# II. Fragmentaryzacja pozioma w SZBD Postgres

Celem zajęć jest zapoznanie się z własnościami SZBD Postgres umożliwiającymi [fragmentaryzację poziomą tabel](https://www.projectmanagement.com/process/popup.cfm?ID=23459&techKid=1) (ang. [sharding](https://en.wikipedia.org/wiki/Shard_(database_architecture))).

## Przygotowanie środowiska

Funkcjonalność fragmentaryzacji poziomej w SZBD Postgres jest zapewniana przez rozszerzenie o nazwie [Citus](https://www.citusdata.com/). Zapoznaj się z [ogólnymi informacjami o tym rozszerzeniu](https://docs.citusdata.com/en/latest/aboutcitus/what_is_citus.html). Rozszerzenie to jest rozwijane przez niezależną firmę, która została wykupiona przez Microsoft, w związku z tym nie jest domyślnie instalowane wraz z oprogramowaniem SZBD Postgres i należy je doinstalować lub skorzystać z gotowego obrazu. Wykorzystamy to drugie rozwiązanie. W celu przygotowania środowiska do dalszej pracy wykonaj poniższe punkty:

1. Zaloguj się do maszyny wirtualnej jako użytkownik rbd używając hasła RBD#7102.
2. Otwórz okno terminala, który nazwiemy terminalem pomocniczym.
3. W pierwszym kroku pobierz plik manifestów za pomocą poniższego polecenia:

wget www.cs.put.poznan.pl/jjezierski/RBDv3/rbd-citus.yaml

1. Otwórz plik manifestów w celu jego przeglądnięcia za pomocą polecenia:

less rbd-citus.yaml

Zawartość tego pliku jest zbliżona do pliku manifestów z poprzedniego tutorialu. W pierwszym manifeście, który opisuje StatefulSet za pomocą klucza *spec.template.spec.containters.image* wskazano na obraz systemu Postgres z zainstalowanym rozszerzeniem Citus. Klucz *spec.template.spec.containters.env* został wykorzystany do przekazania wartości zmiennych środowiskowych umożliwiających konfigurację kontenera. Zmienna POSTGRES\_HOST\_AUTH\_METHOD ustawiona na wartość *trust* umożliwia wykonywanie połączeń do bazy danych bez uwierzytelnienia. Ta metoda jest niezalecana dla systemów produkcyjnych, w takich systemach należy skonfigurować uwierzytelnienie połączeń za pomocą pliku *pg\_hba.conf*.

Na końcu pliku umieszczono dodatkowy manifest, który opisuje kontroler typu bezgłowa usługa (ang. headless Service). Jego nazwa pochodzi od wartości klucza *spec.ClusterIP*, która jest ustawiona na *None*.Zadaniem tego kontrolera jest dostarczenie nazw domenowych dla wdrożonych replik *Pod*, które pasują do wartości klucza *spec.selector* tego kontrolera. Nazwy te będą dostępne wewnątrz klastra Kubernetes. Kubernetes stosuje następującą konwencję nazw domenowych *nazwa-repliki-Pod.nazwa-kontrolera.przestrzeń-nazw*.svc.cluster.local W związku z tym, że nie umieszczaliśmy naszych obiektów w żadnej przestrzeni nazw (ang. Namespace), to zostały one umieszczone w domyślnej przestrzeni nazw (*default*), przykładowa pełna nazwa domenowa pierwszej repliki będzie następująca: pgsql-citus-sts-0.pgsql-rbd-citus.default.svc.cluster.local.

Opuść program *less* wybierając przycisk *q*.

1. Rozpocznij wdrożenie komponentów z pliku manifestów za pomocą polecenia:

kubectl apply -f rbd-citus.yaml

1. Obserwuj postęp wdrożenia *StatefulSet* wykorzystując poniższe polecenie:

kubectl get sts --watch

Wdrożenie wymaga pobrania obrazu kontenera z repozytorium Docker, w związku z tym zajmuje chwilę. Wdrożenie zakończy się w momencie pojawienia się na terminalu wiersza, w którym liczba działających replik będzie równa liczbie żądanych replik, np.:  
pgsql-citus-sts 3/3 1m

W tym momencie przerwij wykonanie polecenia wykorzystując kombinację Ctrl-c.

Wyświetl *Pod*, które zostały utworzone do obsługi replik *StatefulSet*, wykorzystaj następujące polecenie:

kubectl get pods -l app=pgsql-rbd-citus

1. Otwórz trzy kolejne zakładki w oknie terminala.
2. Nazwij te zakładki nazwami kolejnych replik od pgsql-citus-sts-0 do pgsql-citus-sts-2, w tym celu uruchom w każdej zakładce poniższe polecenie, gdzie X oznacza numer repliki od 0 do 2.

