DB Projekt – Daniel Michrowski

Etap1

1. Opis projektu

W projekcie przedstawiam koncepcje bazy danych dla dowolnego przedsiębiorstwa usługowego – handlowego.

Do stworzenie modelu bazy danych wykorzystano opensourc'owy DBMS PostgreSQL ver. 12.4. System jest darmowy, posiada bogate dokumentacje oraz jest wspierany przez szeroką społeczność – z tych powodów wybrałem system PostgreSQL.

Baza danych składa się z następujących tabel:

- Contact_person
- Product
- Producer
- Client
- Place_object
- Supervisor
- Investment
- Sub_investment
- Product_order

Poniżej przedstawiam, krótki opis tabel:

Contact_person – tabela zawierająca kontakt to przedstawicieli handlowych i osób kontaktowych poszczególnych firm producenckich.

Product – tabela zawierająca nazwy produktów, nazwy producentów, parametry techniczne, typ produktu, cena detaliczna itp.

Producer – Informacje o producencie jak: nazwa producenta, rabaty na poszczególne typy produktów.

Client – tabela zawiera listę klientów, z danymi personalnymi oraz kontaktowymi.

Place_object – przechowuje adresy obiektów inwestycji.

Supervisor – w tabeli przechowywane są informacje o pracownikach firmy. Docelowo tabela posłuży do przypisania opiekuna inwestycji.

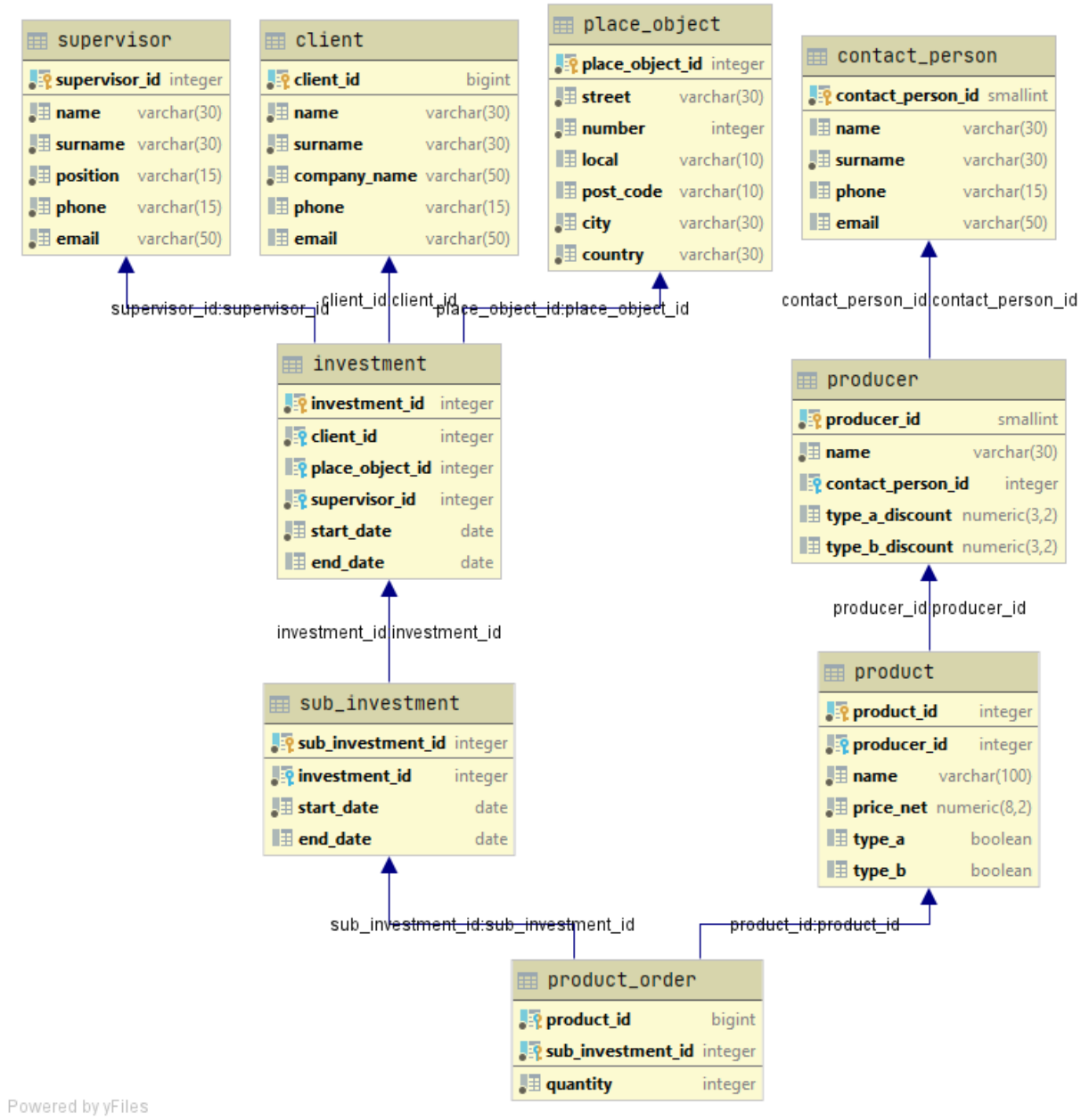
Investment – tabela zawierająca ID obiektu inwestycji, ID klienta oraz ID opiekuna inwestycji oraz czas rozpoczęcia i zakończenia inwestycji.

Sub_investment – tabela zawiera informacje o podinwestycjach. Założono, że jedna inwestycja może grupować pod sobą kilka podinwestycji.

Product_order – tabela przechowująca zamówienia produktów na daną podinwestycje.

2. Schemat ERD (Entity Relationship Diagram)

Schemat przedstawia relacje pomiędzy tabelami wraz z ich atrybutami.



W tabeli Product_order wykorzystano wklucz kompozytowy, składający się z Product_id oraz sub_investment_id. W ten sposób możemy jednoznacznie określić zamówienie – dany produkt może pojawić się tylko raz dla danej podinwestycji.

Daty rozpoczęcia oraz zakończenia inwestycji/podinwestycji zostały zabezpieczone warunkiem - end_date > start_date.

3. Tworzenie bazy danych

Dane są przechowywane w bazie danych „company” w schemacie „company”. Postanowiłem utworzyć nowy schemat grupujący wszystkie tabele oraz określiłem uprawnienia dla użytkowników dla schematu. Tym samym zrezygnowałem z wykorzystania domyślnego schematu „public”, co wydaje się dobrą praktyką, ze względów bezpieczeństwa – tworząc nowego role/user domyślnie jest mu przyznany dostęp do schematu „public”. Tworząc własny schemat, użytkownik tworzący schemat musi nadać nowemu użytkownikowi odpowiednie uprawnienia, aby mógł korzystać z jego zawartości.

```
company=> \dt company.*
            List of relations
 Schema  |      Name       | Type  |  Owner
---------+-----------------+-------+----------
 company | client          | table | postgres
 company | contact_person  | table | postgres
 company | investment      | table | postgres
 company | place_object    | table | postgres
 company | producer        | table | postgres
 company | product         | table | postgres
 company | product_order   | table | postgres
 company | sub_investment  | table | postgres
 company | supervisor      | table | postgres
(9 rows)
```

UTWORZONE TABELE W BAZIE DANYCH ZGODNIE ZE SCHEMATEM ERD.

Nie zdecydowałem się na stworzenie kilku schematów, dla każdej grupy użytkowników, ponieważ w założeniu, każda grupa użytkowników ma mieć dostęp do wszystkich tabel. Ograniczenia dostępu zrealizowałem ustawiając odpowiednie ograniczenia na operacje w schemacie.

```
company=> SELECT * FROM company.client;
 client_id |    name    |  surname   |         company_name          |    phone     |          email
-----------+------------+------------+-------------------------------+--------------+-------------------------
         1 | Beverie    | Nevet      | Graham, Hudson and Hettinger  | 123 557 7970 | bnevet0@github.io
         2 | Kelvin     | Dolder     | Hahn Group                    | 129 842 9118 | kdolder1@oracle.com
         3 | Kaylil     | Lotte      | Lueilwitz Inc                 | 819 330 4006 | klotte2@mapquest.com
         4 | Reeta      | Astman     | Casper-Harber                 | 322 893 4668 | rastman3@pen.io
         5 | Noby       | Theze      | Johnston-Spencer              | 976 549 1871 | ntheze4@ning.com
         6 | Maximilian | Herculeson | Turcotte-Daniel               | 411 442 5859 | mherculeson5@1und1.de
         7 | Lilly      | Billiard   | Goldner, Parker and Dibbert   | 991 112 7980 | lbilliard6@earthlink.net
         8 | Olly       | Bilovsky   | Hartmann LLC                  | 266 957 4203 | obilovsky7@homestead.com
         9 | Nicolea    | Luker      | Stark-Brown                   | 595 190 5940 | nluker8@goo.ne.jp
        10 | Grier      | Dory       | Altenwerth, Kozey and Johnson | 948 913 1268 | gdory9@ovh.net
(10 rows)
```

TABELA „CLIENT” WYPEŁNIONA DANYMI.

Constraint

Poniższe zdjęcia przedstawiają warunek CONSTARINT „product_type” z tabeli „product” oraz jego reakcje przy próbie dodania nowych rekordów do tabeli. Warunek wymaga, aby atrybut type_B był różny od atrybutu type_A (założono, że jednego produkt nie może zakwalifikować do dwóch typów produktu).

Co warto zauważyć, wspomniane atrybuty może przyjąć wartość NULL, co oznacza, że typ produktu nie musi zostać określony.

```sql
CREATE TABLE product (
    product_id SERIAL NOT NULL PRIMARY KEY,
    producer_id INTEGER NOT NULL,
    name VARCHAR(100) NOT NULL,
    price_net NUMERIC(8,2) NOT NULL,
    type_A BOOLEAN,
    type_B BOOLEAN,
    CONSTRAINT product_type CHECK(type_B != type_A),
    FOREIGN KEY (producer_id) REFERENCES producer(producer_id)
);
```

```
company=# \i C:/Users/yoszimitsu/Desktop/product.sql
INSERT 0 1
psql:C:/Users/yoszimitsu/Desktop/product.sql:2: BÅ?Ä‚D:  nowy rekord dla relacji "product" narusz
a ograniczenie sprawdzajÄ.ce "product_type"
DETAIL:  Niepoprawne ograniczenia wiersza (4, 2, Sambuca Cream, 296.60, f, f).
psql:C:/Users/yoszimitsu/Desktop/product.sql:3: BÅ?Ä‚D:  nowy rekord dla relacji "product" narusz
a ograniczenie sprawdzajÄ.ce "product_type"
DETAIL:  Niepoprawne ograniczenia wiersza (5, 3, Water - Perrier, 583.46, f, f).
psql:C:/Users/yoszimitsu/Desktop/product.sql:4: BÅ?Ä‚D:  nowy rekord dla relacji "product" narusz
a ograniczenie sprawdzajÄ.ce "product_type"
DETAIL:  Niepoprawne ograniczenia wiersza (6, 4, Longos - Greek Salad, 201.76, f, f).
INSERT 0 1
INSERT 0 1
psql:C:/Users/yoszimitsu/Desktop/product.sql:7: BÅ?Ä‚D:  nowy rekord dla relacji "product" narusz
a ograniczenie sprawdzajÄ.ce "product_type"
DETAIL:  Niepoprawne ograniczenia wiersza (9, 7, Lamb - Loin Chops, 744.25, t, t).
INSERT 0 1
INSERT 0 1
psql:C:/Users/yoszimitsu/Desktop/product.sql:10: BÅ?Ä‚D:  nowy rekord dla relacji "product" narus
za ograniczenie sprawdzajÄ.ce "product_type"
DETAIL:  Niepoprawne ograniczenia wiersza (12, 10, Wine - Two Oceans Sauvignon, 860.28, f, f).
company=#
```

REAKCJA CONSTRAINT „PRODUCT_TYPE” PRZY WSTAWIANIU DANYCH DO TABELI.

4. Użytkownicy

W bazie danych utworzono 3 grupy użytkowników:

- Management
- Employee
- Intern

Założono, że użytkownik Mangement ma uprawnienia do wszystkich operacji na tabakach oraz ma możliwość dodawania nowych użytkowników oraz tabel.
Employee może wykonywać operacje odczytu, zapisu, edycji oraz usuwania danych ze wszystkich tabel.

Intern ma możliwość jedynie odczytu oraz zapisu danych w tabelach.
Uprawnienia przypisano analogicznie do hierarchii w firmie, chroniąc bazę danych przed błędami użytkowników „Intern”.

Do każdej grupy przypisano poniżej wymienione uprawnienia:

```
CREATE ROLE management;
ALTER ROLE management LOGIN PASSWORD 'management';
ALTER ROLE management CREATEDB;
ALTER ROLE management CREATEROLE;

CREATE ROLE employee;
ALTER ROLE employee LOGIN PASSWORD 'employee';

CREATE ROLE intern;
ALTER ROLE intern LOGIN PASSWORD 'intern';

-- --ACCESS DB
REVOKE CONNECT ON DATABASE company FROM PUBLIC;
GRANT  CONNECT ON DATABASE company TO management, employee, intern;

-- --ACCESS SCHEMA
REVOKE ALL ON SCHEMA company FROM PUBLIC;
GRANT USAGE ON SCHEMA company TO management, employee, intern;

-- --ACCESS TABLES
REVOKE ALL ON ALL TABLES IN SCHEMA company FROM PUBLIC ;
GRANT SELECT, INSERT                ON ALL TABLES IN SCHEMA company TO intern;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA company TO employee;
GRANT ALL                           ON ALL TABLES IN SCHEMA company TO management;

-- --ACCESS SEQUENCES (FROM POSTGRES 8.2)
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA company TO management, employee, intern;
```

PRZYPISYWANE UPRAWNIENIA DLA UŻYTKOWNIKÓW BAZY DANYCH.

Poniższe zdjęcia przedstawiają dostęp do bazy danych użytkownika z grupy ‘Intern”:

```
company=> SELECT current_user;
 current_user
--------------
 intern
(1 row)


company=> DELETE FROM company.client WHERE client_id=1;
BÅ?Ä‚D:  permission denied for table client
company=>
```

```
company=> SELECT current_user;
 current_user
--------------
 intern
(1 row)


company=> UPDATE company.client SET name='ALEX' WHERE client_id=1;
BÅ?Ä‚D:  permission denied for table client
company=>
```

```
company=> SELECT current_user;
 current_user
--------------
 intern
(1 row)


company=> SELECT * FROM company.client;
 client_id |    name    |  surname   |         company_name          |    phone     |          email
-----------+------------+------------+-------------------------------+--------------+-------------------------
         1 | Beverie    | Nevet      | Graham, Hudson and Hettinger  | 123 557 7970 | bnevet0@github.io
         2 | Kelvin     | Dolder     | Hahn Group                    | 129 842 9118 | kdolder1@oracle.com
         3 | Kaylil     | Lotte      | Lueilwitz Inc                 | 819 330 4006 | klotte2@mapquest.com
         4 | Reeta      | Astman     | Casper-Harber                 | 322 893 4668 | rastman3@pen.io
         5 | Noby       | Theze      | Johnston-Spencer              | 976 549 1871 | ntheze4@ning.com
         6 | Maximilian | Herculeson | Turcotte-Daniel               | 411 442 5859 | mherculeson5@1und1.de
         7 | Lilly      | Billiard   | Goldner, Parker and Dibbert   | 991 112 7980 | lbilliard6@earthlink.net
         8 | Olly       | Bilovsky   | Hartmann LLC                  | 266 957 4203 | obilovsky7@homestead.com
         9 | Nicolea    | Luker      | Stark-Brown                   | 595 190 5940 | nluker8@goo.ne.jp
        10 | Grier      | Dory       | Altenwerth, Kozey and Johnson | 948 913 1268 | gdory9@ovh.net
(10 rows)
```

```
company=> SELECT current_user;
 current_user
--------------
 intern
(1 row)


company=> insert into company.client (name, surname, company_name, phone, email)
company-> values ('Beverie', 'Nevet', 'Graham, Hudson and Hettinger', '123 557 7970', 'bnevet0@github.io');
INSERT 0 1
company=>
```

KONTROLA UPRAWNIEŃ DLA UŻYTKOWNIKA INTERN

Jak widać użytkownik Intern nie ma uprawnieni do usuwania oraz edycji danych w tabelach.

