медленный для больших таблиц.
  - Используется для кросс-джойнов.
- Merge Join:
  - Отсортировать обе таблицы и соединить.

Операции в плане. Перемещение данных

- Broadcast:
  - Каждый сегмент пересылает свои данные каждому.
  - У всех сегментов собирается полная копия таблицы.
- Redistribute:
  - Динамическая редистрибуция данных в памяти запроса.
  - Каждый сегмент получает от остальных недостающие строки.
- Gather:
  - Объединение данных с нескольких сегментов на одном (так происходит пересылка результатов мастеру).

Операции в плане

Что делать с информацией из плана

Изучите состав дерева плана и ответьте на вопросы:

- Какие операции с наибольшей стоимостью?
- Верное ли количество строк указано? Совпадает ли значение в теории и на практике?
- Собрана ли статистика? Правильно ли распределены объекты?
- Работает ли partition elimination для партиционированных таблиц (не все ли партиции сканируются)?
- Hash-чейн при hash-джойнах строится по меньшей таблице?
- Есть ли в плане BROADCAST MOTION?
- Если есть NESTED LOOP, нельзя ли его переделать в HASH JOIN?
- Пишутся ли спилл-файлы?

Используйте модуль auto_explain или систему ADBCC для автоматического логирования плана запросов.

Лабораторная работа. Планы запросов

1. Используя таблицы `table1` и `table2` из предыдущих выведите:
   - План запроса, который содержит `redistribute motion`.
   - План запроса, который содержит `nested loop`.

2. Восстановите запрос по плану ниже:

```
Gather Motion 4:1  (slice2; segments: 4)  (cost=0.00..512.49 rows=99939 width=60)
  ->  Hash Join  (cost=0.00..492.43 rows=24985 width=60)
        Hash Cond: (table1.id2 = table2.id2)
        ->  Redistribute Motion 4:4  (slice1; segments: 4)  (cost=0.00..27.62 rows=25014 width=30)
              Hash Key: table1.id2
              ->  Index Scan using table1_pkey on table1  (cost=0.00..23.87 rows=25014 width=30)
                    Index Cond: (id1 < 100000)
        ->  Hash  (cost=435.25..435.25 rows=24985 width=30)
              ->  Seq Scan on table2  (cost=0.00..435.25 rows=24985 width=30)
                    Filter: (id1 < 100000)
Optimizer: Pivotal Optimizer (GPORCA) version 3.88.0
```

ARENADATA

Корпоративная платформа хранения
и обработки больших данных

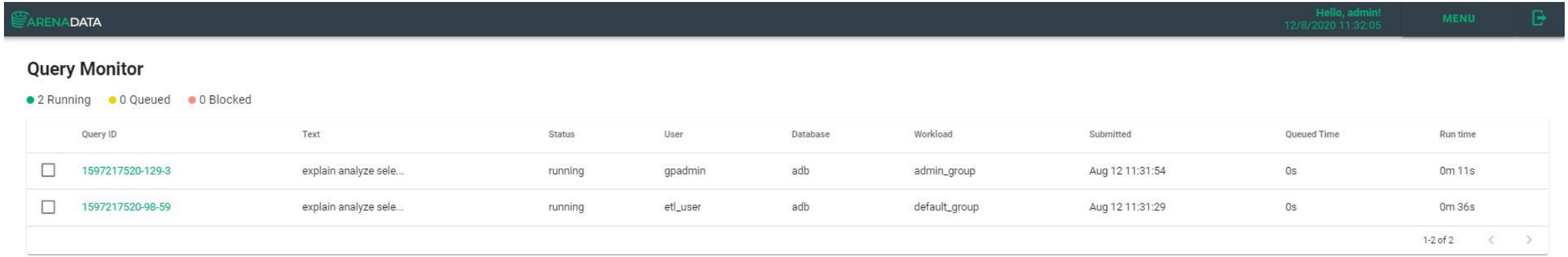
Arenadata Database Command Center

Возможности ADCC

- Позволяет в реальном времени увидеть прогресс выполнения текущих запросов;
- Можно посмотреть план выполнения запросов;
- Возможность просматривать исторические данные;
- Возможность просматривать важные метрики по запросам: перекос данных, количество обработанных строк, объём spill-файлов;
- Отображает список блокировок на объекты;
- Позволяет отменить или прервать выполнение запроса.

Документация: https://docs.arenadata.io/adb/ADCC/index.html

Мониторинг запросов в реальном времени



- Отображает выполняемые запросы в реальном времени и некоторые детали:
- Текст запроса;
- Статус (Running, Queued, Blocked);
- Из-под какого пользователя выполняется;
- В какой базе данных;
- Какая ресурсная группа используется;
- Время запуска запроса;
- Сколько запрос простоял в очереди;
- Время выполнения запроса.

Copyright arenadata.io

Детали выполнения запроса

ARENADATA

Hello, admin!
12/8/2020 12:07:28

MENU

Query Details

Query ID: **1597217520-201-11** ● done Run time 33s

Details

Username	**gpadmin**	Submitted	**Aug 12 12:04:57**
Database	**adb**	Queued Time	**0s**
Workload	**admin_group**	Run Time	**33s**
Planner	**GPORCA**	Est.progress	**97.32%**

Performance

Spill Files (max)	**1.04 GiB**
Spill Files (skew)	**2.30%**

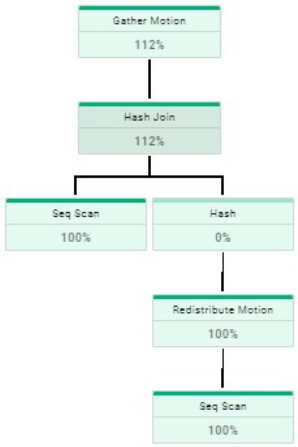
Query text

```
explain analyze select * from foo f join bar b on f.a=b.a;
```

