# ДЗ «Масштабируемая подсистема диалогов»

**Схема данных и шардирования**

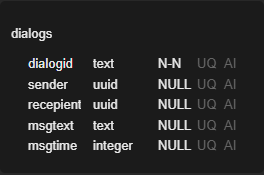
В качестве реализации для шардирования выбран Citus в виде докер контейнера, т.к.:  
 - является расширением postgres, который уже используется в проекте в роли БД;

- из коробки содержит функционал по управлению шардами, не требующим его реализацию в логике сервиса;

- поддерживает подключение как к обычной pg базе;

- рекомендован преподователем на лекции.

Модель данных для сервиса диалогов выбрана единой таблицей:

  
Где id – соcтавлен как объединение uuid отправителя и получателя, что позволяет вычислять диалог к которому принадлежит сообщение (метод gen\_dialogid класса ShardDB).

Для сортировки сообщений в читаемую и понятную цепочку по времени отправки в спецификацию добавлено поле msgtime.

Сообщения шардируются по id диалога, к которому они относятся, чтобы учесть “Эффект Леди Гаги”. Исходил из следующих соображений:

* Люди обычно просматривают все сообщения одного чата (согласно спецификации ДЗ должен возвращаться диалог целиком). Соответственно стоит хранить данные, относящиеся к одному чату на одном шарде.
* Человек одновременно может писать много сообщений, но скорее всего по разным чатам, т.к. мало имеет практического смысла писать слишком большое количество сообщения одному получателю. Поэтому стоит группировать сообщения по идентификатору чата.
* Так как в диалог пишут обычно два человека, то хранить рядом сообщения одного конкретного пользователя (по sender) не логично, так как запросов на сообщения одного конкретного пользователя не будет.
* От добавления в ключ шардирования времени отправки сообщения отказался, чтобы исключить кроссшардовые запросы, т.к. требуется вернуть весь диалог.

**Для равномерного распределения по шардам используется** hashed ключ шардирования по id диалога. Сообщения между парой пользователей будут лежать в одном шарде.

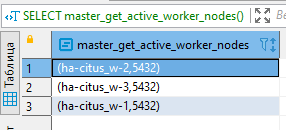
Поскольку Citus при выполнении ребаланса учитывает размер шарда и их количество (стратегия по умолчания), то большие диалоги особо активных пользователей будут равномерно распределены по нодам кластера при ребалансе, исключив перекос в данных.

**Решардинг**

Для решардинга используется встроенные механизмы Citus, возможности проводить решардинг без даунтайма режим репликации pg переключен в логический при инициации БД мастера:  
«alter system set wal\_level = logical»

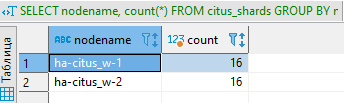
После добавления нового воркера в кластер:  
1. Убеждаемся что новый воркер увидел manager и включил в кластер запросом:

SELECT master\_get\_active\_worker\_nodes();



2. Убедимся, что на новом воркере еще нет шардов:

SELECT nodename, count(\*) FROM citus\_shards GROUP BY nodename;

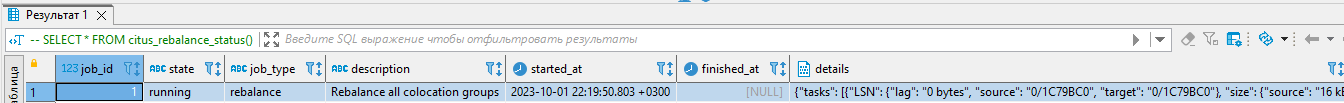


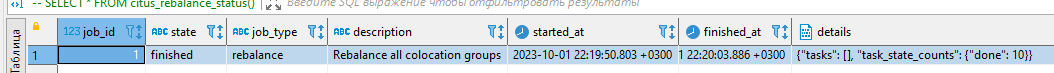
3. Запустим решардирование запросом (решардинг выполняется в фоне):

SELECT citus\_rebalance\_start();

3.1 Периодически выполняем запрос:

SELECT \* FROM citus\_rebalance\_status();

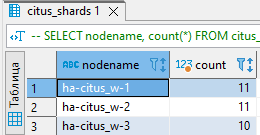




Пока не получим ответ, что все задачи решардирования выполнены, например «"task\_state\_counts": {"done": 18}»

4. Убедимся, что шарды распределены с учетом новой ноды:

SELECT nodename, count(\*) FROM citus\_shards GROUP BY nodename;



Решардинг выполнен без даунтайма

**Запуск сервиса:**

Запуск осуществляется командой:

docker-compose -f ./ docker-compose.citus.yaml up --build –d

При первом запуске init файлом pg создается необходимая таблица, после запуска ее нужно сделать из неешардированную, для этого подключаемся к psql master и выполняем запросы:

SELECT create\_distributed\_table('dialogs', 'dialogid');

После проскалировать необходимое количество нод шардирования, например 2:

docker-compose -f "./docker-compose.citus.yaml" up -d --build --scale citus\_w=2