5. Zapytania SQL

```sql
SELECT

    c.product_id,
    c.sub_investment_id,
    a.price_net,
    c.quantity,
    ROUND(a.price_net * c.quantity, 2) value_net
FROM company.product_order c
INNER JOIN company.product a
ON c.product_id = a.product_id
INNER JOIN company.producer b
ON a.producer_id = b.producer_id
ORDER BY c.product_id;
```

```
company=> SELECT
company->
company-> c.product_id,
company-> c.sub_investment_id,
company-> a.price_net,
company-> c.quantity,
company-> ROUND(a.price_net * c.quantity, 2) value_net
company-> FROM company.product_order c
company-> INNER JOIN company.product a
company-> ON c.product_id = a.product_id
company-> INNER JOIN company.producer b
company-> ON a.producer_id = b.producer_id
company-> ORDER BY c.product_id;
 product_id | sub_investment_id | price_net | quantity | value_net
------------+-------------------+-----------+----------+-----------
          1 |                 1 |    318.27 |        5 |   1591.35
          2 |                 2 |    296.60 |       79 |  23431.40
          3 |                 3 |    583.46 |       43 |  25088.78
          4 |                 4 |    201.76 |        7 |   1412.32
          5 |                 5 |    580.59 |        5 |   2902.95
          6 |                 6 |    595.75 |       71 |  42298.25
          7 |                 7 |    744.25 |      395 | 293978.75
          8 |                 8 |      3.81 |      504 |   1920.24
          9 |                 9 |    816.59 |        5 |   4082.95
         10 |                10 |    860.28 |       57 |  49035.96
(10 rows)
```

PRZYKŁADOWE ZAPYTANIE PRZEDSTAWIAJĄCE WARTOŚĆ ZAMÓWIENIA DANEGO PRODUKTU Z PODINWESTYCJI.

```sql
SELECT
    c.product_id,
    c.sub_investment_id,
    a.price_net,
    b.type_b_discount,
    ROUND(a.price_net * (1 - b.type_b_discount), 2) after_discount,
    c.quantity,
    ROUND(ROUND(a.price_net * (1 - b.type_b_discount), 2) * c.quantity, 2) value_net
FROM company.product_order c
INNER JOIN company.product a
ON c.product_id = a.product_id
INNER JOIN company.producer b
ON a.producer_id = b.producer_id
ORDER BY c.product_id;
```

```
company=>
company=> SELECT
company-> c.product_id,
company-> c.sub_investment_id,
company-> a.price_net,
company-> b.type_b_discount,
company-> ROUND(a.price_net * (1 - b.type_b_discount), 2) after_discount,
company-> c.quantity,
company-> ROUND(ROUND(a.price_net * (1 - b.type_b_discount), 2) * c.quantity, 2) value_net
company-> FROM company.product_order c
company-> INNER JOIN company.product a
company-> ON c.product_id = a.product_id
company-> INNER JOIN company.producer b
company-> ON a.producer_id = b.producer_id
company-> ORDER BY c.product_id;
 product_id | sub_investment_id | price_net | type_b_discount | after_discount | quantity | value_net
------------+-------------------+-----------+-----------------+----------------+----------+-----------
          1 |                 1 |    318.27 |            0.06 |         299.17 |        5 |   1495.85
          2 |                 2 |    296.60 |            0.29 |         210.59 |       79 |  16636.61
          3 |                 3 |    583.46 |            0.54 |         268.39 |       43 |  11540.77
          4 |                 4 |    201.76 |            0.87 |          26.23 |        7 |    183.61
          5 |                 5 |    580.59 |            0.57 |         249.65 |        5 |   1248.25
          6 |                 6 |    595.75 |            0.98 |          11.92 |       71 |    846.32
          7 |                 7 |    744.25 |            0.67 |         245.60 |      395 |  97012.00
          8 |                 8 |      3.81 |            0.13 |           3.31 |      504 |   1668.24
          9 |                 9 |    816.59 |            0.08 |         751.26 |        5 |   3756.30
         10 |                10 |    860.28 |            0.34 |         567.78 |       57 |  32363.46
(10 rows)


company=>
```

PRZYKŁADOWE ZAPYTANIE PRZEDSTAWIAJĄCE WARTOŚĆ ZAMÓWIENIA DANEGO PRODUKTU Z PODINWESTYCJI UWZGLĘDNIAJĄC ZNIŻKI NA DANY PRODUKT.

6. Napotkane problemy

- Client encoding:

```
postgres=# DROP TABLE investment;
ERROR:  character with byte sequence 0xc5 0x81 in encoding "UTF8" has no equivalent in encoding "WIN1252"
postgres=# SET CLIENT_ENCODING TO 'UTF8';
SET
postgres=# DROP TABLE investment;
BÅ?Ä‚D:  nie moÅ¼na usunÄ.Ä╣ tabela investment poniewaÅ¼ inne obiekty zaleÅ¼Ä. od niego
DETAIL:  ograniczenie sub_investment_investment_id_fkey na tabela sub_investment zaleÅ¼y od tabela investment
HINT:  UÅ¼yj DROP ... CASCADE aby usunÄ.Ä╣ wraz z obiektami zaleÅ¼nymi.
postgres=# DROP TABLE investment CASCADE;
UWAGA:  kasowanie kaskadowe do ograniczenie sub_investment_investment_id_fkey na tabela sub_investment
DROP TABLE
postgres=#
```

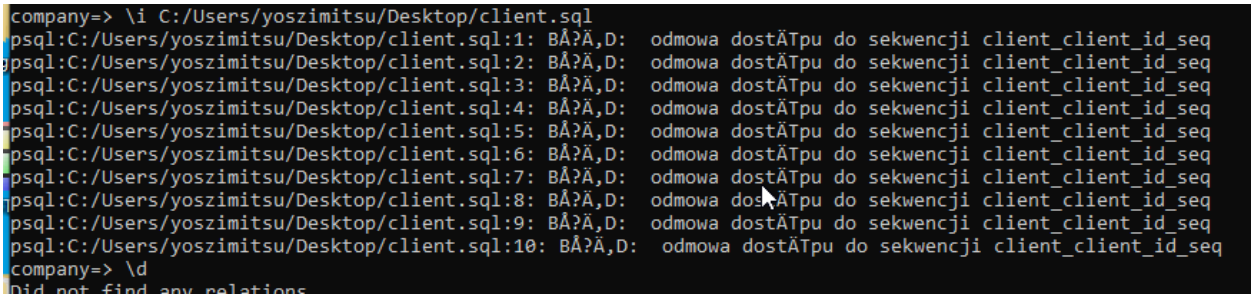
Solution: Konieczna była zmiana typu kodowania klienta z WIN1252 na UTF-8 do poprawne egzekucji komend w Postgres CLI (psql).

- Numerowanie produktów za pomocą SERIAL. Numery ID nie są przypisywane po kolei. Luki powstają, gdy np. wstawiać nowy rekord, dane nie przejdą walidacji. Mimo, że nie dodano nowego produktu, został zajęty dany numer klucza głównego.

```
company=# SELECT * FROM product;
 product_id | producer_id |            name             | price_net | type_a | type_b
------------+-------------+-----------------------------+-----------+--------+--------
          3 |           1 | Pasta - Shells, Medium, Dry |    318.27 | f      | t
          7 |           5 | Table Cloth 54x72 Colour    |    580.59 | f      | t
          8 |           6 | Chinese Foods - Chicken     |    595.75 | t      | f
         10 |           8 | Sage Derby                  |      3.81 | f      | t
         11 |           9 | Cape Capensis - Fillet      |    816.59 | t      | f
         13 |           1 | Pasta - Shells, Medium, Dry |    318.27 | f      | t
         14 |           2 | Sambuca Cream               |    296.60 | f      |
         15 |           3 | Water - Perrier             |    583.46 | f      | t
         16 |           4 | Longos - Greek Salad        |    201.76 | t      | f
         17 |           5 | Table Cloth 54x72 Colour    |    580.59 | f      | t
         18 |           6 | Chinese Foods - Chicken     |    595.75 | t      | f
         19 |           7 | Lamb - Loin Chops           |    744.25 | t      |
         20 |           8 | Sage Derby                  |      3.81 | f      | t
         21 |           9 | Cape Capensis - Fillet      |    816.59 | t      | f
         22 |          10 | Wine - Two Oceans Sauvignon |    860.28 |        |
(15 rows)
```

Solution: Nie rozwiązano

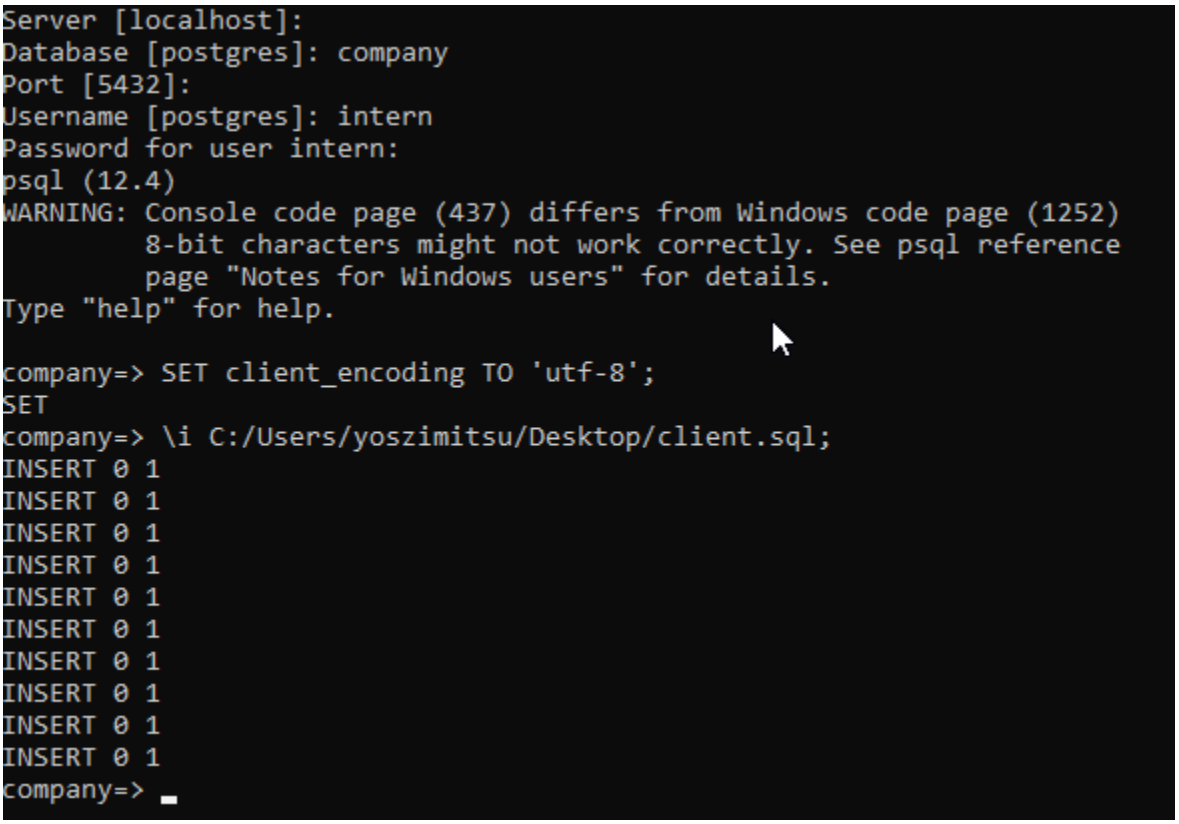
- Odmowa dostępu przy próbie wstawienia danych



Solution:

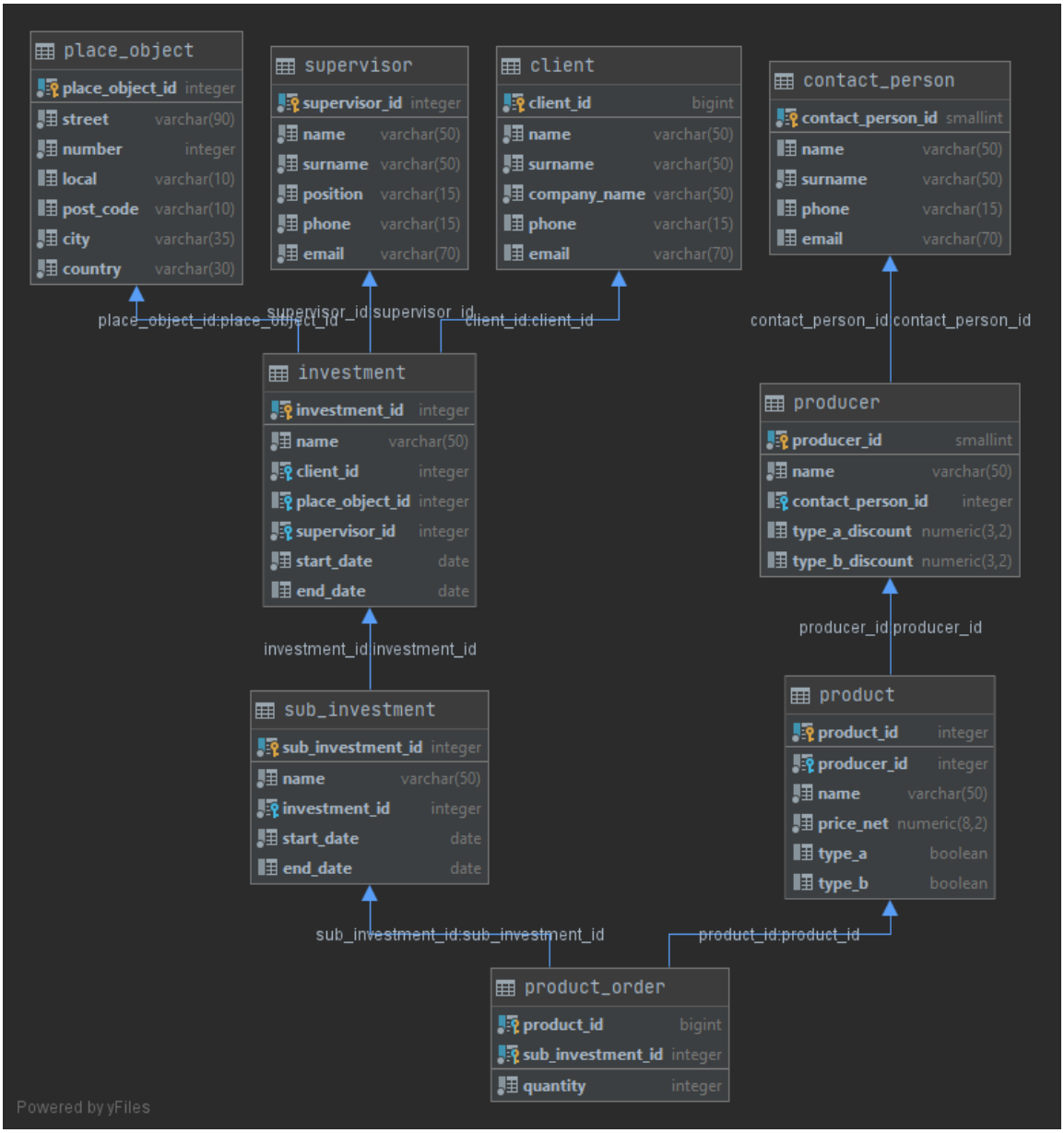
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA company TO management, employee, intern;

Nadanie uprawnień do użycia dla wszystkich sekwencji w schemacie. (wymagane od PostgresSQL 8.2)



Poprawki Etap1

Schemat ERD



Po etapie pierwszym dodano atrybut name do tabeli investment i sub_investment. Dodatkowo zmieniono typy (wielkość VARCHAR) niektórych zmiennych jak: street, city. Zmiana poprawiła bazę pod względem walidacji danych, lecz nie wpłynęła na strukturę bazy danych.

Dodatkowo tabele uzupełniono danymi dopasowanymi do update'owanych tabel.

Wprowadzono dodatkowe ograniczenie na tabele producer

```
ALTER TABLE company.producer ADD CONSTRAINT unique_constraints UNIQUE (name);
```

Etap2

1. Perspektywy

Utworzono perspektywę investment_info, która przedstawia informacje o inwestycji w praktyczny sposób. Widok zawiera informacje z tabeli investment, opiekuna projektu (supervisor) oraz adres obiektu inwestycji, wywołany przez specjalnie utworzoną funkcję.

Przed stworzeniem widoku została dodana funkcja zwracająca adres obiektu z poszczególnych kolumn tabeli place_object.

```
CREATE OR REPLACE FUNCTION company.address() RETURNS VARCHAR
LANGUAGE SQL
AS $$
    SELECT
    CONCAT('st. ', p.street, ' ',number, ' ,', post_code, ' ,', city, ' ,', country)
    FROM company.place_object AS p;
$$;
```

PROCEDURA COMAPNY.ADDRESS() ZWRACAJĄCA PEŁEN ADRES OBIEKTU.

