



POLITECHNIKA  
LUBELSKA  
WYDZIAŁ ELEKTROTECHNIKI  
I INFORMATYKI

## ZARZĄDZANIE BAZAMI SQL I NoSQL

*Projekt systemu wypożyczalni filmów*

Zespół projektu:

1. Mateusz Kierepka, Gr. 6.6
2. Hubert Kiszka, Gr. 6.6 – kierownik
3. Mateusz Kłos, Gr. 6.6

Prowadzący projekt:

mgr inż. Albert Rachwał

Lublin, 2025

## Spis treści

1. Opis słowny i motywacje wykonania bazy danych.....	3
2. Wyodrębnienie niezbędnych tabel.....	4
3. Diagram bazy danych .....	5
4. Wypełnienie bazy danymi.....	6
5. Prezentacja działania zapytań pobierających dane z bazy .....	7
6. Prezentacja działania zapytań edytujących i usuwających dane .....	13
7. Zarządzanie użytkownikami, prezentacja działania użytkowników z różnymi zestawami uprawnień.....	14
8. Kopia zapasowa, import i eksport .....	16
9. Przykład tworzenia wyzwalaczy .....	17

# 1. Opis słowny i motywacje wykonania bazy danych

Wypożyczalnia filmów wymaga odpowiedniej bazy danych, która umożliwi efektywne zarządzanie biblioteką filmów i danymi użytkowników. Głównym zadaniem bazy jest przechowywanie danych klientów, którzy korzystają z usług wypożyczalni, informacji o filmach dostępnych do wypożyczenia, a także szczegółów dotyczących historii oglądania oraz opinii użytkowników na temat filmów. Ponadto przechowuje informacje na temat planów subskrypcji, metod płatności i historii transakcji.

Użytkownicy wypożyczalni będą mogli tworzyć swoje konta, zarządzać swoimi subskrypcjami, wypożyczać wybrane przez siebie pozycje filmowe, zapisywać je jako ulubione, wystawiać oceny i pisać recenzje oraz śledzić historię oglądania.

Baza przechowuje szczegółowe informacje na temat filmów takie jak tytuł, opis, kategoria, rok wydania, obsada, reżyser, ocena i dostępne wersje językowe. Dzięki temu użytkownicy mogą łatwo wyszukiwać i filtrować filmy według różnych kryteriów.

Użytkownicy mogą wybrać jeden z planów subskrypcyjnych, które różnią się ceną, okresem trwania i oferowanymi korzyściami lub alternatywnie mają możliwość wypożyczania pojedynczych filmów bez konieczności subskrypcji.