Locks and Blocks

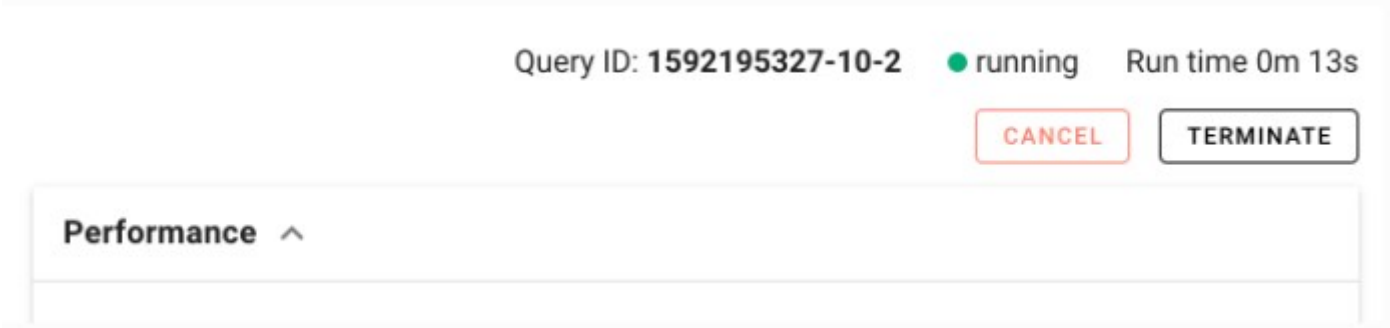
holding 6 locks

PLAN&PROGRESS TEXTUAL PLAN



Cost
4950.21
Relation
--
Row Skew
0.30%
Est. Tuples
63733248
Actual Tuples
72000026
Operation Keys
Hash Cond: (a = a) Join Filter: true Filter: true

Отмена выполнения запроса



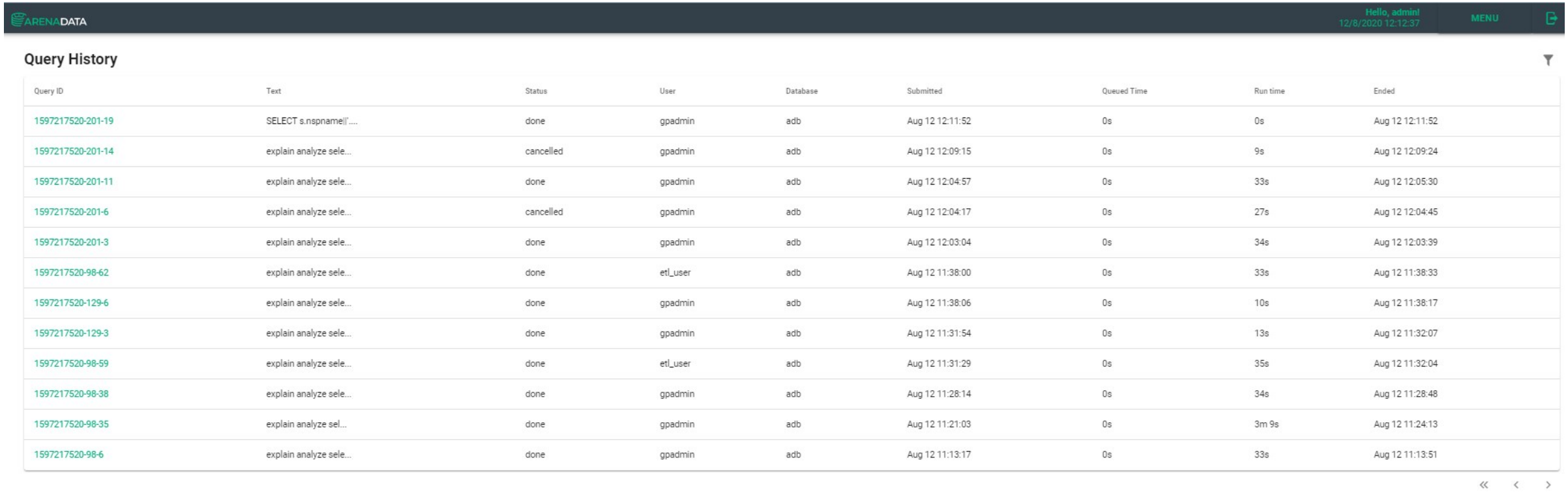
- Cancel – отмена выполнения запроса путем вызова функции pg_cancel_backend, отменяющей запрос в обслуживающем процессе.
- Terminate – отмена выполнения запроса путем вызова функции pg_terminate_backend, завершающей обслуживающий процесс, в котором выполняется запрос.

Операции Cancel и Terminate доступны исходя из прав доступа, определенных для пользователя.

Copyright arenadata.io

История выполнения запросов

- С помощью истории запросов можно просмотреть все детали выполнения запроса
- Для поиска запросов существует панель фильтров, позволяющая ограничивать вывод истории запросов



Уровни доступа ADCC

- **manage users** – доступ к администрированию (создание, изменение, удаление пользователей).
- **view all queries** – возможность видеть все выполняемые в кластере ADB запросы.
- **kill all queries** – возможность отменять выполнение любого запроса в кластере ADB.
- **view self queries** – возможность видеть собственные запросы (запросы, выполненные ролью ADB с именем, идентичным логину текущего пользователя ADBCC или названию LDAP-группы, в которую входит пользователь).
- **kill self queries** – возможность отменять выполнение собственных запросов (запросов, выполненных ролью ADB с именем, идентичным логину текущего пользователя ADBCC или названию LDAP-группы, в которую входит пользователь).

