

Tomasz Mycielski

Zaawansowane systemy baz danych – projekt

Case study: Firma transportowa Eltrans

Eltrans to prężnie działająca firma transportowa, operująca zarówno na rynku krajowym, jak i międzynarodowym. Firma dysponuje liczną flotą, z czego większość stanowią tiry. Eltrans zatrudnia wielu pracowników, w większości kierowców, lecz także personel administracyjny, mechaników oraz specjalistów ds. logistyki.

Głównym wyzwaniem dla Eltrans jest efektywne zarządzanie flotą pojazdów i personelem. Firma potrzebuje narzędzia do monitorowania wydajności, optymalizacji tras, kontroli kosztów oraz wykrywania nadużyć (fraudów). Dodatkowo, Eltrans chce wzmocnić swoje możliwości analityczne, aby podejmować trafniejsze decyzje biznesowe oparte na danych.

Baza danych, która ma usprawnić działanie firmy, będzie centralnym punktem gromadzenia i analizy informacji o pracownikach, pojazdach, trasach i historycznym stanie pojazdów podczas całego cyklu ich eksploatacji. Dzięki niej, Eltrans będzie w stanie:

- Analizować pracę kierowców
- Monitorować wykorzystanie pojazdów
- Analizować opłacalność poszczególnych tras i zleceń, z uwzględnieniem specyfiki różnych typów pojazdów
- Generować raporty i analizy wspierające podejmowanie decyzji strategicznych
- Wykrywać nadużycia, których dopuszczają się pracownicy firmy

Wdrożenie tego systemu pozwoli Eltrans na zwiększenie efektywności operacyjnej, redukcję kosztów oraz poprawę jakości świadczonych usług. To z kolei przełoży się na wzmocnienie pozycji firmy na konkurencyjnym rynku transportowym, umożliwiając jej lepsze wykorzystanie zróżnicowanej floty pojazdów i efektywniejsze zarządzanie zasobami ludzkimi, oraz oczywiście zwiększenie zysków.

RDBMS

Na potrzeby tego projektu do zarządzania relacyjną bazą danych wykorzystany zostanie system **PostgreSQL**. Używałem tego systemu już wcześniej i nie miałem z nim problemu. Postgres ma ogromną społeczność, więc nawet jeśli napotkam jakiś problem, to na pewno szybko znajdę jego rozwiązanie w internecie. Korzystanie z Postgresa jest darmowe (open source), a instalacja jest dziecinnie prosta. Korzystam z **helm** (chart od Bitnami) do utworzenia instancji Postgresa na moim klastrze **k3s**, do którego dostęp mam przez WireGuard (Tailscale).

Screenshoty z działającego Postgresa:

```
postgres=# \l
```

Name	Owner	Encoding	Locale Provider	List of databases					Access privileges
				Collate	Ctype	Locale	ICU Rules		
postgres	postgres	UTF8	libc	en_US.UTF-8	en_US.UTF-8				
template0	postgres	UTF8	libc	en_US.UTF-8	en_US.UTF-8				=c/postgres +
template1	postgres	UTF8	libc	en_US.UTF-8	en_US.UTF-8				postgres=Ctc/postgres +
									=c/postgres +
									postgres=Ctc/postgres

