

УТВЕРЖДЕН
643.72410666.00067-07 99 01-ЛУ

ЗАЩИЩЕННАЯ СИСТЕМА УПРАВЛЕНИЯ
БАЗАМИ ДАННЫХ «ЈАТОВА»

Инструкция по миграции компонента «ja_Hipe_Cluster».

643.72410666.00067-07 99 02

Листов 25

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

АННОТАЦИЯ

Настоящий документ является инструкцией-дополнением к существующему руководству по настройке «Компонент ja_Hipe_Cluster. Горизонтальное масштабирование. Часть 11» и описывает рекомендации и процедуры по миграции на новую версию компонента .

Настоящее руководство предназначено для администраторов СУБД, специалистов по информационной безопасности и носит рекомендательный характер.

Документ применяется при работе с кластерами ja_Hipe_Cluster версии 10.x и выше, а также при миграции СУБД «Jatoba» с версии 5 на 6.

Степени важности примечаний, применяемые в данном документе:



Важная информация – указания, требующие особого внимания



Дополнительная информация – указания, позволяющие упростить работу с изделием



Все примеры в данном документе приведены для СУБД «Jatoba» версии ядра 6.x, для других версий все шаги выполняются аналогично, разница состоит в именах директорий.

Например, СУБД «Jatoba» версии 6.x по умолчанию устанавливается в директорию:

- ОС Windows – «C:\Program Files\GIS\Jatoba\6\bin»;
- ОС Linux – «/usr/jatoba-6/bin».



Важная информация

Для сертифицированной версии СУБД «Jatoba» поддерживается работа только на ОС, указанных в формуляре на поставку!

СОДЕРЖАНИЕ

1. Актуальность версий	4
2. Обновление компонента	5
2.1. Обновление пакета компонента.....	5
2.2. Обновление расширения БД	5
3. Переход на новую версию СУБД «Jatoba».....	7
3.1. Подготовительные процедуры.....	7
3.2. Создание резервной копии метаданных	7
4. Миграция существующего приложения на работу с компонентом ja_Hipe_Cluster	10
4.1. Определение типа таблиц	10
4.1.1. Локальные таблицы	10
4.1.2. Справочные или глобальные таблицы	10
4.1.3. Распределенные таблицы	11
4.2. Совместное размещение таблиц	12
4.3. Выбор ключа распределения	12
4.3.1. Распределение по первичному ключу.....	13
4.3.2. Распределение по внешнему ключу.....	13
5. Примеры миграции	15
5.1. Пример для распределения по первичному ключу	15
5.2. Пример распределения по внешнему ключу	16
5.2.1. Добавление ключей распределения	17
5.2.2. Заполнение повторно созданных столбцов	17
5.2.3. Включение столбца распределения в ключи	19
5.2.4. Добавление ключа распределения в запросы	20
Термины и определения	22
Перечень сокращений.....	24

1. АКТУАЛЬНОСТЬ ВЕРСИЙ

Используйте актуальную версию ПО СУБД «Jatoba» и компонента ja_Hipe_Cluster, регулярно проверяйте выпуск обновлений.

Чем старше версия используемого программного обеспечения, тем больше времени было у злоумышленников на то, чтобы найти в ней уязвимости и «эксплойты» и, соответственно, тем уязвимее будет информационная система.

Чтобы защитить системы от потенциальных угроз, необходимо своевременно устанавливать обновления безопасности, закрывающие известные уязвимости в программном обеспечении.

Для проверки используемой версии СУБД «Jatoba» можно воспользоваться командой в терминале ОС:

Пример команды

```
/usr/jatoba-6/bin/postgres --version
```

При подключении к СУБД, можно воспользоваться следующей функцией:

Пример функции

```
SELECT jatoba_version();
```

Для получения информации о версии компонента ja_Hipe_Cluster необходимо подключиться к БД, в которую он установлен, и выполнить запрос к системному каталогу:

Пример запроса

```
SELECT citus_version();
```

В командной строке psql можно воспользоваться метакомандой:

Пример метакоманды

```
\dx
```

2. ОБНОВЛЕНИЕ КОМПОНЕНТА



Обновление компонента ja_Hipe_Cluster выполняется перед обновлением СУБД «Jatoba».

2.1. Обновление пакета компонента



В целях защиты целостности данных в процессе обновления компонент ja_Hipe_Cluster приостанавливает выполнение распределённых запросов до перезапуска службы СУБД и выполнения команды ALTER EXTENSION.

