Politechnika Warszawska

Wydział Elektryczny

Kierunek Informatyka Stosowana

Migracja MySQL do PostgreSQL

Wykonał:
Aliaksandr Karolik

Sprawdzający:
dr inż. Krzysztof Hryniów

20 listopada 2021

Spis treści

1 Wybrana baza danych do migracji

Obecnie baza została stworzona w oparciu o MySQL, zdecydowałem, że docelową bazą danych będzie PostgreSQL. Tak samo, jak wcześniej baza danych zostanie skonfigurowana w kontenerze dockerowym. Wybraną wersję PostgreSQL była 10.5 (Debian 10.5-2.pgdg90+1).

2 Porównanie MySQL z PostgreSQL

2.1 Typy baz

PostgreSQL to obiektowo-relacyjny system zarządzania bazą danych (ORDBMS), w którym nacisk położono na rozszerzalność i zgodność ze standardami. PostgreSQL jest zgodny z ACID, transakcyjny, posiada aktualizowane i zmaterializowane widoki, wyzwalacze oraz klucze obce. Obsługuje również funkcje i procedury składowane.

MySQL jest systemem zarządzania relacyjną bazą danych (RDBMS) typu open-source. Podobnie jak PostgreSQL i wszystkie inne relacyjne bazy danych, MySQL używa tabel jako głównego komponentu i posiada mniej więcej taki sam zestaw funkcji jak PostgreSQL. Nowsze wersje MySQL (5.7+) obsługują nawet niektóre funkcje noSQL.

2.2 Przechowywane typy danych

Podczas analizy różnicy pomiędzy tymi bazami danych rzuciło mi się w oczy to że Postgres oferuje szerszą gamę typów danych niż MySQL. MySQL wspiera następujące typy danych: Numeric, date/time, character, spatial, JSON natomiast Postgre obsługuje również te typy danych oprócz spatial, ale dodatkowo wspiera takie typy danych jak XML, HSTORE, arrays, ranges, composite. Mając to na uwadze podczas tworzenia aplikacji musimy się zastanowić czy aplikacja operuje na unikalnych typach danych, które są dostępne MySQL, jeżeli nie i potrzebujemy obsługi danych niestrukturyzowanych to PostgreSQL może być lepszym wyborem.

2.3 Indeksy

Również ciekawą różnicę stanowią indeksy w tych bazach. Oby dwie bazy wspierają indeksy jednak algorytmy, które są używane podczas indeksowania są różne. MySQL wspiera B-tree; R-tree, hash jednak Postgre udostępnia szerszą listę algorytmów do indeksowania takich jak GiST, SP-GiST, GIN, and BRIN.

2.4 Bezpieczeństwo

Również ciekawostką dla mnie bezpieczeństwo w tych bazach. Obie bazy danych wspierają zarządzanie użytkownikami i grupami oraz nadawanie uprawnień według roli. PostgreSQL obsługuje filtrowanie klientów w oparciu o IP oraz uwierzytelnianie za pomocą PAM i Kerberos, natomiast MySQL obsługuje PAM, natywne usługi windowsowe

oraz LDAP do uwierzytelniania użytkowników. Pod względem bezpieczeństwa obie bazy danych posiadają porównywalne opcje.

2.5 Różnicy dialektowe

Różnice dialektowe które zauważyłem pomiędzye PostgreSQL i MySQL są następne:

- W PostgreSQL wielkość liter w zapytaniach ma znaczenie. Czyli WHERE name = 'John' oraz WHERE name = 'john' nie jest tym samym. W przypadku MySQL tego nie ma.
- W przypadku stosowania cudzysłowów MySQL radzi z name = 'John' oraz name = "John". PostgreSQL działa tylko z name = 'John'.
- W przypadku współpracy z danymi typu datatime. PostgreSQL mają w nazwach funkcji dolne podkreślenie czyli CURRENT_DATE() CURRENT_TIME(). MySQL ma funkcje standardowe czyli CURDATE() CURTIME().