PS1="\[\e]0;pgsql-citus-sts-X\a\]\u@\h:\w\$ "

1. W terminalach pgsql-citus-sts-1 i pgsql-citus-sts-2 będziemy wykonywać operacje na bazach danych uruchomionych w replikach pgsql-citus-sts-1 i pgsql-citus-sts-2. Bazy te będą służyć do przechowywania fragmentów poziomych danych (ang. shard). W terminalu pgsql-citus-sts-0 będziemy wykonywać operacje na bazie danych uruchomionej w replice pgsql-citus-sts-0.Baza ta będzie służyć do rozpraszania danych między różne fragmenty i koordynowania poleceń SQL kierowanych do rozproszonych poziomo fragmentów.

## Przygotowanie schematu danych i załadowanie danych do klastra baz danych

Przykładowe dane opisują pomiary temperatury i są zgromadzone w 3 tabelach: *organizations*, *loggers*, *measurements*. Rejestratory temperatury (ang. logger) należą do określonej organizacji, zaś pomiary temperatury (ang. measurements) są zbierane przez określone rejestratory.

Dane te zostaną rozproszone w poszczególnych bazach danych tworzących klaster Citus, składającym się z jednego węzła koordynatora i wielu węzłów roboczych. Kluczowym krokiem z punkty widzenia użyteczności i wydajności mechanizmu fragmentaryzacji poziomej danych na węzły robocze jest wybór kryterium rozpraszania. Kryterium tym powinna być kolumna lub zbiór kolumn, która dzieli dane na logicznie rozłączne zbiory. W przypadku przykładowych danych jest to identyfikator organizacji. Jest to typowy wybór dla aplikacji udostępniającej usługi typu [*Software as a Service*](https://pl.wikipedia.org/wiki/Software_as_a_Service), gdzie dobrym kryterium rozpraszania jest identyfikator klienta takiej usługi. Rozszerzenie Citus wymaga denormalizacji schematu bazy danych tak aby wszystkie tabele, które będą fragmentaryzowane poziomo zawierały kryterium rozpraszania. Dodatkowo, kryterium rozpraszania powinno wchodzić w skład klucza podstawowego takich tabel.

1. W terminalu pomocniczym pobierz i rozkompresuj schemat oraz przykładowe dane:

wget https://www.cs.put.poznan.pl/jjezierski/RBDv3/loggers.zip

unzip loggers.zip

1. Zapoznaj się ze schematem przykładowych danych znajdującym się w pliku ~/loggers/loggers.sql. Zwróć uwagę na definicję kluczy podstawowych oraz obcych.
2. Skopiuj katalog *loggers* do katalogu */data* repliki pgsql-citus-sts-0.   
   Użyj poniższego polecenia:

kubectl cp loggers pgsql-citus-sts-0:/data

1. W terminalu pgsql-citus-sts-0 wykonaj poniższe polecenie aby uruchomić powłokę w replice pgsql-citus-sts-0. Replika ta będzie obsługiwać węzeł koordynatora klastra Citus.

kubectl exec -it pgsql-citus-sts-0 -- /bin/bash

1. W tym samym terminalu uruchom narzędzie *psql* w celu przyłączenia się do bazy danych koordynatora, wykorzystaj następujące polecenie:

psql -U postgres

1. Analogicznie jak punkcie 4, w zakładkach pgsql-citus-sts-1 i pgsql-citus-sts-2 uruchom powłoki replik pgsql-citus-sts-1 i pgsql-citus-sts-2.
2. Tak samo jak punkcie 5, w zakładkach pgsql-citus-sts-1 i pgsql-citus-sts-2 uruchom narzędzie *psql* w celu przyłączenia się do baz danych węzłów roboczych klastra Citus.
3. W narzędziu *psql*  repliki pgsql-citus-sts-0 wskaż na koordynatora klastra Citus wykonując poniższe polecenie:

SELECT citus\_set\_coordinator\_host('pgsql-citus-sts-0.pgsql-rbd-citus', 5432);

Zauważ, że użyto tylko początkowy fragment nazwy domenowej. Jest to możliwe dzięki temu, że Kubernetes wstrzyknął do kontenera plik /etc/resolve.conf o następującej zawartości (sprawdź to samodzielnie):  
search default.svc.cluster.local svc.cluster.local cluster.local

nameserver …

options ndots:5

Za przeszukiwanie domen odpowiedzialna jest dyrektywa *search.*

1. W bazie danych koordynatora skonfiguruj klaster baz danych dołączając do niego bazy danych węzłów roboczych. W tym celu wykorzystaj poniższe polecenia:

SELECT \* from citus\_add\_node('pgsql-citus-sts-1.pgsql-rbd-citus', 5432);

SELECT \* from citus\_add\_node('pgsql-citus-sts-2.pgsql-rbd-citus', 5432);

1. Sprawdź poprawność wykonania poprzedniego kroku przez wydanie w bazie koordynatora polecenia:

select \* from citus\_get\_active\_worker\_nodes();

1. Utwórz w koordynatorze obiekty ze skryptu loggers.sql, wykonując w terminalu pgsql-citus-sts-0 poniższe polecenie:

\i /data/loggers/loggers.sql

1. W bazie danych koordynatora, z wykorzystaniem funkcji [*create\_distributed\_table*](https://docs.citusdata.com/en/latest/reference/user_defined_functions.html#create-distributed-table), zdefiniuj fragmentaryzację tabeli *organizations*:

SELECT create\_distributed\_table('organizations', 'or\_id');

1. W bazie danych koordynatora, zdefiniuj fragmentaryzację tabeli *loggers*, zwróć uwagę na parametr *colocate\_with*, który umożliwia rozproszenie wierszy tabeli loggers w taki sposób aby każdy wiersz tej tabeli został umieszczony w tym samym węźle co wiersz opisujący organizację, do której należy dany rejestrator:

SELECT create\_distributed\_table('loggers', 'lo\_or\_id', colocate\_with => 'organizations');

1. W bazie danych koordynatora, zdefiniuj fragmentaryzację tabeli *measurements,* nie zapomnij o kolokacji z tabelą *organizations.* [Raport]
2. W bazie danych koordynatora załaduj dane do przykładowych tabel:

\i /data/loggers/organizations.dmp

\i /data/loggers/loggers.dmp

\i /data/loggers/measurements.dmp

1. Policz poziome fragmenty rozproszonych tabel, w tym celu wykonaj w bazie koordynatora poniższe zapytanie.

SELECT table\_name, count(\*)

FROM citus\_shards

GROUP BY table\_name;

1. Przeglądnij alokację przykładowych fragmentów tabeli *measurements*, w tym celu wykonaj w bazie koordynatora poniższe zapytanie.

SELECT shardid, shard\_name, nodename

FROM citus\_shards

WHERE table\_name = 'measurements'::regclass

ORDER BY shardid

LIMIT 10;

Czy fragmenty zostały zaalokowane w bazie danych koordynatora? [Raport]

1. Nazwy fragmentów z wyniku powyższego zapytania są nazwami fizycznych tabel w bazach danych węzłów roboczych i można przeglądać ich zawartość w tych węzłach z wykorzystaniem polecenia SELECT. Wykorzystamy tę własność i policzymy liczbę wierszy w wybranych fragmentach tabeli *measurements* w bazie danych węzła roboczego *pgsql-citus-sts-1*. W tym celu uruchom poniższe polecenie w narzędziu *psql* węzła roboczego *pgsql-citus-sts-1*.

SELECT format(

'SELECT ''%I'' AS shard\_name, count(\*) AS row\_count FROM %I;',

shard\_name,

shard\_name

)

FROM citus\_shards

WHERE table\_name = 'measurements'::regclass

and nodename= 'pgsql-citus-sts-1.pgsql-rbd-citus'

ORDER BY shardid

LIMIT 5

\gexec

Czy wiersze tabeli *measurements* zostały równomiernie rozproszone między fragmenty? Dlaczego? [Raport]

1. W bazie danych koordynatora sprawdź możliwość modyfikacji wierszy rozproszonych na poszczególne bazy danych klastra:

UPDATE loggers SET lo\_description='Fridge #23' where lo\_id=622 and lo\_or\_id=138;

## Optymalizacja zapytań kierowanych do fragmentaryzowanych poziomo tabel

W tym punkcie zostanie sprawdzony sposób optymalizacji poleceń SQL, które są kierowane do tabel fragmentaryzowanych poziomo.