После обновления компонента необходимо перезапустить СУБД и выполнить команду ALTER EXTENSION.

В редких случаях, если при обновлении возникнет ошибка, можно отключить проверку версии разделяемой библиотеки расширения и самого расширения с помощью параметра конфигурации citus.enable_version_checks.

Обновление пакета компонента ja_Hipe_Cluster осуществляется средствами пакетного менеджера ОС.

Для разных типов пакетных менеджеров команда обновления немного отличается. Ниже приведен пример для ОС на основе пакетного менеджера apt-get:

- 1) Обновить список доступных пакетов в репозитории операционной системы:

```
sudo apt-get update
```

- 2) Обновить пакет с расширением:

```
sudo apt-get install --only-upgrade jatoba6-ja-hipe-cluster
```

- 3) Перезапустить службу СУБД:

```
sudo systemctl restart jatoba-6
```

2.2. Обновление расширения БД

Необходимо обновить расширения ja_Hipe_Cluster в каждом экземпляре БД.

Для обновления версии расширения необходимо подключиться к БД с использованием psql и выполнить следующие команды:

- 1) Обновить расширение:

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

```
ALTER EXTENSION citus UPDATE;
```

2) Проверить версию расширения:

```
SELECT citus_version();
```

3) При обновлении до ja_Hipe_Cluster 11.x или более поздней версии на узле-координаторе выполнить:

```
CALL citus_finish_citus_upgrade();
```



При обновлении до ja_Hipe_Cluster 11.x с более ранней основной версии процедура `citus_finish_citus_upgrade()` обеспечит наличие правильной схемы данных и метаданных на всех рабочих узлах кластера. Выполнение процедуры обновления может занять несколько минут в зависимости от объёма метаданных, которые потребуют синхронизации.

3. ПЕРЕХОД НА НОВУЮ ВЕРСИЮ СУБД «ЈАТОВА»



Обновление компонента ja_Hipe_Cluster выполняется перед обновлением СУБД «Jatoba».

В случае, если применяется версия ja_Hipe_Cluster 10.0 или 10.1, во избежание возникновения ошибок и проблем, запрещается обновлять версию СУБД «Jatoba». Необходимо предварительно выполнить обновление версии компонента до 10.2 (см. раздел 2), а затем выполнить обновление СУБД «Jatoba».

3.1. Подготовительные процедуры

В переменных окружения ОС необходимо указать следующие пути и каталоги:

- Существующий каталог данных СУБД:

```
export OLD_PG_DATA=/var/lib/jatoba/5/data
```

- Существующий каталог установки СУБД:

```
export OLD_PG_PATH=/var/lib/jatoba/5
```

- Каталог данных СУБД после обновления:

```
export NEW_PG_DATA=/var/lib/jatoba/6/data
```

- Каталог установки СУБД после обновления

```
export NEW_PG_PATH=/var/lib/jatoba/6
```

3.2. Создание резервной копии метаданных



Приведенные далее процедуры выполняются на каждом узле высоконагруженного кластера, если не указано иное.

Создание резервной копии метаданных выполняется в следующей последовательности:

- 1) На узле-координаторе кластера подключиться к БД и выполнить:

```
SELECT citus_prepare_pg_upgrade();
```

2) Настроить новый экземпляр СУБД согласно документу «Руководство администратора» 643.72410666.00067-07 95 01.

3) В конфигурационном файле postgresql.conf указать общую библиотеку компонента для предварительной загрузки в СУБД:

```
shared_preload_libraries = 'citus'
```



На данном этапе запрещается создавать расширение компонента в новом экземпляре СУБД и запускать сервер СУБД

4) Остановить службу СУБД на старом сервере:

```
systemctl stop jatoba-6
```

5) На сервере с новой версией СУБД проверить совместимость при помощи следующей команды:

```
$NEW_PG_PATH/bin/pg_upgrade -b $OLD_PG_PATH/bin/ -B $NEW_PG_PATH/bin/ -d $OLD_PG_DATA -D $NEW_PG_DATA --check
```

Результатом выполнения команды должно быть сообщение “Clusters are compatible”. Если это не так, необходимо исправить ошибки, прежде чем продолжить. Следует убедиться в том, что:

- каталог, указанный в переменной окружения NEW_PG_DATA, содержит пустую базу данных, инициализированную новой версией СУБД «Jatoba»;
- расширение компонента не создано.

