МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ

ФГБОУ ВО «Уральский государственный экономический университет»

В.П. Часовских

Формализация информации и BigData

02.03.03 - Математическое обеспечение и администрирование информационных систем, направленность (профиль) - разработка и администрирование информационных систем

СУБД для информационных систем цифровой экономики

Екатеринбург 2023

1. СУБД для информационных систем цифровой экономике

Администрирование ИС цифровой экономики и прежде всего для сквозной цифровой технологии Big Data определяется применяемой СУБД. В учебном пособие будем рассматривать СУБД Greenplum и SQL Server 2019.

1.1. Основные сведения о СУБД Greenplum

СУБД Greenplum – open-source продукт, массивно-параллельная реляционная СУБД для хранилищ данных с гибкой горизонтальной масштабируемостью и столбцовым хранением данных на основе PostgreSQL.

Благодаря своим архитектурным особенностям и мощному оптимизатору запросов, СУБД Greenplum отличается особой надежностью и высокой скоростью обработки SQL-запросов над большими объемами данных, поэтому эта MPP-СУБД широко применяется для аналитики Big Data в промышленных масштабах.

Эксплуатация системы началась в 2005 г. фирмой США «Greenplum». В 2018 году компания РФ «Arenadata» разработчик первого отечественного дистрибутива Apache Hadoop (программный проект с открытым исходным кодом, предназначенный для эффективной обработки больших пакетов данных), выпустила собственную MPP-СУБД Arenadata DB на основе Greenplum, адаптировав ее для корпоративного использования.

До недавнего времени распространённой структурой системами для аналитической обработки больших объемов данных являются SPM (symmetric multiprocessing) или системы симметричные мультипроцессорной архитектуры показана на рис. 1.

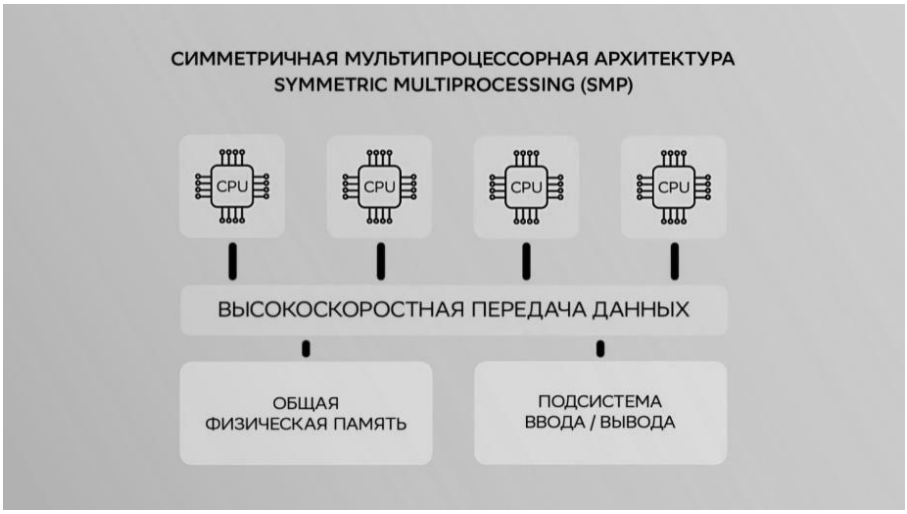


Рис. 1. SMP архитектура

Главная особенностью систем с такой архитектурой является наличие общих физических ресурсов, которые разделяются между несколькими процессорами сравнимой производительности.

Память служит, в частности, для передачи сообщений между процессорами и при этом все вычислительные устройства при обращение к памяти имеют равные права поэтому структура называется симметричной.

Большинство популярных СУБД таких как Oracle, Sybase и PostgreSQL , реализованы именно в такой архитектуре.

PostgreSQL принято обозначать

SMP - системы обладают рядом преимуществ, таких как высокая скорость обмена данными между процессорами за счет наличия общей памяти, простота и универсальность обслуживания и относительно невысокая цена.

Существенным недостатком таких систем является плохая масштабируемость.

Каждый раз, когда необходимо увеличить скорость обработки данных, нам необходимо увеличивать мощность сервера за счет увеличения числа процессоров.

Как правило, данная операция является дорогостоящей, а в некоторых случаях и вовсе невозможно из-за физических ограничений в конфигурации сервера.

Для решения проблемы масштабируемости в системах аналитической обработки данных стали применять архитектуру MPP, или архитектуру массивно-параллельной обработки данных, показанную на рис.2.

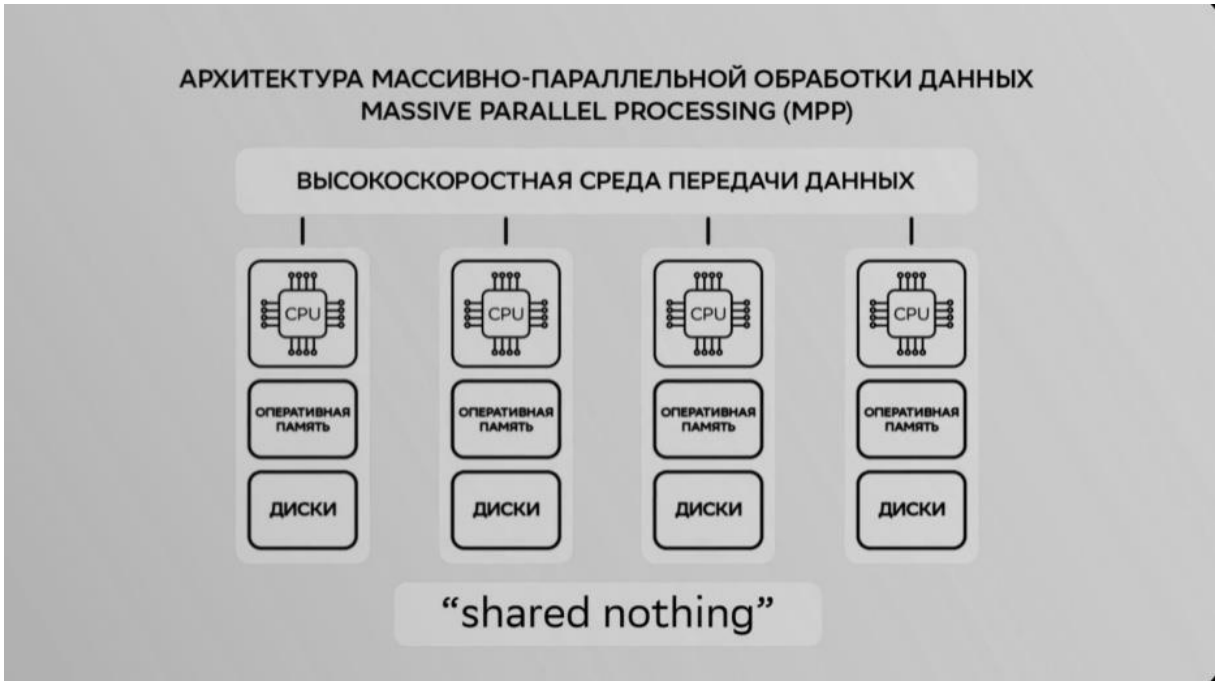


Рис.2. MPP - архитектура

В такой архитектуре система состоит из нескольких независимых узлов, соединенных по сети. При этом в каждом вычислительном узле процессор обладает своими собственными физическими ресурсами, такими как память и диски, которые не разделяются с другими узлами. Именно поэтому такая архитектура также называется Shared Nothing (Ничего общего).