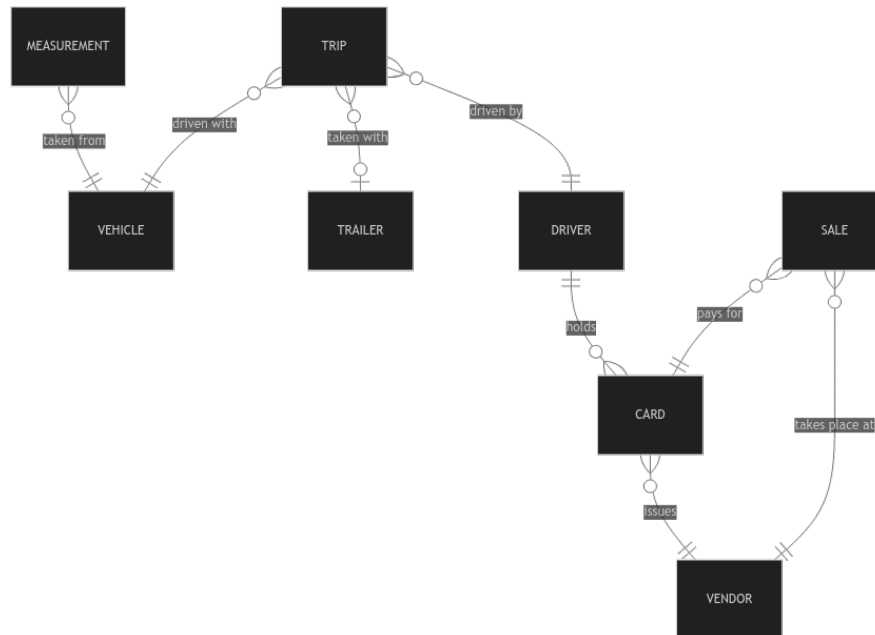
(3 rows)

```
> kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
postgres-postgresql-0	1/1	Running	0	11m

Projekt bazy danych

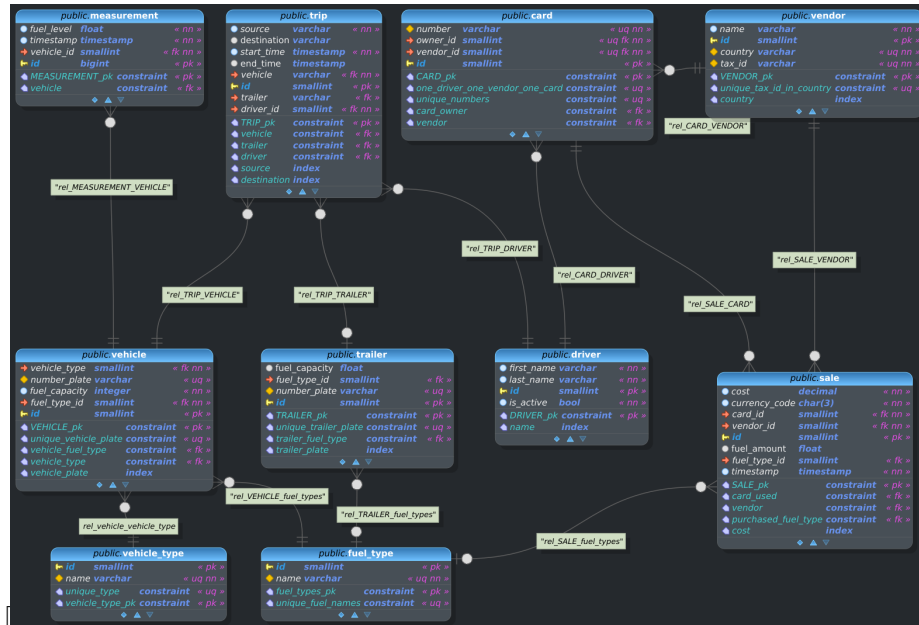
ERD



Implementacja bazy danych

Kod, który nie został umieszczony w tym sprawozdaniu dostępny jest na GitHub.

Schemat logiczny



Wszystkie tabele są w jednym schemacie bazodanowym.

Tabele niezależne

1. measurement

Zawiera pomiary poziomu paliwa podczas jazdy samochodu.

2. trip

Zawiera informacje o trasach pokonanych przez samochody.

3. card

Zawiera informacje o kartach flotowych, których używają kierowcy podczas transakcji na stacjach paliw.

4. vendor

Zawiera informacje o dostawcach paliwa, z którymi firma ma podpisane umowy.

5. vehicle

Zawiera informacje o samochodach, którymi operuje firma.

6. trailer

Zawiera informacje o naczepach, którymi operuje firma.

7. driver

Zawiera informacje o kierowcach zatrudnionych w firmie.

8. sale

Zawiera informacje o transakcjach zakupu paliwa dokonanych przez kierowców.

Tablice słownikowe

1. vehicle_type
2. fuel_type

Odstępstwa od 3NF

- W tabeli trip:
 - **source** i **destination** są przechowywane jako varchar zamiast referencji do tabeli locations

Uzasadnienie:

- Uniknięcie skomplikowanych joinów przy wyszukiwaniu tras
- Lokalizacje mogą być bardzo zmienne i nie zawsze standardowe (np. adresy klientów)
- Dodano indeksy hash aby przyspieszyć wyszukiwanie po tych kolumnach

- W tabeli sale:
 - **currency_code** jest przechowywane bezpośrednio zamiast referencji do tabeli walut

Uzasadnienie:

- Lista walut jest względnie stała (ISO 4217)
- Uniknięcie dodatkowego joina przy każdym zapytaniu o sprzedaż
- Typ **char(3)** zapewnia poprawny format kodu waluty

- W tabeli vendor:
 - **country** jest przechowywane jako varchar zamiast referencji do tabeli krajów

Uzasadnienie:

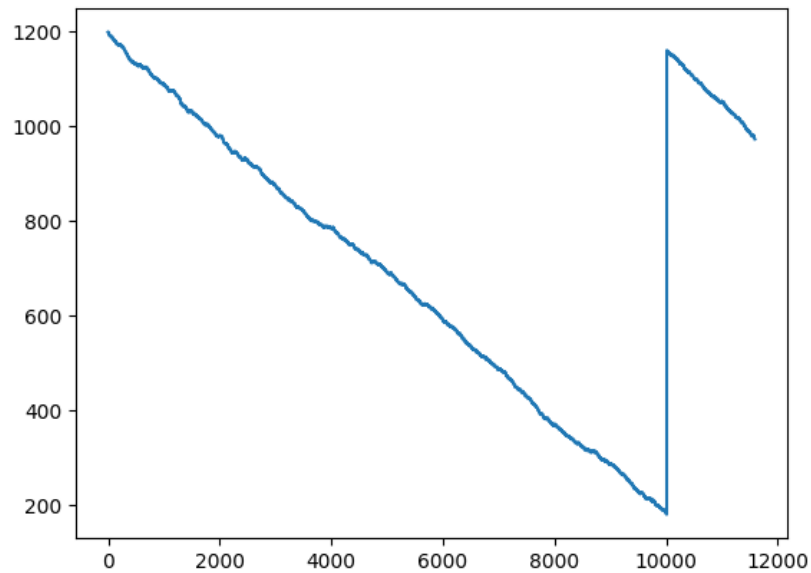
- Lista krajów jest względnie stała
- Często używane w zapytaniach (dodano indeks hash)
- Uniknięcie dodatkowego joina przy wielu zapytaniach

Istotne decyzje projektowe

- Użycie `number_plate` jako klucza obcego w powiązaniach pojazdów i naczep z przejazdami:
 - Zamiast używać `id`, użyłem numeru rejestracyjnego jako klucza obcego
 - Uzasadnienie: Szybsze wyszukiwanie po numerach rejestracyjnych (które są często używane w biznesie transportowym) bez konieczności joinów
- Separacja `vehicle_type` i `fuel_type` do osobnych tabel:
 - Zamiast przechowywać te informacje jako teksty w tabeli `vehicle`
 - Uzasadnienie: zapewnia spójność danych
- Użycie `timestamp` zamiast `datetime`:
 - Uzasadnienie: Uniknięcie problemów ze strefami czasowymi (`timestamp` przechowuje UTC)
- Struktura kart paliwowych:
 - Karta jest powiązana zarówno z kierowcą jak i vendorem
 - Constraint `one_driver_one_vendor_one_card` zapewnia że kierowca może mieć tylko jedną kartę u danego vendora
 - Uzasadnienie: Odzwierciedla rzeczywisty model biznesowy gdzie kierowcy mogą mieć różne karty od różnych dostawców

Generowanie danych

Do wygenerowania danych wykorzystałem własne skrypty SQL i Python. Aby zwiększyć realizm danych wielokrotnie wykorzystywałem losowanie z rozkładu normalnego (na przykład do generowania cen paliwa). Na szczególną uwagę zasługuje program, który generuje pomiary poziomu paliwa podczas przejazdów samochodów oraz zapisy transakcji zakupu paliwa dokonanych przez kierowców gdy poziom był niski.



Widoczny na wykresie szum został wprowadzony do danych celowo w celu symulacji niedokładności przyrządów pomiarowych oraz zmiennego spalania pojazdu podczas podróży.

Użytkownicy

Stworzyłem użytkowników symbolizujących działy w firmie.