Kod perspektywy z użyciem funkcji address() przedstawiono poniżej:

```
CREATE OR REPLACE VIEW company.investment_info as
SELECT
    c.investment_id,
    c.name,
    k.company_name client,
    company.address_procedure() address,
    c.start_date,
    c.end_date,
    CONCAT(s.name, ' ', s.surname) supervisor
FROM company.investment c
INNER JOIN company.place_object p
ON c.place_object_id = p.place_object_id
INNER JOIN company.supervisor s
ON c.supervisor_id = s.supervisor_id
INNER JOIN company.client k
ON c.client_id = k.client_id
ORDER BY c.investment_id;
```

PERSPEKTYWA COMPANY.INVESTMENT_INFO.

```
company=# SELECT * FROM company.investment_info;
 investment_id |   name   |            client             |                       address                       | start_date |  end_date  |    supervisor
---------------+----------+-------------------------------+-----------------------------------------------------+------------+------------+-------------------
             1 | invest1  | Graham, Hudson and Hettinger  | st. Thierer 1 ,411 45 ,Muka┼Öov ,Czech Republic       | 2018-11-28 | 2020-10-11 | Ogdon Kauschke
             2 | invest2  | Hahn Group                    | st. Bowman 2 , ,Tarrafal ,Cape Verde                | 2016-12-10 | 2020-10-12 | Chauncey Drewe
             3 | invest3  | Lueilwitz Inc                 | st. Rieder 3 ,58884-000 ,Catol├- do Rocha ,Brazil    | 2016-04-18 | 2020-10-10 | Jewel Dik
             4 | invest4  | Casper-Harber                 | st. Gerald 4 ,93285 ,Saint-Denis ,France            | 2019-11-28 | 2020-10-12 | Cori Follacaro
             5 | invest5  | Johnston-Spencer              | st. Morning 5 , ,Andros Town ,Bahamas               | 2016-07-26 | 2020-10-11 | Walliw Regler
             6 | invest6  | Turcotte-Daniel               | st. Orin 6 ,57351 ,Kam─ülia ,Pakistan                | 2017-08-10 | 2020-10-12 | Stirling Williams
             7 | invest7  | Goldner, Parker and Dibbert   | st. Stone Corner 7 ,50770 ,─Çrifw─üla ,Pakistan        | 2017-03-05 | 2020-10-13 | Danella Valti
             8 | invest8  | Hartmann LLC                  | st. Scott 8 ,97701 ,Ranua ,Finland                  | 2019-06-25 | 2020-10-12 | Katina Lubbock
             9 | invest9  | Stark-Brown                   | st. Erie 9 , ,D─ürayy─ü ,Syria                         | 2018-01-08 | 2020-10-10 | Ladonna Matyasik
            10 | invest10 | Altenwerth, Kozey and Johnson | st. Tomscot 10 , ,San Antonio del Monte ,El Salvador | 2017-03-14 | 2020-10-13 | Iormina Lodeke
(10 rows)
```

WYWOŁANIE PERSPEKTYWY COMPANY.INVESTMENT_INFO.

Drugą perspektywą o nazwie product_order_value przedstawia zamówienia z danymi zawierającymi ceny produktów, ilość oraz wartość końcową.

```sql
CREATE VIEW company.product_orders_value as
SELECT
    c.sub_investment_id,
    s.name sub_investment_name,
    p.name product_name,
    a.price_net,
    ( ROUND(a.price_net *
     (1 - CASE
      WHEN (COALESCE(a.type_a, false)) IS TRUE THEN
                b.type_a_discount
      WHEN  (COALESCE(a.type_b, false)) IS TRUE THEN
                b.type_b_discount
      ELSE
                0
      END
     ),2)
     ) after_discount,
    c.quantity,
    ROUND(
    ROUND(a.price_net *
     (1 - CASE
      WHEN (COALESCE(a.type_a, false)) IS TRUE THEN
                b.type_a_discount
      WHEN  (COALESCE(a.type_b, false)) IS TRUE THEN
                b.type_b_discount
      ELSE
                0
      END
     ),2) * c.quantity, 2) value_net
FROM company.product_order c
INNER JOIN company.product a
ON c.product_id = a.product_id
INNER JOIN company.sub_investment s
ON c.sub_investment_id = s.sub_investment_id
INNER JOIN company.product p
ON a.product_id = p.product_id
INNER JOIN company.producer b
ON a.producer_id = b.producer_id
ORDER BY c.product_id;
```

PERSPEKTYWA COMPANY.PRODUCT_ORDER_VALUE.

Podjąłem próbę wydzielenia powielanej części kodu do oddzielnej funkcji, lecz niestety nie udało mi się utworzyć tej funkcji w prawidłowy sposób. Więcej informacji w sekcji „Napotkane problemy”.

```
company=# SELECT * FROM company.product_orders_value;
 sub_investment_id | sub_investment_name |         product_name          | price_net | after_discount | quantity | value_net
-------------------+---------------------+-------------------------------+-----------+----------------+----------+-----------
                 2 | sub_invest2         | Sambuca Cream                 |    296.60 |         296.60 |       79 |  23431.40
                 3 | sub_invest3         | Water - Perrier               |    583.46 |         268.39 |       43 |  11540.77
                 4 | sub_invest4         | Longos - Greek Salad          |    201.76 |         135.18 |        7 |    946.26
                 5 | sub_invest5         | Table Cloth 54x72 Colour      |    580.59 |         249.65 |        5 |   1248.25
                 6 | sub_invest6         | Chinese Foods - Chicken       |    595.75 |         142.98 |       71 |  10151.58
                 7 | sub_invest7         | Lamb - Loin Chops             |    744.25 |         334.91 |      395 | 132289.45
                 8 | sub_invest8         | Sage Derby                    |      3.81 |           3.31 |      504 |   1668.24
                 9 | sub_invest9         | Cape Capensis - Fillet        |    816.59 |         293.97 |        5 |   1469.85
                10 | sub_invest10        | Wine - Two Oceans Sauvignon   |    860.28 |         860.28 |       57 |  49035.96
(9 rows)
```

WYWOŁANIE PERSPEKTYWY COMPANY.PRODUCT_ORDER_VALUE.

2. Indeksy

Indeksy przyśpieszają przetwarzanie zapytań SQL i powinny być nakładane na kolumny, w których wartości często się powtarzają lub są często wczytywane. Klucze główne tabel są automatycznie indeksowane.

Postanowiono stworzyć 3 indeksy:

- Dla wartości name i surname tabeli supervisor
- Dla wartości name tabeli investment
- Dla wartości name tabeli sub_investment

Wartości name z tabel investmetn i sub_investment są wykorzystywane w widokach i są ważnymi kolumnami z punktu widzenia logiki biznesowej.

Indeks supervisor_idx został dodatkowo oznaczony jako UNIQUE co oznacza, że wartości „imię” i „nazwisko” nie mogą być duplikowane. Indeks UNIQUE znacząco przyspiesza przetwarzanie wartości.

```
company=# CREATE UNIQUE INDEX supervisor_idx ON supervisor (name, surname);
BÅ?Ä‚D:  relacja "supervisor" nie istnieje
company=# CREATE UNIQUE INDEX supervisor_idx ON company.supervisor (name, surname);
CREATE INDEX
company=# CREATE  INDEX invest_name_idx ON company.investment (name);
CREATE INDEX
company=# CREATE  INDEX sub_invest_name_idx ON company.sub_investment (name);
CREATE INDEX
```

```
 company    | supervisor          | supervisor_idx                 |          | CREATE UNIQUE INDEX supervisor_idx ON company.supervisor USING btree (name, surname)
 company    | investment          | invest_name_idx                |          | CREATE INDEX invest_name_idx ON company.investment USING btree (name)
 company    | sub_investment      | sub_invest_name_idx            |          | CREATE INDEX sub_invest_name_idx ON company.sub_investment USING btree (name)
(129 rows)
```

INDEKSY UTWORZONE W BAZIE DANYCH COMPANY.

3. Procedury, funkcje, wyzwalacze

Wyjaśnienie różnic:

Funkcja musi zwracać wartość a w procedurze składowanej jest opcjonalna (procedura może zwrócić zero lub n wartości). Funkcje mogą mieć tylko parametry wejściowe a procedury mogą mieć parametry wejściowe/wyjściowe. Funkcje mogą być wywoływane z procedury, ale procedura nie może być wywołana z funkcji.

Jedną ze stworzonych funkcji jest funkcja address(), która zwraca adres w postach łańcucha znaków z poszczególnych kolumn tabeli place_object.

```
company=# CREATE OR REPLACE FUNCTION company.address(index INT) RETURNS VARCHAR
company-# LANGUAGE SQL
company-# AS $$
company$#     SELECT
company$# CONCAT('st. ', p.street, ' ',number, ' ,', post_code, ' ,', city, ' ,', country)
company$# FROM company.place_object AS p
company$# WHERE p.place_object_id = index;
company$# $$;
CREATE FUNCTION
company=# SELECT company.address(2);
              address
-------------------------------------
 st. Bowman 2 , ,Tarrafal ,Cape Verde
(1 row)


company=# SELECT company.address(4);
                address
----------------------------------------
 st. Gerald 4 ,93285 ,Saint-Denis ,France
(1 row)
```

FUNKCJA COMAPNY.ADDRESS(ID) ZWRACAJĄCA ADDRESS OBIEKTU Z REKORDU ID.

Utworzony procedurę new_employee, która pozwala na wprowadzenie nowych pracowników do tabeli supervisor.

```
company=# CALL company.new_employee('daniel', 'michrowski', 'test', 'test', 'test@test.pl');
CALL
company=# SELECT * FROM company.supervisor;
 supervisor_id |   name   |   surname   | position |    phone     |            email
---------------+----------+-------------+----------+--------------+-----------------------------
             1 | Ogdon    | Kauschke    | engineer | 228 911 2936 | okauschke0@wired.com
             2 | Chauncey | Drewe       | engineer | 474 860 1874 | cdrewe1@independent.co.uk
             3 | Jewel    | Dik         | engineer | 974 941 9143 | jdik2@google.cn
             4 | Cori     | Follacaro   | engineer | 752 997 3851 | cfollacaro3@a8.net
             5 | Walliw   | Regler      | engineer | 789 582 4255 | wregler4@nydailynews.com
             6 | Stirling | Williams    | engineer | 925 222 8698 | swilliams5@mashable.com
             7 | Danella  | Valti       | engineer | 317 883 0721 | dvalti6@ucsd.edu
             8 | Katina   | Lubbock     | engineer | 857 635 9630 | klubbock7@lulu.com
             9 | Ladonna  | Matyasik    | engineer | 146 450 5182 | lmatyasik8@chronoengine.com
            10 | Iormina  | Lodeke      | engineer | 365 179 3664 | ilodeke9@google.co.jp
            12 | daniel2  | michrowski2 | test2    | test2        | test2@test.pl
            13 | daniel   | michrowski  | test     | test         | test@test.pl
(12 rows)
```

PROCEDURA COMPANY.NEW_EMPLOYEE POZWALA NA ZAPIS NOWYCH PRACOWNIKÓW TO TABELI SUPERVISOR.

Kolejną procedurą jest new_product and_producer pozwalająca na wprowadzenie produktu od nowego producenta. W procedurze zastosowano obsługę błędów. Poniższe zdjęcia przedstawiają kod procedury:

```
CREATE OR REPLACE PROCEDURE company.new_product_and_producer(
product_name VARCHAR(50), price_net NUMERIC(8,2), type_a BOOLEAN, type_b BOOLEAN,
producer_name VARCHAR(50), type_a_discount NUMERIC(3,2), type_b_discount NUMERIC(3,2))
AS $$
declare
    v_state   TEXT;
    v_msg     TEXT;
    v_detail  TEXT;
    v_hint    TEXT;
    v_context TEXT;
    begin

        INSERT INTO company.producer(
            name,
            type_a_discount,
            type_b_discount)
        VALUES(
            producer_name,
            type_a_discount,
            type_b_discount);

        INSERT INTO company.product(
            producer_id,
            name,
            price_net,
            type_a,
            type_b)
        VALUES(
            (SELECT producer_id FROM company.producer WHERE name = producer_name),
            product_name,
            price_net,
            type_a,
            type_b);
```

```
    exception when others then

        get stacked diagnostics
            v_state   = returned_sqlstate,
            v_msg     = message_text,
            v_detail  = pg_exception_detail,
            v_hint    = pg_exception_hint,
            v_context = pg_exception_context;

        raise notice E'Got exception:
            state  : %
            message: %
            detail : %
            hint   : %
            context: %', v_state, v_msg, v_detail, v_hint, v_context;

        raise notice E'Got exception:
            SQLSTATE: %
            SQLERRM: %', SQLSTATE, SQLERRM;

        raise notice '%', message_text;

    end;
$$ LANGUAGE plpgsql;
```

KOD PROCEDUR Z OBSŁUGĄ BŁĘDÓW.

Procedura zwraca szczegółowe informacje o błędzie. Komenda “exception when others then” obsługuje występujące błędy oraz zarządza transakcjami. Jeśli w czasie przetwarzania procedury wystąpi błąd, transakcje zostaną cofnięte (rollback). Jeśli procedura przebiegnie pomyślnie transakcje są zatwierdzane (commit).

Poniżej przedstawiono wywołania procedury new_product_and_producer:

```
company=# CALL company.new_product_and_producer('soap', 3.45, true, false, 'Dove', 0.25, 0.0);
CALL
company=# SELECT * FROM company.product;
 product_id | producer_id |            name             | price_net | type_a | type_b
------------+-------------+-----------------------------+-----------+--------+--------
          1 |           1 | Pasta - Shells, Medium, Dry |    318.27 | f      | t
          2 |           2 | Sambuca Cream               |    296.60 | f      |
          3 |           3 | Water - Perrier             |    583.46 | f      | t
          4 |           4 | Longos - Greek Salad        |    201.76 | t      | f
          5 |           5 | Table Cloth 54x72 Colour    |    580.59 | f      | t
          6 |           6 | Chinese Foods - Chicken     |    595.75 | t      | f
          7 |           7 | Lamb - Loin Chops           |    744.25 | t      |
          8 |           8 | Sage Derby                  |      3.81 | f      | t
          9 |           9 | Cape Capensis - Fillet      |    816.59 | t      | f
         10 |          10 | Wine - Two Oceans Sauvignon |    860.28 |        |
         13 |          15 | soap                        |      3.45 | t      | f
(11 rows)


company=# SELECT * FROM company.producer;
 producer_id |     name     | contact_person_id | type_a_discount | type_b_discount
-------------+--------------+-------------------+-----------------+-----------------
           1 | Agimba       |                 1 |            0.45 |            0.06
           2 | Brightbean   |                 2 |            0.18 |            0.29
           3 | Kazu         |                 3 |            0.07 |            0.54
           4 | LiveZ        |                 4 |            0.33 |            0.87
           5 | Roomm        |                 5 |            0.35 |            0.57
           6 | Skipstorm    |                 6 |            0.76 |            0.98
           7 | Thoughtworks |                 7 |            0.55 |            0.67
           8 | Dabfeed      |                 8 |            0.42 |            0.13
           9 | Tazz         |                 9 |            0.64 |            0.08
          10 | Divanoodle   |                10 |            0.93 |            0.34
          15 | Dove         |                   |            0.25 |            0.00
(11 rows)
```

WYWOŁANIE PROCEDURY Z POZYTYWNYM REZULTATEM.

Poniżej przedstawiono obsługę błędów. W tym przypadku błąd dotyczył próby wstawienia nazwy producenta, która istniała już w tabeli.

```
company=# CALL company.new_product_and_producer('soap', 3.45, true, false, 'Dove', 0.25, 0.0);
UWAGA:  Got exception:
           state  : 23505
           message: podwA3jna wartoÅ>Ä╡ klucza narusza ograniczenie unikalnoÅ>ci "unique_constraints"
           detail : Klucz (name)=(Dove) juÅ% istnieje.
           hint   :
           context: wyraÅ%enie SQL "INSERT INTO company.producer(
               name,
               type_a_discount,
               type_b_discount)
           VALUES(
               producer_name,
               type_a_discount,
               type_b_discount)"
funkcja PL/pgSQL company.new_product_and_producer(character varying,numeric,boolean,boolean,character varying,
UWAGA:  Got exception:
           SQLSTATE: 23505
           SQLERRM: podwA3jna wartoÅ>Ä╡ klucza narusza ograniczenie unikalnoÅ>ci "unique_constraints"
BÅ?Ä,D:  kolumna "message_text" nie istnieje
LINE 1: SELECT message_text
               ^
QUERY:  SELECT message_text
CONTEXT:  funkcja PL/pgSQL company.new_product_and_producer(character varying,numeric,boolean,boolean,characte
company=#
```

ZWRÓCENIE BŁĘDU PRZY WYWOŁANIU PROCEDURY NEW_PROCUDT_AND_PRODUCER.