уровень доступа \ роль	Owner	Administrator	Advanced User	Regular User
manage users	+			
view all queries	+	+	+	
kill all queries	+	+		
view self queries				+
kill self queries			+	+

ARENADATA

Корпоративная платформа хранения
и обработки больших данных

Common Table Expressions

CTE: Возможности

- Можно представить как временное представление, которое существует только на время выполнения запроса;
- Внутри запроса определяются с помощью параметра WITH:

```
WITH regional_sales AS (
                        SELECT region, SUM(amount) AS total_sales
                        FROM orders
                        GROUP BY region
                        )
SELECT * FROM regional_sales;
```

- Помогают разбить большой запрос с подзапросами на меньшие части, улучшая читабельность кода;
- Могут улучшать производительность запросов, выполняя код внутри CTE только один раз;
- Могут быть использованы при выполнение команд SELECT, INSERT, UPDATE или DELETE ;
- Могут быть использованы предикаты из внешнего запроса (GPORCA);
- Поддерживаются рекурсивные (только Postgres query optimizer) и вложенные CTE (GPORCA).

Copyright arenadata.io

CTE: Пример с несколькими CTE

```
WITH
  regional_sales AS (
     SELECT region, SUM(amount) AS total_sales
     FROM orders
     GROUP BY region
  ),
 top_regions AS (
     SELECT region
     FROM regional_sales
     WHERE total_sales > (SELECT SUM(total_sales)/10 FROM regional_sales)
  )
SELECT region,
    product,
    SUM(quantity) AS product_units,
    SUM(amount) AS product_sales
FROM orders
WHERE region IN (SELECT region FROM top_regions)
GROUP BY region, product;
```


Copyright arenadata.io

CTE: Пример с рекурсией

```
WITH RECURSIVE letters AS
  (
   SELECT ASCII('A') AS code, CHR(ASCII('A')) AS symbol
     UNION ALL
   SELECT code + 1, CHR(code + 1) FROM letters
   WHERE code + 1 <= ASCII('Z')
  )
SELECT code, symbol FROM letters;
```


Copyright arenadata.io

CTE: Пример с DML

```
WITH del_rows  AS  (
                DELETE FROM item
                WHERE name = 'book'
                RETURNING * )
SELECT * FROM del_rows;
```

ARENADATA

Корпоративная платформа хранения
и обработки больших данных

Статистика

Статистика

Актуальная статистика является необходимым условием оптимального построения плана для выполнения запроса. Без неё планы будут строиться неверно и могут быть проблемы с распределением памяти.

Когда необходимо запускать сбор статистики:

- После загрузки данных.
- После создания индексов.
- После операция `INSERT`, `UPDATE` и `DELETE`, которые привели к значительному изменению данных.

Примерный объём изменений в таблице, при котором следует обновить статистику – 0.5%.

В базовой конфигурации статистика собирается только при первой вставке данных в таблицу.

При последующих операциях с данными необходимо вручную обновлять статистику при помощи команды `ANALYZE`.

Команда ANALYZE

- `ANALYZE` – команда для сбора статистики
  - Без параметров – запускает сбор статистики по всем таблицам в БД.
  - С указанием имени таблицы – собирает статистику только по заданной таблице.
  - С указанием таблицы и колонок – собирает статистику только по указанным колонкам.
- Сохраняет статистику в системную таблицу `pg_statistic`.
- `pg_stats` – представление, которое покажет, какая статистика была собрана для таблицы.
- `pg_stat_operations` - представление, которое подскажет, какие операции и когда были над объектом в том числе и `ANALYZE`.

Нюансы

- GPORCA использует статистику на уровне root-партиции, а Postgres Planner на уровне leaf-партиций;
- Статистика не собирается автоматически для партиционированных таблиц, если загрузка происходит через root-партицию;
- Если загрузка производилась в child leaf партицию, то для этой партиции будет собрана статистика, если она ещё не собиралась для партиционированной таблицы;
- ANALYZE ROOTPARTITION <*root_partition*> - соберёт статистику только по root-партиции;
- ANALYZE <root_partition> - соберёт статистику и по root-партиции и по child-партициям;
- ANALYZE <*leaf_partition*> - соберёт статистику только по child-партициям;
- При добавлении новых партиций, статистика будет автоматически обновляться на уровне root-партиции , если для всех существующих child-партиций есть статистика.
- GPORCA не поддерживает рекурсивные CTE и работу с внешними партициями.

Автоматизация анализа в ADB

- `0 11 * * * /home/gpadmin/arenadata_configs/operation.sh analyze`
  - Выполняет analyzedb по всем таблицам в БД, кроме тех, чьи схемы внесены в таблицу: arenadata_toolkit.operation_exclude
  - Работает по кругу.
  - Начинает работать по крону, завершает по переменной analyze_stop.

- analyzedb – обвёртка поверх ANALYZE:
  - Ведёт статистику по таблицам, которые уже проанализировала и не собирает статистику по тем партициям, которые не изменялись.
  - Может выполнять параллельный сбор статистики в несколько потоков.