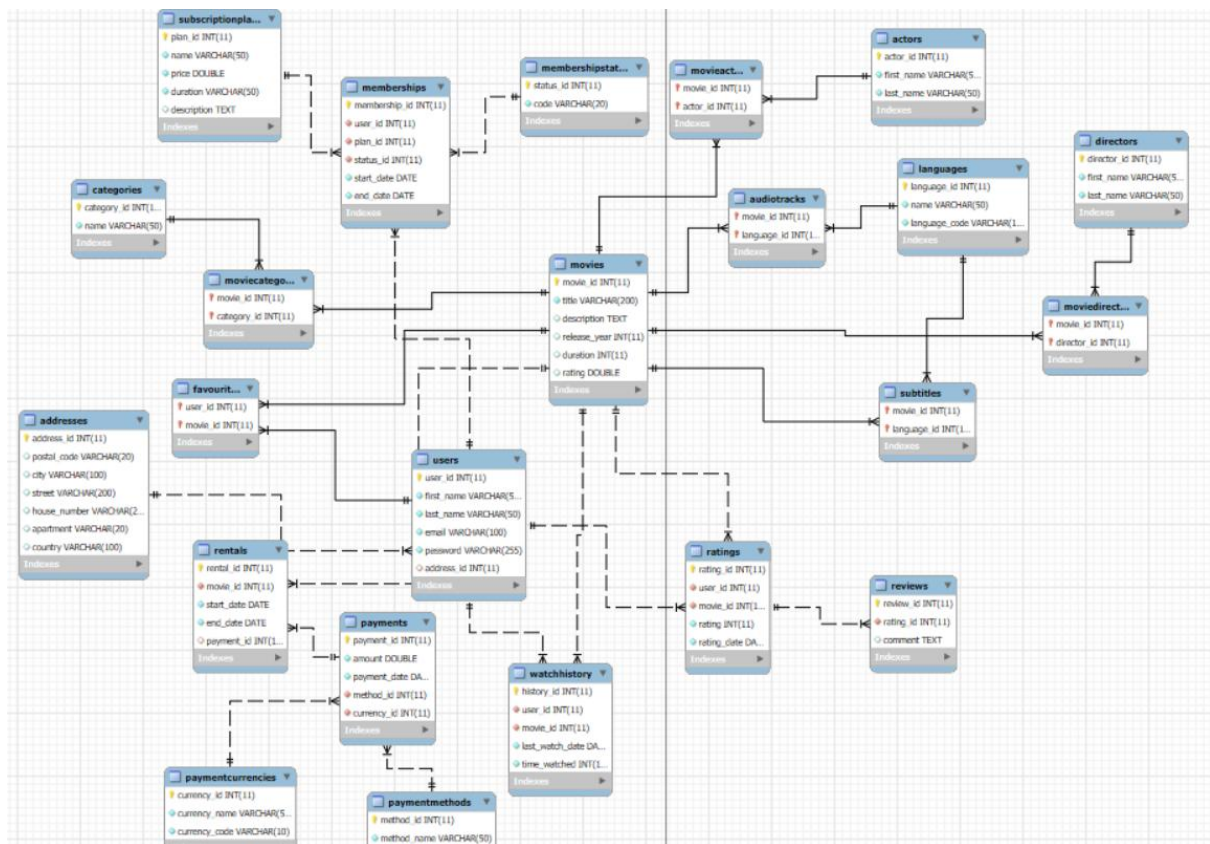
Przechowywanie historii oglądania filmów przez użytkowników platformy umożliwia śledzenie ich aktywności, co pozwala na lepsze dostosowanie oferty do ich preferencji oraz analizę obecnych trendów.

## 2. Wyodrębnienie niezbędnych tabel

Baza danych składa się z 23 tabel, które tworzą spójny system zarządzania treściami, użytkownikami i płatnościami:

- PaymentCurrencies - przechowuje obsługiwane waluty
- PaymentMethods - zawiera dostępne metody płatności
- Categories - przechowuje różne gatunki filmowe
- SubscriptionPlans – zawiera dostępne plany subskrypcji dla użytkowników
- MembershipStatus - definiuje statusy subskrypcji
- Languages - przechowuje dostępne języki w systemie
- Addresses - zawiera szczegółowe informacje o adresach użytkowników
- Users - przechowuje dane osobowe, dane logowania użytkowników oraz powiązanie z adresami
- Actors - zawiera dane o aktorach
- Directors - przechowuje dane o reżyserach
- Movies - zawiera dane o filmach dostępnych w serwisie
- Subtitles - przechowuje napisy do filmów w różnych językach
- AudioTracks - zawiera dubbing do filmów w różnych językach
- Favourites - przechowuje ulubione filmy użytkowników
- Memberships - zawiera dane o członkostwie użytkownika, dacie rozpoczęcia i zakończenia subskrypcji oraz statusie
- Ratings - przechowuje oceny filmów (1-5)
- Reviews - zawiera recenzje filmów
- WatchHistory – przechowuje historię oglądanych filmów przez użytkownika.
- Payments – zawiera informacje o dokonanych płatnościach
- Rentals - przechowuje informacje o wypożyczonych filmach
- MovieCategory – zawiera filmy z określonych kategorii
- MovieActor - przechowuje aktorów grających w danych filmach
- MovieDirector – zawiera reżyserów określonych filmów

### 3. Diagram bazy danych



Rysunek 1. Diagram bazy danych dla projektu wypożyczalni filmów

## 4. Wypełnienie bazy danymi

Baza została wypełniona przykładowymi danymi, w załączniku do pracy został zamieszczony kod zawierający zapytania INSERT.