Poniżej przedstawiono obsługę błędu związaną z wprowadzeniem nowego produktu, który przypisano jednocześnie do grupy a oraz b (ograniczenie product_type sprawdza czy produkt nie jest przypisany do obu grup). W przykładzie można zauważyć, że pomimo że, błąd wystąpił przy wstawianiu danych do drugiej tabeli (w procedurze dane są wstawiane do dwóch tabel), dane które zostały pomyśle wstawione o pierwszej tabeli zostały wycofane. Wywołanie SELECT * FROM company.producer potwierdza, że producent „Nivea' nie został dodany.

```
company=# CALL company.new_product_and_producer('soap', 3.45, true, true, 'Nivea', 0.25, 0.0);
UWAGA:  Got exception:
           state  : 23514
           message: nowy rekord dla relacji "product" narusza ograniczenie sprawdzajÄ.ce "product_type"
           detail : Niepoprawne ograniczenia wiersza (14, 18, soap, 3.45, t, t).
           hint   :
           context: wyraÅ%enie SQL "INSERT INTO company.product(
               producer_id,
               name,
               price_net,
               type_a,
               type_b)
           VALUES(
               (SELECT producer_id FROM company.producer WHERE name = producer_name),
               product_name,
               price_net,
               type_a,
               type_b)"
funkcja PL/pgSQL company.new_product_and_producer(character varying,numeric,boolean,boolean,character varying,numeric,numeric), wiersz 19 w wyraÅ%enie SQL
UWAGA:  Got exception:
           SQLSTATE: 23514
           SQLERRM: nowy rekord dla relacji "product" narusza ograniczenie sprawdzajÄ.ce "product_type"
BÅ?Ä,D:  kolumna "message_text" nie istnieje
LINE 1: SELECT message_text
               ^
QUERY:  SELECT message_text
CONTEXT:  funkcja PL/pgSQL company.new_product_and_producer(character varying,numeric,boolean,boolean,character varying,numeric,numeric), wiersz 52 w RAISE
company=# SELECT * FROM company.producer;
 producer_id |     name      | contact_person_id | type_a_discount | type_b_discount
-------------+---------------+-------------------+-----------------+-----------------
           1 | Agimba        |                 1 |            0.45 |            0.06
           2 | Brightbean    |                 2 |            0.18 |            0.29
           3 | Kazu          |                 3 |            0.07 |            0.54
           4 | LiveZ         |                 4 |            0.33 |            0.87
           5 | Roomm         |                 5 |            0.35 |            0.57
           6 | Skipstorm     |                 6 |            0.76 |            0.98
           7 | Thoughtworks  |                 7 |            0.55 |            0.67
           8 | Dabfeed       |                 8 |            0.42 |            0.13
           9 | Tazz          |                 9 |            0.64 |            0.08
          10 | Divanoodle    |                10 |            0.93 |            0.34
          15 | Dove          |                   |            0.25 |            0.00
(11 rows)


company=#
```

TRANSKACJA Z TABELI COMPANY.PRODUCER ZOSTAŁA COFNIĘTA PO POJAWIENIU SIĘ BŁĘDU.

4. Wyzwalacz

Wyzwalacze pozwalają na zautomatyzowanie pewnych poleceń w przypadku wystąpienia określonego działania w bazie danych.

Poniżej przedstawiono trigger, który usuwa zależności sub_investment przed próbą usunięcia rekordu z investment. Zapobiega to przed pojawieniem się błędu, o powiązanych elementach w innych tabelach. Funkcja, która jest wyzwalana przez trigger, wykonuje polecenie równoznaczne z DELETE … CASCADE.

```
CREATE OR REPLACE FUNCTION company.clear_sub_investement() RETURNS TRIGGER AS $$
   BEGIN
      DELETE FROM company.product_order WHERE sub_investment_id = (
        SELECT sub_investment_id FROM company.sub_investment WHERE investment_id = OLD.investment_id);
      DELETE FROM company.sub_investment WHERE investment_id = OLD.investment_id;
      RETURN OLD;
   END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER clear_sub_investement BEFORE DELETE ON company.investment
FOR EACH ROW EXECUTE PROCEDURE company.clear_sub_investement();
```

WYZWALACZ USUWAJĄCY ODNIESIENIA PRZY USUWANIU REKORDU Z TABELI COMPANY.INVESTMENT.

```
company=# SELECT * FROM company.investment;
 investment_id |   name    | client_id | place_object_id | supervisor_id | start_date |  end_date
---------------+-----------+-----------+-----------------+---------------+------------+------------
             1 | invest1   |         1 |               1 |             1 | 2018-11-28 | 2020-10-11
             2 | invest2   |         2 |               2 |             2 | 2016-12-10 | 2020-10-12
             3 | invest3   |         3 |               3 |             3 | 2016-04-18 | 2020-10-10
             4 | invest4   |         4 |               4 |             4 | 2019-11-28 | 2020-10-12
             5 | invest5   |         5 |               5 |             5 | 2016-07-26 | 2020-10-11
             6 | invest6   |         6 |               6 |             6 | 2017-08-10 | 2020-10-12
             7 | invest7   |         7 |               7 |             7 | 2017-03-05 | 2020-10-13
             8 | invest8   |         8 |               8 |             8 | 2019-06-25 | 2020-10-12
             9 | invest9   |         9 |               9 |             9 | 2018-01-08 | 2020-10-10
            10 | invest10  |        10 |              10 |            10 | 2017-03-14 | 2020-10-13
(10 rows)


company=# SELECT * FROM company.sub_investment;
 sub_investment_id |     name     | investment_id | start_date |  end_date
-------------------+--------------+---------------+------------+------------
                 1 | sub_invest1  |             1 | 2017-02-16 | 2019-09-05
                 2 | sub_invest2  |             2 | 2017-12-20 | 2019-09-27
                 3 | sub_invest3  |             3 | 2017-10-18 | 2019-11-09
                 4 | sub_invest4  |             4 | 2017-04-05 | 2020-03-17
                 5 | sub_invest5  |             5 | 2017-11-14 | 2020-10-13
                 6 | sub_invest6  |             6 | 2017-10-14 | 2019-01-07
                 7 | sub_invest7  |             7 | 2017-12-26 | 2019-11-17
                 8 | sub_invest8  |             8 | 2017-12-25 | 2019-12-18
                 9 | sub_invest9  |             9 | 2017-02-27 | 2019-12-10
                10 | sub_invest10 |            10 | 2017-02-05 | 2019-11-10
(10 rows)


company=# SELECT * FROM company.product_order;
 product_id | sub_investment_id | quantity
------------+-------------------+----------
          1 |                 1 |        5
          2 |                 2 |       79
          3 |                 3 |       43
          4 |                 4 |        7
          5 |                 5 |        5
          6 |                 6 |       71
          7 |                 7 |      395
          8 |                 8 |      504
          9 |                 9 |        5
         10 |                10 |       57
(10 rows)
```

STAN TABEL PRZED USUNIĘCIEM REKORDU Z INVESTMENT.

```
company=# DELETE FROM company.investment WHERE investment_id = 1;
DELETE 1
company=# SELECT * FROM company.investment;
 investment_id |   name    | client_id | place_object_id | supervisor_id | start_date |  end_date
---------------+-----------+-----------+-----------------+---------------+------------+------------
             2 | invest2   |         2 |               2 |             2 | 2016-12-10 | 2020-10-12
             3 | invest3   |         3 |               3 |             3 | 2016-04-18 | 2020-10-10
             4 | invest4   |         4 |               4 |             4 | 2019-11-28 | 2020-10-12
             5 | invest5   |         5 |               5 |             5 | 2016-07-26 | 2020-10-11
             6 | invest6   |         6 |               6 |             6 | 2017-08-10 | 2020-10-12
             7 | invest7   |         7 |               7 |             7 | 2017-03-05 | 2020-10-13
             8 | invest8   |         8 |               8 |             8 | 2019-06-25 | 2020-10-12
             9 | invest9   |         9 |               9 |             9 | 2018-01-08 | 2020-10-10
            10 | invest10  |        10 |              10 |            10 | 2017-03-14 | 2020-10-13
(9 rows)


company=# SELECT * FROM company.sub_investment;
 sub_investment_id |     name     | investment_id | start_date |  end_date
-------------------+--------------+---------------+------------+------------
                 2 | sub_invest2  |             2 | 2017-12-20 | 2019-09-27
                 3 | sub_invest3  |             3 | 2017-10-18 | 2019-11-09
                 4 | sub_invest4  |             4 | 2017-04-05 | 2020-03-17
                 5 | sub_invest5  |             5 | 2017-11-14 | 2020-10-13
                 6 | sub_invest6  |             6 | 2017-10-14 | 2019-01-07
                 7 | sub_invest7  |             7 | 2017-12-26 | 2019-11-17
                 8 | sub_invest8  |             8 | 2017-12-25 | 2019-12-18
                 9 | sub_invest9  |             9 | 2017-02-27 | 2019-12-10
                10 | sub_invest10 |            10 | 2017-02-05 | 2019-11-10
(9 rows)


company=# SELECT * FROM company.product_order;
 product_id | sub_investment_id | quantity
------------+-------------------+----------
          2 |                 2 |       79
          3 |                 3 |       43
          4 |                 4 |        7
          5 |                 5 |        5
          6 |                 6 |       71
          7 |                 7 |      395
          8 |                 8 |      504
          9 |                 9 |        5
         10 |                10 |       57
(9 rows)
```

ZADZIAŁANIE TRIGGERA. SUB_INVESTMENT, PODPIĘTE POD USUNIĘTY REKORD Z INVESTMENT, ZOSTAŁY USUNIĘTE Z TABELI SUB_INVESTMENT ORAZ PRODUCT_ORDER.

Trigger wykorzystano również do zapisu logów odnośnie do zmiany cen produktów. Utworzono nową tabelę do zapisu logów.

```
CREATE TABLE company.log_price_update(
    log_id BIGSERIAL NOT NULL,
    log_date TIMESTAMP NOT NULL,
    product_id INTEGER NOT NULL,
    product_name VARCHAR(50) NOT NULL,
    old_price NUMERIC(8,2) NOT NULL,
    new_price NUMERIC(8,2) NOT NULL,
    product_producer VARCHAR(50) NOT NULL);
```

NOWA TABELA LOG_PRICE_UPDATE DO ZAPISU ZMIAN CEN PRODUKTÓW.

Tabele będzie zawierać date i czas zmiany ceny produktu (log_date). W takim przypadku warto zmienić strefę czasową na GMT. Jest to dobra praktyka dotycząca zapisu dat i godzin patrząc pod względem aplikacji działających na poziomie globalnym.

```
company=# SET TIMEZONE='GMT';
SET
company=# SHOW TIMEZONE;
 TimeZone
----------
 GMT
(1 row)


company=#
```

ZAMIANA STREFY CZASOWEJ NA GMT.

```
CREATE OR REPLACE FUNCTION company.log_price_update() RETURNS TRIGGER AS $$
   BEGIN
     INSERT INTO company.log_price_update (log_date, product_id, product_name, old_price, new_price, product_producer)
     VALUES (NOW(), OLD.product_id, OLD.name, OLD.price_net, NEW.price_net, OLD.producer_id);
      RETURN OLD;
      RETURN NEW;
   END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER log_price_update_trigger AFTER UPDATE ON company.product
FOR EACH ROW EXECUTE PROCEDURE company.log_price_update();
```

FUNKCJA COMPANY.LOG_PRICE_UPDATE WYWOLYWANA PRZY ZMIANIE CEN PRODUKTÓW.

Poniżej przedstawiono rezultat utworzonego triggera. W tabeli log_price_update zapisaliśmy informacje o zmianie ceny produktu.

```
company=# UPDATE company.product SET price_net = 33.45 WHERE product_id = 1;
UPDATE 1
company=# SELECT * FROM company.log_price_update;
 log_id |          log_date          | product_id |        product_name         | old_price | new_price | product_producer
--------+----------------------------+------------+-----------------------------+-----------+-----------+------------------
      5 | 2020-11-22 11:41:26.191782 |          1 | Pasta - Shells, Medium, Dry |      3.45 |     33.45 | 1
(1 row)


company=#
```

ZAPIS LOGÓW PRZY ZMIANIE CEN PRODUKTÓW.

5. Automatyzacja działania

W ramach automatyzacji działania w bazie danych, wykonano automatyczny backup. Do wykonaniu backup wykorzystano mechanizm z bazy danych PostgreSQL pg_dump.

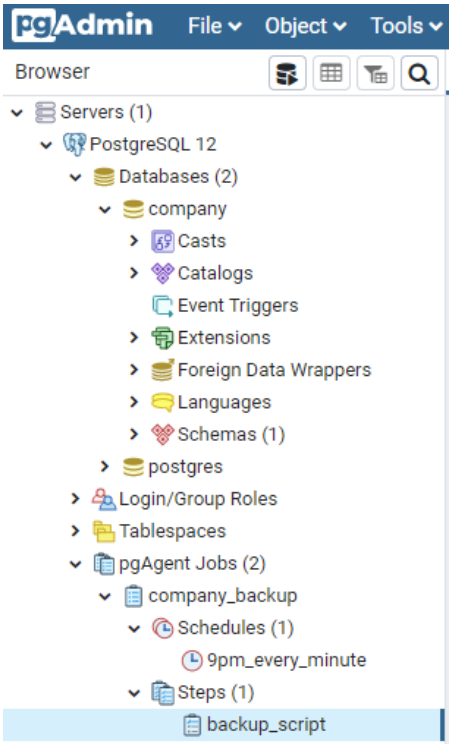
Pierwszym krokiem jest napisanie komendy obejmującej maszynę, na której ma zostać wykonana procedura (localhost), użytkownika (management) oraz ścieżkę do zapisu plików backup.

```
cd C:\\Users

pg_dump -h "localhost" -U "management" -f "C:\\Users\\yoszimitsu\\Documents\\it_projects\\Company_DB\\DB\\backup\\company_backup
```

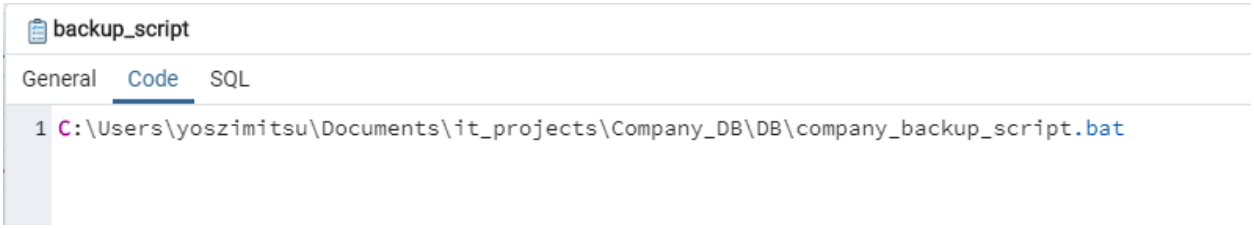
SKRYPT TWORZĄCY BACKUP BAZY DANYCH COMPANY.

Po przygotowaniu odpowiedniego skryptu i zapisaniu go w formacie .bat można przejść do automatyzacji procesu. Do tego celu można wykorzystać pgAgent ze środowiska PostgreSQL lub Task Scheduler ze środowiska Windows.