В MPP -системах вычислительная мощность и объем хранения данных увеличиваются за счет добавления в систему дополнительных вычислительных узлов.

Данный подход позволяет достигнуть линейный рост производительности и в зависимости от количества узлов в системе.

СУБД Greenplum переназначена до хранения и обработки больших объемов данных методом распределения данных и обработки запросов на нескольких серверах.

Данная СУБД лучше всего подходит для построения корпоративных хранилищ данных, решение аналитических задач и задач машинного обучения и искусственного интеллекта.

В основе СУБД Greenplum лежит СУБД PostgreSQL.

По сути СУБД Greenplum представляет из себя множество модифицированных экземпляров дисковых баз данных PostgreSQL, работающих совместно как одна связанная система управления базами данных.

в большинстве случаев похож на PostgreSQL, например, в части синтаксиса SQL, функции параметров, конфигурации и функциональности для конечного пользователя. Пользователи базы данных взаимодействуют с СУБД Greenplum также как с обычной СУБД PostgreSQL.

Внутреннее устройство СУБД PostgreSQL было изменено или дополнено для поддержки параллельной структуры баз данных СУБД Greenplum.

Например, компоненты системного каталога, оптимизатора, исполнителя запросов и диспетчера транзакции были изменены и улучшены, чтобы они могли выполнять запросы одновременно в параллель во всех экземплярах базы данных.

СУБД Greenplum является продуктом с открытым кодом, который развивается не только основной компании разработчикам, но и другими участниками.

На основе открытого исходного кода многие компании делают свои собственные сборки и продают коммерческие версии СУБД, добавляя в них свои компоненты и осуществляя поддержку пользователей.

СУБД Greenplum является исключительно программным решением. Он поставляется в виде дистрибутива, который может быть установлен на различные серверные платформы. Производительность напрямую зависит от оборудования, на котором он установлен.

СУБД Greenplum предоставляет широкий выбор инструментов для решения аналитических задач, показанных на рис. 3.

Рис. 3. Типы и формат данных, поддерживаемых СУБД Greenplum

СУБД Greenplum обладает большим количеством различных коннекторов, которые предоставляют доступ к другим источникам данных, таким как другие СУБД-компоненты экосистемы Hadoop и стриминговые источники данных, например, Кафка, показанные на рис.4.

Рис.4. Источник данных СУБД Greenplum

Hadoop — проект фонда Apache Software Foundation, свободно распространяемый набор утилит, библиотек и фреймворк для разработки и выполнения распределённых программ, работающих на кластерах из сотен и тысяч узлов.

Также в СУБД Greenplum можно разрабатывать хранимые процедуры не только на встроенном процедурном языке SQL, но и на многих других популярных языках программирования, которые будут компилироваться и выполняться внутри базы данных.

Также есть возможность разрабатывать процедуры, которые будут работать в контейнерах.

Такой подход позволяет сделать разработку безопасной путем изолирования исполняемого кода от операционной системы, на которой установлена СУБД. Языки программирования, поддерживаемые СУБД Greenplum показаны на рис. 5.

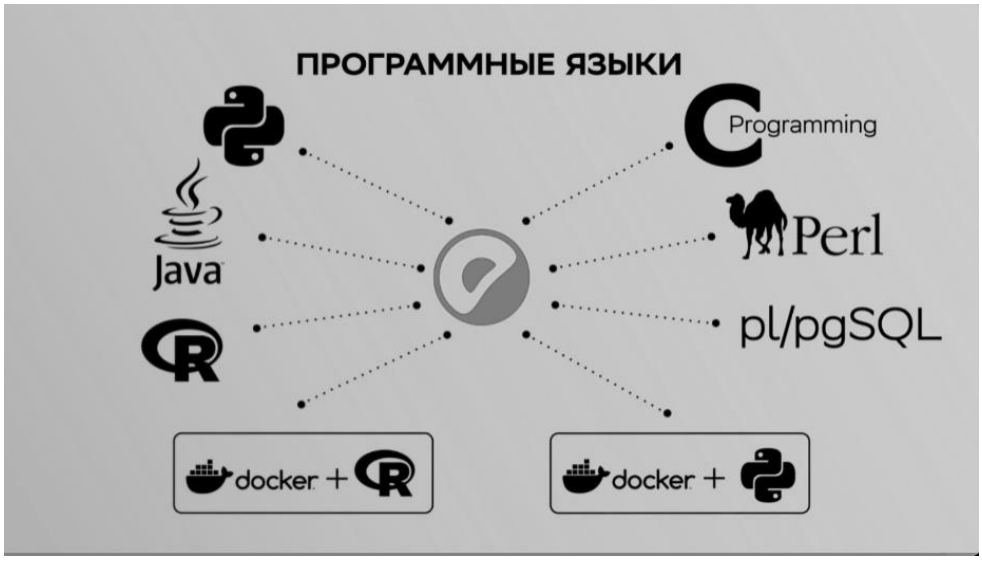


Рис.5. Языки программирования, поддерживаемые СУБД Greenplum

1.2. Архитектура СУБД Greenplum

СУБД Greenplum представляет из себя множество модифицированных баз данных пост, которые расположены на нескольких серверах и взаимодействуют друг с другом, образуя единую, цельную базу данных, как показано на рис. 6.

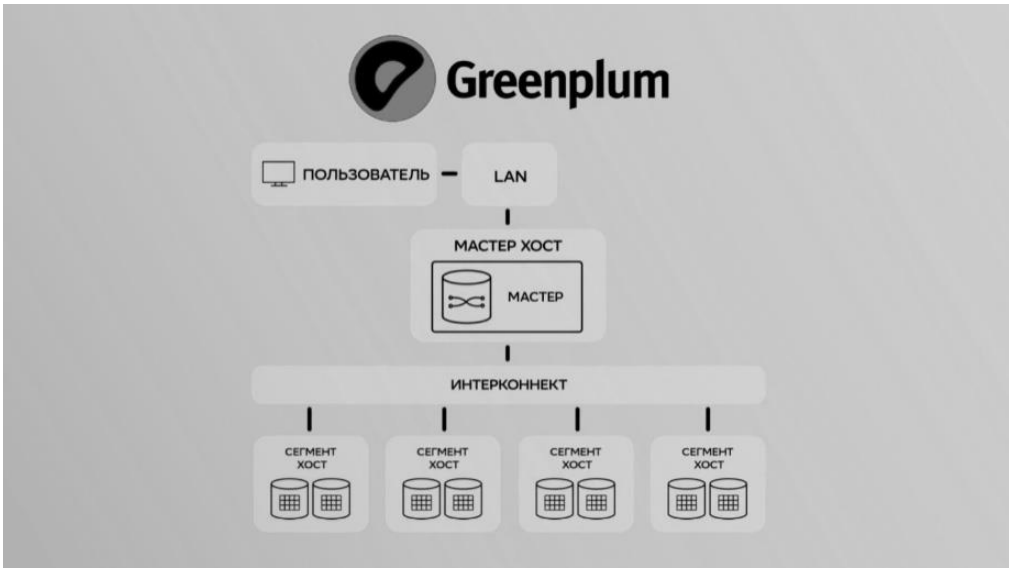


Рис. 6. Архитектура СУБД Greenplum

Логическая архитектура СУБД Greenplum выглядит следующим образом:

1. Мастер - это входная точка для СУБД Greenplum — это экземпляр базы данных СУБД PostgreSQL, к которому клиенты подключаются и отправляют свои SQL-запросы. Мастер координирует работу с другими экземплярами базы данных системы, которые называются «сегменты». Пользователи взаимодействуют с мастером также, как если бы это была обычная база данных СУБД PostgreSQL. Подключение может выполняться при помощи клиентских программ, таких как:

PSQL (SQL СУБД PostgreSQL);

JDBC (Java DataBase Connectivity — соединение с базами данных на Java);

ODBC (Open Database Connectivity - программный интерфейс (API) доступа к базам данных, разработанный компанией Microsoft).

На мастере располагается глобальный системный каталог — это набор системных таблиц, которые содержат метаданные о всех объектах базы данных СУБД Greenplum.

Важно отметить, что мастер не хранит каких-либо пользовательских данных, все пользовательские данные хранятся на сегментах. Функции

мастера заключаются в следующем: мастер выполняет авторизацию клиентских соединений, обрабатывает входящие SQL-команды, распределяет нагрузку между сегментами, координирует результаты, возвращаемое каждым сегментом, выполняет финальные операции над данными предоставляет конечный результат клиенту.

2. Сегменты — это отдельные экземпляры базы данных СУБД PostgreSQL, которые хранят определенную порцию пользовательских данных и выполняют большую часть обработки запросов.

Когда пользователь подключается к базе данных через мастер и запускает запрос. На каждом сегменте создаются процессы для обработки этого запроса. Затем каждый сегмент параллельно обрабатывает свою порцию данных и возвращает финальный результат своей работы на мастер-сегменты работают на серверах, которые называются сегмент-хосты или ноды.

На каждом ноде обычно располагается от двух до восьми сегментов Greenplum. Количество сегментов конфигурируется при первоначальной установке системы и напрямую зависит от профиля рабочих нагрузок системы и технических характеристик сервера, таких как количество и производительность ядер процессора, объема оперативной и постоянной памяти и сетевых интерфейсов.

Предполагается, что все ноды системы будут настроены идентично ключом к достижению максимальной производительности от СУБД Greenplum, является равномерное распределение данных и рабочих нагрузок по большому количеству сегментов с одинаковой производительностью. Это необходимо для того, чтобы все сегменты одновременно начинали работать над задачей и одновременно завершали свою работу. В этом случае ни один из узлов не станет узким горлышком всей системы.

3.Интерконнект это сетевой слой архитектуры СУБД Greenplum. Интерконнект обеспечивает взаимосвязь между мастером-сегментами и сетевой инфраструктурой, в которой развернут кластер.

Для обеспечения высокой производительности системы рекомендуемая пропускная способность сети должна быть не менее 10.0 гигабит.

По умолчанию обмен данными происходит по протоколу UDP (User Datagram Protocol — протокол пользовательских блоков информации; один из ключевых элементов набора сетевых протоколов для Интернета).

Также программное обеспечение СУБД Greenplum выполняет дополнительную проверку пакетов поверх UDP. Таким образом, надежность остается такой же, как у TCP (Transmission Control Protocol - один из основных протоколов передачи данных Интернета). При этом производительность и возможности масштабирования намного выше.

1.3. Дистрибьюция

Дистрибьюция или распределении данных — это один из важнейших факторов быстродействия системы. В СУБД Greenplum общее время выполнения запроса измеряется временем выполнения на всех сегментах. Так как обработка на сегментах выполняется в параллель то максимальная скорость системы ограничивается работой самого медленного сегмента. Если данные распределены неравномерно, сегменты с большим количеством данных будут выполнять работу дольше, что снижает общую производительность. Таким образом, для повышения производительности необходимо стремиться к равномерному распределению данных по всем сегментам системы, рис. 7.

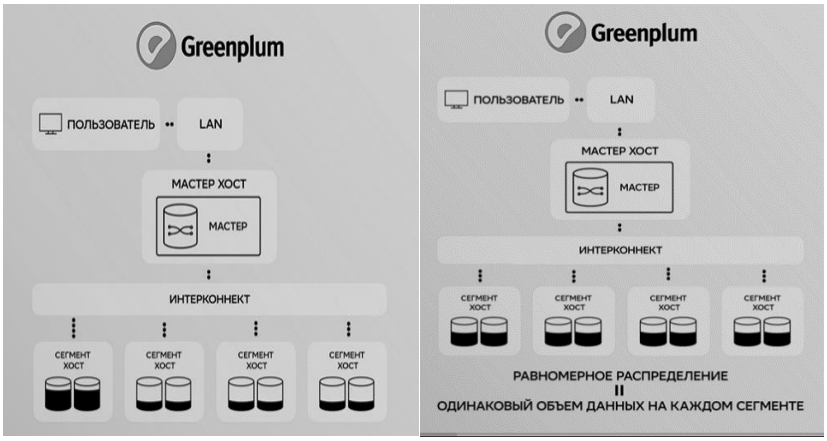


Рис. 7. Неравномерное и равномерное распределение данных по сегментам

Понимание и правильное использование такой ключевой особенности СУБД Greenplum, как распределение данных по всем сегментам с учетом особенностей ваших данных поможет вам добиться наилучшей производительности. Даже одна таблица в СУБД Greenplum хранится не на одном, а на нескольких сегментах. Каждый сегмент хранит уникальную порцию данных. Для каждой таблицы задается своя политика дистрибьюции. Рассмотри пример на рис.7.