1. W bazie danych koordynatora wydaj poniższe zapytanie, zwróć uwagę, że w warunku połączeniowym znajduje się odniesienie do kryterium rozpraszania:

select lo\_description, me\_time, me\_temperature

from loggers join measurements on (lo\_id=me\_lo\_id and lo\_or\_id=me\_or\_id)

where me\_time between timestamp '2017-03-15 08:13' and timestamp '2017-03-15 09:18'

order by me\_time;

1. Sprawdź plan wykonania powyższego zapytania:

EXPLAIN select lo\_description, me\_time, me\_temperature

from loggers join measurements on (lo\_id=me\_lo\_id and lo\_or\_id=me\_or\_id)

where me\_time between timestamp '2017-03-15 08:13' and timestamp '2017-03-15 09:18'

order by me\_time;

Na ile zadań zostało podzielone zapytanie? Jaki jest szacowany koszt wykonania jednego zadania? [Raport]

1. Wydaje, że można uzyskać wzrost wydajności wykonywanego polecenia przez utworzenie indeksu na atrybucie *me\_time*. W bazie danych koordynatora wykonaj poniższe polecenie:

CREATE INDEX me\_time\_idx on measurements(me\_time);

1. Ponownie sprawdź plan zapytania. Czy indeks został wykorzystany? Czy zmienił się szacowany koszt wykonania jednego zadania? [Raport]
2. Dodaj w warunku selekcji zapytania kryterium rozpraszania. Jaka nastąpiła zmiana w planie? [Raport]

EXPLAIN select lo\_description, me\_time, me\_temperature

from loggers join measurements on (lo\_id=me\_lo\_id and lo\_or\_id=me\_or\_id)

where me\_time between timestamp '2017-03-15 08:13' and timestamp '2017-03-15 09:18'

and lo\_or\_id=143

order by me\_time;

## Tabele referencyjne

Zapoznaj się ze skryptem ~/loggers/logger\_types.sql. Zawiera on definicję tabeli *logger\_types*. Wiersze tej tabeli opisują rejestratory wszystkich organizacji, z tego powodu nie można jej bezpośrednio fragmentować poziomo. W związku z tym, że tabela *loggers\_types* zawiera niewiele wierszy lepszym sposobem jest jej replikacja na wszystkie bazy danych klastra. Rozszerzenie Citus nazywa taki rodzaj tabel tabelami referencyjnymi. W schemacie fragmentowanej poziomo bazy danych mają on analogiczne znaczenie jak tabele wymiarów w hurtowni danych.

1. W bazie danych koordynatora uruchom skrypt /data/loggers/logger\_types.sql z wykorzystaniem poniższego polecenia:

\i /data/loggers/logger\_types.sql

1. W bazie danych koordynatora wykorzystaj funkcję [create\_reference\_table](https://docs.citusdata.com/en/latest/reference/user_defined_functions.html#create-reference-table) do wskazania, że tabela *loggers\_types* jest tabelą referencyjną:

SELECT create\_reference\_table('logger\_types');

1. W bazie danych koordynatora załaduj dane do tabeli *loggers\_types*:

\i /data/loggers/logger\_types.dmp

1. Wykorzystując poniższe polecenie sprawdź ile fragmentów tej tabeli zostało utworzone bazach danych węzłów roboczych.

SELECT shard\_name, nodename

FROM citus\_shards

WHERE table\_name = 'logger\_types'::regclass;

Porównaj zawartość tych fragmentów z zawartością tabeli *loggers\_types*. [Raport]

1. W bazie danych koordynatora zmodyfikuj nazwę jednego z typów rejestratora:

UPDATE logger\_types SET lt\_name='TypRejestratora2' WHERE lt\_name='LoggerType2';

1. Sprawdź czy zmiany zostały przeniesione do fragmentów w bazach danych węzłów roboczych. [Raport]
2. Sprawdź plan poniższego zapytania, które odwołuje się do tabeli *loggers\_types*. Ile baz danych klastra było zaangażowanych w wykonanie tego zapytania? [Raport]

explain

select lt\_name, lo\_description, me\_time, me\_temperature

from loggers join measurements on (lo\_id=me\_lo\_id and lo\_or\_id=me\_or\_id)

join logger\_types on(lt\_id=lo\_lt\_id)

where me\_time between timestamp '2017-03-15 08:13' and timestamp '2017-03-15 09:18'

and lo\_or\_id=143

order by me\_time;

## Rozpraszanie połączeń do klastra Citus

1. Metadane klastra Citus są zarządzane i składowane w bazie danych koordynatora. Domyślnie są one replikowane na węzły robocze. Zabieg ten umożliwia zlecanie zapytań i poleceń DML nie tylko w sesjach koordynatora ale również w sesjach węzłów roboczych. Polecenia DDL są obsługiwane jedynie przez sesje koordynatora. Przetestuj ten mechanizm i wykonaj w zakładce pgsql-citus-sts-1 następujące polecenie:

UPDATE loggers SET lo\_description='**New** Fridge #23' where lo\_id=622 and lo\_or\_id=138;

1. Skorzystamy z usługi *LoadBalancer* klastra Kubernetes aby przyłączyć się do jednego węzłów klastra Citus. Uruchom w terminalu pomocniczym następujące polecenie:

psql -h localhost -U postgres

1. Za pomocą poniższego polecenia wydanego w terminalu pomocniczym ustal adres IP węzła klastra Kubernetes, na którym działa baza danych do której się przyłączyłeś.