Poniżej przedstawiono przykładowe polecenia wypełniające bazę danymi:

```
1  ✓ INSERT INTO SubscriptionPlans
2    (plan_id, name, price, duration, description)
3    VALUES
4    (1, 'Basic', 9.99, '1 month', 'Basic streaming plan with standard definition'),
5    (2, 'Standard', 14.99, '1 month', 'HD streaming on two devices'),
6    (3, 'Premium', 19.99, '1 month', 'Ultra HD streaming on four devices'),
7    (4, 'Annual Basic', 99.99, '1 year', 'Basic plan with annual discount'),
8    (5, 'Student', 7.99, '1 month', 'Discounted plan for students'),
9    (6, 'Family', 24.99, '1 month', 'Up to 5 profiles, family sharing');
```

*Rysunek 2. Zapytanie INSERT wypełniające tabelę SubscriptionPlans*

```
1  ✓ INSERT INTO Users
2    (user_id, first_name, last_name, email, password, address_id)
3    VALUES
4    (1, 'Jan', 'Kowalski', 'jan.kowalski@example.com', 'hashed_password_1', 1),
5    (2, 'Maria', 'Smith', 'maria.smith@example.com', 'hashed_password_2', 2),
6    (3, 'Pierre', 'Dubois', 'pierre.dubois@example.com', 'hashed_password_3', 3),
7    (4, 'Emily', 'Johnson', 'emily.johnson@example.com', 'hashed_password_4', 4),
8    (5, 'Anna', 'Nowak', 'anna.nowak@example.com', 'hashed_password_5', 1),
9    (6, 'John', 'Doe', 'john.doe@example.com', 'hashed_password_6', 2),
10   (7, 'Sophie', 'Martin', 'sophie.martin@example.com', 'hashed_password_7', 3),
11   (8, 'Michael', 'Brown', 'michael.brown@example.com', 'hashed_password_8', 4);
```

*Rysunek 3. Zapytanie INSERT wypełniające tabelę Users*

```
1  ✓ INSERT INTO Directors (director_id, first_name, last_name) VALUES
2    (1, 'Steven', 'Spielberg'),
3    (2, 'Christopher', 'Nolan'),
4    (3, 'Quentin', 'Tarantino'),
5    (4, 'Martin', 'Scorsese'),
6    (5, 'David', 'Fincher'),
7    (6, 'Greta', 'Gerwig'),
8    (7, 'Denis', 'Villeneuve'),
9    (8, 'Wes', 'Anderson');
```

*Rysunek 4. Zapytanie INSERT wypełniające tabelę Directors*

## 5. Prezentacja działania zapytań pobierających dane z bazy

### 1. 10 Najpopularniejszych filmów

```
1 SELECT
2     m.title,
3     COUNT(f.user_id) AS favourites_count
4 FROM movies m
5 JOIN favourites f ON m.movie_id = f.movie_id
6 GROUP BY m.movie_id, m.title
7 ORDER BY favourites_count DESC
8 LIMIT 10;
```

	title character varying (200)	favourites_count bigint
1	The Wolf of Wall Street	2
2	Pulp Fiction	2
3	Forrest Gump	2
4	Inception	2
5	Lady Bird	1
6	Parasite	1
7	Interstellar	1
8	Gone Girl	1
9	The Social Network	1
10	The Grand Budapest Hotel	1