```
CREATE USER financial WITH PASSWORD 'financial';
CREATE USER compliance WITH PASSWORD 'compliance';
```

Następnie nadałem im dostęp do widoków, które zostały stworzone dla ich działów.

```
GRANT SELECT ON <widok> TO <uzytkownik>
REVOKE ALL ON SCHEMA public FROM <uzytkownik>
```

Testy dostępów

```
> PGPASSWORD=financial psql -h raspberrypi -p 30956 -U financial -d eltrans
psql (14.13 (Homebrew), server 17.0)
WARNING: psql major version 14, server major version 17.
        Some psql features might not work.
Type "help" for help.

eltrans=> select * from fuel_cost_per_driver_hour limit 10;
 driver_id | first_name | last_name | fuel_spending_per_hour
-----+-----+-----+-----
          10 | Ewa       | Nowak     | 89.69
          14 | Tomasz    | Woźniak   | 94.65
           8 | Marek     | Michalski | 110.19
           4 | Paweł     | Kowalski  | 147.46
           5 | Joanna    | Kowalczyk | 151.26
          13 | Marek     | Kozłowski | 153.24
           1 | Elżbieta  | Wiśniewska | 155.02
           9 | Józef     | Nowakowski | 163.58
           3 | Joanna    | Kowalska  | 168.10
           6 | Elżbieta  | Adamczyk  | 168.58
(10 rows)

eltrans=> select * from driver;
ERROR: permission denied for table driver
eltrans=>
```

```
> PGPASSWORD=compliance psql -h raspberrypi -p 30956 -U compliance -d eltrans
psql (14.13 (Homebrew), server 17.0)
WARNING: psql major version 14, server major version 17.
        Some psql features might not work.
Type "help" for help.






eltrans=> select * from suspected_fraudsters;
 first_name | last_name | vehicle_reg | timestamp | fuel_stolen
-----+-----+-----+-----+-----
Paweł      | Kowalski  | WAGVE03    | 2024-01-01 16:36:00 | 70.81
Aleksandra | Kowalczyk | WALWR09    | 2024-01-01 16:11:30 | 274.45
Aleksandra | Kowalczyk | WALWR09    | 2024-01-04 15:28:45 | 66.64
Tomasz     | Woźniak   | WAMDC09    | 2024-01-04 16:52:09 | 119.28
Elżbieta   | Wiśniewska | WAMOA01    | 2024-01-01 16:53:54 | 60.21
Adam        | Wiśniewski | WANUY09    | 2024-01-01 16:24:18 | 192.40
Adam        | Wiśniewski | WANUY09    | 2024-01-07 16:47:09 | 220.61
Marek       | Kozłowski | WAOTM08    | 2024-01-04 15:39:51 | 134.67
Jan         | Mazur     | WATUW05    | 2024-01-07 16:05:21 | 179.74
Elżbieta   | Adamczyk  | WAXIU05    | 2024-01-04 15:04:09 | 52.69
Katarzyna  | Pawłowska | WAXJH03    | 2024-01-01 15:53:21 | 79.20
Katarzyna  | Pawłowska | WAXJH03    | 2024-01-04 16:28:15 | 94.00
(12 rows)

eltrans=> select * from driver;
ERROR: permission denied for table driver
eltrans=>
```

Przykładowe zapytania

Którzy kierowcy wykonali najwięcej tras w danym miesiącu?

```
SELECT driver.first_name,  
       driver.last_name,  
       count(*) AS trips_taken  
FROM driver  
JOIN trip ON driver.id=trip.driver_id  
WHERE trip.start_time>='2024-01-01'  
      AND trip.end_time<'2024-02-01'  
GROUP BY driver.first_name,  
         driver.last_name  
ORDER BY trips_taken DESC  
LIMIT 10;
```

#	 A-Z first_name 	A-Z last_name 	123 trips_taken 
1	Zbigniew 	Nowak	3
2	Adam	Kowalczyk	3
3	Łukasz	Szymański	3
4	Joanna	Wiśniewska	3
5	Marcin	Kwiatkowski	3
6	Marcin	Woźniak	2
7	Adam	Kowalski	2
8	Andrzej	Lewandowski	2
9	Grzegorz	Wójcik	2
10	Katarzyna	Zielińska	2

Jaka jest średnia cena oleju napędowego w Polsce?

W tym zapytaniu wykorzystane zostało **podzapytanie**.

```
SELECT ft.name AS "Nazwa paliwa",  
       ROUND(CAST(sum(cost) / sum(fuel_amount) AS numeric), 2) AS "Cena paliwa [EUR]"  
FROM sale s  
JOIN fuel_type ft ON ft.id = s.fuel_type_id  
WHERE vendor_id IN  
      (SELECT id  
       FROM vendor  
       WHERE country = 'Poland')
```