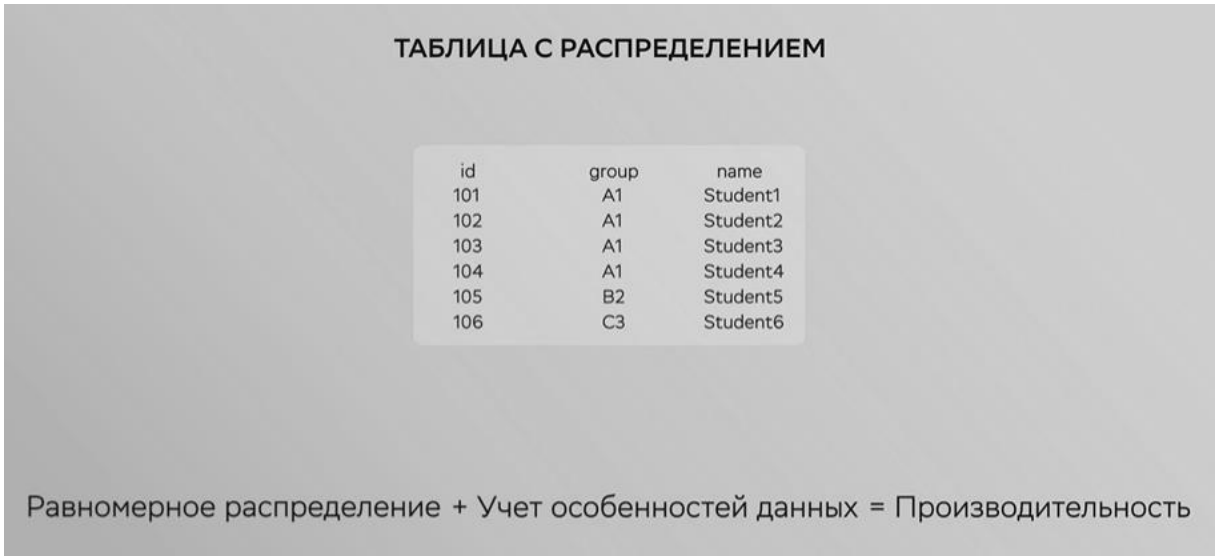


Рис. 7. Таблица с распределением

Способ, согласно которому строки распределяются по сегментам задается при создании таблицы в команде Create Table, а также может быть изменен впоследствии командой Alter Table.

Ключ дистрибьюции — это поле или набор полей, по значениям которых определяется, на каком сегменте будет храниться существующая строка таблицы.

При выборе ключа дистрибьюции необходимо учитывать 2 основных момента:

Ключ должен обеспечивать равномерность распределения данных;

Добиться локальности операций.

Под локальными операциями подразумевается ситуации, когда соединяемые таблицы имеют равный ключ дистрибьюции, что позволяет выполнять соединение локально на сегментах без необходимости перераспределять данные, что, в свою очередь, дает существенное ускорение выполнения запроса СУБД Greenplum.

Существует 3 вида дистрибьюции:

- distributed by;
- distributed randomly;
- distributed replicated.

Distributed by — это дистрибьюция по hash указанных полей. Идея соединения с помощью хеширования состоит в поиске подходящих строк с помощью заранее подготовленноё хеш-таблицы.

Хеш-таблица позволяет сохранять пары, составленные из ключа хеширования и значения, а затем искать значения по ключу за фиксированное время, не зависящее от размера хеш-таблицы.

В нашем примере из значения колонки или нескольких колонок вычисляется хэш-значения, по которому в дальнейшем определяется, на какой сегмент попадает запись. В нашем примере вычисления хеш-значения осуществляется по значениям одной или двух колонок.

Distributed randomly - это метод, при котором планировщик сам равномерно размещает данные по всем сегментам. Данный метод лучше всего подходит для таблиц, в которых отсутствует или сложно определить ключ, который обеспечит равномерное распределение данных. Важно

учесть, что при таком алгоритме соединения с другими таблицами будет всегда вызывать перераспределение данных, так как строки распределены случайным образом.

Distributed replicated – это метод, при котором на каждом сегменте хранится полная копия таблицы.

Такой метод подходит для небольших таблиц, например, справочников, которые регулярно участвуют в соединениях с другими таблицами.

Хранение полной копии таблицы на каждом сегменте позволяет сэкономить время, которое в противном случае было бы потрачено на перераспределение во время выполнения запроса.

При выборе ключа дистрибьюции необходимо руководствоваться следующими правилами:

- в качестве ключа дистрибьюции необходимо выбирать поле с большой селективностью, то есть поля с большим количеством уникальных значений. Как правило, под этот критерий подходит поля с идентификаторами. Если данные распределены по ключу с низкой селективностью, то велика вероятность неравномерного распределения данных, так как одинаковые значения будут располагаться на одном сегменте;

- в ключе не должно быть Null или значений по умолчанию. Иначе все Null или другие значения по умолчанию будут попадать на один сегмент, что приведет к перекосу данных.

Рассмотрим примеры рис. 7.

Выберем в качестве ключа дистрибьюции поле ID. В результате распределения на каждый сегмент падает ровно по две строки. Распределение получается равномерное, а значит, ключ дистрибьюции хороший.

Выберем в качестве ключа дистрибьюции поле group. На рис. 8 показаны результаты.

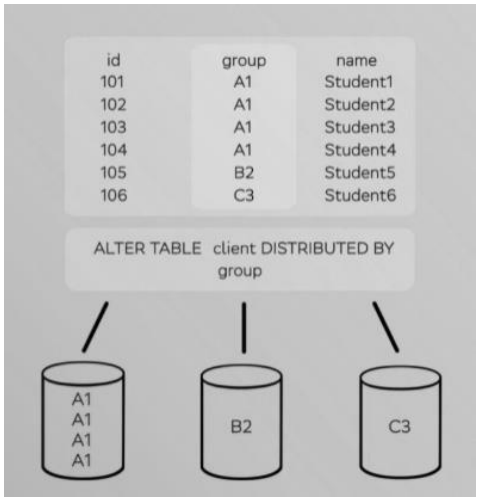


Рис. 8. Ключ дистрибьюции поле group

После распределения мы видим, что на одном из сегментов данных в 4 раза больше, чем на других. Как следствие, и работать этот сегмент будет в 4 раза дольше. Делаем вывод о том, что ключ выбран не оптимально. Проверить равномерность распределения строк между сегментами можно, используя служебный столбец GP сегмент ID, который присутствует в каждой таблице

SELECT gp_segment_id, count (1)
FROM sales
GROUP BY gp.segmentjd

В данном столбце содержится идентификатор сегмента, на котором лежит строка пример запроса с группировкой по полю GP сегмент ID для определения равномерности распределения строк по сегментам, приведен на экране. По результату запроса видно, что распределение данных таблицы Sails практически равномерное, рис. 9.