### 2. Liczba użytkowników z aktywną subskrypcją

```
1 SELECT
2     sp.name AS subscription_plan,
3     COUNT(m.user_id) AS users_count
4 FROM memberships m
5 JOIN subscriptionplans sp ON m.plan_id = sp.plan_id
6 JOIN Membershipstatus ms ON m.status_id = ms.status_id
7 WHERE ms.code = 'Active'
8 GROUP BY sp.plan_id, sp.name
9 ORDER BY users_count DESC;
```

	subscription_plan character varying (50)	users_count bigint
1	Standard	2
2	Premium	2
3	Basic	1
4	Annual Basic	1
5	Student	1
6	Family	1

### 3. Liczba zarobków z subskrypcji z konkretnego miesiąca

```
1 SELECT
2     COALESCE(SUM(amount), 0) AS total_revenue
3 FROM payments
4 WHERE payment_date >= '2024-02-01' AND payment_date < '2024-03-01';
```

	total_revenue
	double precision
1	47.97

#### 4. Osoby, które obejrzały najwięcej filmów

```

1  SELECT
2      u.user_id,
3      u.first_name,
4      u.last_name,
5      COUNT(wh.history_id) AS watch_count
6  FROM users u
7  JOIN watchHistory wh ON u.user_id = wh.user_id
8  GROUP BY u.user_id, u.first_name, u.last_name
9  ORDER BY watch_count DESC
10 LIMIT 10;

```

	user_id	first_name	last_name	watch_count
	[PK] integer	character varying (50)	character varying (50)	bigint
1	8	Michael	Brown	1
2	7	Sophie	Martin	1
3	1	Jan	Kowalski	1
4	5	Anna	Nowak	1
5	4	Emily	Johnson	1
6	2	Maria	Smith	1
7	6	John	Doe	1
8	3	Pierre	Dubois	1

#### 5. Filmy z najwyższą średnią ocen (liczymy tylko te które mają ponad 10 ocen wystawionych)

```

1  SELECT
2      m.movie_id,
3      m.title,
4      AVG(r.rating) AS average_rating,
5      COUNT(r.rating_id) AS ratings_count
6  FROM movies m
7  JOIN ratings r ON m.movie_id = r.movie_id
8  GROUP BY m.movie_id, m.title
9  HAVING COUNT(r.rating_id) > 10
10 ORDER BY average_rating DESC
11 LIMIT 10;

```

	movie_id	title	average_rating	ratings_count
	[PK] integer	character varying (200)	numeric	bigint

#### 6. Najbardziej popularne kategorie bazując na czasie oglądania

```

1  SELECT
2      c.name AS category,
3      SUM(wh.time_watched) AS total_minutes_watched,
4      COUNT(DISTINCT wh.user_id) AS unique_viewers
5  FROM categories c
6  JOIN moviecategory mc ON c.category_id = mc.category_id
7  JOIN movies m ON mc.movie_id = m.movie_id
8  JOIN watchhistory wh ON m.movie_id = wh.movie_id
9  GROUP BY c.category_id, c.name
10 ORDER BY total_minutes_watched DESC;

```



	category character varying (50)	total_minutes_watched bigint	unique_viewers bigint
1	Drama	53820	6
2	Thriller	44160	5
3	Science Fiction	28380	3
4	Action	9240	1
5	Romance	8520	1

## 7. Zyski na podstawie metody płatności oraz walucie za ostatni kwartał

```

1 SELECT
2     pm.method_name,
3     pc.currency_name,
4     COUNT(p.payment_id) AS transactions_count,
5     SUM(p.amount) AS total_amount
6 FROM payments p
7 JOIN paymentmethods pm ON p.method_id = pm.method_id
8 JOIN paymentcurrencies pc ON p.currency_id = pc.currency_id
9 WHERE p.payment_date >= CURRENT_DATE - INTERVAL '3 months'
10 GROUP BY pm.method_name, pc.currency_name
11 ORDER BY total_amount DESC;