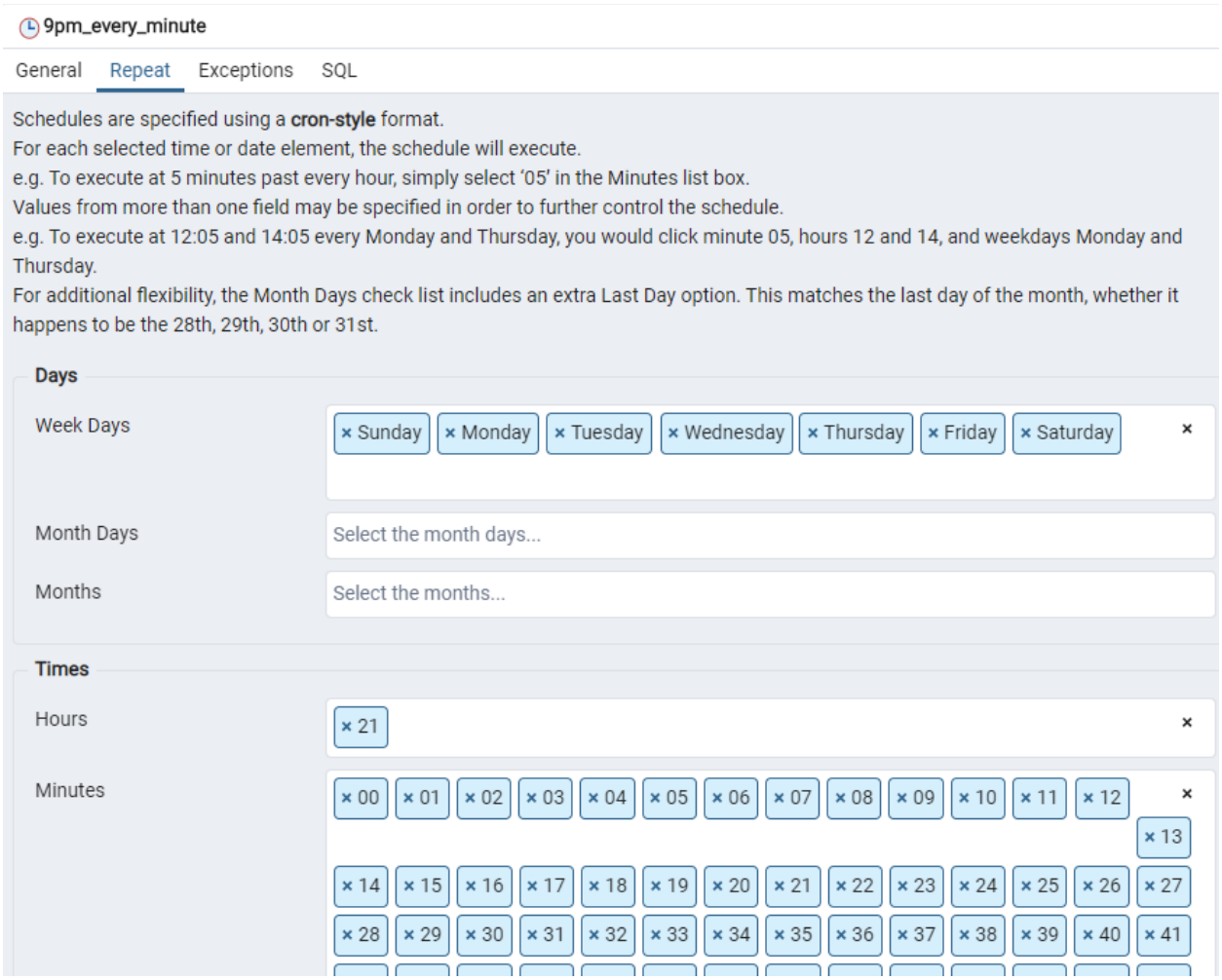
PANEL PGADMIN Z ZAKŁADKĄ PGADENT JOBS

Przy konfiguracji Job'a można wpisać komendę SQL lub odnieść się do pliku z systemu. W tym przypadku wskazano na wcześniej przygotowany skrypt.



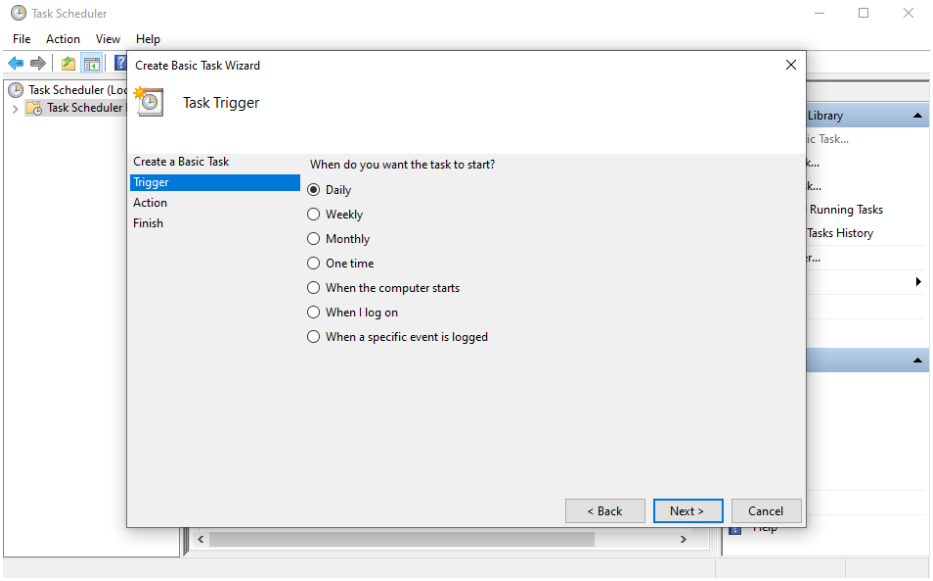
KONFIGURACJA JOB'A.

Jon będzie się wykonywał w godzinach określonych w Schedules.



KONFIGURACJA SCHEDULES.

Również do tego celu można wykorzystać Task Scheduler z Windows.



KONFIGURACJA TASK SCHEDULER.

Niezależnie od wyboru narzędzia do automatyzacji, częścią wykonawczą jest skrypt zawierający narzędzie pg_dump z środowiska PostgreSQL.

6. Napotkane problemy

Podjąłem próbę stworzenia funkcji zwracającej zniżki na produkt w zależności od typu produktu (a lub b). Niestety SELECT type_a FROM company.product zwraca całą kolumnę wartości i nie może być przyrównana do true. Wartości musiałby być sprawdzane po kolei, następnie musiałaby być z nich stworzona cala kolumna i dopiero przekazana do widoku product_order_value, gdzie miała zastąpić zwielokrotniony kod.

```
CREATE OR REPLACE FUNCTION company.product_type_disc() RETURNS TABLE (discount NUMERIC(3,2))
LANGUAGE plpgsql
AS $$
BEGIN
    CASE
      WHEN (COALESCE((SELECT type_a FROM company.product), false)) IS TRUE THEN
                SELECT type_a_discount FROM company.producer;
      WHEN  (COALESCE((SELECT type_b FROM company.product), false)) IS TRUE THEN
                SELECT type_b_discount FROM company.producer;
      ELSE

    END CASE;
END
$$;
```

Poprawki Etap2

Po drugim etapie dodano indeksy na start_date i end_date dla investiment i sub_investment. Daty mogą być zmiennymi często używanymi do filtracji inwestycji patrząc ze strony biznesowej. Dodanie indeksów może znacząco przyśpieszyć takie operacje filtrowania.

```
CREATE  INDEX sub_investment_start_date ON company.sub_investment (start_date);
CREATE  INDEX sub_investment_end_date ON company.sub_investment (end_date);

CREATE  INDEX investment_start_date ON company.investment (start_date);
CREATE  INDEX investment_end_date ON company.investment (end_date);
```

Poprawiono również błąd przy funkcji z obsługą błędów. Błąd był spowodowany złym przypisaniem zmiennej.

```
        raise notice E'Got exception:
            state  : %
            message: %
            detail : %
            hint   : %
            context: %', v_state, v_msg, v_de

        raise notice E'Got exception:
            SQLSTATE: %
            SQLERRM: %', SQLSTATE, SQLERRM;

        raise notice '%', message_text;

    end;
$$ LANGUAGE plpgsql;
```

```
        raise notice E'Got exception:
            state  : %
            message: %
            detail : %
            hint   : %
            context: %', v_state, v_msg, v_detail, v_hint, v_context;

        raise notice E'Got exception:
            SQLSTATE: %
            SQLERRM: %', SQLSTATE, SQLERRM;

        raise notice '%', v_msg;

    end;
$$ LANGUAGE plpgsql;
```

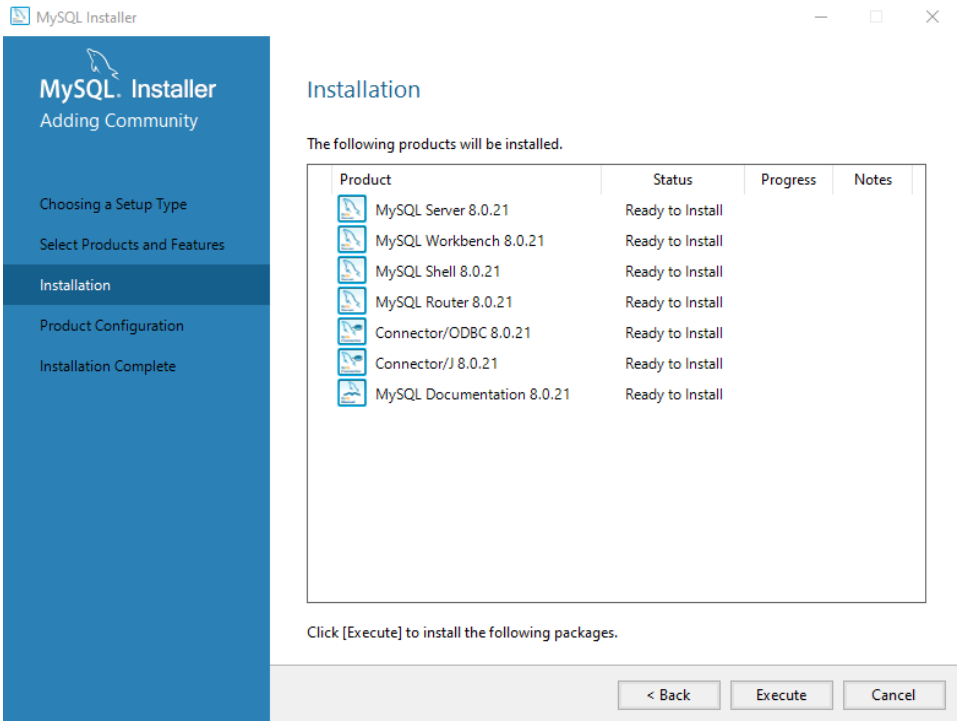
Po poprawce obsługa błędów przebiega prawidłowo.

```
company=> CALL company.new_product_and_producer('soap', 3.45, true, true, 'Dove', 0.25, 0.0);
UWAGA:  Got exception:
            state  : 23505
            message: podwA3jna wartoÅ>Ä╣ klucza narusza ograniczenie unikalnoÅ>ci "unique_constraints"
            detail : Klucz (name)=(Dove) juÅ¼ istnieje.
            hint   :
            context: wyraÅ¼enie SQL "INSERT INTO company.producer(
            name,
            type_a_discount,
            type_b_discount)
        VALUES(
            producer_name,
            type_a_discount,
            type_b_discount)"
funkcja PL/pgSQL company.new_product_and_producer(character varying,numeric,boolean,boolean,character varying,numeric,numeric), wiersz 10 w wyraÅ¼enie SQL
UWAGA:  Got exception:
            SQLSTATE: 23505
            SQLERRM: podwA3jna wartoÅ>Ä╣ klucza narusza ograniczenie unikalnoÅ>ci "unique_constraints"
UWAGA:  podwA3jna wartoÅ>Ä╣ klucza narusza ograniczenie unikalnoÅ>ci "unique_constraints"
CALL
company=>
```

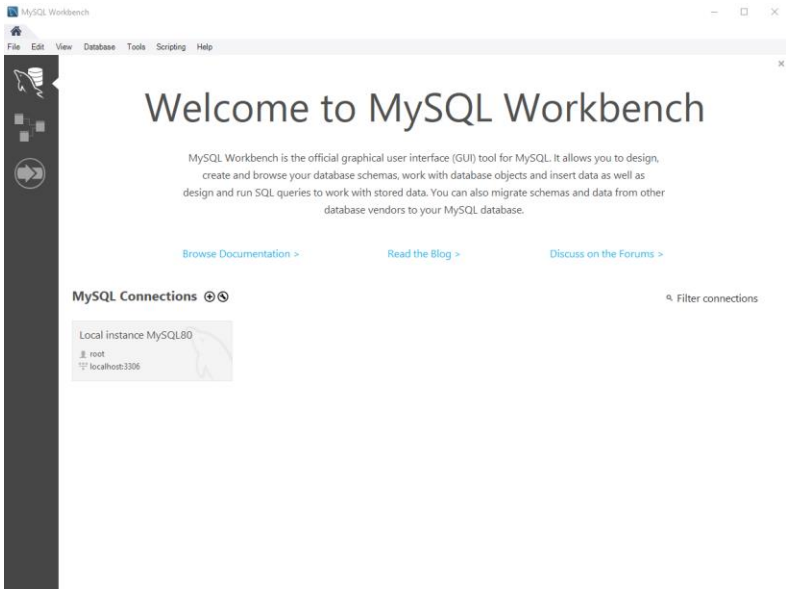
Etap3

1. Instalacja środowiska MySQL

Po pierwszej części projektu przygotowanego w PostgreSQL zdecydowano się na migracje do MySQL.



INSTALACJA MYSQL



PANEL MYSQL WORKBENCH PO INSTALACJ.

2. Różnice pomiędzy PostgreSQL a MySQL

PostgreSQL i MySQL należą do baz danych o otwartym kodzie źródłowym i darmowej licencji, choć MySQL oferuję również wersje komercyjną.

PostgreSQL lepiej implementuje standard SQL, przez co jest postrzegany jako pewniejszy DBMS. Jest wykorzystywany w aplikacjach każdego rodzaju, od małych aplikacji internetowych po duże aplikacje wymagające analizy danych. Za wadę PostgreSQL względem MySQL można uznać wolniejszą operację odczytu danych. Za to posiada szereg zalet jak możliwość tworzenia własnych typów danych, szybsze operacje zapisu danych, zapewnia lepszą ochronę przy transakcjach oraz w spełnia zasady ACID.

MySQL jest szybką i lekką bazą danych, lecz poprzez mniejszą zgodność ze standardem SQL należy uważać przy większych zastosowaniach. Za to nadaje się idealnie do małych i średnich aplikacji internetowych, na których chcemy dokonywać podstawowych operacji na danych. MySQL zapewnia doskonałą szybkość odczytu danych.

Wybór spomiędzy tych dwóch DBMS zależy od potrzebnych zastosowań. Jeśli nie ma potrzeby korzystania z szerszych możliwości PostgreSQL oraz nie w bazie danych nie będą wykonywane skomplikowane operacje można skierować się w stronę MySQL. Lecz jeśli baza danych może przyjąć duże rozmiary oraz istnieje faktyczna potrzeba zaawansowanych operacji na danych lepiej postawić na PostgreSQL.

Istnieją różnice w dialekcie pomiędzy dwoma systemami baz danych. Problemy wydają się być większe przy przenoszeniu zapytań z MySQL po PostgreSQL niż w drugą stronę chociażby z racji na kwestie z apostrofami/cudzysłowami.

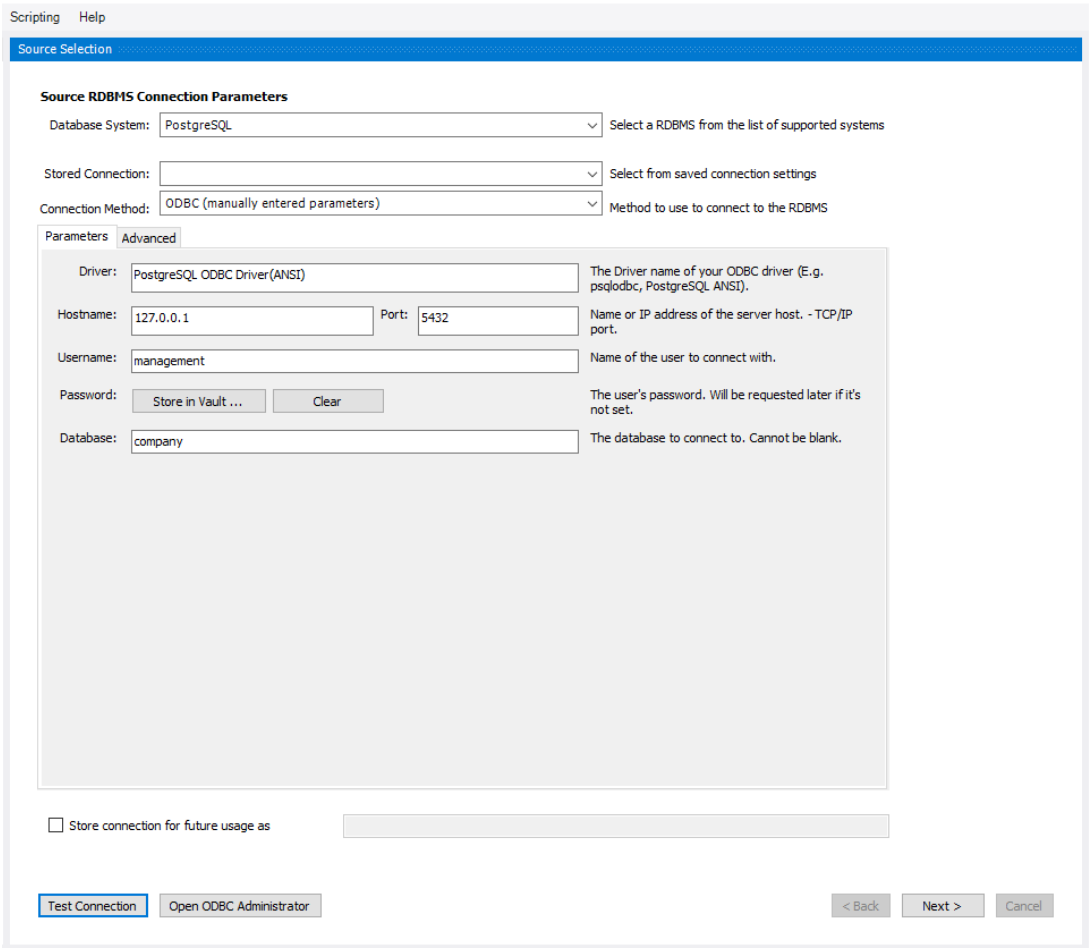
3. Migracja

Dostępnych jest kilka narzędzi pozwalających na migrację bazy danych z PostgreSQL do MySQL. W pierwszej kolejności wypróbowano MySQL Workbench, ponieważ jest to natywny client do DBMS MySQL. Wypróbowano również zewnętrzne narzędzie ESF Database Migration Toolkit.