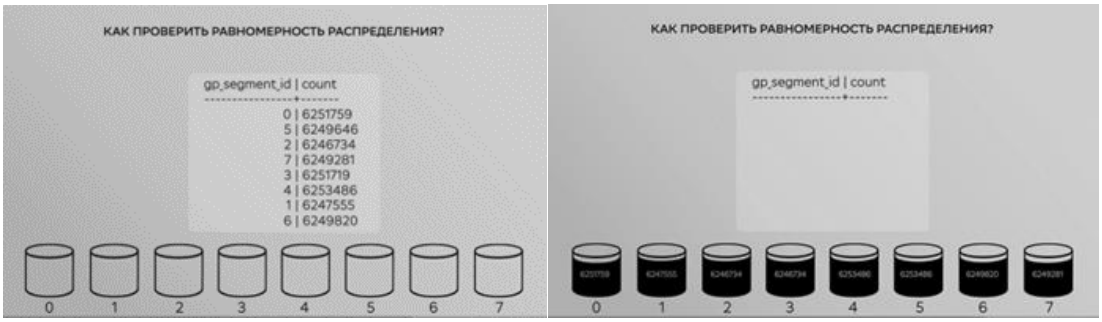
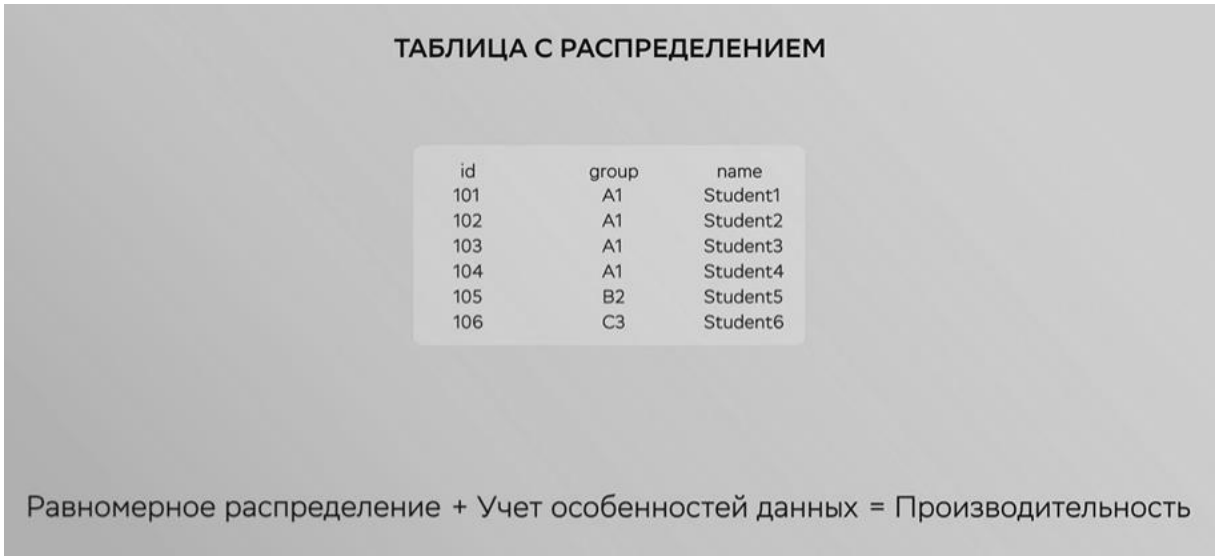


Рис. 9. Равномерное распределение данных таблицы



Ключ распределения можно изменить без пересоздания таблицы с помощью предложения

ALTER TABLE tabi SET DISTRIBUTED BY (coll)

ALTER TABLE tab! SET DISTRIBUTED RANDOMLY

Также есть возможность перераспределения данных по прежнему ключу дистрибьюции при помощи опции

ALTER TABLE tabi SET WITH (REORGANIZE=true)

Эта опция будет полезна в двух случаях: при добавлении нового сегмента и для избежания раздутия таблиц, результат на рис. 10.

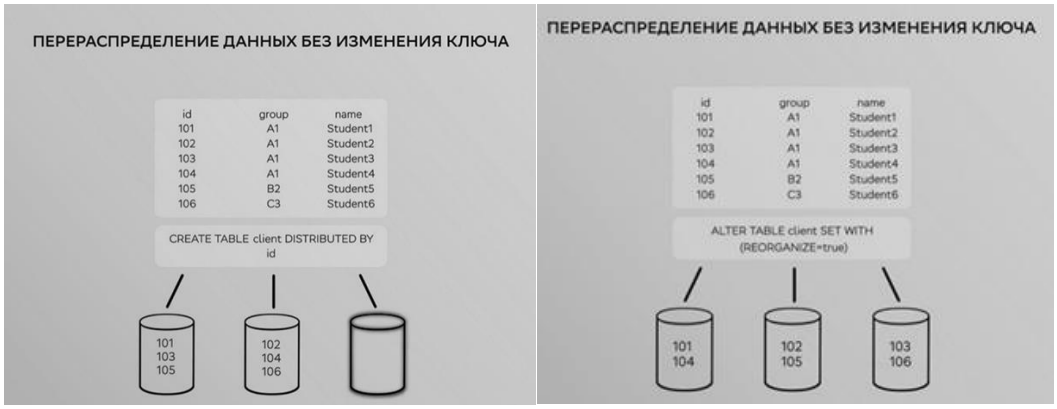


Рис. 10. Перераспределение данных без изменения ключа.

При добавлении нового сегмент хоста в кластер возникает необходимость заново перераспределить данные, чтобы избежать перекоса

в распределении данных. Когда новый сегмент фактически не задействован для хранения данных, а все данные по-прежнему распределены по старым сегментам.

Некоторые важные рекомендации, которые помогут вам правильно распределять данные:

- явно задавайте способ и ключ распределения при создании таблицы, в том числе для временных таблиц;
- используйте RANDOMLY-распределение для небольших таблиц, где нет хороших кандидатов на ключ распределения из одного атрибута;
- избегайте использования в качестве ключа распределения атрибутов, которые будете часто указывать в предложении WHERE и атрибутов с типом данных date, timestamp;
- старайтесь использовать в качестве ключа распределения тип данных integer (предпочтительнее, чем string) и столбец, который будете использоваться как ключ соединения (JOIN ON).

1.4. Партиционирование

Партиционирование, или секционирование, представляет собой логическое разделение таблицы на части по определенному критерию.

Применение партиционирования на больших таблицах значительно улучшает производительность запросов, а также упрощает сопровождение базы данных благодаря возможности разделять перемещать и компактно хранить данные в зависимости от их востребованности.

Партиционированные таблицы, также, как и любые таблицы в СУБД Greenplum, имеют ключ дистрибьюции.

Важно различать понятия: **дистрибьюция** - это распределение данных по сегментам с целью равномерно распределить нагрузку по всем узлам кластер; **партиционирование** - это метод оптимизации, при котором на каждом сегменте таблица делится на части с целью увеличения

производительности запросов за счет выборочного сканирования. Различия дистрибьюции и партиционирования показано на рис. 11.

Рис. 11. Различия дистрибьюции и партиционирования

Партиционирование распространяется на все сегменты и выполняется на каждом сегменте одинаково.

Например, если таблица при создании поделена на 6 партиций, то на каждом сегменте кластера будет по 6 партиций.

Когда таблица партиционирована, запрос выполняется гораздо быстрее за счет того, что на каждом сегменте сканируется только часть таблицы.