3 Konfiguracja PostgreSQL

```
postgres:
    image: postgres:10.5
    restart: always
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    logging:
      options:
        max-size: 10m
        max-file: "3"
    ports:
      - '5438:5432'
    volumes:
      - postgres-data:/var/lib/postgresql/data
```

4 Migracja

Jako główne narzędzie do migracji wybrałem **airbyte.io** również była podjęta próba migracji przy pomocy **PgLoader** ale nie udało się do końca poprawnie skonfigurować to narzędzie. Ze względu na brak czasu proszedłem w stronę **airbyte.io**. Szczerzę mówiąc, to narzędzie było najlepszym do tej pory softem, który używałem. Cały proces instalacji polegał na pobraniu git repozytorium oraz odpalenia docker-compose. Po odpaleniu komendy bezpośrednio mogłem zajmować się konfiguracją procesu migracji.

Proces polegał na ustawieniu odpowiedniego źródła oraz punktu koncowego, czyli źródło MySQL, oraz PostgreSQL jak punkt końcowy. Na rysunku 1 została przedstawiona część konfiguracji narzędzia. Definicja punktu końcowego wygląda identycznie, jak i źródła.

Zalety narzędzia to:

- łatwe postawienie nie mogę tego jeszcze raz nie powiedzieć, bo to było najlepsze doświadczenia z nowym narzędziem w ciągu całych studiów, duży szacunek dla developerów,
- przeniesienie wszystkich danych z wykreowaniem ich hashy, aby mieć pewność, że cały proces przeszedł poprawnie,
- możliwość ustawienia cyklicznej migracji danych. Podczas testowania zostało to ustawione na manualny tryb. Na rysunku 2 jest przedstawiona możliwość konfiguracji,
- możliwość wprowadzenia normalizacji podczas migracji danych.

Każde narzędzie ma jakieś wady i to narzędzie nie jest wyjątkiem, podczas migracji nie są przenoszone triggery, procedury, funkcji, indeksy. Ale trudno mi powiedzieć, że to jest wadą, bo z informacji z internetu wynika, że proces migracji nie polega na przeniesieniu funkcjonalność, polega on na przeniesieniu danych i z tym zadaniem narzędzie świetnie poradziło.

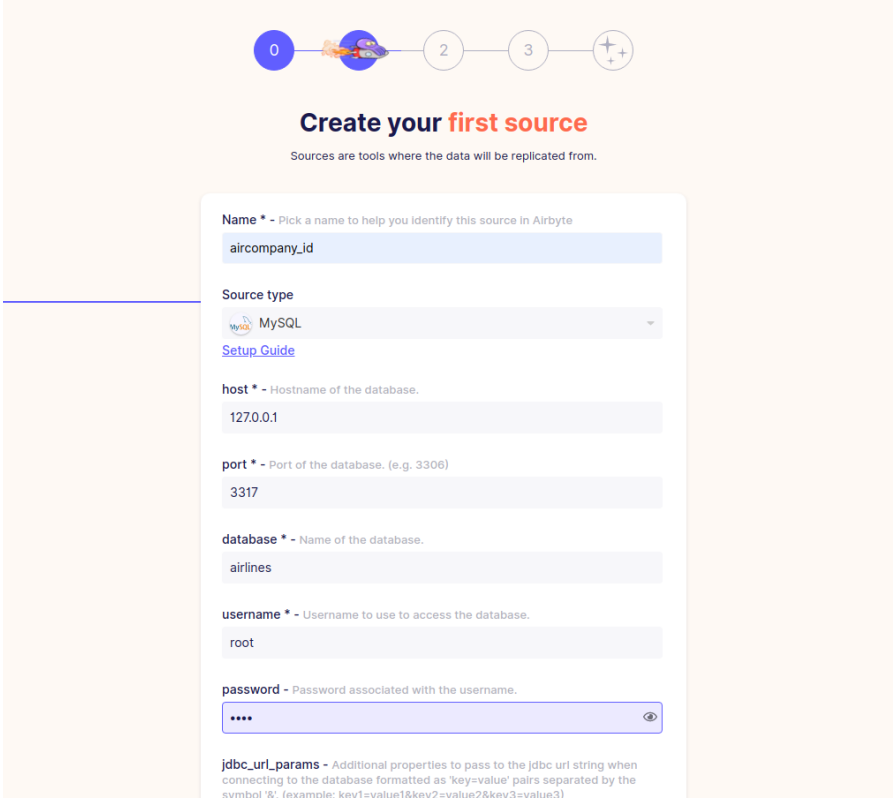
Dodatkowym minusem narzędzia jest generowania dużego zbioru dodatkowych danych typu tabeli które zawierają informacje odnośnie złódłowych tabel zazwyczaj takie tabeli mają prefikts airbyte_raw. Przykład takiej tabeli jest przedstawiony na rysunku 3 taka tabela zawiera hashe które są walidowane przez narzędzie podczas synchronizacji.