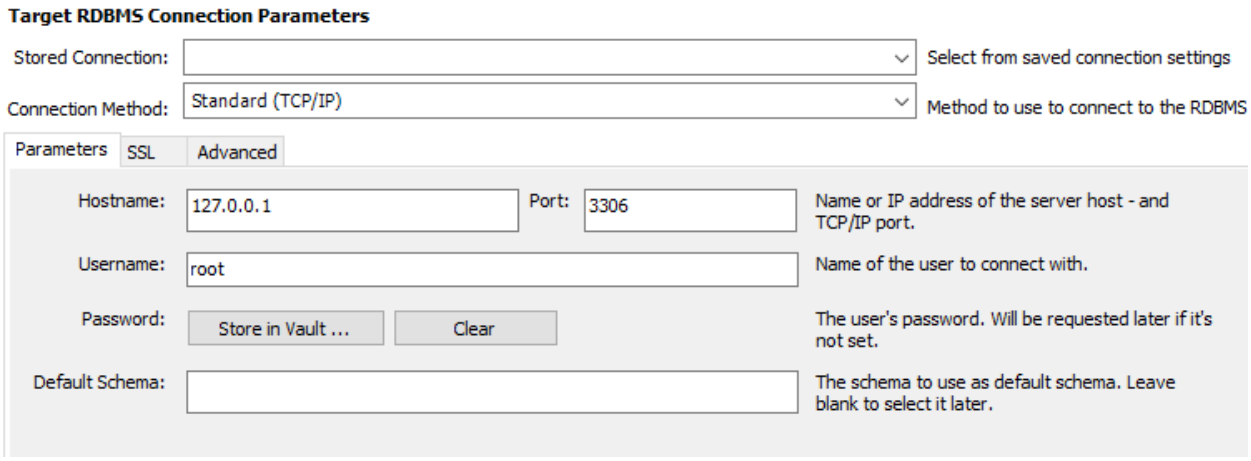
3.1 MySQL Workbench Migration

MySQL Workbench udostępnia narzędzie Migration, które pozwala na przeprowadzenie migracji spośród kilku systemów DB.

Pierwszym krokiem jest wybranie źródła oraz docelowej bazy danych. Aby była możliwa migracja z PostgreSQL trzeba zainstalować odpowiedni Driver (ODBC).

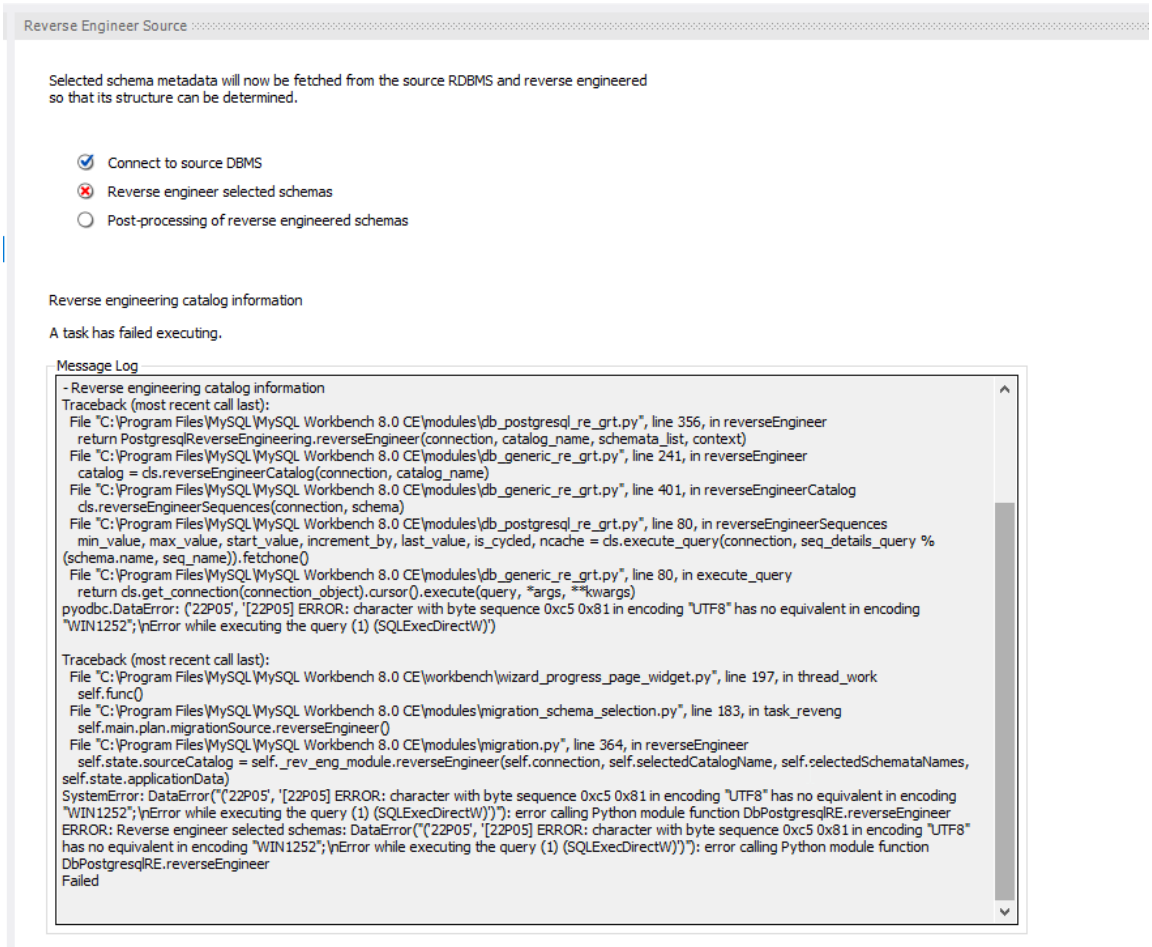


KONFIGURACJA BAZY ŹRÓDŁOWEJ W WORKBENCH MIGRATION.



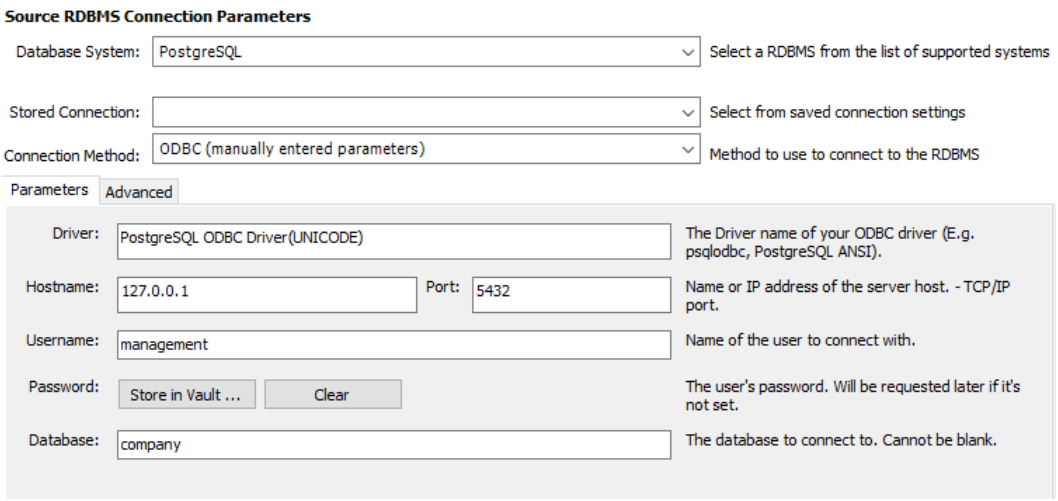
KONFIGURACAJ BAZY DOCELOWEJ

Pierwszym napotkanym problemem był odpowiedni wybór Drivera. Okazało się, że pojawił się problem z kodowaniem znaków.



PROBLEM KODOWANIA PODCZAS MIGRACJI.

Rozwiązaniem było zmiana Drivera na typ UNICODE. Wpłynęło to na rodzaj kodowania znaków. Tym samym problem został rozwiązany.



ZMIANA DRIVERA NA UNICODE.

Następnym problemem był odczytanie „min_value” z tabeli systemowych PostgreSQL.

Fix File `"C:\Program Files\MySQL\MySQL Workbench 6.3 CE\modules\db_postgresql_re_grt.py"` lines ~70 change to

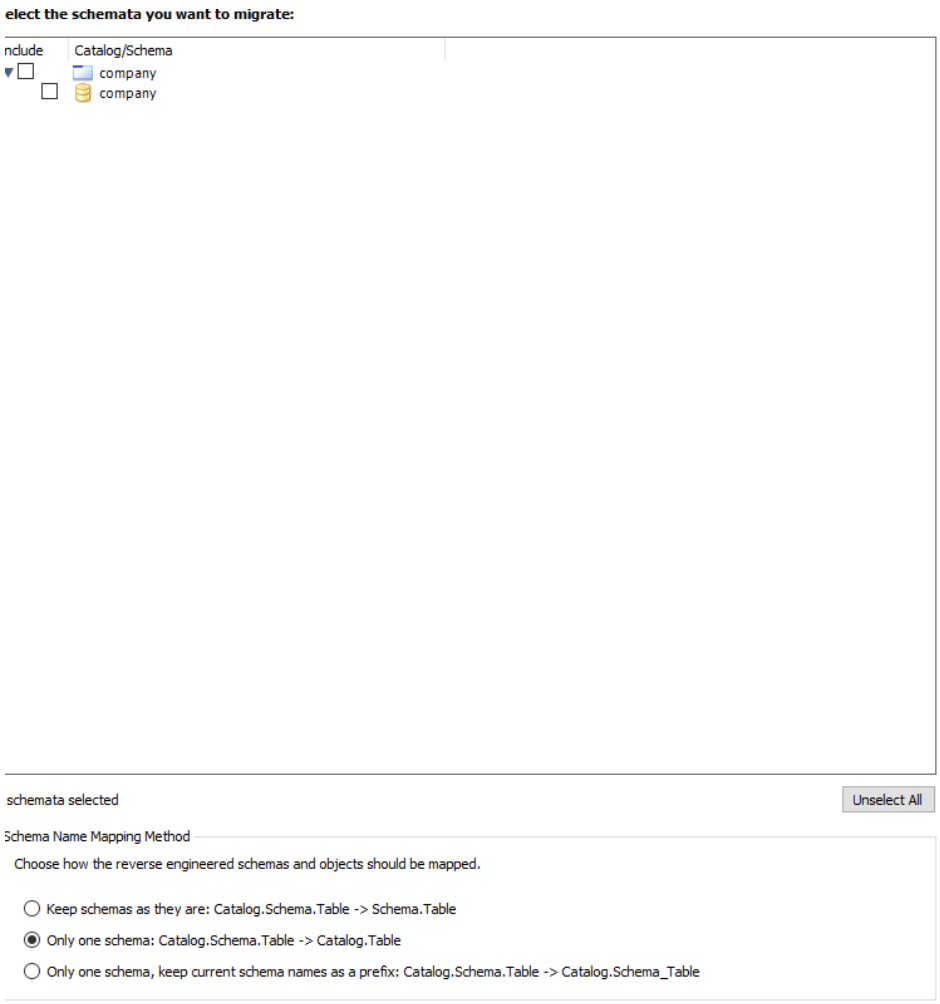
```
seq_details_query = """SELECT min_value, max_value, start_value,
increment_by, last_value, cycle as is_cycled, cache_size as cache_value
FROM pg_catalog.pg_sequences
WHERE schemaname = '%s' AND sequencename = '%s' """
```

EDYCJA PLIKÓW ŹRÓDŁOWYCH MYSLQ DOSTOSOWUJĄC JE DO POSTGRESQL

Rozwiązaniem była edycja plików źródłowych MySQL i odpowiednie przypisanie nazw zmiennych. Różnice mogły pojawić się wraz z nowymi wersjami PostgreSQL.

Edycja pliku db_postgresql_re_grt.py rozwiązała problem.

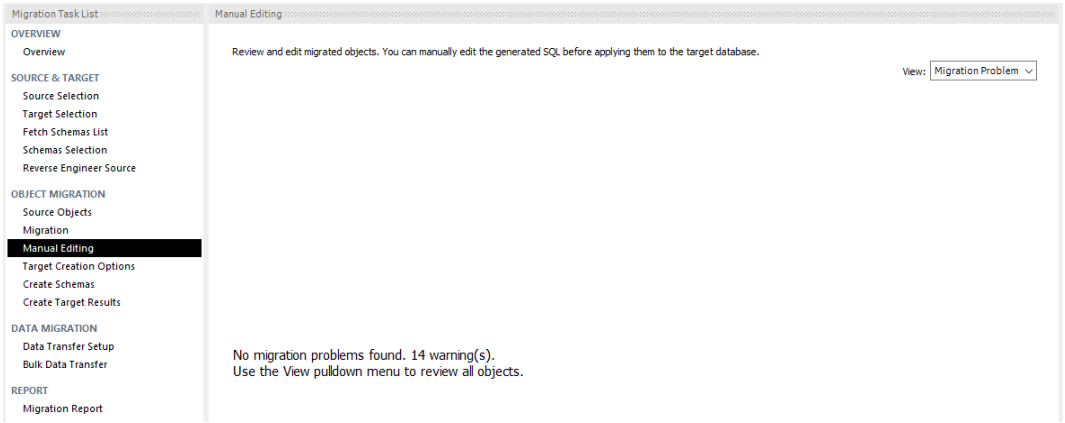
Następnie sprawdzana jest konfiguracja. Po pomyślnej konfiguracji należy wybrać schemat bazy danych, do migrowania.



WYBRÓ SCHEMATU DO MIGRACJI.

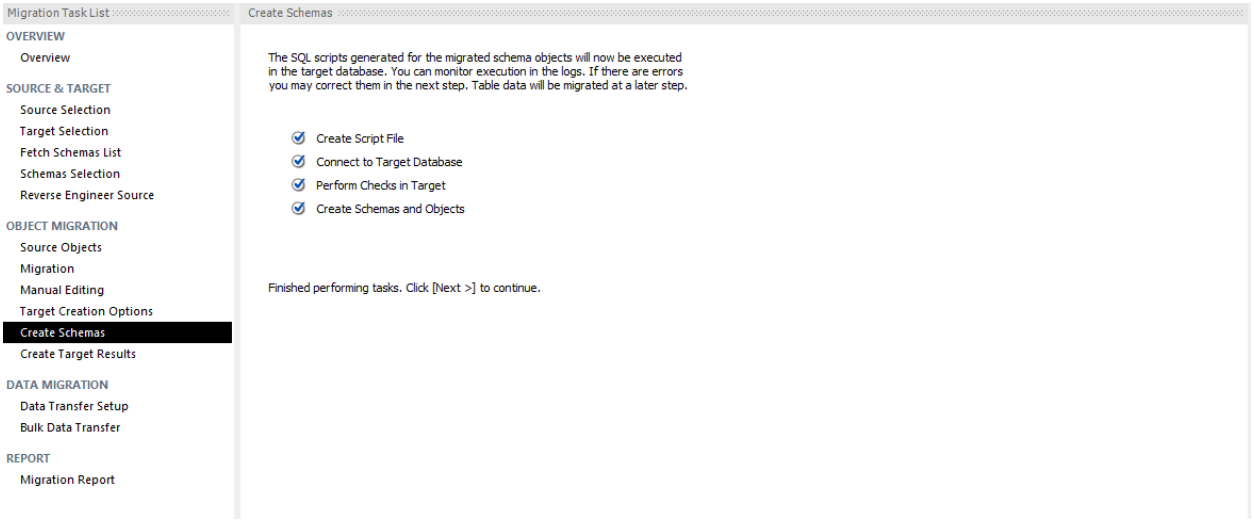
Podczas próby odczytu schematu z PostgreSQL występował „crash” w MySQL Workbench i aplikacja zamykała się bez podania żadnego błędu.