Дистрибьюция в СУБД Greenplum обязательно для всех таблиц, то есть все таблицы в СУБД Greenplum являются распределенными и имеют ключ дистрибьюции.

В партиционирование - это лишь опция, которую можно применять к таблице в целях оптимизации.

В СУБД Greenplum поддерживаются следующие виды позиционирования:

PARTITION BY RANGE - по диапазону значений
PARTITION BY LIST – по списку значений
PARTITION BY ... SUBPARTITION BY - многоуровневое

Рассмотрим таблицу, содержащую записи о клиентах учебного заведения, например, студентах университета, рис. 12.

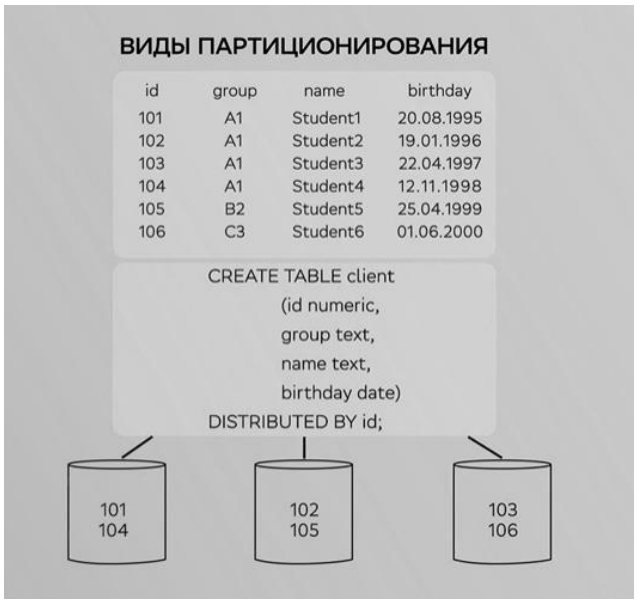


Рис. 12. Таблица – студенты университета. Создание таблицы.

Согласно заданной политике дистрибьюции, таблица равномерно распределена по сегментам на основе идентификаторов студентов.

Допустим, что для построения отчетности периодически требуется анализировать информацию о всех учащихся того или иного возраста. Вместе с тем мы хотим, чтобы система выдавала результаты запросов для разных возрастов как можно быстрее.

Эту задачу поможет решить Range. Партиционирование по году рождения, рис. 13.

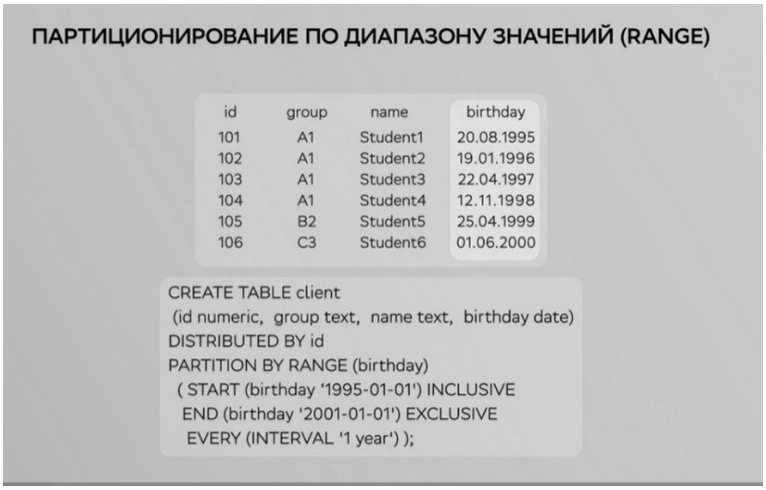


Рис. 13. Партиционирование по году рождения

Вид партиционирования задается при создании таблицы с помощью выражения Partition By Range. Диапазон значений задан выражением start и **end**, а шаг каждой партийцы — выражением **every** интервал. После создания таблицы на каждом сегменте будут созданы по 6 партийцы и хранящих записей по году.

Если в запросе к таблице есть условия фильтрации записи и погоду, то база данных в рамках каждого сегмента сразу обращается к той партийцы, которая соответствует нужному году, в связи с чем запрос выполняется намного быстрее, рис. 14.

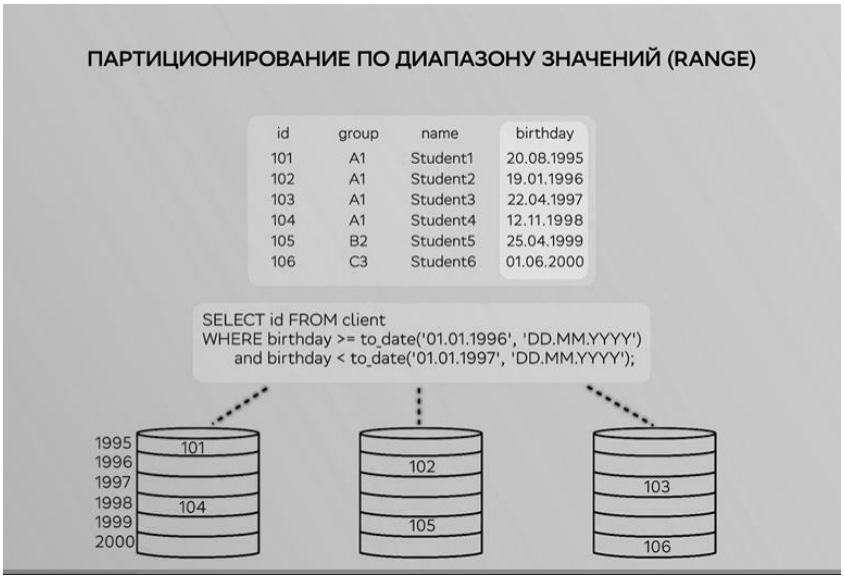


Рис. 14. Партиционирования по диапазону значений

В случае партиционирования типа List разделения данных выполняется на основе списка значений.

Например, мы можем применить к той же таблице партиционирование по полю «группа студентов».

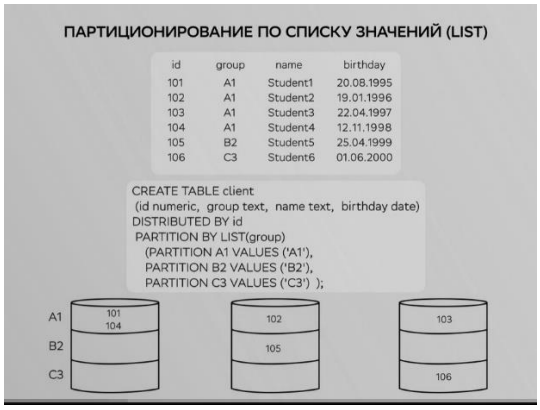


Рис. 15. Партиционирование по списку значений

Разделение партийцы и по значениям выполняется с помощью выражения **Partition Values**.