SELECT inet\_server\_addr();

1. Sprawdź nazwę domenową tego adresu. Będziesz potrzebować w tym celu narzędzia, które nie są zainstalowane w obrazie kontenera. Uruchom w trybie interaktywnym samodzielny Pod z obrazem kontenera *dnsutils*. W nowej zakładce terminala uruchom poniższe polecenie:

kubectl run -it dnsutils \

--image gcr.io/kubernetes-e2e-test-images/dnsutils:1.3

Następnie w interaktywnej sesji z nowym Pod wykorzystaj polecenie nslookup do sprawdzenia nazwy domenowej adresu IP, adres IP podaj jako argument polecenia nslookup.

1. Otwórz nową zakładkę terminala w celu nawiązania kolejnej sesji z klastrem Citus w taki sam sposób jak w punkcie 2.
2. Sprawdź IP i nazwę domenową węzła klastra Citus, do których się przyłączyłeś.
3. Wykonaj dowolne operacje modyfikacji danych za pomocą sesji, który ustanowiłeś w punkcie 2 i 5.
4. Czy zaobserwowałeś jakieś różnice w funkcjonalności tych sesji? [Raport]

## Dodawanie nowej bazy danych do klastra Citus

W celu wydajnego obsłużenia zwiększającej się ilości danych i zwiększającego się obciążenia systemu można dodać nową bazę danych do klastra.

1. W terminalu pomocniczym zwiększ liczbę replik *StatefulSet* do czterech. W tym celu wykonaj poniższe polecenie:

kubectl scale sts pgsql-citus-sts --replicas 4

Powyższe polecenie, zostało wykorzystane w celu poglądowym. Z punktu widzenia właściwej metodyki definicji klastra Kubernetes należałoby zmodyfikować manifest ustalając nową liczbę replik i następnie ponownie go wdrożyć.

1. W bazie danych koordynatora dodaj nową bazę danych do klastra:

SELECT \* from citus\_add\_node('pgsql-citus-sts-3.pgsql-rbd-citus', 5432);

1. Sprawdź jakie zostały utworzone fragmenty tabel w bazie danych nowego węzła roboczego. [Raport]
2. Od wersji 11 Citus Community wspiera balansowanie fragmentów rozproszonych tabel. Służy do tego funkcja [citus\_rebalance\_start](https://docs.citusdata.com/en/stable/develop/api_udf.html#citus-rebalance-start). Operacja ta jest wysoce współbieżna, kopiowanie fragmentów odbywa się z wykorzystaniem mechanizmu logicznej replikacji, a dostęp do fragmentów jest blokowany przez krótki czas potrzebny na zmodyfikowanie metadanych. Replikacja logiczna wymaga generowania dziennika transakcyjnego ze szczegółowością na poziomie *logical*. Obecnie bazy danych działają na poziomie *replica*, który jest niewystarczający na potrzeby replikacji logicznej. W związku z tym uruchom poniższe polecenie w każdej z 4 baz danych aby ustawić poziom *logical*.

alter system set wal\_level=logical;

1. W celu zaaplikowania zmian wprowadzonych za pomocą polecenia ALTER SYSTEM SET należy zrestartować wszystkie bazy danych, w tym celu wykonaj w terminalu poniższe polecenie:

kubectl rollout restart statefulset/pgsql-citus-sts

Monitoruj postęp restartu za pomocą polecenia:

kubectl rollout status statefulset/pgsql-citus-sts

Poczekaj na zakończenie restartu.

1. Rozpocznij balansowanie uruchamiając w bazie koordynatora poniższe polecenie:

SELECT citus\_rebalance\_start();

1. Monitoruj status realizacji operacji balansowania uruchamiając w koordynatorze poniższe polecenie:

SELECT details FROM citus\_rebalance\_status();

Poczekaj na zakończenie balansowania.

1. Sprawdź nowe rozproszenie fragmentów między węzły robocze, w tym celu wykonaj w koordynatorze poniższe polecenie:

SELECT table\_name, nodename, count(\*)

FROM citus\_shards

GROUP BY table\_name, nodename

order by table\_name, nodename;

Czy rozproszenie fragmentów między węzły robocze jest równomierne? Dlaczego? [Raport]