6) Обновить версию СУБД:

```
$NEW_PG_PATH/bin/pg_upgrade -b $OLD_PG_PATH/bin/ -B $NEW_PG_PATH/bin/ -d $OLD_PG_DATA -D $NEW_PG_DATA
```

7) Компонент ja_Hipe_Cluster имеет функционал настройки конфигурации TLS. Для этого необходимо скопировать конфигурационный файл и подготовленные сертификаты:

```
cp $OLD_PG_DATA/postgresql.auto.conf $NEW_PG_DATA/  
cp $OLD_PG_DATA/server.crt $NEW_PG_DATA/
```



```
cp $OLD_PG_DATA/server.key $NEW_PG_DATA/
```

8) Запустить новый экземпляр СУБД:

```
systemctl start jatoba-6
```



На данном этапе запрещается выполнять какие-либо запросы к новому экземпляру СУБД.

9) Подключиться к СУБД на новом узле-координаторе и выполнить восстановление метаданных на:

```
SELECT citus_finish_pg_upgrade();
```

4. МИГРАЦИЯ СУЩЕСТВУЮЩЕГО ПРИЛОЖЕНИЯ НА РАБОТУ С КОМПОНЕНТОМ JA_HIPE_CLUSTER

Компонент ja_Hipe_Cluster расширяет возможности СУБД «Jatoba» за счёт функциональности работы с распределенными таблицами, но шардирование само по себе не является тем решением, которое безусловно масштабирует все рабочие нагрузки.

С целью достижения желаемой производительности миграция существующего приложения на работу с компонентом ja_Hipe_Cluster может потребовать внесения изменений в схему данных и запросы.

4.1. Определение типа таблиц

В распределенном кластере компонента ja_Hipe_Cluster существует три вида таблиц:

- локальные;
- справочные;
- распределенные.

Особенности этих типов таблиц, их назначение, возможности и ключевые ограничения приводятся далее.

4.1.1. Локальные таблицы

Локальные таблицы – это обычные реляционные таблицы на узле-координаторе, для которых не применяются никакие механизмы шардирования. Компонент ja_Hipe_Cluster не меняет планы выполнения запросов, в которых участвуют только локальные таблицы и таким образом, функциональность таких таблиц полностью идентична функциональности, доступной в классической нераспределенной СУБД.

Основное назначение локальных таблиц заключается в обеспечении возможности частичного распределения данных. Зачастую даже очень большие приложения имеют всего несколько по-настоящему больших таблиц, требующих шардирования. По этой причине нет необходимости в существующем приложении распределять все таблицы и основную часть из них можно оставить без изменений.

4.1.2. Справочные или глобальные таблицы

Справочные или глобальные таблицы (reference table) – это относительно небольшие таблицы, полная физическая копия которых находится на каждом узле.

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

Чтение данных из справочных («глобальных») таблиц всегда выполнится локально – любой запрос на чтение к такой таблице будет выполняться без взаимодействия с другими узлами кластера.

В противоположность этому, запись в такие таблицы всегда глобальная и любая операция записи в такую таблицу потребует взаимодействия со всеми узлами кластера.

В результате, чтение из таких таблиц будет иметь предсказуемое линейное масштабирование, пропорциональное количеству узлов. Однако запись в справочные («глобальные») таблицы не масштабируется и деградирует с ростом числа узлов в кластере и показателей скорости сетевого взаимодействия между ними.

Исходя из указанного, основным назначением справочных («глобальных») таблиц является хранение небольших данных. Для этих данных, как правило, выполняются операции чтения и редко – операции записи. Можно считать, что операции записи в такие таблицы допустимы только в рамках некоторых административных операций и не допустимы как часть целевой пишущей нагрузки приложения.

4.1.3. Распределенные таблицы

Распределенные (или шардированные таблицы) – это крупные таблицы, которые также как и имеющиеся в СУБД «Jatoba» секционированные таблицы, разделены на несколько физически независимых сегментов (так называемых «шардов»).

Как и в секционированной таблице, каждый сегмент распределенной таблицы фактически является независимой таблицей. Каждый «шард» хранится в единственном экземпляре на одном из узлов – по этой причине размер таблицы будет примерно равен ее размеру в «нераспределенном» варианте и не зависеть от количества узлов.