```
AND ft.name = 'diesel'
GROUP BY ft.name;
```

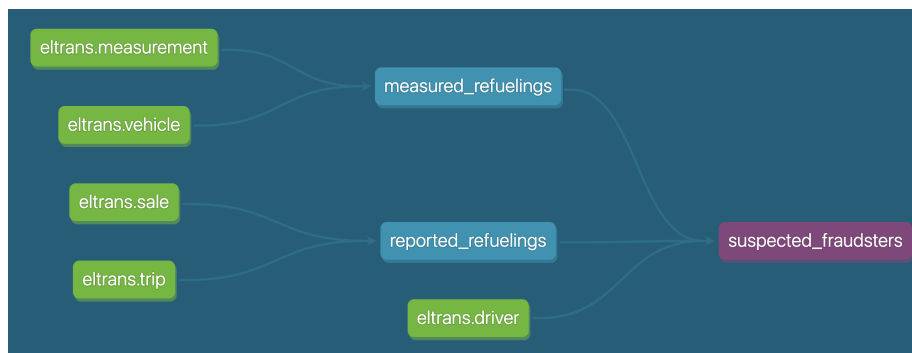
#	A-Z Nazwa paliwa ↓↑	123 Cena paliwa [EUR] ↓↑
1	diesel	1.5

Perspektywy

Utworzyłem sześć perspektyw.

Wykrywanie fraudów

Kierowcy kupują paliwo na stacjach paliw przy użyciu firmowych kart flotowych. Kierowca może próbować zatankować część paliwa do własnego kanistra aby w ten sposób “dorobić” sobie bonus do pensji. Jest to oczywiście kradzież. W celu jej wykrywania stworzone zostały perspektywy.



```
SELECT * FROM reported_refuelings;
```

#	123 fuel_added ↓↑	timestamp ↓↑	A-Z vehicle_reg ↓↑	123 driver_id ↓↑
1	1249.16	2024-01-01 16:53:55.000	WAMOA01	1
2	754.14	2024-01-04 14:56:43.000	WAMOA01	1
3	939.3	2024-01-07 15:21:34.000	WAMOA01	1
4	1381.07	2024-01-01 16:24:19.000	WANUY09	2
5	921.61	2024-01-04 15:23:46.000	WANUY09	2
6	1409.22	2024-01-07 16:47:10.000	WANUY09	2
7	1175.58	2024-01-01 16:42:01.000	WAYHI08	3
8	1045.44	2024-01-04 15:22:16.000	WAYHI08	3
9	1061.8	2024-01-01 16:36:01.000	WAGVE03	4
10	841.77	2024-01-04 14:59:01.000	WAGVE03	4
11	975.54	2024-01-01 15:26:46.000	WALNO01	5
12	862.92	2024-01-04 15:13:19.000	WALNO01	5

SELECT * FROM measured_refuelings;

#	A-Z vehicle_reg ↓↑	123 fuel_added ↓↑	timestamp ↓↑
1	WAMOA01	1188.9534073964683	2024-01-01 16:53:54.000
2	WAMOA01	754.0765134427759	2024-01-04 14:56:42.000
3	WAMOA01	938.4067551316404	2024-01-07 15:21:33.000
4	WANUY09	1188.66922327344	2024-01-01 16:24:18.000
5	WANUY09	922.1991233817599	2024-01-04 15:23:45.000
6	WANUY09	1188.6094756769883	2024-01-07 16:47:09.000
7	WAYHI08	1132.7650542949686	2024-01-01 16:42:00.000
8	WAYHI08	1003.6869848406427	2024-01-04 15:22:15.000
9	WAGVE03	990.9931671729654	2024-01-01 16:36:00.000
10	WAGVE03	841.5433555682264	2024-01-04 14:59:00.000
11	WALNO01	975.6183475425687	2024-01-01 15:26:45.000
12	WALNO01	863.3450326906885	2024-01-04 15:13:18.000
13	WALNO01	1007.9772090774426	2024-01-07 15:12:54.000

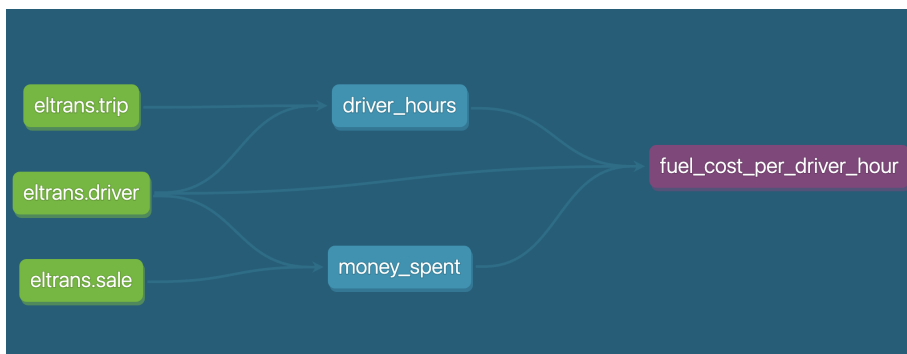
SELECT * FROM suspected_fraudsters;

Enter a SQL expression to filter results, e.g. column_name = 'v'

#	A-Z first_name ↓↑	A-Z last_name ↓↑	A-Z vehicle_reg ↓↑	timestamp ↓↑	123 fuel_stolen ↓↑
1	Paweł	Kowalski	WAGVE03	2024-01-01 16:36:00.000	70.81
2	Aleksandra	Kowalczyk	WALWR09	2024-01-01 16:11:30.000	274.45
3	Aleksandra	Kowalczyk	WALWR09	2024-01-04 15:28:45.000	66.64
4	Tomasz	Woźniak	WAMDC09	2024-01-04 16:52:09.000	119.28
5	Elżbieta	Wiśniewska	WAMOA01	2024-01-01 16:53:54.000	60.21
6	Adam	Wiśniewski	WANUY09	2024-01-01 16:24:18.000	192.4
7	Adam	Wiśniewski	WANUY09	2024-01-07 16:47:09.000	220.61
8	Marek	Kozłowski	WAOTM08	2024-01-04 15:39:51.000	134.67
9	Jan	Mazur	WATUW05	2024-01-07 16:05:21.000	179.74
10	Elżbieta	Adamczyk	WAXIU05	2024-01-04 15:04:09.000	52.69
11	Katarzyna	Pawłowska	WAXJH03	2024-01-01 15:53:21.000	79.2
12	Katarzyna	Pawłowska	WAXJH03	2024-01-04 16:28:15.000	94

Koszt paliwa na godzinę jazdy kierowcy

Firma chce nagradzać kierowców którzy wybierają tanie stacje i nie mają ciężkiej nogi. W tym celu skorzystać można z perspektyw, które informują o średnim koszcie paliwa spalonego przez danego kierowcę na godzinę jazdy.