Również narzędzie dodaje hasze w tabeli docelowe które przy pomocy których można zwalidować działanie narzędzia. Na rysunku 4 jest przedstawiony proces synchronizacji jednego rekordu jak widać hasze innych rekordów nie zostały zmienione co świadczy o tym że nie były one nadpisywane. Dalej zostanie przedstawiony proces przepisywania funkcjonalności bazy MySQL do PostgreSQL.

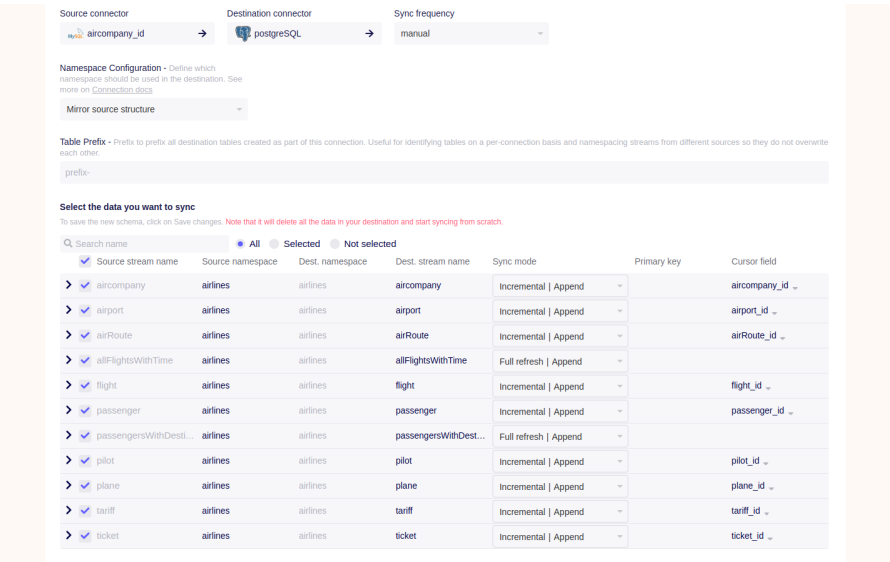
4.1 Migracji funkcji

Migrację funcjonalności rozpocząłem od przeniesienia funkcji i miałem bardzo dobre nastawienia, bo wydawało mi się, że to będzie dosyć proste zadanie, ale okazało się wręcz przeciwnie. Pierwsze napotkany problem był taki, że wybrana wersja PostgreSQL nie miała funkcji wbudowanej do obliczenia różnicy pomiędzy datami przypominam, że zaimplementowana funkcja obliczała długość lotu.

Podczas rozwiązywania tego problemu niszczyły mnie bezsensowne komunikaty o błędach które dostawałem od PostgreSQL a czasami zdarzała się sytuacja, że napisana funkcja była akceptowana przez bazę, ale po uruchomieniu wyrzucała błąd. Nie rozumiem czemu błąd nie był wyświetlany przy kroku dodawania funkcji. Na rysunku 5 został zaprezentowany wynik działania funkcji jak widać uzyskany wynik jest poprawny.