Разделение таблиц на «шарды» производится по значению одного из полей таблицы – так называемому «ключу распределения». Строки имеющие одно и тоже значение ключа распределения всегда будут находиться в одном «шарде» и, как следствие, на одном узле.

Планировщик запросов позволяет строить оптимальные планы по таким распределенным таблицам.

Данная особенность позволяет линейно масштабировать как запросы на запись, так и запросы на чтение, но только в случае, если в этих запросах в условиях WHERE или ON будет фигурировать условие равенства по ключу распределения. В другом же случае, когда

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

в транзакции выполняется массовое чтение или запись нескольких «шардов», эффект масштабирования будет либо отсутствовать, либо определяться не связанными со структурой хранения данных факторами. К одному из таких факторов будет относиться суммарный размер буферного кэша в кластере. Так например, даже «не масштабируемый» запрос ко множеству шардов будет выполняться «быстрее», если суммарный размер буферного кэша на всех узлах кластера позволит полностью разместить данные в оперативной памяти.

4.2. Совместное размещение таблиц

Совместное размещение позволяет локализовать выполнение операции соединения данных (оператор JOIN) в пределах того узла, на котором физически расположены записи. Без подобной «локализации» подавляющее большинство операций JOIN можно выполнить только передавая по сети «как есть» значительную часть данных с одного узла на другой. Это в свою очередь почти всегда будет означать более медленное время выполнения таких запросов в сравнении с базой данных без шардирования.

Для ручного управления назначением группы совместного размещения таблицы используется необязательный параметр `colocate_with` функции `create_distributed_table`.

Для того чтобы обеспечить совместное размещение таблиц необходимо:

- 1) Создать распределенные таблицы:

```
SELECT create_distributed_table('stores', 'store_id');
```

- 2) Определить таблицы, которые будут размещены совместно:

```
SELECT create_distributed_table('orders', 'store_id', colocate_with => 'stores');  
  
SELECT create_distributed_table('products', 'store_id', colocate_with => 'stores');
```

4.3. Выбор ключа распределения

Следующим шагом при переходе приложения на компонент `ja_Hipe_Cluster` является определение подходящих ключей распределения.

Концептуально существует две основные стратегии – распределение по первичному ключу и распределение по внешнему ключу.

4.3.1. Распределение по первичному ключу

Стратегия распределения по первичному является наиболее простой и по своей сути приводит к равномерному (по количеству записей) распределению таблиц по «шардам» и, как следствие, узлам.

Соответственно при выборе такой стратегии можно ожидать, что данные будут распределены по узлам кластера равномерно.

Недостатком данной стратегии является невозможность локализовать сложные запросы на одном узле в случае, если соединение нескольких таблиц будет выполняться не по первичному ключу (операции JOIN редко выполняются по первичным ключам обеих таблиц).

4.3.2. Распределение по внешнему ключу

Данная стратегия более известна как «мультитенантное» распределение. Некий идентификатор «клиента» (также называемого «тенантом») присутствует во всех таблицах в виде внешнего ключа, заданного как явном, так и в неявном виде (то есть с определением ограничения REFERENCES или без него).

Под «тенантом» в такой стратегии подразумевается некоторая абстрактная бизнес-сущность – это может быть отдельный клиент приложения, группа клиентов в одной организации, или даже использующие приложение отдельные организации.

В отличие от стратегии распределения по первичному ключу, стратегия распределения по внешнему ключу позволяет локализовать сложные запросы на одном узле, ведь как правило операции JOIN выполняются именно по внешнему, а не первичному ключу.

К недостаткам же стратегии распределения по внешнему ключу относится в первую очередь необходимость явно указывать ключ распределения в конструкциях WHERE или ON всех запросов. В противном случае этого запрос будет отправлен на все узлы кластера.

Кроме того, эта распределение по внешнему ключу в большинстве случаев предполагает существенную неравномерность распределения данных на узлах в тех случаях, когда количество выделенных «тенантов» мало.

Следует дополнительно упомянуть, что ограничения уникальности и внешнего ключа для значений, отличных от `tenant_id`, представляют проблему в любой распределенной системе, поскольку сложно гарантировать, что ни один из двух узлов не принимает одно и то же уникальное значение. По этой причине, вместо «глобальных» ограничений применяют более слабую модель – уникальность в пределах «тенанта».

5. ПРИМЕРЫ МИГРАЦИИ

5.1. Пример для распределения по первичному ключу

В качестве простейшего примера будет рассмотрена классическая транзакционная задача типа «Дебет-Кредит».