Rozwiązaniem był downgrade do wersji 8.12.



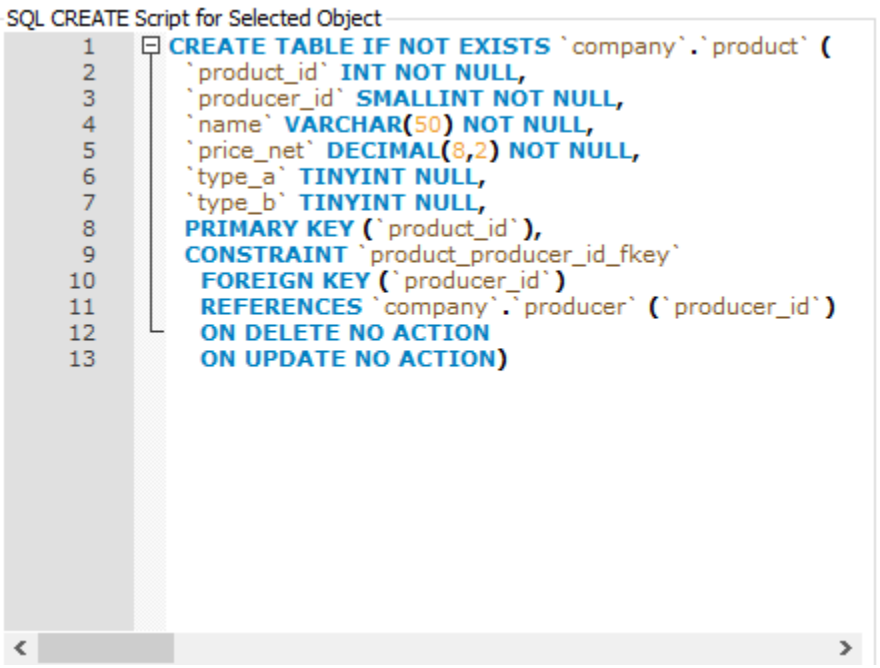
IFORMACJA O UKONCZENIU PROCESU MIGRACJI (POBRANIA ZAWARTOSCI Z POSTGRESQL).

Po przeinstalowaniu wersji MySQL na starszą, udało się odczytać schemat z PostgreSQL.



TWORZENIE ZMIGROWANEGO SCHEMATU w MYSQL.

Przed migracją jest możliwość podejrzenia zapytań SQL, które przygotowało narzędzie do migracji. Jeśli zauważymy błędy lub braki, na tym etapie można je uzupełnić.



SQL CREATE Script for Selected Object

```sql
CREATE TABLE IF NOT EXISTS `company`.`producer` (
  `producer_id` SMALLINT NOT NULL,
  `name` VARCHAR(50) NOT NULL,
  `contact_person_id` SMALLINT NULL,
  `type_a_discount` DECIMAL(3,2) NULL,
  `type_b_discount` DECIMAL(3,2) NULL,
  PRIMARY KEY (`producer_id`),
  UNIQUE INDEX `unique_constraints` (`name` ASC) VISIBLE,
  CONSTRAINT `producer_contact_person_id_fkey`
    FOREIGN KEY (`contact_person_id`)
    REFERENCES `company`.`contact_person` (`contact_person_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

SQL CREATE Script for Selected Object

```sql
CREATE TABLE IF NOT EXISTS `company`.`investment` (
  `investment_id` INT NOT NULL,
  `name` VARCHAR(50) NOT NULL,
  `client_id` BIGINT NOT NULL,
  `place_object_id` INT NULL,
  `supervisor_id` INT NOT NULL,
  `start_date` DATE NOT NULL,
  `end_date` DATE NULL,
  PRIMARY KEY (`investment_id`),
  INDEX `invest_name_idx` (`name` ASC) VISIBLE,
  CONSTRAINT `investment_client_id_fkey`
    FOREIGN KEY (`client_id`)
    REFERENCES `company`.`client` (`client_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `investment_place_object_id_fkey`
    FOREIGN KEY (`place_object_id`)
    REFERENCES `company`.`place_object` (`place_object_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `investment_supervisor_id_fkey`
    FOREIGN KEY (`supervisor_id`)
    REFERENCES `company`.`supervisor` (`supervisor_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

SQL CREATE Script for Selected Object

```sql
CREATE TABLE IF NOT EXISTS `company`.`sub_investment` (
  `sub_investment_id` INT NOT NULL,
  `name` VARCHAR(50) NOT NULL,
  `investment_id` INT NOT NULL,
  `start_date` DATE NOT NULL,
  `end_date` DATE NULL,
  PRIMARY KEY (`sub_investment_id`),
  INDEX `sub_invest_name_idx` (`name` ASC) VISIBLE,
  CONSTRAINT `sub_investment_investment_id_fkey`
    FOREIGN KEY (`investment_id`)
    REFERENCES `company`.`investment` (`investment_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

ZAPYTANIA TWORZACE TABELE W MYSQL. BRAK OGARNICZEŃ, KTÓRE BŁY W POSTGRESQL.

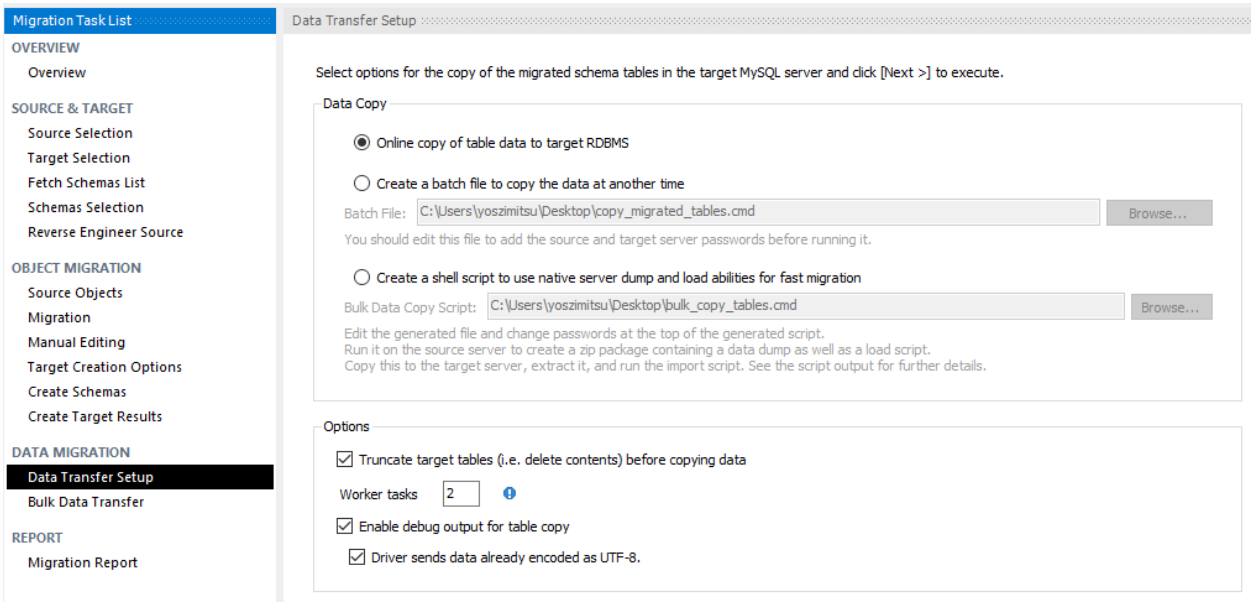
W powyżej przedstawionych tabelach zabrakło ograniczeń, które były zdefiniowane w PostgreSQL. Poniżej przedstawiono jedno z ograniczeń, którego zabrakło.