```

method_name character varying (50)	currency_name character varying (50)	transactions_count bigint	total_amount double precision
---------------------------------------	---	------------------------------	----------------------------------

## 8. Rozkład użytkowników na podstawie kraju pochodzenia

```

1 SELECT
2     a.country,
3     COUNT(DISTINCT u.user_id) AS users_count,
4     COUNT(DISTINCT wh.movie_id) AS movies_watched,
5     AVG(wh.time_watched) AS avg_watch_time_minutes
6 FROM users u
7 JOIN addresses a ON u.address_id = a.address_id
8 LEFT JOIN watchhistory wh ON u.user_id = wh.user_id
9 GROUP BY a.country
10 ORDER BY users_count DESC;

```

country character varying (100)	users_count bigint	movies_watched bigint	avg_watch_time_minutes numeric
1 United States	4	4	8700.0000000000000000
2 France	2	2	9360.0000000000000000
3 Poland	2	2	9270.0000000000000000

## 9. Reżyserowie których filmy cieszą się największym zainteresowaniem

```

1 SELECT
2     d.director_id,
3     CONCAT(d.first_name, ' ', d.last_name) AS director_name,
4     COUNT(DISTINCT m.movie_id) AS movies_count,
5     AVG(m.rating) AS avg_movie_rating,
6     COUNT(DISTINCT wh.user_id) AS total_viewers,
7     SUM(wh.time_watched) AS total_watch_minutes
8 FROM directors d
9 JOIN moviedirector md ON d.director_id = md.director_id
10 JOIN movies m ON md.movie_id = m.movie_id
11 LEFT JOIN WatchHistory wh ON m.movie_id = wh.movie_id
12 GROUP BY d.director_id, d.first_name, d.last_name
13 ORDER BY total_viewers DESC
14 LIMIT 10;

```

	director_id [PK] integer	director_name text	movies_count bigint	avg_movie_rating numeric	total_viewers bigint	total_watch_minutes bigint
1	3	Quentin Tarantino	4	8.7500000000000000	3	27960
2	2	Christopher Nolan	2	8.5000000000000000	2	18540
3	5	David Fincher	2	8.0000000000000000	1	7200
4	1	Steven Spielberg	1	9.0000000000000000	1	8520
5	7	Denis Villeneuve	1	8.0000000000000000	1	9840
6	8	Wes Anderson	1	8.0000000000000000	0	[null]
7	6	Greta Gerwig	1	8.0000000000000000	0	[null]

## 10. Najczęściej wypożyczane filmy z ostatniego kwartału

```

1  SELECT
2      m.movie_id,
3      m.title,
4      COUNT(r.rental_id) AS rental_count
5  FROM movies m
6  JOIN rentals r ON m.movie_id = r.movie_id
7  GROUP BY m.movie_id, m.title
8  ORDER BY rental_count DESC
9  LIMIT 10;

```

	movie_id [PK] integer	title character varying (200)	rental_count bigint
1	8	Blade Runner 2049	1
2	7	Parasite	1
3	1	Inception	1
4	5	Interstellar	1
5	4	Pulp Fiction	1
6	2	Forrest Gump	1
7	6	The Social Network	1
8	3	The Wolf of Wall Street	1

## 11. Filmy z napisami po angielsku i niemiecku

```

1  SELECT
2      m.movie_id,
3      m.title,
4      m.release_year
5  FROM movies m
6  JOIN subtitles s1 ON m.movie_id = s1.movie_id
7  JOIN languages l1 ON s1.language_id = l1.language_id
8  JOIN subtitles s2 ON m.movie_id = s2.movie_id
9  JOIN languages l2 ON s2.language_id = l2.language_id
10 WHERE l1.name = 'English' AND l2.name = 'German'
11 ORDER BY m.release_year DESC;