```
SELECT * FROM driver_hours;
```

#	A-Z first_name ↓↑	A-Z last_name ↓↑	123 hours_driven ↓↑	123 driver_id ↓↑
1	Elżbieta	Wiśniewska	26.75263501749999999	1
2	Adam	Wiśniewski	26.24400638444...	2
3	Joanna	Kowalska	18.769775878888889	3
4	Paweł	Kowalski	18.493900615277778	4
5	Joanna	Kowalczyk	26.494782386944444	5
6	Elżbieta	Adamczyk	17.443545604722222	6

```
SELECT * FROM money_spent;
```

#	A-Z first_name ↓↑	A-Z last_name ↓↑	123 driver_id ↓↑	123 total_spending ↓↑
1	Elżbieta	Wiśniewska	1	4147.2
2	Adam	Wiśniewski	2	5603.63
3	Joanna	Kowalska	3	3155.21
4	Paweł	Kowalski	4	2727.02
5	Joanna	Kowalczyk	5	4007.63
6	Elżbieta	Adamczyk	6	2940.67
7	Katarzyna	Pawłowska	7	3862.43

```
SELECT * FROM fuel_cost_per_driver_hour;
```

#	123 driver_id ↓↑	A-Z first_name ↓↑	A-Z last_name ↓↑	123 fuel_spending_per_hour ↓↑
1	10	Ewa	Nowak	89.69
2	14	Tomasz	Woźniak	94.65
3	8	Marek	Michalski	110.19
4	4	Paweł	Kowalski	147.46
5	5	Joanna	Kowalczyk	151.26
6	13	Marek	Kozłowski	153.24
7	1	Elżbieta	Wiśniewska	155.02
8	9	Józef	Nowakowski	163.58
9	2	Adam	Wiśniewski	166.37

Indeksy

W bazie stworzyłem indeksy aby przyspieszyć niektóre zapytania, które dają istotne informacje z punktu widzenia biznesowego. Tam, gdzie zapytania mają formę przyrównania do konkretnej wartości (na przykład numer rejestracyjny) wykorzystany został indeks typu hash. W przeciwnym wypadku (oraz w indeksie złożonym z kilku kolumn ponieważ takich nie obsługuje hash) wykorzystałem indeks btree.

trip

Utworzone zostały indeksy typu hash na kolumnach **source** i **destination** aby przyspieszyć zapytania o miejsca, gdzie jeżdżą pojazdy.

vendor

Utworzony został indeks typu hash na kolumnie **country** aby przyspieszyć zapytania o kraje w których zarejestrowane są działalności dostawców paliwa.

sale

Utworzony został indeks btree na kolumnie `cost` zawierający wartości `cost`, `fuel_amount` i `vendor_id` aby przyspieszyć zapytania o średnie ceny paliwa u dostawców.

driver

Utworzony został indeks złożony typu btree na kolumnach `first_name` i `last_name` aby przyspieszyć zapytania o kierowców przy użyciu ich imion (w przeciwieństwie do ich id).

vehicle i trailer

W obu tabelach utworzono indeksy typu hash na kolumnach z numerami rejestracyjnymi.

Benchmark

Bez indeksu na kolumnie `timestamp` w `measurement` wykonanie zapytania `SELECT min(m.timestamp) FROM measurement m` zajmowało około 900 milisekund. Po dodaniu indeksu btree na tę kolumnę czas wykonania tego zapytania spadł do 12 milisekund! To redukcja o niemal **99%**!

Etap 2

Procedura do dodawania kierowców

```
CREATE OR REPLACE PROCEDURE add_driver(  
    p_first_name VARCHAR,  
    p_last_name  VARCHAR,  
    p_is_active  BOOLEAN DEFAULT true  
)  
LANGUAGE plpgsql  
AS $$  
BEGIN  
    IF p_first_name IS NULL OR p_last_name IS NULL THEN  
        RAISE EXCEPTION 'First name and last name cannot be null';  
    END IF;  
  
    INSERT INTO driver (first_name, last_name, is_active)  
    VALUES (p_first_name, p_last_name, p_is_active);  
END;  
$$;
```

```

30
31 begin transaction;
32
33 call add_driver('Eryk', 'Elektryczny', true);
34 select count(*) from driver d where d.first_name = 'Eryk';
35
36 rollback;

```

Ln 30, Col 1, Pos 5

Result - 1

Enter a SQL expression to filter results, e.g. column_name=10

#	123 count
1	1

VALUE

Funkcja eur_to_pln

```

CREATE OR REPLACE FUNCTION eur_to_pln(amount numeric)
RETURNS numeric
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN ROUND(amount * 4.37, 2);
END;
$$;

```

```

33
34 select s.cost as "cost [EUR]", eur_to_pln(s.cost) as "cost [PLN]"
35 from sale s where currency_code='EUR';

```

Ln 33, Col 1

Result - 1

Enter a SQL expression to filter results, e.g. column_name=10

#	123 cost [EUR]	123 cost [PLN]
1	1738.25	7596.15
2	1006.38	4397.88
3	1402.57	6129.23
4	1948.4	8514.51
5	1415.23	6184.56

Backup

```
PGPASSWORD=$POSTGRES_PASSWORD pg_dump -h 127.0.0.1 -p 5432 -U postgres -d eltrans > eltrans.
```

Wykorzystałem program `pg_dump` do utworzenia kopii zapasowej mojej bazy danych. Poprawność utworzonej kopii zapasowej zweryfikowałem tworząc nowy klaster kubernetes (minikube), instalując tam Postgresa, a następnie wykonując plik `.sql` z kopią zapasową na tym Postgresie.