Эта задача удобна тем, что ее реализация уже встроена в любую СУБД на основе PostgreSQL в виде утилиты `pgbench`. Задача имеет множество ценных с диагностической точки зрения вариантов тестирования, и в связи с этим является стандартной основой для множества тестов с общеизвестной количественной интерпретацией результатов.

По условиям этой задачи в БД присутствует четыре таблицы:

- филиалы банков (branches);
- кассы (tellers);
- клиенты (accounts);
- история операций (history)

Таблицы `branches`, `tellers` и `accounts` имеют первичные ключи (`bid`, `tid` и `aid`) и поле баланса (`bbalance`, `tbalance` и `abalance`).

Каждая транзакция представляет операцию начисления или снятия некоторой произвольной суммы со счета случайного клиента, кассы и филиала банка.

В стандартных условиях этой задачи предполагается, что на один филиал банка приходится десять касс и сто тысяч клиентов, а выбор случайных значений происходит по равномерному закону распределения.

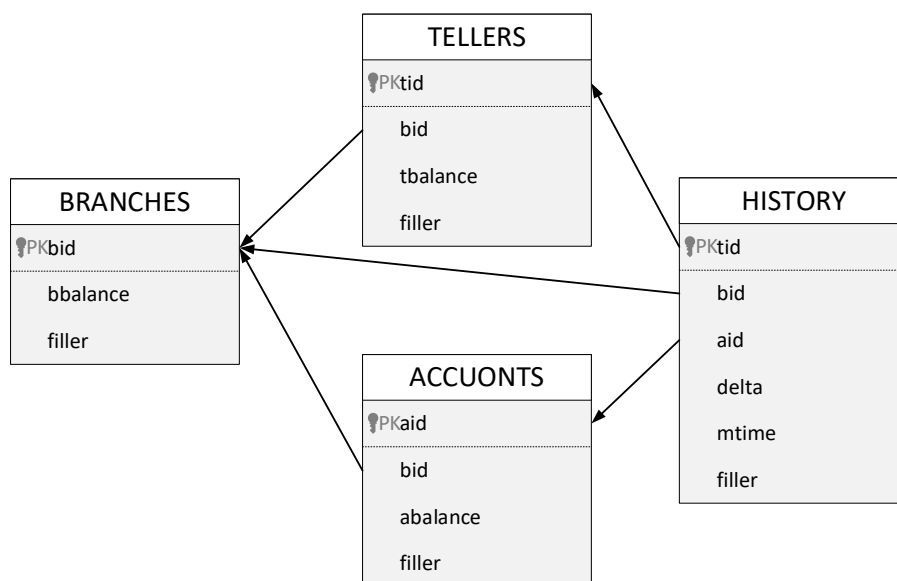


Рисунок 5.1 – Пример для распределения по первичному ключу

5.2. Пример распределения по внешнему ключу

Ниже приведен фрагмент упрощённой схемы многопользовательского приложения, похожего на Etsy или Shopify, где каждый «тенант» – это магазин, где подчеркнутые элементы – это первичные ключи. А выделенные курсивом элементы это – внешние ключи.

В этом примере хранилища являются естественным «тенантом». Идентификатор «тенанта» в данном случае – *store_id*. После распределения таблиц в кластере, надо чтобы строки, относящиеся к одному и тому же хранилищу, располагались вместе на одних и тех же узлах.