На каждом сегменте создается по 3 партийцы. Для групп A1, B2 и C3. Внутри каждой партийцы есть студенты с разными годами рождения. Например, запись о студенте 4 хранится в сегменте 1. «» в партийцы A1. Этот способ партиционирования будет удобен, если требуется часто строить запросы с фильтрацией по группе студентов, то, кроме того, в одной таблице можно реализовать многоуровневое партиционирование, при котором будут использоваться комбинации обоих типов. Проводится по группе студентов, а внутри группы — по году рождения, рис. 16.

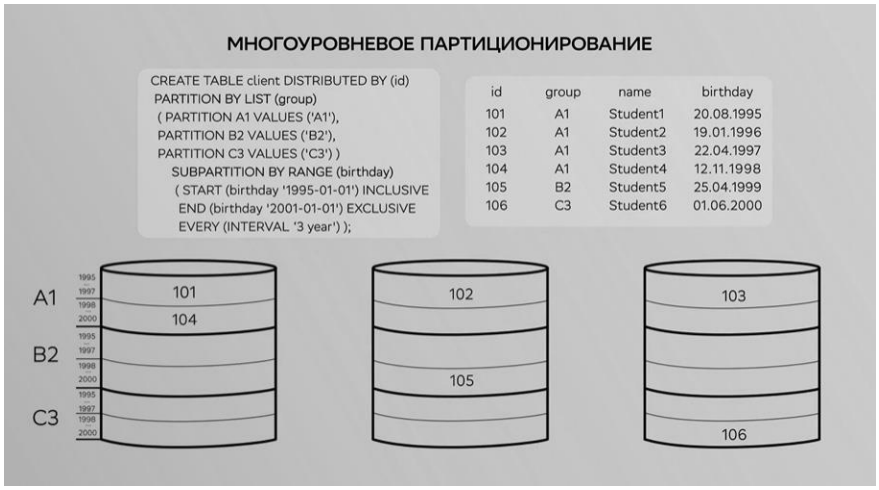


Рис. 16. Многоуровневое партиционирование

Имейте в виду, что использование многоуровневого партиционирования может привести к созданию избыточного количества партийцев, что, в свою очередь, может замедлить работу всей базы данных.

В при создании партиционированной таблицы можно указывать

- Отличающиеся имена;
- Отличающиеся диапазоны;
- Разные опции хранения (ориентация, компрессия);
- Можно создать партицию по умолчанию (DEFAULT) для значений, не удовлетворяющих условиям других партиций.

Нельзя:

- Указать разные ключи дистрибьюции для партиций одной таблицы;
- Поменять тип партиционирования без пересоздания таблицы

Рассмотрим пример, рис. 17 стратегии партиционирования с применением разных опций хранения данных.

Рис. 17. Стратегия партиционирования с применением разных опций хранения данных.

Наиболее свежие данные хранятся без компрессии для того, чтобы можно было быстро выполнять операции обновления.

Старые данные хранятся с более сильными алгоритмами сжатия. Например, для данных трехмесячной давности применяется низкая степень сжатия, что обеспечивает хорошую производительность, так как выполняется мало операций ввода-вывода.

Для данных годичной давности применяется колоночная ориентация с более высокой степенью компрессии, что повышает производительность при выборке по подмножеству колонок.

Для данных глубиной 2 года используется колоночная ориентация и максимальная компрессия. При этом производительность ниже, так как при выполнении запросов строки приходится разжимать.

Архивные данные хранятся за пределами базы данных в виде файлов.

Обращаем внимание, что цель партиционирования состоит в исключении чтения лишних партийций, то есть минимизации количества сканируемых партиций при выполнении запроса.

Следовательно, наиболее эффективным будет такое позиционирование, которое максимально приближено к логике выполнения запроса.

Последовательность операции при выполнении запроса отражается в плане запроса, который выводится с помощью команды Explan.

Рассмотрим план выполнения запросов в случае таблицы с партиционированием или без него, рис. 18.

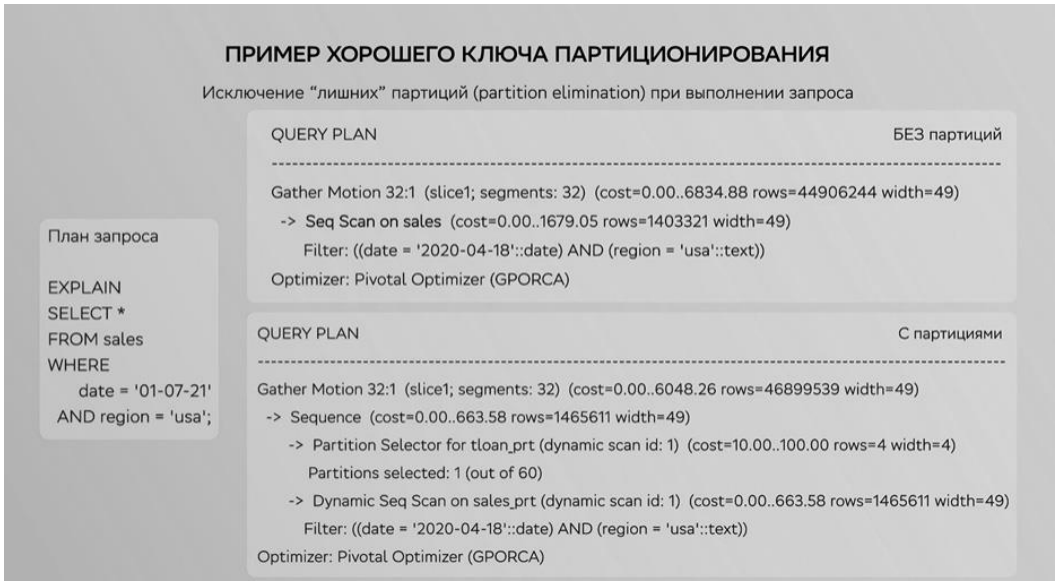


Рис. 18. План выполнения запросов при партиционирование

В плане запроса для таблицы без партиционирования сканируется вся таблица, поэтому стоимость выполнения запроса получается высокой. Для таблицы с партиционированием мы видим, что сканируется только одна нужная партиция, в связи с чем стоимость запроса низкая, и, как следствие, запрос выполняется намного быстрее.

Рассмотрим синтаксис запросов - для разных партиций можно указывать разные интервалы и опции хранения.

На рис.19 для партийцев с интервалом год или 3 месяца задана колоночная ориентация хранения и тип компрессии залип с разными коэффициентами сжатия.