```

	movie_id [PK] integer	title character varying (200)	release_year integer
1	8	Blade Runner 2049	2017
2	11	The Grand Budapest Hotel	2014
3	2	Forrest Gump	1994
4	4	Pulp Fiction	1994

## 12. Najgorzej oceniane filmy

```

1  SELECT
2      m.movie_id,
3      m.title,
4      AVG(r.rating) AS average_rating,
5      COUNT(r.rating_id) AS rating_count
6  FROM movies m
7  JOIN ratings r ON m.movie_id = r.movie_id
8  GROUP BY m.movie_id, m.title
9  ORDER BY average_rating ASC
10 LIMIT 10;

```

	movie_id [PK] integer	title character varying (200)	average_rating numeric	rating_count bigint
1	8	Blade Runner 2049	7.0000000000000000	1
2	3	The Wolf of Wall Street	7.0000000000000000	1
3	1	Inception	8.0000000000000000	1
4	6	The Social Network	8.0000000000000000	1
5	5	Interstellar	9.0000000000000000	1
6	4	Pulp Fiction	9.0000000000000000	1
7	2	Forrest Gump	9.0000000000000000	1
8	7	Parasite	9.0000000000000000	1

13. Użytkownicy, których subskrypcja kończy się w ciągu najbliższego miesiąca

```

1  SELECT
2      u.user_id,
3      u.first_name,
4      u.last_name,
5      u.email,
6      m.end_date,
7      sp.name AS plan_name
8  FROM Users u
9  JOIN memberships m ON u.user_id = m.user_id
10 JOIN subscriptionplans sp ON m.plan_id = sp.plan_id
11 JOIN membershipstatus ms ON m.status_id = ms.status_id
12 WHERE ms.code = 'Active'
13 AND m.end_date BETWEEN CURRENT_DATE AND (CURRENT_DATE + INTERVAL '30 days')
14 ORDER BY m.end_date;

```

	user_id integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)	end_date date	plan_name character varying (50)
--	--------------------	--------------------------------------	-------------------------------------	----------------------------------	------------------	-------------------------------------

14. Aktorzy, którzy wystąpili w co najmniej 3 filmach

```

1  SELECT
2      a.actor_id,
3      a.first_name,
4      a.last_name,
5      COUNT(ma.movie_id) AS movie_count
6  FROM actors a
7  JOIN movieactor ma ON a.actor_id = ma.actor_id
8  GROUP BY a.actor_id, a.first_name, a.last_name
9  HAVING COUNT(ma.movie_id) >= 3
10 ORDER BY movie_count DESC;

```

	actor_id [PK] integer	first_name character varying (50)	last_name character varying (50)	movie_count bigint
1	3	Leonardo	DiCaprio	4

## 15. Najpopularniejsze filmy wśród użytkowników z polski

```

1  SELECT
2      m.movie_id,
3      m.title,
4      a.country,
5      COUNT(wh.history_id) AS watch_count
6  FROM movies m
7  JOIN watchhistory wh ON m.movie_id = wh.movie_id
8  JOIN users u ON wh.user_id = u.user_id
9  JOIN addresses a ON u.address_id = a.address_id
10 WHERE a.country = 'Poland'
11 GROUP BY m.movie_id, m.title, a.country
12 ORDER BY watch_count DESC
13 LIMIT 10;

```

	movie_id integer	title character varying (200)	country character varying (100)	watch_count bigint
1	1	Inception	Poland	1

## 16. Średnia ocena filmów po kategoriach

```

1  SELECT
2      c.name AS category_name,
3      COUNT(DISTINCT m.movie_id) AS movies_count,
4      AVG(r.rating) AS average_rating
5  FROM categories c
6  JOIN moviecategory mc ON c.category_id = mc.category_id
7  JOIN movies m ON mc.movie_id = m.movie_id
8  JOIN ratings r ON m.movie_id = r.movie_id
9  GROUP BY c.category_id, c.name
10 ORDER BY average_rating DESC;