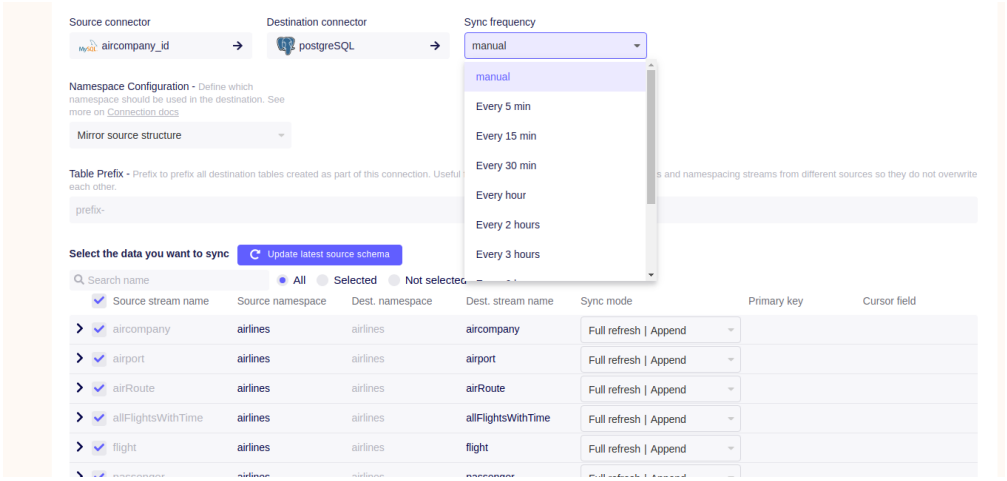


(a) Ustawienie źródła

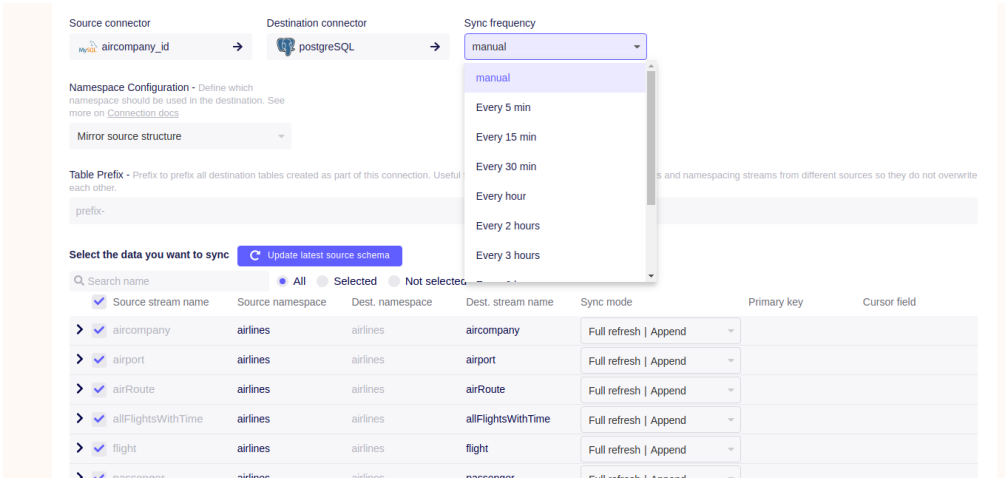


(b) Wybór tabeli które mają być przeniesione

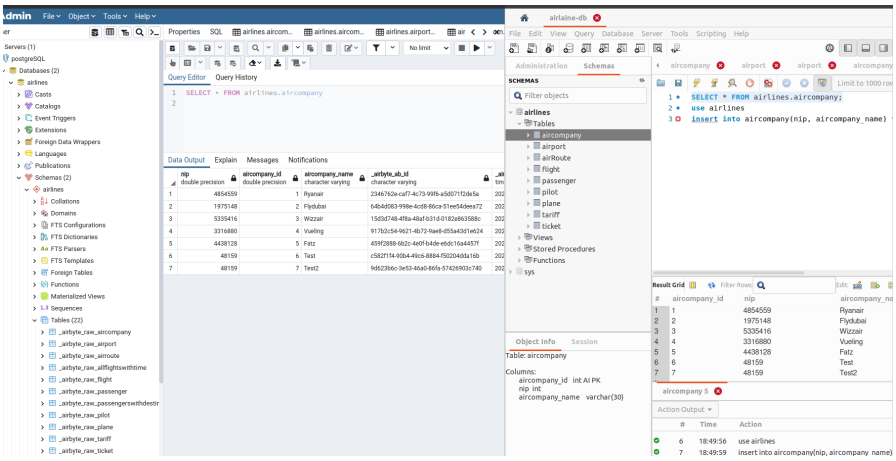
Rysunek 1: Konfiguracja airbyte.io



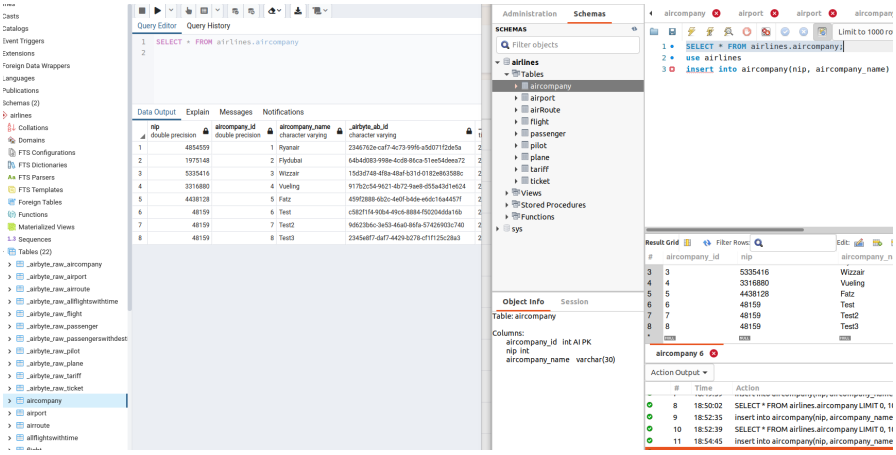
Rysunek 2: Konfiguracja migracji



Rysunek 3: Konfiguracja migracji

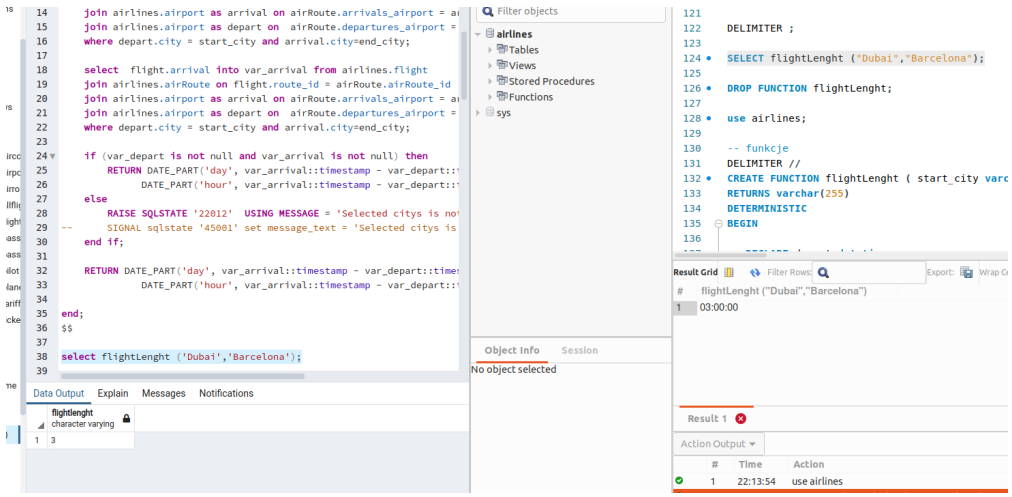