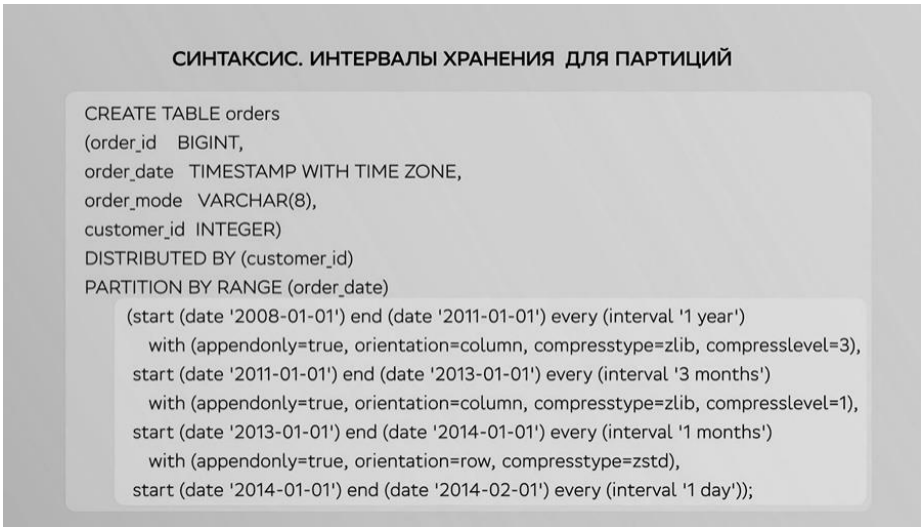


Рис.19. Синтаксис запроса

Для партийцы с интервалом один месяц применяется ориентация по строкам и другой тип компрессии без явного указания коэффициента сжатия.

Для партийцы с интервалом один день используется способ хранения по умолчанию.

Рассмотрим синтаксис изменение партийцы, рис. 20.

Рис. 20. Добавление партиции.

Для изменения партийцы используется команда **Alter table** с дополнительными опциями.

ALTER TABLE sales RENAME PARTITION FOR ('2016-01-01') TO jan16;

Очистка партиции.

ALTER TABLE table.prtmixed TRUNCATE PARTITION year2019;

Очистка партиции.

ALTER TABLE tabLe.prt.mixed TRUNCATE PARTITION year2019;

ALTER TABLE tabLe.prt.mixed TRUNCATE PARTITION FOR ('2019-04-28'::date);

Для очистки партийцы и применяется команда Trancate Partition. В примере показано, как очистить портицу с названием яр 2019.0 или партийцу, содержащую дату 28.0 апреля 2019.0 года.

Удаление партиции.

ALTER TABLE cardUst DROP PARTITION n.a;

ALTER TABLE card.list DROP DEFAULT PARTITION;

Удаление партиции.

ALTER TABLE cardUst DROP PARTITION n.a IF EXIST;

ALTER TABLE card.list DROP DEFAULT PARTITION IF EXIST;

Для удаления партиции используется команда DROP PARTITION.

В примере показано удаления обычной и дефолтной партии. При желании в тексте команды можно указать опцию IF EXIST, которая поможет избежать сообщения об ошибке при выполнении команды в случае отсутствия удаляемой партиции. Для изменения опции хранения в отдельной партии можно создать таблицу с нужной опцией хранения, вставить в нее данные из этой пардиции и затем заменить всю прежнюю партицию на только что созданную таблицу с помощью команды ALTER TABLE.

Разделение партиции.

ALTER TABLE sales SPLIT PARTITION FOR ('2021-01-01') AT ('2021-01-16')

INTO (PARTITION jan21.1to15, PARTITION jan21.16to31);

```
ALTER TABLE sales SPLIT DEFAULT PARTITION START ('2021-01-01')
INCLUSIVE
END ('2021-02-01') EXCLUSIVE
INTO (PARTITION jan21, default partition);
CREATE TABLE sales (trans.id int, date date, amount decimal(9,2), region
text) DISTRIBUTED BY (trans.id) PARTITION BY RANGE (date)
SUBPARTITION BY LIST (region) SUBPARTITION TEMPLATE
( SUBPARTITION usa VALUES ('usa'), SUBPARTITION asia VALUES
('asia'), SUBPARTITION europe VALUES ('europe'), DEFAULT
SUBPARTITION other.regions) ( START (date '2014-01-01') INCLUSIVE
END (date '2014-04-01') EXCLUSIVE EVERY (INTERVAL '1 month'));
```

Для разделения одной партиции на две части используется команда ALTER TABLE … SPLIT PARTITION. В начале примера выполняется разделение партийцы, в которой хранятся данные за январь, на две партии. В первой будут храниться данные с первого по 15 января, а во второй — с шестнадцатого по 31, если в вашей таблице есть дефолт-партийцы, то добавить новую партицию в такой таблице можно только путем разделения дефолт партийцы, как это показано во втором части. В этом случае в предложении INTO в качестве второй позиции необходимо указать дефолтпартицию.

ПРИМЕРЫ DDL(Data Definition Language**)-ОПЕРАЦИЙ.**

Создание и изменение шаблонов субпартиций.

```
ALTER TABLE sales ADD PARTITION "4"
START ('2014-04-01') INCLUSIVE
END ('2014-05-01') EXCLUSIVE ;
```

При создании таблицы возможно определение шаблона, по которому будут автоматически создаваться наборы субпартийций при добавлении новых партийций.

Создание и изменение шаблонов субпартиций.

```
ALTER TABLE sales ADD PARTITION "4"
START ('2014-04-01') INCLUSIVE
END ('2014-05-01') EXCLUSIVE ;
ALTER TABLE sales SET SUBPARTITION TEMPLATE ( SUBPARTITION usa VALUES ('usa'), SUBPARTITION asia VALUES ('asia'), SUBPARTITION europe VALUES ('europe'), SUBPARTITION africa VALUES ('africa'), DEFAULT SUBPARTITION regions);
```

В приведенном скрипте создания шаблон определяется с помощью предложения SUBPARTITION TEMPLATE, в котором мы определяем 4 субпартийца и для разных регионов. Теперь, если мы добавляем в таблицу партийцу и следующим скриптом, то эта партийция по шаблону автоматически разделится на 4 субпартийцы и для всех регионов.

Просмотр информации о партициях.

Информация о ключах партиционирования

```
SELECT * FROM pg_catalog.pg_partitions WHERE tablename = 'sales';
```

С помощью представления о pg Partitions можно получить подробную информацию о партиционированных таблицах и их иерархии.

Представление pg Partition Colance отображает информацию о колонках, которые используются для партиционирования.

Рекомендации по партиционированию данных.

Когда использовать.

Большая таблица.

Фильтры запросов (WHERE) исключают часть партиций.

Нужно "скользящее окно" данных при периодической загрузке архивировании.

Рекомендации по партиционированию данных

Как использовать.

Нужно:

1. Предпочитать RANGE, а не LIST.
2. Партиционировать по часто используемому столбцу.
3. Убедиться в исключении ненужных партиций в плане запроса (EXPLAIN).
4. Выбирать наилучший способ физического хранения для разных партиций (например, по строкам/колонкам).

Не рекомендуется:

1. Создавать большое количество партиций.
2. Использовать дефолтную партицию.
3. Использовать многоуровневое партиционирование.
4. Секционировать по столбцу ключа дистрибьюции.
5. Создавать много партиций при колоночной ориентации (количество физических файлов = сегменты * столбцы * партиции).