```

	category_name character varying (50)	movies_count bigint	average_rating numeric
1	Action	1	9.0000000000000000
2	Romance	1	9.0000000000000000
3	Drama	6	8.5000000000000000
4	Science Fiction	3	8.0000000000000000
5	Thriller	5	7.8000000000000000

## 6. Prezentacja działania zapytań edytujących i usuwających dane

1. Zapytanie edytujące cenę planu subskrypcji o id = 1

```
1  ✓ UPDATE SubscriptionPlans SET price = 19.99  
2  WHERE plan_id = 1;
```

2. Zapytanie usuwające użytkownika o id = 5

```
1  DELETE FROM Users WHERE user_id = 5;
```

## 7. Zarządzanie użytkownikami, prezentacja działania użytkowników z różnymi zestawami uprawnień

```
1 CREATE ROLE admin_user WITH LOGIN PASSWORD 'admin';
2 CREATE ROLE regular_user WITH LOGIN PASSWORD 'user';
3 CREATE ROLE basic_user WITH LOGIN PASSWORD 'basic_user';
4
5 GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO admin_user;
6 GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO admin_user;
7
8 GRANT CONNECT ON DATABASE baza TO regular_user;
9 GRANT USAGE ON SCHEMA public TO regular_user;
10 GRANT SELECT ON Movies, Categories, SubscriptionPlans, MembershipStatus, Languages TO regular_user;
11 GRANT SELECT, INSERT ON Ratings, Reviews, Favourites, WatchHistory TO regular_user;
12 GRANT SELECT, UPDATE ON Users TO regular_user;
13
14 GRANT CONNECT ON DATABASE baza TO basic_user;
15 GRANT USAGE ON SCHEMA public TO basic_user;
16 GRANT SELECT ON Movies, Categories, SubscriptionPlans, MembershipStatus, Languages TO basic_user;
17 GRANT SELECT, INSERT ON Favourites, WatchHistory TO basic_user;
18 GRANT SELECT, UPDATE ON Users TO basic_user;
```

Rysunek 5. Stworzenie użytkowników i przypisanie im praw

Query Query History

---

1 set role admin\_user

---

Data Output Messages Notifications

---

SET

Query returned successfully in 68 msec.

Rysunek 6. Ustawienie roli admin\_user

Query Query History

---

1 SELECT \* from directors

---

Data Output Messages Notifications

---

	director_id [PK] integer	first_name character varying (50)	last_name character varying (50)
1	1	Steven	Spielberg
2	2	Christopher	Nolan
3	3	Quentin	Tarantino
4	4	Martin	Scorsese
5	5	David	Fincher
6	6	Greta	Gerwig
7	7	Denis	Villeneuve
8	8	Wes	Anderson

Rysunek 7. Dostęp do tabeli 'directors' (dostęp do wszystkich)

Query Query History

1 **set role** regular\_user

Data Output Messages Notifications

SET

Query returned successfully in 140 msec.

*Rysunek 8. Ustawienie roli 'regular user'*

Query Query History

1 **SELECT** \* **from** directors

Data Output Messages Notifications

ERROR: permission denied for table directors

*Rysunek 9. Brak uprawnień do tabeli 'directors'*

Query Query History

1 **set role** basic\_user

Data Output Messages Notifications

SET

Query returned successfully in 57 msec.