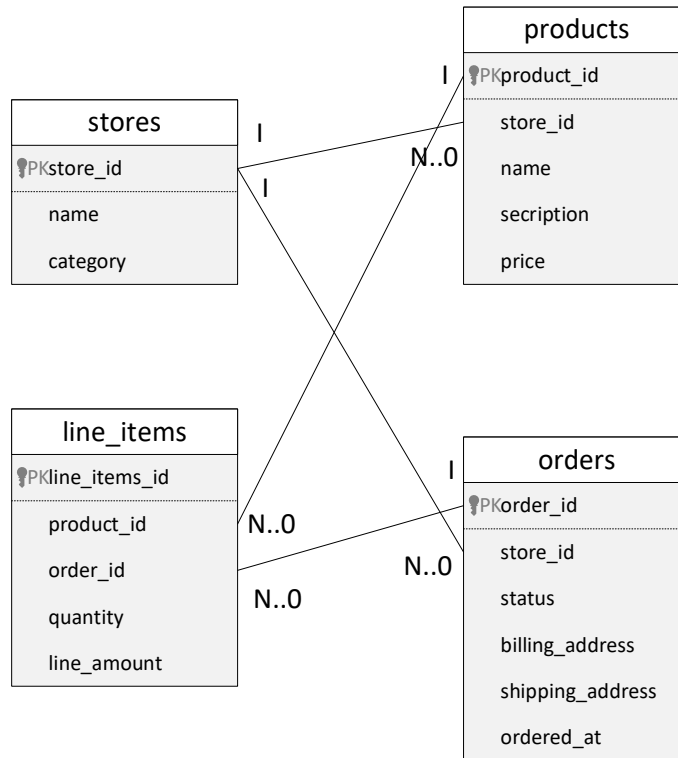


Рисунок 5.2 – Пример распределения по внешнему ключу

5.2.1. Добавление ключей распределения

После определения объёма необходимых изменений, следующим важным шагом является изменение структуры данных существующей базы.

Сначала таблицы, требующие заполнения, модифицируются с помощью добавления столбца, необходимого для ключа распределения.

В примере таблицы stores и products содержат store_id и готовы к распределению. В таблице line_items, поскольку она нормализована, отсутствует store_id. Если надо распределить товары по store_id, то этот столбец необходим.

```
ALTER TABLE line_items ADD COLUMN store_id uuid;
```

Столбец распределения должен иметь одинаковый тип во всех таблицах, например, нельзя смешивать int и bigint.

Типы столбцов должны совпадать для обеспечения корректного размещения данных.

5.2.2. Заполнение повторно созданных столбцов

После обновления схемы необходимо заполнить отсутствующие значения столбца tenant_id в таблицах, в которые этот столбец был добавлен.

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

Данная процедура в терминологии ja_Hipe_Cluster называется «backfilling». В приведенном примере таблица line_items требует значений для store_id.

После заполнения таблицы line_items получаем недостающие значения из запроса на соединение с таблицей orders:

```
UPDATE line_items
SET store_id = orders.store_id
FROM line_items
INNER JOIN orders
WHERE line_items.order_id = orders.order_id;
```

Одновременная обработка всей таблицы может привести к чрезмерной нагрузке на базу данных и нарушить выполнение других запросов.

Заполнение отсутствующих значений может выполняться медленно. Один из способов исправить ситуацию – создание функций, которая заполняет данные небольшими пакетами данных за раз:

```
CREATE FUNCTION backfill_batch()
RETURNS void LANGUAGE sql AS $$
WITH batch AS (
SELECT line_items_id, order_id
FROM line_items
WHERE store_id IS NULL
LIMIT 10000
FOR UPDATE
SKIP LOCKED
)
UPDATE line_items AS li
SET store_id = orders.store_id
FROM batch, orders
WHERE batch.line_item_id = li.line_item_id
AND batch.order_id = orders.order_id;
```

```
$$;
```

Созданную функцию можно затем вызывать, например, с помощью инструментария `pg_cron`:

```
SELECT cron.schedule('* /15 * * * *', 'SELECT backfill_batch()');
```

После того как процесс заполнения завершиться задание `pg_cron` можно отключить:

```
SELECT cron.unschedule(42);
```

Где 42 – это номер задания в `cron.unschedule`.

В конце необходимо обновить код приложения и запросы, для того чтобы они учитывали изменения схемы данных.

5.2.3. Включение столбца распределения в ключи

Компонент `ja_Hipe_Cluster` не может обеспечить соблюдение ограничений уникальности, если уникальный индекс или первичный ключ не содержат столбец распределения.

Таким образом, необходимо изменить первичные и внешние ключи, в нашем примере, включив в них `store_id`.

Ниже приведён пример базовых команд SQL для преобразования простых ключей в составные в базе данных для разработки.

```
BEGIN;

-- удалить простые первичные ключи (каскадом во внешние ключи)
ALTER TABLE products DROP CONSTRAINT products_pkey CASCADE;
ALTER TABLE orders DROP CONSTRAINT orders_pkey CASCADE;
ALTER TABLE line_items DROP CONSTRAINT line_items_pkey CASCADE;

-- пересоздать первичные ключи, включив в них столбец возможного
распределения
ALTER TABLE products ADD PRIMARY KEY (store_id, product_id);
ALTER TABLE orders ADD PRIMARY KEY (store_id, order_id);
ALTER TABLE line_items ADD PRIMARY KEY (store_id, line_item_id);
```

```
-- пересоздать внешние ключи, включив в них столбец возможного ра  
спределения  
ALTER TABLE line_items ADD CONSTRAINT line_items_store_fkey  
FOREIGN KEY (store_id) REFERENCES stores (store_id);  
ALTER TABLE line_items ADD CONSTRAINT line_items_product_fkey  
FOREIGN KEY (store_id, product_id) REFERENCES products (store_id,  
product_id);  
ALTER TABLE line_items ADD CONSTRAINT line_items_order_fkey  
FOREIGN KEY (store_id, order_id) REFERENCES orders (store_id, ord  
er_id);  
COMMIT;
```