```
PGPASSWORD="$POSTGRES_PASSWORD" psql --host 127.0.0.1 -U postgres -d postgres
List of schemas
Name | Owner
-----
public | pg_database_owner
(1 row)

postgres=# select datname from pg_database;
 datname
-----
 postgres
 template1
 template0
(3 rows)

postgres=# select count(*) from public.
public.card
public.card_id_seq
public.driver
public.driver_hours
public.driver_id_seq
public.fuel_cost_per_driver_hour
public.fuel_type
public.fuel_type_id_seq
public.measured_refuelings
postgres=# select count(*) from public.driver;
 count
-----
      26
(1 row)

postgres=#
```

Złożona procedura

```
CREATE OR REPLACE PROCEDURE add_fuel_card(
    p_driver_id integer,
    p_vendor_id integer,
    p_card_number varchar
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_driver_exists boolean;
    v_is_active boolean;
    v_card_exists boolean;
BEGIN
    SELECT EXISTS (
        SELECT 1
        FROM driver
        WHERE id = p_driver_id::smallint
    ) INTO v_driver_exists;

    IF NOT v_driver_exists THEN
```

```

        RAISE EXCEPTION 'Driver with ID % does not exist', p_driver_id;
    END IF;

    SELECT is_active
    FROM driver
    WHERE id = p_driver_id::smallint
    INTO v_is_active;

    IF NOT v_is_active THEN
        RAISE EXCEPTION 'Driver with ID % is not active', p_driver_id;
    END IF;

    SELECT EXISTS (
        SELECT 1
        FROM card
        WHERE owner_id = p_driver_id::smallint
        AND vendor_id = p_vendor_id::smallint
    ) INTO v_card_exists;

    IF v_card_exists THEN
        RAISE EXCEPTION 'Driver % already has a fuel card with vendor %', p_driver_id, p_vendor_id;
    END IF;

    SELECT EXISTS (
        SELECT 1
        FROM card
        WHERE number = p_card_number
    ) INTO v_card_exists;

    IF v_card_exists THEN
        RAISE EXCEPTION 'Card number % is already in use', p_card_number;
    END IF;

    INSERT INTO card (number, owner_id, vendor_id)
    VALUES (p_card_number, p_driver_id::smallint, p_vendor_id::smallint);

    RAISE NOTICE 'Successfully added new fuel card for driver %', p_driver_id;

EXCEPTION
    WHEN foreign_key_violation THEN
        RAISE EXCEPTION 'Vendor with ID % does not exist', p_vendor_id;

    WHEN OTHERS THEN
        RAISE EXCEPTION 'An unexpected error occurred: %', SQLERRM;
END;
$$;

```



Ta procedura dodaje nową kartę dla kierowcy. Sprawdza ona najpierw:

- czy kierowca istnieje?
- czy kierowca jest aktywny?
- czy vendor istnieje?
- czy kierowca nie ma już karty u tego vendora?

Wykorzystane są do tego tabele:

- driver
- card
- vendor

Procedura ma zaimplementowaną obsługę błędów. Przykład błędu związanego z dodaniem drugiej karty dla tego samego kierowcy u tego samego vendora:



Error executing query:
SQL Error [P0001]: ERROR: An unexpected error occurred: Driver 25 already has a fuel card with vendor 2
Where: PL/pgSQL function add_fuel_card(integer,integer,character varying) line 64 at RAISE

CLOSEDETAILSRETRY

W Postgresie procedury domyślnie wykonują się w transakcji, nie trzeba tego implementować samodzielnie w samym kodzie procedury.

Trigger

```
CREATE OR REPLACE FUNCTION validate_fuel_type()
RETURNS TRIGGER AS $$
DECLARE
    vehicle_fuel_type SMALLINT;
BEGIN
    SELECT v.fuel_type_id INTO vehicle_fuel_type
    FROM trip t
    JOIN vehicle v ON v.number_plate = t.vehicle
    WHERE t.driver_id = (SELECT owner_id FROM card WHERE id = NEW.card_id)
    ORDER BY t.start_time DESC
    LIMIT 1;

    IF NEW.fuel_type_id != vehicle_fuel_type THEN
        RAISE EXCEPTION 'Invalid fuel type for vehicle. Expected fuel type ID: %', vehicle_fuel_type;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER validate_fuel_type_trigger
    BEFORE INSERT ON sale
    FOR EACH ROW
    WHEN (NEW.fuel_type_id IS NOT NULL)
    EXECUTE FUNCTION validate_fuel_type();
```

Ten trigger uniemożliwia dodanie zakupu paliwa innego typu, niż przypisany do samochodu, którym porusza się dany kierowca.



Error executing query:

SQL Error [P0001]: ERROR: Invalid fuel type for vehicle. Expected fuel type ID: 1
Where: PL/pgSQL function validate_fuel_type() line 15 at RAISE

CLOSE

DETAILS

RETRY

Różnica między dialektami SQL Postgres i MySQL

1. Typy danych

<i>-- PostgreSQL</i>	<i>/ MySQL</i>
TIMESTAMP WITH TIME ZONE	DATETIME (brak stref czasowych)
SERIAL	AUTO_INCREMENT
VARCHAR bez limitu	VARCHAR wymaga limitu
NUMERIC/DECIMAL(p,s)	DECIMAL(p,s) z mniej precyzyjnymi obliczeniami
ARRAY	Brak natywnych tablic
JSON, JSONB	JSON (przed MySQL 8.0 jako TEXT)

2. Funkcje

```
-- Konkatenacja stringów
-- PostgreSQL
SELECT 'Hello' || ' ' || 'World';

-- MySQL
SELECT CONCAT('Hello', ' ', 'World');
```

```
-- Praca z datami
-- PostgreSQL
SELECT EXTRACT(EPOCH FROM timestamp_column);

-- MySQL
SELECT UNIX_TIMESTAMP(timestamp_column);
```

2. Triggery

```
-- PostgreSQL
```

```

CREATE TRIGGER nazwa_triggera
AFTER INSERT ON tabela
FOR EACH ROW
EXECUTE FUNCTION nazwa_funkcji();