(a) Tabela przed dodanie rekordu testowego



(b) Tabela po dodaniu testowego rekordu i synchronizacji

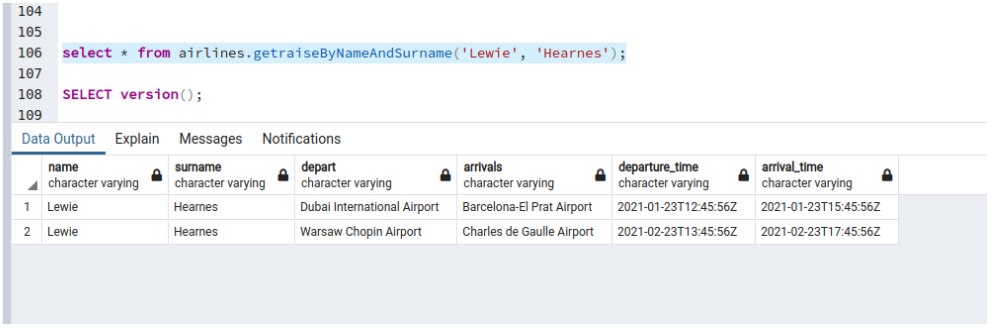
Rysunek 4: Przykład synchronizacji



Rysunek 5: Konfiguracja migracji

4.2 Migracja procedur

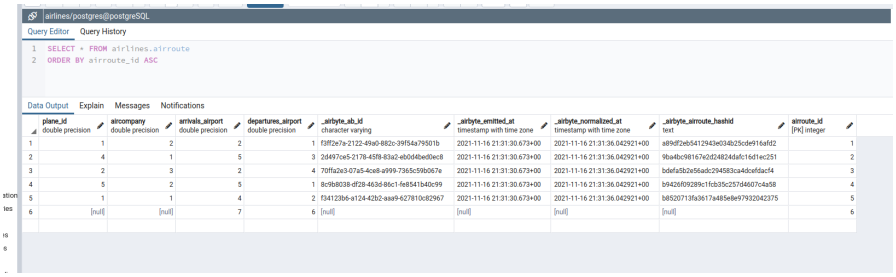
Podczas migracji procedur okazało się, że w wersji 10.5 PostgreSQL, która była używana nie ma procedur i trzeba tworzyć funkcje