5.2.4. Добавление ключа распределения в запросы

После того, как ключ распределения будет присутствовать во всех соответствующих таблицах, необходимо включить его в запросы приложения.

Следующие шаги необходимо выполнить, используя копию приложения, работающую в среде разработки, и протестировать совместно с компонентом ja_Hipe_Cluster.

После того, как будет обеспечена работа приложения с компонентом ja_Hipe_Cluster, можно планировать перенос данных из исходной базы данных в реальный кластер ja_Hipe_Cluster.

- Код приложения и любые другие процессы приема данных, которые записывают данные в таблицы, должны быть обновлены с учетом новых столбцов.
- Запуск набора тестов приложения на основе измененной схемы в Citus — хороший способ определить в каких областях кода необходимо внести изменения.
- Рекомендуется включить ведение журналов базы данных. Журналы могут помочь обнаружить случайные перекрестные запросы в многопользовательском приложении, которые следует преобразовать в запросы для каждого клиента.

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

Перекрестные запросы между «шардами» поддерживаются, но в многопользовательском приложении большинство запросов должны быть направлены на один узел. Для простых запросов на выборку, обновление и удаление это означает, что условие WHERE должно фильтроваться по идентификатору клиента. Тогда компонент ja_Hipe_Cluster сможет эффективно выполнять эти запросы на одном узле.

В приведенном ниже примере подчеркнутые элементы – это первичные ключи, а выделенные курсивом элементы – это внешние ключи.