-- MySQL
CREATE TRIGGER nazwa_triggera
AFTER INSERT ON tabela
FOR EACH ROW
BEGIN
    -- kod triggera bezpośrednio tutaj
END;

```

- PostgreSQL wymaga osobnej funkcji dla triggera
- MySQL zawiera kod triggera bezpośrednio w definicji
- MySQL nie ma INSTEAD OF triggerów
- PostgreSQL pozwala na triggerzy na widokach

4. Sekwencje

```

-- PostgreSQL
CREATE SEQUENCE seq_name;
CREATE TABLE table_name (
    id integer DEFAULT nextval('seq_name')
);

-- MySQL
CREATE TABLE table_name (
    id integer AUTO_INCREMENT
);

```

Najważniejszą różnicą między Postgres a MySQL jest wykorzystanie wątków do obsługi połączeń z klientami w MySQL vs procesów w Postgres. Nie ma to jednak znaczenia dla nas przy migracji.

Wybrane narzędzia do migracji

1. Airbyte

- Obsługa ponad 350 źródeł i miejsc docelowych danych
- Przyrostowa synchronizacja danych (przenosi tylko zmienione dane)
- Elastyczna obsługa schematów danych
- Możliwość budowania własnych konektorów
- Dostępne jako open-source

Wszechstronne narzędzie, szczególnie przydatne w przypadku złożonych migracji z wielu źródeł. Duża elastyczność dzięki możliwości tworzenia własnych konektorów.

2. Azure Migrate

- Dedykowane do migracji do chmury Azure
- Centralna kontrola projektów migracji
- Szacowanie kosztów migracji
- Optymalizacja wydajności w środowisku Azure
- Wiele dostępnych źródeł danych

Dobre narzędzie jeśli ktoś pracuje z Azure. Niestety płatne, ale za to daje dużo informacji o szacowanych kosztach.

3. AWS Database Migration Service

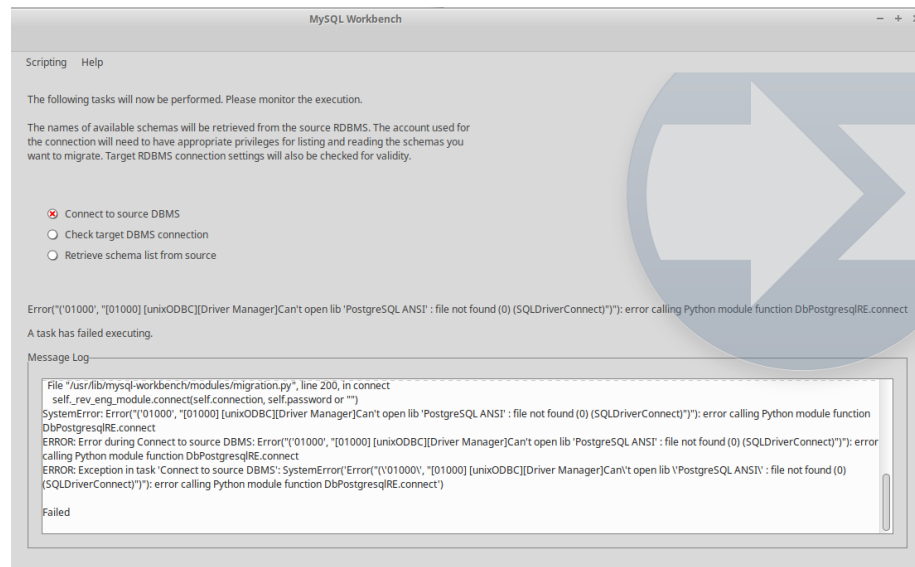
- Ciągła replikacja danych
- Albo batchowa
- Pełna integracja z usługami AWS
- Wiele dostępnych źródeł danych

Podobnie jak rozwiązanie od Azure, tylko w świecie AWS.

Migracja

MySQL Workbench

Próba migracji przy użyciu narzędzia MySQL Workbench zakończyła się niepowodzeniem, zarówno na MacOS, NixOS, Ubuntu i Windows 10.