Лабораторная работа. Статистика

1. Убедитесь, что таблицы `table1` и `table2` существуют в базе данных с помощью команды `\d+`
2. Установите значение выделенной памяти для запроса в рамках сессии: `set statement_mem = 20000;`
3. Вставьте данные в таблицу table1: `INSERT INTO table1 SELECT gen, gen, gen::text || 'text1', gen::text || 'text2' FROM generate_series (1000000,4000000) gen;`
4. Постройте план выполнения запроса с помощью EXPLAIN ANALYZE следующего запроса: `SELECT * FROM table1 t1 join table2 t2 on t1.id1 = t2.id1;`
5. Выполните сбор статистики для таблицы `table1`.
6. Снова постройте план выполнения запроса с помощью EXPLAIN ANALYZE следующего запроса: `SELECT * FROM table1 t1 join table2 t2 on t1.id1 = t2.id1;`
7. Сравните полученные планы. Обратите внимание на ожидаемое число строк в плане и сравните его с актуальным (rows out) для таблицы table1. Также обратит внимание по какой таблице строится hash.
8. Обратите внимание на показатели `Memory used` и `Memory wanted` в первом плане и во втором.

ARENADATA

Корпоративная платформа хранения
и обработки больших данных

Индексы

Индексы в ADB: общие сведения

- Индекс – объект в СУБД, создаваемый с целью повышения производительности при выборке данных с условиями.
- Сам индекс – это отдельная таблица, содержащая ссылки на строки основной таблицы.
- В ADB доступны следующие виды: BTREE, BITMAP, GIST, GIN.
  - BTREE – используется по умолчанию, если не указывать тип. Подходит для OLTP-профиля нагрузки.
  - BITMAP – используйте если:
    - Колонка(и) содержит(ат) от 100 до 100 000 уникальных значений (низкая кардинальность данных).
    - Чтения намного больше, чем UPDATE (не OLTP).
  - GIST – очень специфические случаи (https://www.postgresql.org/docs/9.4/indexes-types.html);
  - GIN – обратный индекс. Он работает с типами данных, значения которых не являются атомарными, а состоят из элементов.
- Будьте аккуратны с LIKE: ‘%text’ и ‘text%’ могут вести себя по-разному.

Индексы в ADB: нюансы и применимость

- Оптимизация производительности с помощью индексов в DWH должна выполняться в последнюю очередь!
- Индексы подходят для близкой к OLTP-профилю нагрузке: частый выбор небольшого числа строк с WHERE.
- Уменьшая random_page_cost можно форсировать использование индекса планировщиком (но этого лучше не делать).
- Поиск по индексу умеет разжимать только необходимые строки;
- При загрузке данных в таблицу с индексами удалите индексы, загрузите данные, затем создайте индексы заново. Это будет быстрее, чем загружать данные в таблицу.
- FILLFACTOR – для часто обновляемых таблиц снижайте значение.

Индексы в ADB: синтаксис создания

```
CREATE [UNIQUE] INDEX name ON table

       [USING btree|bitmap|gist|spgist|gin]

       ( {column_name | (expression)} [COLLATE parameter] [opclass] [ ASC | DESC ]

       [ NULLS { FIRST | LAST } ] [, ...] )

       [ WITH ( storage_parameter = value [, ... ] ) ]

       [TABLESPACE tablespace]

       [WHERE predicate]
```

Лабораторная работа. Создание индекса

1. Создайте таблицу table4 (колоночная, сжатие `zstd` уровня 1):

```
id1 int,
id2 int,
gen1 text,
gen2 text.
```

2. Вставьте в таблицу данные:

```
INSERT INTO table4 SELECT gen, gen, 'text' || gen::text,'text' || gen::text FROM generate_series(1,2000000) gen;
```

3. Создайте индексы, замеряя время их создания:

```
id1 – btree.
id2 – bitmap.
gen1 – btree.
gen2 – bitmap.
```

ARENADATA

Корпоративная платформа хранения
и обработки больших данных

Транзакции

Проблема потери консистенстности

- Alice переводит 100.00 Bob’y:

```
UPDATE accounts SET balance = balance - 100.00 WHERE name = 'Alice';

UPDATE accounts SET balance = balance + 100.00 WHERE name = 'Bob';
```

- В середине что-то пошло не так. Оба остались без денег.

Решение

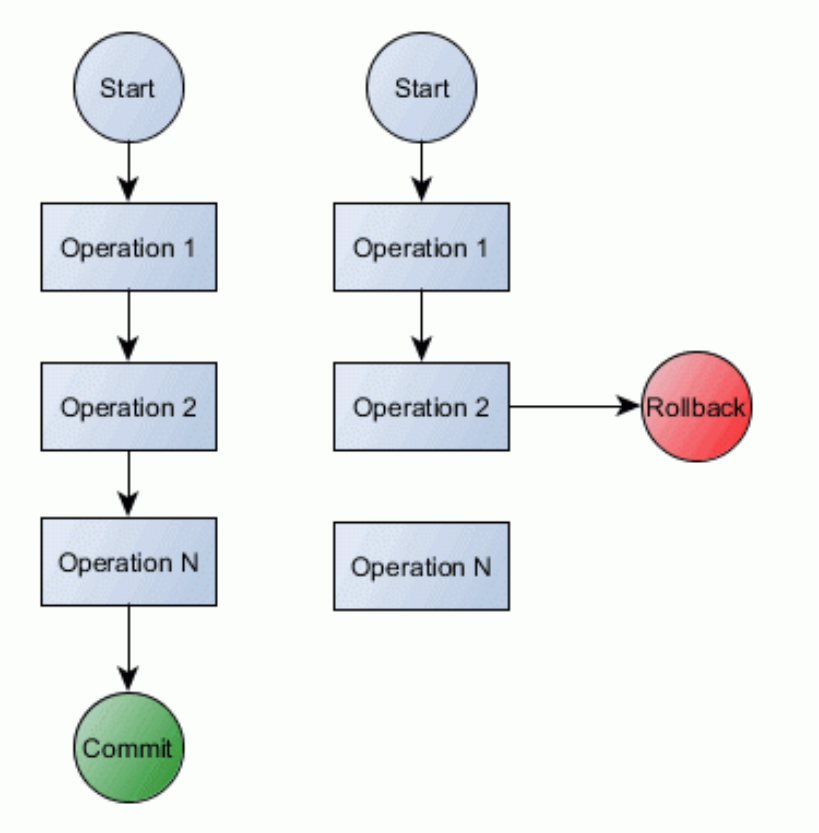
- Alice переводит 100.00 Bob’y:

```
BEGIN;

UPDATE accounts SET balance = balance - 100.00 WHERE name = 'Alice';

UPDATE accounts SET balance = balance + 100.00 WHERE name = 'Bob';

COMMIT;
```