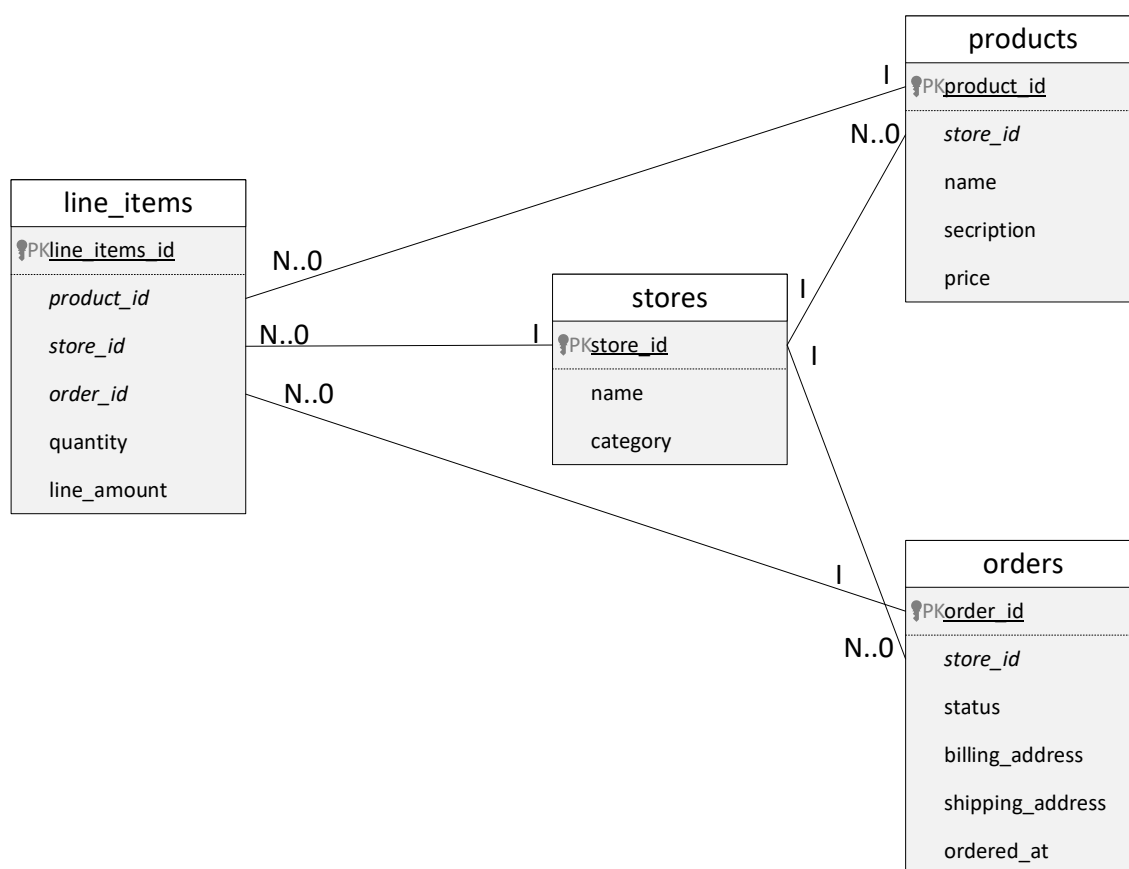


Рисунок 5.3 – Пример распределения

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Аутентификационная информация — информация, используемая при аутентификации субъекта доступа или объекта доступа.

Аутентификация – действия по проверке подлинности субъекта доступа и/или объекта доступа, а также по проверке принадлежности субъекту доступа и/или объекту доступа предъявленного идентификатора доступа и аутентификационной информации (ГОСТ Р 58833-2020).

Администратор СУБД – субъект доступа, выполняющий административные функции в СУБД и наделенный правами:

- создавать учетные записи пользователей системы управления базами данных;
- модифицировать, блокировать и удалять учетные записи пользователей системы управления базами данных;
- назначать права доступа пользователям системы управления базами данных к объектам доступа системы управления базами данных;
- управлять конфигурацией системы управления базами данных;
- создавать, подключать базы данных.

Администратор СУБД имеет атрибут SUPERUSER и/или обладает системной учетной записью «postgres».

Администратор БД – субъект доступа, выполняющий административные функции в БД и наделенный правами:

- создавать учетные записи пользователей базы данных;
- модифицировать, блокировать и удалять учетные записи пользователей базы данных;
- управлять конфигурацией базы данных;
- назначать права доступа пользователям базы данных (пользователей информационной системы) к объектам доступа базы данных;

- создавать резервные копии базы данных и восстанавливать базу данных из резервной копии;
- создавать, модифицировать и удалять процедуры (программный код), хранимые в базе данных.

Администратор БД имеет атрибут CREATEROLE, и возможные атрибуты BYPASSRLS, REPLICATION, а также прочие системные привилегии относительно БД, кроме атрибута CREATEDB.

Безусловная блокировка пользователя – это ограничение пользователя в возможности устанавливать новую сессию с СУБД. Безусловная блокировка имеет приоритет над ограничениями, накладываемыми парольными политикам (блокировка вследствие истечения срока действия пароля, временные блокировки при исчерпании попыток ввода пароля и т.п.), применяется независимо от них и не зависит от применяемого метода аутентификации пользователей. Снятие безусловной блокировки не снимает блокировок по парольным политикам и наоборот.

Завершение сессии пользователя – принудительное завершение открытой сессии пользователя с БД/СУБД в заданном режиме.

Пользователь БД - субъект доступа, имеющий доступ к ограниченному перечню БД и объектов БД. Имеющий следующий набор привилегий:

- создавать и манипулировать объектами доступа БД (таблица, запись или столбец, поле, представление и иные объекты доступа);
- выполнять процедуры (программный код), хранимые в БД.

Пользователь БД имеет обязательный атрибут LOGIN.

Пользователь СУБД – см. «Пользователь БД». Для СУБД эти понятия идентичны. СУБД не разграничивает пользователей по отдельным БД. Все пользователи общие, доступ к отдельным БД определяется настройками доступа.

Роль – субъект доступа в БД/СУБД, наделенный определенным набором привилегий (чаще всего употребляется как обобщение группы пользователей для выполнения определенного набора действий в БД/СУБД).

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

SQL	–	Structured Query Language
БД	–	База данных
КС	–	Контрольные суммы
КЦ	–	Контроль целостности
ОС	–	Операционная система
СУБД	–	Система управления базами данных
ФСТЭК России	–	Федеральная служба по техническому и экспортному контролю России

[illegible]

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------