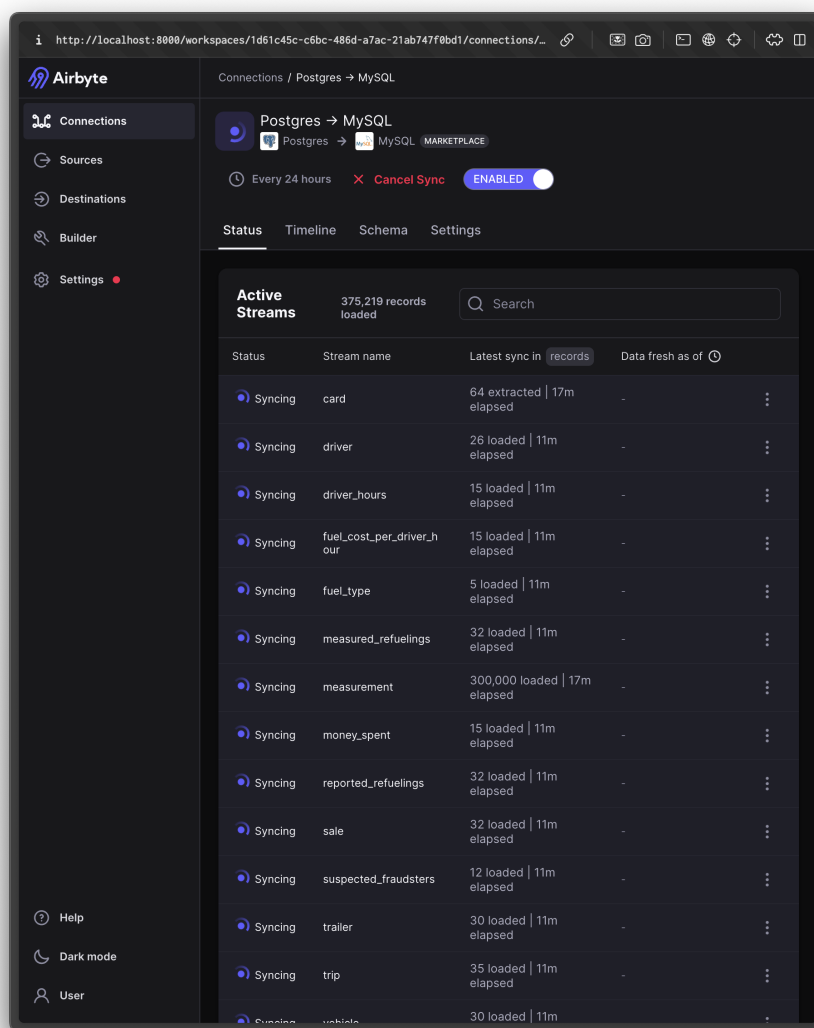


Airbyte

Drugą próbę podjąłem przy użyciu narzędzia Airbyte. Zainstalowałem Airbyte na swoim komputerze i skonfigurowałem dostęp do obu RDBMSów. Migracja

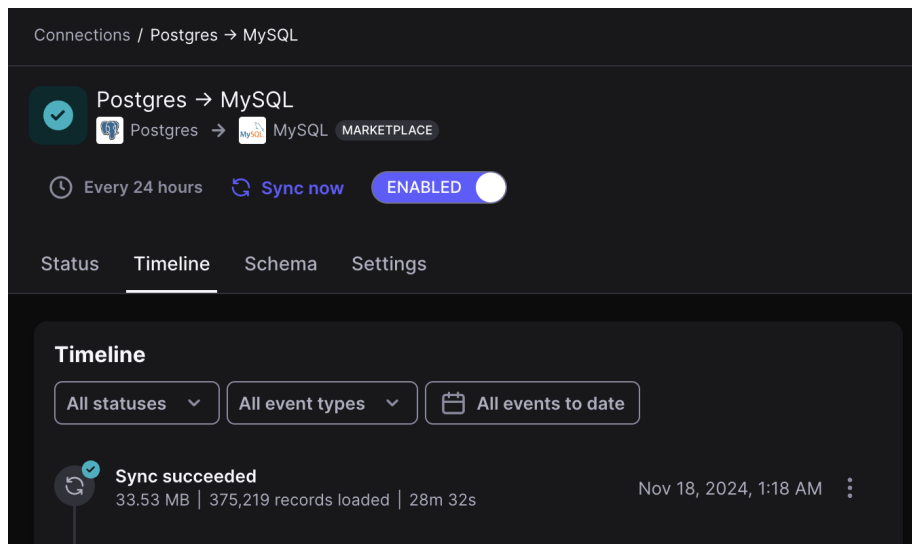
przy użyciu Airbyte jest czasochłonna – moja baza miała niecałe 400000 rekordów i ważyła 33.5 MB, a jej migracja zajęła ponad 28 minut! Prawdopodobnie ze względu na niskowydajny sprzęt na którym są bazy. Na szczęście migracja jest dość prosta. Nie miałem żadnych problemów ze zmigrowaniem danych pomiędzy RDBMSami.

Dashboard Airbyte w trakcie migracji:

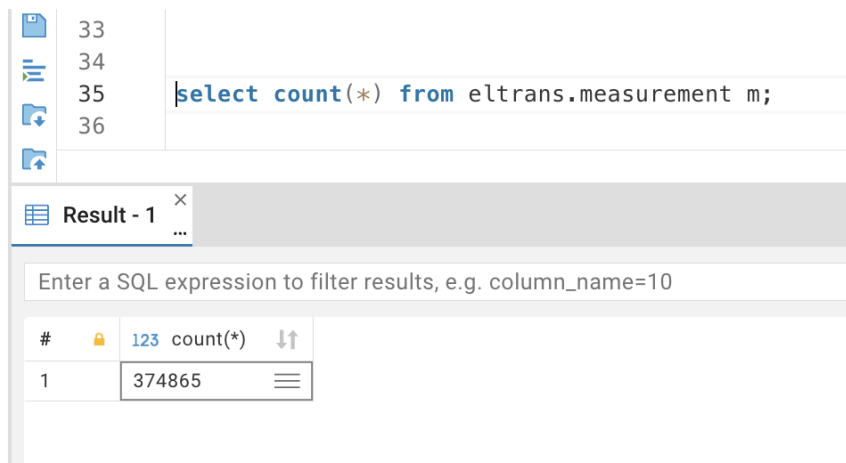


Status	Stream name	Latest sync in	records	Data fresh as of
Syncing	card	64 extracted 17m elapsed	-	-
Syncing	driver	26 loaded 11m elapsed	-	-
Syncing	driver_hours	15 loaded 11m elapsed	-	-
Syncing	fuel_cost_per_driver_hour	15 loaded 11m elapsed	-	-
Syncing	fuel_type	5 loaded 11m elapsed	-	-
Syncing	measured_refuelings	32 loaded 11m elapsed	-	-
Syncing	measurement	300,000 loaded 17m elapsed	-	-
Syncing	money_spent	15 loaded 11m elapsed	-	-
Syncing	reported_refuelings	32 loaded 11m elapsed	-	-
Syncing	sale	32 loaded 11m elapsed	-	-
Syncing	suspected_fraudsters	12 loaded 11m elapsed	-	-
Syncing	trailer	30 loaded 11m elapsed	-	-
Syncing	trip	35 loaded 11m elapsed	-	-
Syncing	vehicle	30 loaded 11m elapsed	-	-



Informacja o udanej migracji:



Liczba pomiarów poziomu paliwa w docelowym RDBMS jest poprawna:



Database link Przy użyciu Airbyte zrealizowałem także replikację danych co godzinę (choć da się nawet co minutę).

 Postgres →  MySQL MARKETPLACE

StatusTimelineSchemaSettings

Settings

Connection name

Name for your connection

Postgres → MySQL

Schedule type

How you want your syncs to be triggered

Scheduled

Replication frequency

How often your data will sync to your destination

Every hour

[Advanced settings >](#)

23