- В середине что-то пошло не так. Транзакция откатилась (`abort, rollback`). Alice вернулись деньги.

Транзакции

- Транзакции — это группа операций на чтение/запись, выполняющихся только если все операции из группы успешно выполнены.



ACID

- **`Atomicity`** – транзакции атомарны, то есть либо все изменения транзакции фиксируются (commit), либо все откатываются (rollback).

- **`Consistency`** – транзакции не нарушают согласованность данных, то есть они переводят базу данных из одного корректного состояния в другое.

- **`Isolation`** – работающие одновременно транзакции не влияют друг на друга, то есть многопоточная обработка транзакций производится таким образом, чтобы результат их параллельного исполнения соответствовал результату их последовательного исполнения.

- **`Durability`** – если транзакция была успешно завершена, никакое внешнее событие не должно привести к потере совершенных ей изменений.

Изоляция

Уровень изоляции - свойство транзакции, которое показывает, насколько транзакция пренебрегает изменениями в данных, сделанными другими транзакциями. Устанавливается после открытия транзакции:

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ WRITE;
```

Уровни изоляции:

- **`READ UNCOMMITED`** – чтение незафиксированных изменений как своей транзакции, так и параллельных транзакций (не может быть применен в ADB).
- **`READ COMMITTED`** – чтение всех изменений своей транзакции и закоммиченных изменений параллельных транзакций.
- **`REPEATABLE READ`** – все операции текущей транзакции могут видеть только строки, которые были закоммичены только до первой операции в текущей транзакции. В обычных СУБД возможно фантомное чтение, но в Greenplum оно не допустимо на этом уровне.
- **`SERIALIZABLE` (установить можно, но ведёт себя как `REPEATABLE READ`)** – все операции текущей транзакции могут видеть только строки, которые были закоммичены только до первой операции в текущей транзакции. Возможны ошибки сериализации, приложение должно быть готово повторить транзакцию.

Уровни доступа: **`READ ONLY`** и **`READ WRITE`**.

Лабораторная работа. Уровни изоляции

Уровень READ COMMITED:

1. Создайте таблицу table5: create table table5 (id int, state text);
2. Вставьте данные: insert into table5 values (1,'insert 1'),(2, 'insert 2');
3. Откройте две транзакции.
4. В первой транзакции выведите содержимое таблицы table5.
5. Во второй транзакции проведите UPDATE: update table5 set state = 'update 1 transaction 2' where id=1;
6. В первой транзакции выведите содержимое таблицы table5.
7. Закоммитьте вторую транзакцию.
8. В первой транзакции выведите содержимое таблицы table5.
9. Завершите первую транзакцию любым способом.

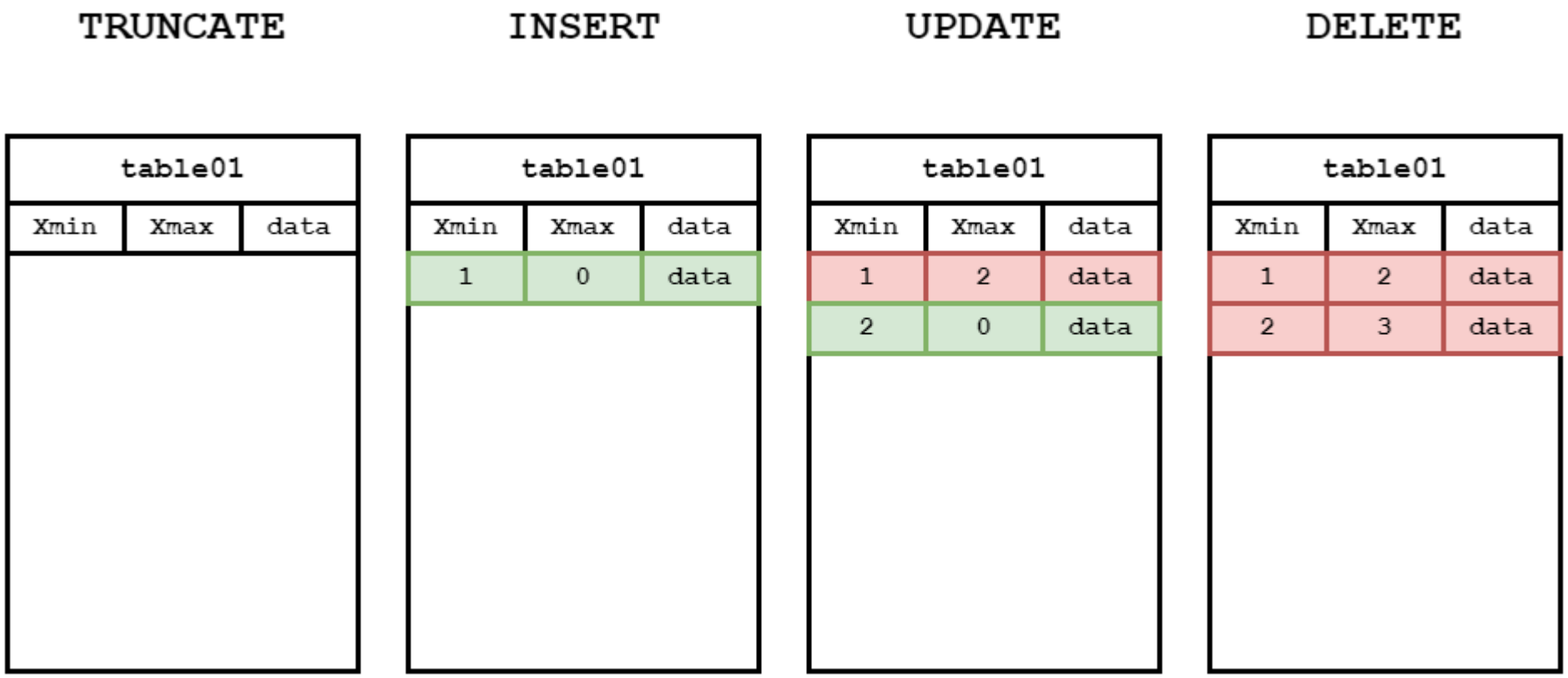
Уровень REPEATEBLE READ:

1. Откройте две транзакции.
2. Переведите первую транзакцию в режим REPEATABLE READ.
3. В первой транзакции выведите содержимое таблицы table5.
4. Во второй транзакции проведите UPDATE: update table5 set state = 'update 2 transaction 2' where id=1;
5. Закоммитьте вторую транзакцию.
6. В первой транзакции выведите содержимое таблицы table5.
7. Завершите первую транзакцию любым способом.
8. Выведите содержимое таблицы table5.

MVCC

- Все транзакции в системе имеют последовательные номера.
- Существует глобальный реестр транзакций, содержащий информацию о том, какие транзакции находятся в процессе выполнения, а какие были подвергнуты откату.
- Для каждой записи таблицы существуют технические записи `Xmin` и `Xmax`, хранящие информацию о транзакциях, модифицировавших эту запись:
  - `Xmin` – идентификатор транзакции, добавившей запись в таблицу.
  - `Xmax` – идентификатор транзакции, удалившей запись из таблицы.
- В heap-таблицах эти записи хранятся как поля, AO-таблицы хранят эти данные в дополнительной скрытой таблице.
- `Операция UPDATE` реализована через удаление строки и вставка новой (измененной).
- Строки не удаляются, а лишь помечаются как удаленные, оставаясь в таблице.
- В запросе на выборку для новой транзакции будут выводиться только те строки, которые не помечены удаленными (с учетом уровня изоляции).

MVCC


TRUNCATE
INSERT
UPDATE
DELETE
table01
Xmin
Xmax
data
1
0
data
1
2
data
2
0
data
1
2
data
2
3
data

MVCC