```
CREATE TABLE company.sub_investment (
    sub_investment_id SERIAL NOT NULL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    investment_id INTEGER NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE,
    CONSTRAINT check_dates CHECK(end_date > start_date),
    FOREIGN KEY(investment_id) REFERENCES company.investment(investment_id)
);
```

OGRANICZENIE W TABELU SUB_INVESTMENT SPRAWDZAŁO CZY DATA KONCOWA NIE JEST WCZEŚNIEJSZA NIŻ POCZĄTKOWA. JENDO Z UTRACONYCH OGRANICZEŃ W CZASIE MIGRACJI.

Po zaakceptowaniu schematu, tabele zostały utworzone w bazie MySQL.

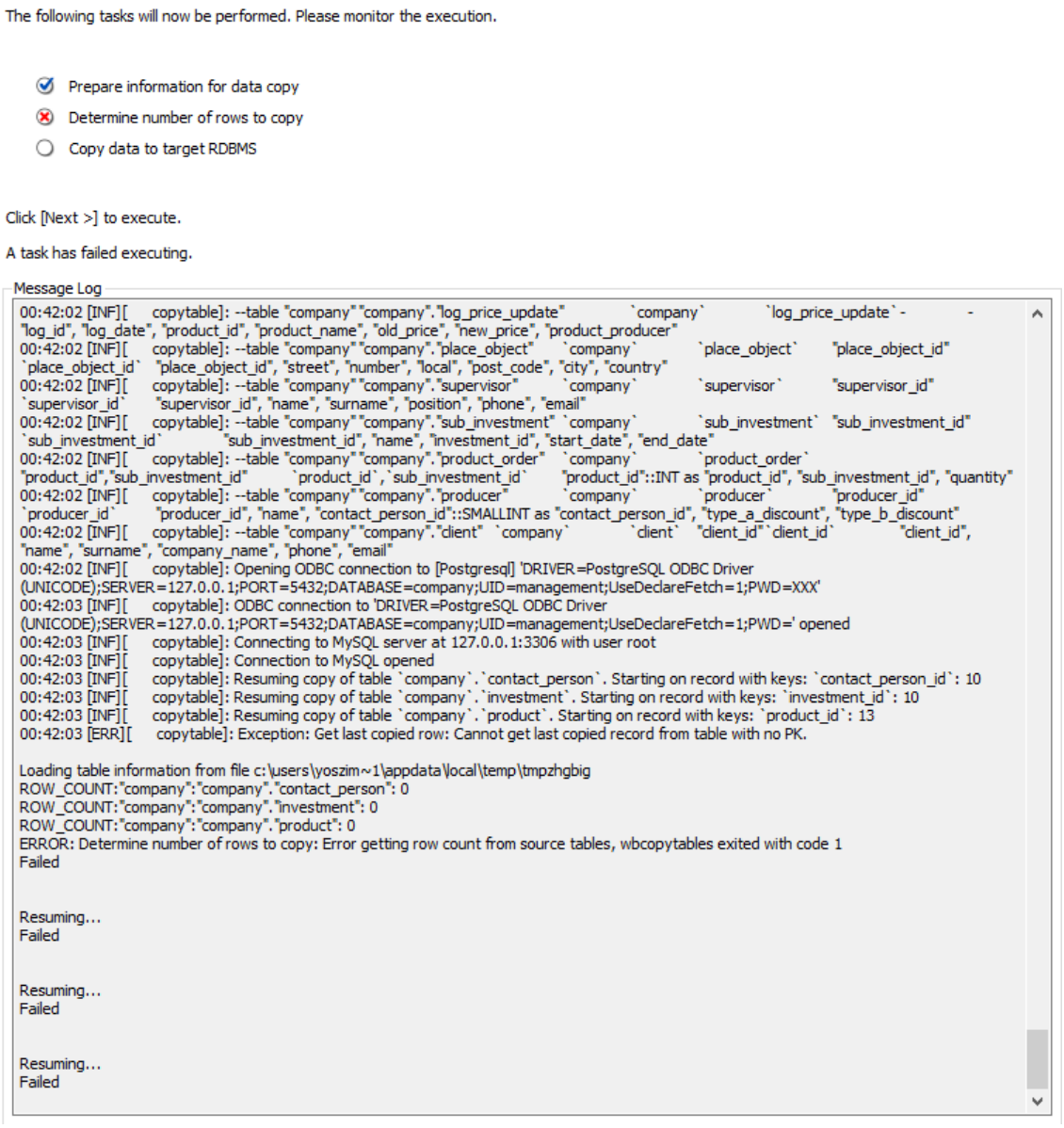
Następnym krokiem było przeniesienie zawartości tabel.



KONFIGURACJA PRZENIESIEMIA ZAWARTOŚCI TABEL DO MYSQL.

REZULAT PRZENOSZENIA DANYCH.

Poniższe zdjęcie przedstawia rezultat przenoszenia danych. Dane z 8 z 10 tabel zostały przeniesione bez problemu. Problem pojawił się przy tabeli producer oraz place_object. Niestety nie udało się rozwiązać tego problemu.

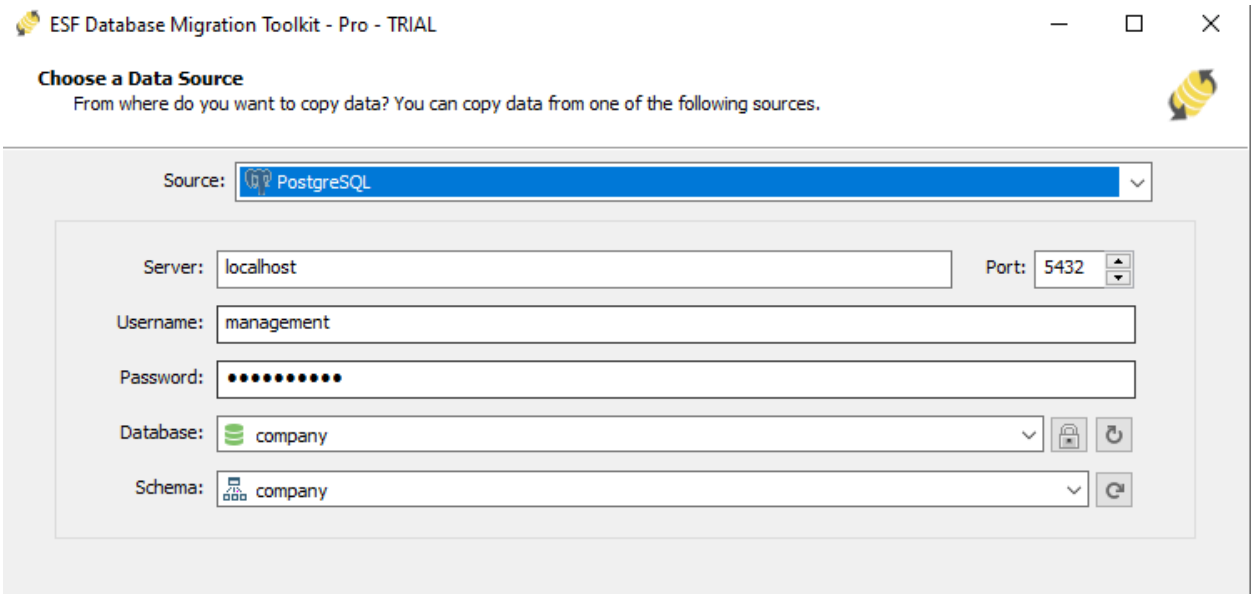


BŁAD PRZY ZAKONCZENIU MIGRACJI DANYCH.

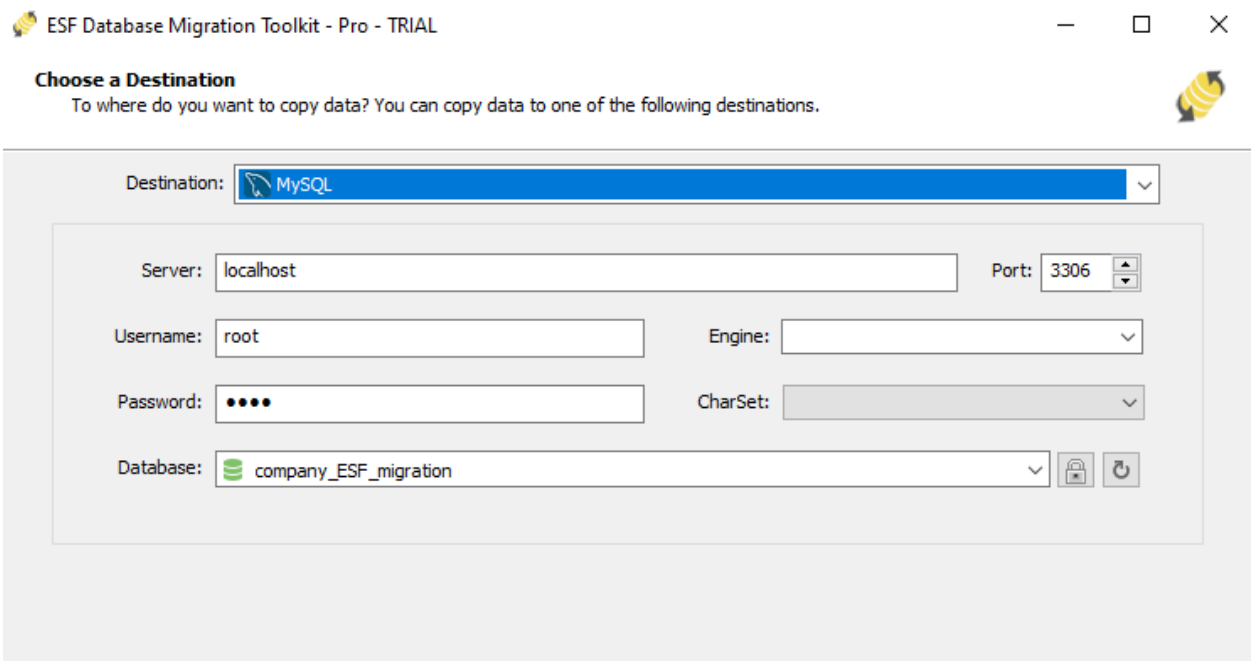
Procesu migracji (formalnie) również nie został zakończony. Przy próbie przejścia do końcowego etapu migracji jakim jest generowanie raportu pojawiał się komunikat „Resumin… Failed”. Być może migracja danych musi się odbyć bez problemu by przejść do ostatniego etapu.

3.2 ESF Database Migration Toolkit

Spróbowano również migracji za pomocą ESF Database Migration Toolkit. Pierwszym krokiem było wybór źródła oraz docelowej bazy danych.

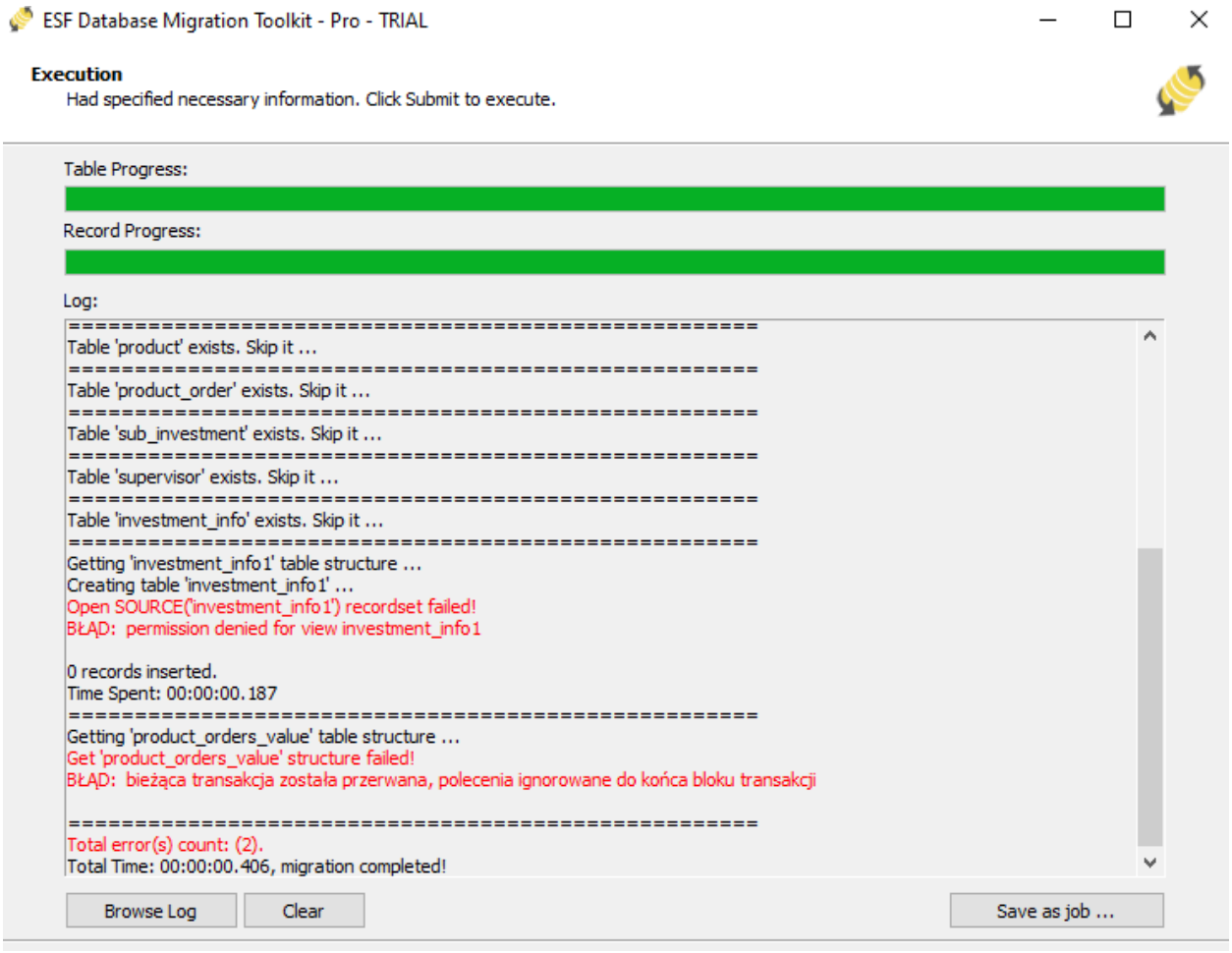


KONFIGURACJA ŹRÓDŁOWEJ BAZY DANYCH.



KONFIGURACJA DOCELOWEJ BAZY DANYCH.

Konfiguracja była o wiele szybsza i prostsza.



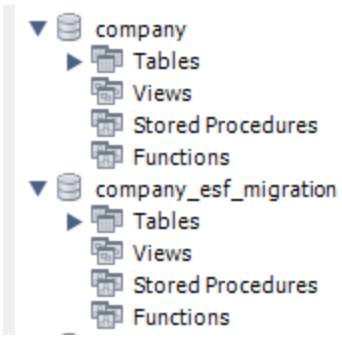
REZULTAT MIGRACJI.

Jedyny błędy jakie wystąpiły były związane z perspektywą product_orders_value oraz investment_info (problem prawdopodobnie wynikał z niewystarczających uprawnień użytkownika „management”).

Poza tym wszystkie dane zostały zmigorwane do MySQL.

3.3 Podsumowanie migracji

Po weryfikacji schematów w MySQL okazało się, że, zmigrowane zostały tylko tabele oraz dane (pomijając place_object oraz producent przy migracji przez Workbench). Perspektywy zostały przeniesione tylko przez narzędzie ESF i zapisano jako tabele. Procedury oraz funkcje nie zostały przeniesione.



ZMIGROWANE SCHEMATY W ŚRODOWISKU MYSQL WORKBENCH.

Result Grid | Filter Rows: | Edit: | Export/Import:

supervisor_id	name	surname	position	phone	email
1	Ogdon	Kauschke	engineer	228 911 2936	okauschke0@wired.com
2	Chauncey	Drewe	engineer	474 860 1874	cdrewe1@independent.co.uk
3	Jewel	Dik	engineer	974 941 9143	jdik2@google.cn
4	Cori	Follacaro	engineer	752 997 3851	cfollacaro3@a8.net
5	Walliw	Regler	engineer	789 582 4255	wregler4@nydailynews.com
6	Stirling	Williams	engineer	925 222 8698	swilliams5@mashable.com
7	Danella	Valti	engineer	317 883 0721	dvalti6@ucsd.edu
8	Katina	Lubbock	engineer	857 635 9630	klubbock7@lulu.com
9	Ladonna	Matyasik	engineer	146 450 5182	lmatyasik8@chronoengine.com
10	Iormina	Lodeke	engineer	365 179 3664	ilodeke9@google.co.jp
12	daniel2	michrow...	test2	test2	test2@test.pl
13	daniel	michrowski	test	test	test@test.pl
NULL	NULL	NULL	NULL	NULL	NULL

ZAWARTOŚĆ TABELI SUPERVISOR ZE SCHEMATU COMPANY_ESF_MIGRATION.

```
company=# Select * from company.supervisor;
 supervisor_id |   name    |   surname    | position |    phone     |            email
---------------+-----------+--------------+----------+--------------+-----------------------------
             1 | Ogdon     | Kauschke     | engineer | 228 911 2936 | okauschke0@wired.com
             2 | Chauncey  | Drewe        | engineer | 474 860 1874 | cdrewe1@independent.co.uk
             3 | Jewel     | Dik          | engineer | 974 941 9143 | jdik2@google.cn
             4 | Cori      | Follacaro    | engineer | 752 997 3851 | cfollacaro3@a8.net
             5 | Walliw    | Regler       | engineer | 789 582 4255 | wregler4@nydailynews.com
             6 | Stirling  | Williams     | engineer | 925 222 8698 | swilliams5@mashable.com
             7 | Danella   | Valti        | engineer | 317 883 0721 | dvalti6@ucsd.edu
             8 | Katina    | Lubbock      | engineer | 857 635 9630 | klubbock7@lulu.com
             9 | Ladonna   | Matyasik     | engineer | 146 450 5182 | lmatyasik8@chronoengine.com
            10 | Iormina   | Lodeke       | engineer | 365 179 3664 | ilodeke9@google.co.jp
            12 | daniel2   | michrowski2  | test2    | test2        | test2@test.pl
            13 | daniel    | michrowski   | test     | test         | test@test.pl
(12 rows)
```

ZAWARTOŚĆ TABLI SUPERVISOR W ŚRODOWISKU POSTGRESQL.

Należałoby dostosować wszystkie procedury oraz funkcje do składni MySQL.

Podjęto próbę przeniesienia funkcji. Niestety natrafiłem na problem ze składnia, których nie udało mi się poprawić.

```
DELIMITER $$

CREATE FUNCTION company_esf_migration.address() RETURNS VARCHAR
BEGIN
    DECLARE address VARCHAR;

    SELECT CONCAT("st. ", street, " ", number, " ,", post_code, " ,", city, " ,", country) INTO address
    FROM company_esf_migration.place_object;

    RETURN address;number
END$$

DELIMITER ;
```

PRÓBA PRZENIESIENIA FUNKCJI ADDRESS() DO MYSQL.

12:41:50 CREATE FUNCTION company_esf_migration.address() RETURNS VARCHAR BEGIN DECLARE address VARCHAR; SELECT CONCAT("st. ", street, " ", number, " ,", post_code, " ,", city, " ,", country) INTO address FROM company_esf_migration.place_object; RETURN address;number END Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'BEGIN DECLARE address VARCHAR; SELECT CONCAT("st. ", street, " ", numb' at line 2 0.000 sec

BŁĄD ZWRACANY PRZEZ MYSQL.

Funkcja z obsługą błędów należałoby napisać zupełnie od nowa, ponieważ wykorzystywała narzędzie wbudowane w PostgreSQL.

3.4 Wnioski

Migracja zakończyła się częściowym powodzeniem. Udało się przenieść schemat bazy danych, wszystkie tabel z odpowiednimi typami oraz kluczami. Dane zawarte w tabelach również zostały poprawnie przeniesione.

Perspektywy zostały przeniesione jako tabele do bazy MySQL. Funkcje, procedury i wyzwalacze nie zostały przeniesione. Należałoby zaimplementować je zgodnie ze składnią MySQL.

Narzędzie MySQL Workbench Migration przysporzyło sporo problemów. Dane z 2 tabel nie zostały przeniesione, wskazując niezrozumiałe błędy. Problemy wynikały również z działania samego narzędzia pokazując, że nie jest to najlepsze narzędzie do migracji.

O wiele sprawniej przebiegła migracja używając ESF Database Migration Toolkit. Wszystkie dane zostały przeniesione poprawnie, bez żadnych problemów konfiguracyjnych.

4. Połączenie bazodanowe

Do połączenia bazodanowego pomiędzy PostgreSQL i MySQL można użyć narzędzie Foreign Data Wrapper.

https://github.com/EnterpriseDB/mysql_fdw/blob/master/README.md

Niestety przy próbie konfiguracji narzędzia wystąpił błąd.

```
PS C:\Users\yoszimitsu\Desktop\mysql_fdw-master> make USE_PGXS=1
process_begin: CreateProcess(NULL, mysql_config --include, ...) failed.
Makefile:16: pipe: No error
process_begin: CreateProcess(NULL, mysql_config --libs, ...) failed.
Makefile:17: pipe: No error
process_begin: CreateProcess(NULL, uname, ...) failed.
Makefile:27: pipe: No error
Makefile:44: *** PostgreSQL 9.5, 9.6, 10, 11, 12, or 13 is required to compile this extension.  Stop.
PS C:\Users\yoszimitsu\Desktop\mysql_fdw-master> postgres --version
postgres (PostgreSQL) 12.4
PS C:\Users\yoszimitsu\Desktop\mysql_fdw-master>
```

BŁĄD PRZY PRÓBIE KONFIGURACJI NARZĘDZIA FOREIGN DATA WRAPPER W ŚRODOWISKU WINDOWS.

Prawdopodobnie narzędzie jest kompatybilne z systemami Linux. W systemie Windows nie udało mi się rozwiązać problemu.

Istnieją narzędzie ‘dblink’ w PostgreSQL, które pozwalają na połączenie między bazami danych w systemie PostgreSQL.

```
company=# CREATE EXTENSION dblink WITH SCHEMA company;
CREATE EXTENSION
company=#
```

```
company=# CREATE SERVER server_company_remote FOREIGN DATA WRAPPER dblink_fdw OPTIONS (host 'remote', dbname 'remote', p
ort '5432');
CREATE SERVER
company=#
```

```
company=# CREATE USER MAPPING FOR postgres SERVER server_company_remote OPTIONS (user 'root',password 'root');
CREATE USER MAPPING
company=#
```

```
company=# GRANT USAGE ON FOREIGN SERVER server_company_remote TO postgres;
GRANT
company=#
```

```
company=# SELECT * from company.dblink('host=localhost user=postgres password=123 dbname=company','select * from com
pany.client') AS x(client_id integer,name VARCHAR(50), surname VARCHAR(50),  company_name VARCHAR(50), phone VARCHAR
(15), email VARCHAR(70));
 client_id |    name    |  surname   |         company_name          |    phone     |          email
-----------+------------+------------+-------------------------------+--------------+-------------------------
         1 | Beverie    | Nevet      | Graham, Hudson and Hettinger  | 123 557 7970 | bnevet0@github.io
         2 | Kelvin     | Dolder     | Hahn Group                    | 129 842 9118 | kdolder1@oracle.com
         3 | Kaylil     | Lotte      | Lueilwitz Inc                 | 819 330 4006 | klotte2@mapquest.com
         4 | Reeta      | Astman     | Casper-Harber                 | 322 893 4668 | rastman3@pen.io
         5 | Noby       | Theze      | Johnston-Spencer              | 976 549 1871 | ntheze4@ning.com
         6 | Maximilian | Herculeson | Turcotte-Daniel               | 411 442 5859 | mherculeson5@1und1.de
         7 | Lilly      | Billiard   | Goldner, Parker and Dibbert   | 991 112 7980 | lbilliard6@earthlink.net
         8 | Olly       | Bilovsky   | Hartmann LLC                  | 266 957 4203 | obilovsky7@homestead.com
         9 | Nicolea    | Luker      | Stark-Brown                   | 595 190 5940 | nluker8@goo.ne.jp
        10 | Grier      | Dory       | Altenwerth, Kozey and Johnson | 948 913 1268 | gdory9@ovh.net
        11 | test       | test       | test                          | 123          | test@test.pl
(11 rows)
```

POLACZENIE BAZ DANYCH W SRODOWISKU POSTGRESQL Z UŻYCIEM DBLINK.