zamiast procedur, które zwracają void. Podczas migracji znów dużo czasu było stracone na walce z dialektem oraz funkcjami wbudowanymi. Ale procedury zostały poprawnie przeniesione. Na rysunkach 6 oraz 7 zostały przedstawione wyniki działania procedur, jak widać, wszystko działa poprawnie.



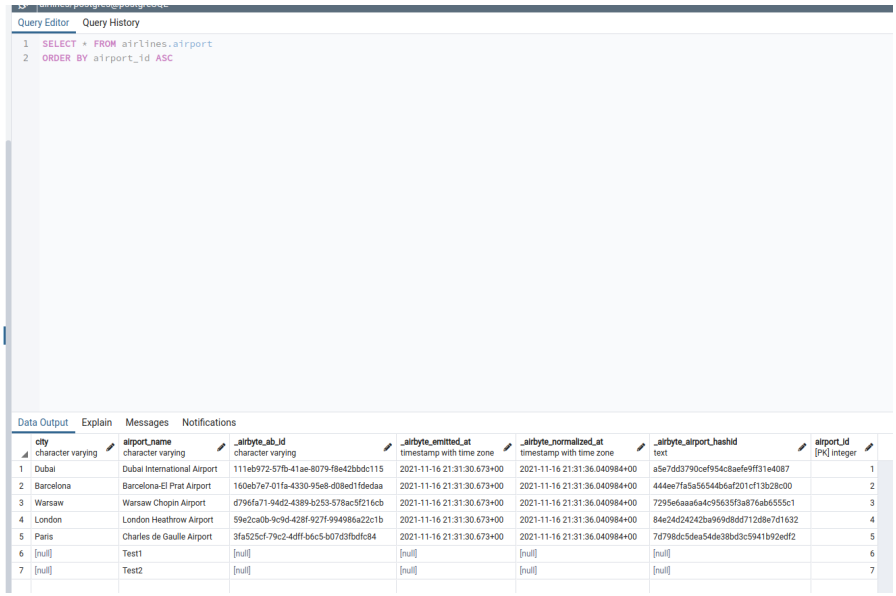
Rysunek 6: Wynik procedury po odnalezeniu lotów dla pasażera