```
adb=# SET gp_select_invisible = TRUE; -- смотрим все записи в таблице (god mode on)

adb=# insert into foo values (1,'insert 1'),(2, 'insert 2');
INSERT 0 2

adb=# select xmin,xmax,* from foo;
  xmin  | xmax | id |  state
--------+------+----+----------
 110408 |    0 |  1 | insert 1
 110409 |    0 |  2 | insert 2

adb=# update foo set state = 'update 1' where id=1;
UPDATE 1

adb=# select xmin,xmax,* from foo;
  xmin  |  xmax  | id |  state
--------+--------+----+----------
 110409 |      0 |  2 | insert 2
 110408 | 110421 |  1 | insert 1 -- «закрытая» запись
 110421 |      0 |  1 | update 1
```

Обслуживание таблиц. VACUUM и VACUUM FULL

- VACUUM удаляет закрытые записи, оставляя вместо них пустое место;
  - VACUUM для AO-таблиц ведёт себя как VACUUM FULL, если bloat>10%.
- VACUUM FULL удаляет закрытые записи и сжимает таблицы. Операция блокирует таблицу в эксклюзивном режиме.
- При регулярном VACUUM в VACUUM FULL нет необходимости – пустые места перезаписываются.
- Системные каталоги тоже нуждаются в регулярном VACUUM – частота зависит от количества DDL-операций.
- Bloat – распухание таблиц в связи с большим количеством закрытых строк.
- Текущий bloat приведён в gp_toolkit.gp_bloat_diag (нужна актуальная статистика).

Автоматизация VACUUM в ADB

```
0     6       *       *       *       /home/gpadmin/arenadata_configs/operation.sh vacuum
```

- Делает VACUUM по всем таблицам в БД, кроме тех, чьи схемы внесены в таблицу: arenadata_toolkit.operation_exclude
- Работает по кругу.
- Начинает работать по крону, завершает по переменной vacuum_stop.

```
4     */8       *       *       *       /home/gpadmin/arenadata_configs/run_sql.sh
/home/gpadmin/arenadata_configs/vacuum_catalog_...
```

- Делает VACUUM по всем таблицам системного каталога.
- Отредактируйте частоту запуска согласно вашему количеству DDL-операций.

Нюансы MVCC

- `TRUNCATE` и `DELETE` без условий – две большие разницы. Используйте TRUNCATE.
- Не используйте `SET gp_select_invisible = TRUE;` для поиска bloat.
- Настройте алерты на Wraparound в мониторинге.
- Если `VACUUM` долго не выполнялся регламентно, выполните `VACUUM FULL`.
- При `ROLLBACK` записи проставляется специальный флаг (не виден внутри СУБД).

Лабораторная работа. MVCC

1. Создайте таблицу table6: `create table table6 (id int, state text);`
2. Вставьте данные: `INSERT INTO table6 values (1,'insert 1'),(2, 'insert 2');`
3. Откройте две транзакции.
4. В первой транзакции выполните update: `update table6 set state='update 1 transaction 1' where id=1;`
5. В первой транзакции выполните update: `update table6 set state='update 2 transaction 1' where id=1;`
6. В первой транзакции включите режим просмотра закрытых строк.
7. В первой транзакции получите вывод xmin, xmax и остальных столбцов.
8. Во второй транзакции получите вывод xmin, xmax и остальных столбцов.
9. Откатите первую транзакцию, закройте вторую транзакцию и сессию.
10. Получите вывод xmin, xmax и остальных столбцов.
11. Включите режим просмотра закрытых строк.
12. Получите вывод xmin, xmax и остальных столбцов.
13. Выполните update: `update table6 set state='update 3 transaction 1' where id=1;`
14. Получите вывод xmin, xmax и остальных столбцов. Постарайтесь объяснить его.

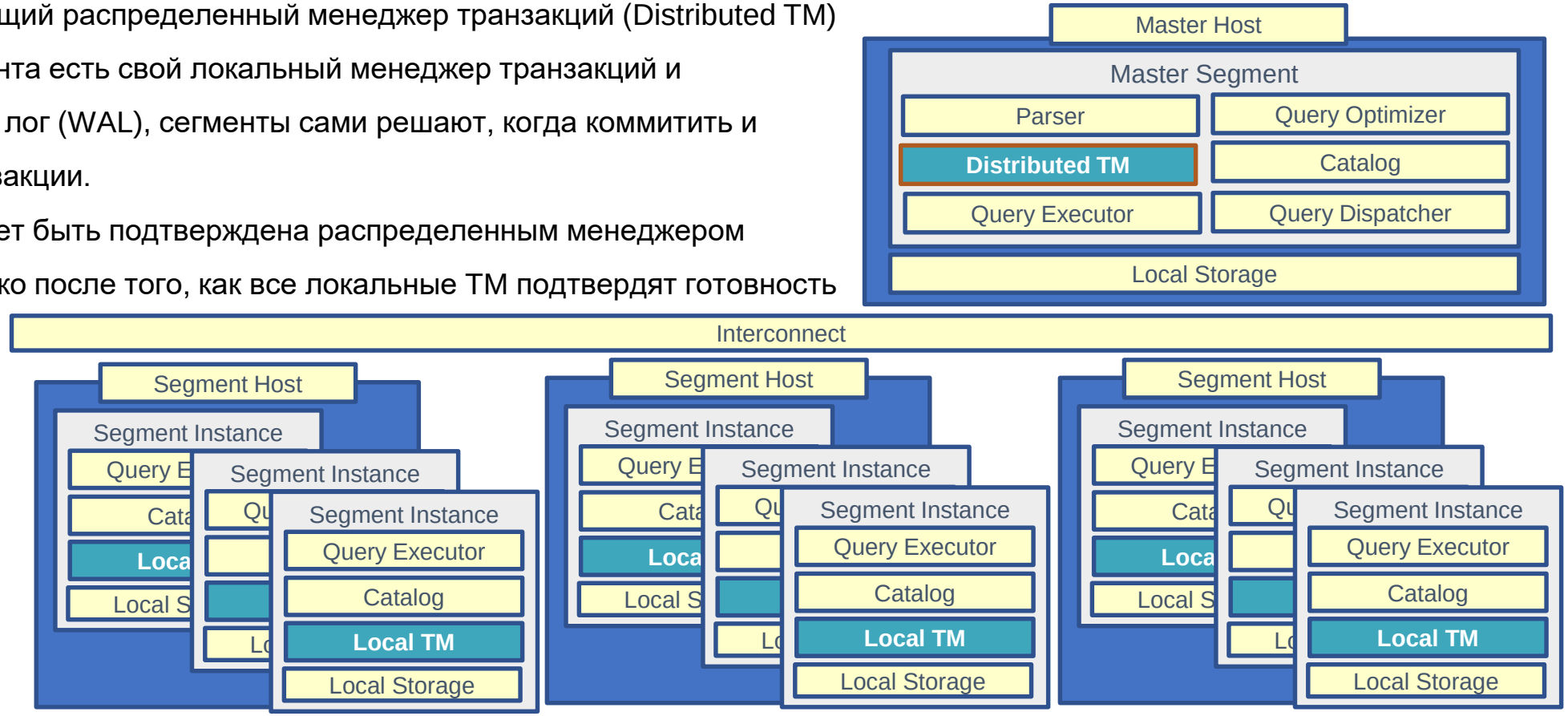
ARENADATA

Корпоративная платформа хранения
и обработки больших данных

Блокировки

Транзакционность в MPP-архитектуре

- ADB – транзакционная система.
- Мастер контролирует COMMIT/ABORT/ROLLBACK на сегментах. На нем размещается общий распределенный менеджер транзакций (Distributed TM)
- У каждого сегмента есть свой локальный менеджер транзакций и транзакционный лог (WAL), сегменты сами решают, когда коммитить и прерывать транзакции.
- Транзакция может быть подтверждена распределенным менеджером транзакций только после того, как все локальные TM подтвердят готовность к коммиту.



ARENADATA

Транзакции: блокировки объектов

Requested lock mode	Current lock mode								Associated SQL Commands
	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE	
ACCESS SHARE								X	SELECT
ROW SHARE							X	X	SELECT FOR SHARE
ROW EXCLUSIVE						X	X	X	INSERT, COPY
SHARE UPDATE EXCLUSIVE					X	X	X	X	VACUUM (without FULL), ANALYZE
SHARE				X	X	X	X	X	CREATE INDEX
SHARE ROW EXCLUSIVE			X	X	X	X	X	X	ALTER TABLE
EXCLUSIVE		X	X	X	X	X	X	X	DELETE, UPDATE, SELECT FOR UPDATE
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X	ALTER TABLE, DROP TABLE, TRUNCATE, REINDEX, CLUSTER, VACUUM FULL

Ручная блокировка

Команда LOCK позволяет получить блокировку в ручном режиме:

```
LOCK [TABLE] name [, ...] [IN lockmode MODE] [NOWAIT]
```

где *lockmode* - один из вариантов уровня блокировки:

```
ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE
| SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE
```

Мониторинг блокировок

- С помощью представления `pg_locks`;

```
adb=# select locktype, database, relation, pid, mode, granted from pg_locks;
  locktype   | database | relation |  pid  |        mode         | granted
-------------+----------+----------+-------+---------------------+---------
 relation    |    16384 |    11343 | 38358 | AccessShareLock     | t
 virtualxid  |          |          |  6253 | ExclusiveLock       | t
 virtualxid  |          |          |  8674 | ExclusiveLock       | t
 relation    |    16384 |    11343 |  8674 | AccessShareLock     | t
 virtualxid  |          |          | 38358 | ExclusiveLock       | t
 relation    |    16384 |   941139 |  8674 | AccessExclusiveLock | t
 relation    |    16384 |   941139 |  6253 | ExclusiveLock       | f
 virtualxid  |          |          | 17267 | ExclusiveLock       | t
```

- Если выполнить `JOIN` с таблицей `pg_stat_activity`, то можно получить больше информации по блокировкам (запрос, пользователь, имя приложения).
- С помощью функции `gp_dist_wait_status()`.
- Записи в стандартном логе PostgreSQL (при включенном параметре log_lock_waits):

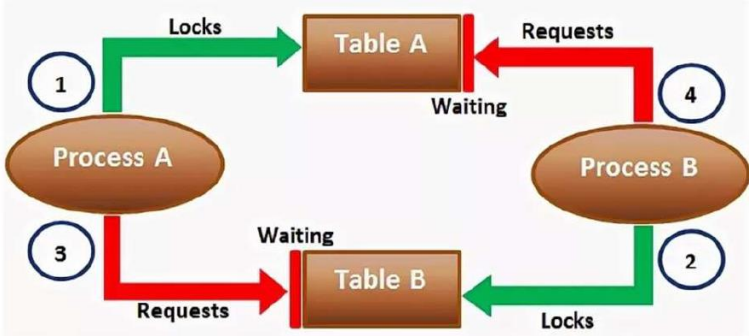
2019-09-26 12:11:47.754885 UTC,"gpadmin","adb",p20841,th1014880384,"[local]",,2019-09-26 12:05:12 UTC,0,con1771,cmd6,seg-1,,dx1600,,sx1,"LOG","00000","process 20841 still waiting for ExclusiveLock on relation 941105 of database 16384 after 1000.091 ms",,,,,,"update sales set id2 = 122 where id1=2;",8,,"proc.c",1212,

Лабораторная работа. Блокировки

1. Убедитесь, что таблицы table5 и table6 присутствуют в базе данных;
2. Откройте две транзакции;
3. В первой транзакции выполните update: update table5 set state='lock 1 transaction 1' where id=1;
4. Во второй транзакции выполните select: select * from table5;
5. Выполнилась ли операция Select?
6. Во второй транзакции выполните update: update table5 set state='lock 2 transaction 1' where id=1;
7. Выполнилась ли операция Update? Почему?
8. В первой транзакции выполните COMMIT;
9. Проверьте состояние операции Update во второй транзакции. Выполнилась ли эта операция? Почему?
10. Во второй транзакции выполните COMMIT;

Global Deadlock Detector

- Позволяет выполнять в разных транзакциях операции `UPDATE`, `DELETE` и `SELECT...FOR UPDATE` для одних и тех же `HEAP`-таблиц. Уровень блокировки меняется на `Row Exclusive`.
- Для включения необходимо поменять значение параметра `gp_enable_global_deadlock_detector`.
- Если обнаружится взаимная блокировка, то одна из транзакций откатится (та, которая вызвала дедлок).



- При наличии взаимной блокировки возникнет ошибка.
- Настройка СУБД `deadlock_timeout` определяет таймаут для разрешения ситуации (1 секунда по умолчанию).
- Для разрешения ситуации одна из транзакций прерывается.

Лабораторная работа. Global Deadlock Detector

Включите global deadlock detector и перезапустите кластер, выполнив команды под gpadmin в операционной системе мастер-сервера:

- `gpconfig -c gp_enable_global_deadlock_detector -v on –masteronly`
- `gpstop –sar`

После этого вызовите взаимную блокировку:

1. Откройте две транзакции.
2. В первой транзакции выполните update: `update table6 set state= 'lock 1 transaction 1' where id=1;`
3. Во второй транзакции выполните update: `update table5 set state='lock 1 transaction 2' where id=1;`
4. В первой транзакции выполните update: `update table5 set state='lock 2 transaction 1' where id=1;`
5. Во второй транзакции выполните update: `update table6 set state='lock 2 transaction 2' where id=1;`
6. Убедиться, что сработал механизм разрешения взаимных блокировок.

Конец третьей части