4.3 Migracja trigera

Migracja triggerów nie obyła się bez przygód, czyli stabilnie PostgreSQL niszczył mnie error handlingiem i komunikatami o błędach. Drugą ciekawostką dla mnie było



(a) Ustawienie źródła



(b) Wybór tabeli które mają być przeniesione

Rysunek 7: Wynik działania procedury dodającej airRoute

to, że w PostgreSQl nie definiuje się bezpośrednio triggera a proces wygląda tak, że najpierw jest definiowana funkcja, która będzie odpalana w triggerze a później dopiero sam trigger.

Na rysunku 8 został przedstawiony kod oraz wynik działania triggera.

```
111  CREATE OR REPLACE FUNCTION triger_function_for_passenger()
112    RETURNS TRIGGER
113    LANGUAGE PLPGSQL
114    AS
115  $$
116  declare
117  rowcount int;
118  BEGIN
119
120      SELECT COUNT(*)
121      INTO rowcount
122      FROM airlines.passenger
123      where passenger.passport = new.passport;
124      IF rowcount > 0 THEN
125          RAISE SQLSTATE '45001'  USING MESSAGE = 'Passport must be unique. This passenger alredy in database';
126      END IF;
127      RETURN NEW;
128  END;
129  $$
130
131  CREATE TRIGGER check_passport
132    BEFORE insert
133    ON airlines.passenger
134    FOR EACH ROW
135    EXECUTE PROCEDURE triger_function_for_passenger();
136  insert into airlines.passenger(name,surname, passport) values('Lewie','Hearnes','2G3WD58236');
```

Data Output | Explain | Messages | Notifications

```
ERROR:  Passport must be unique. This passenger alredy in database
CONTEXT:  PL/pgSQL function triger_function_for_passenger() line 11 at RAISE
SQL state: 45001
```

Rysunek 8: Wynik oraz implementacja triggera

5 Wnioski

Cały proces okazał się dużo bardziej skomplikowany niż na początku myślałem po analizie informacji w internecie. Różnice w składni zapytań były bardzo małe, jednak różnice przy deklaracji funkcji, procedur oraz triggerów były bardzo duże względem MySQL. Wiązało się to z dużo, dużo większym debagowaniem oraz szukaniem rozwiązania podczas translacji, która w ciągu całej migracji niszczyła komórki nerwowe mojego mózgu. Nowością dla mnie było rozbicie wyzwalacza na deklarację funkcji oraz deklarację wyzwalacza wywołującego tę funkcję.

Szczerzę mówiąc, ta migracja bardzo mocno zniechęciła mnie do korzystania z PostgreSQL i jest mi bardzo smutno, bo akurat w pracy zaczynają używać tej bazy.

Również nie udało mi się skonfigurować połączenie dblink pomiędzy bazami danych związane to głównie było z brakiem czasu oraz doświadczenia w konfigurowaniu takiego połączenia. Dodatkowe problemy podczas konfiguracji dblinku stanowił docker to również było związane z brakiem doświadczenia konfiguracji kontenerów dla baz danych.