*Rysunek 10. Ustawienie roli 'basic\_user'*

Query Query History

1 **SELECT** \* **from** ratings

Data Output Messages Notifications

ERROR: permission denied for table ratings

*Rysunek 11. Brak uprawnień do tabeli 'ratings'*

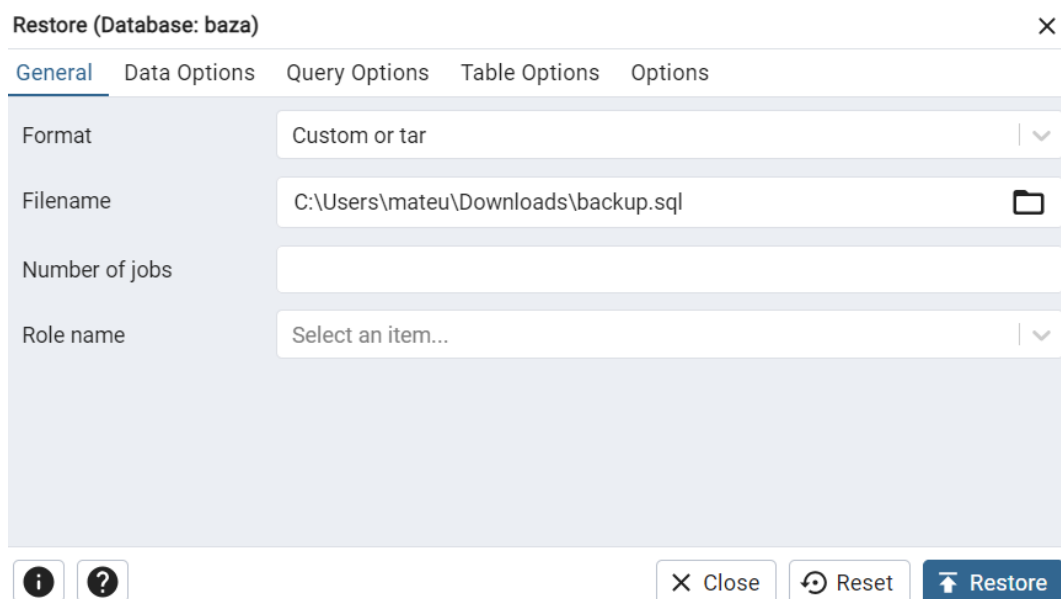
## 8. Kopia zapasowa, import i eksport

Kopię zapasową tworzymy za pomocą polecenia:

```
C:\Users\mateu>pg_dump -h localhost -p 5432 -U postgres -d test -F c -f backup.sql
```

*Rysunek 12. Stworzenie kopii zapasowej*

Funkcja `pg_dump` służy do tworzenia kopii zapasowych bazy danych PostgreSQL, następnie określamy hosta na którym działa serwer (w tym przypadku `localhost`, bo serwer bazy działa na kontenerze Docker) oraz port serwera (domyślnie 5432). Komenda `-U postgres` wskazuje użytkownika PostgreSQL, który ma uprawnienia do wykonania kopii zapasowej. Później wybieramy bazę, której kopię zapasową chcemy wykonać (w naszym przypadku baza ma nazwę „test”), ustawiamy format kopii na `custom` i określamy nazwę dla nowo utworzonej kopii `backup.sql`.



Restore (Database: baza) X

General Data Options Query Options Table Options Options

Format Custom or tar

Filename C:\Users\mateu\Downloads\backup.sql

Number of jobs

Role name Select an item...

Close Reset Restore

*Rysunek 13. Import z pliku .sql*



## 9. Przykład tworzenia wyzwalaczy

1. Przykład wyzwalacza, który usuwa automatycznie film z ulubionych filmów, jeżeli film został usunięty z platformy

```
1  CREATE OR REPLACE FUNCTION remove_favourite_function()
2  RETURNS TRIGGER AS $$
3  BEGIN
4      DELETE FROM favourites WHERE movie_id = OLD.movie_id;
5      RETURN OLD;
6  END;
7  $$ LANGUAGE plpgsql;
8
9  CREATE TRIGGER remove_favourite
10 AFTER DELETE ON movies
11 FOR EACH ROW
12 EXECUTE FUNCTION remove_favourite_function();
```

2. Przykład wyzwalacza, który automatycznie update'uje rating jak pojawi się nowa opinia o filmie

```
1  CREATE OR REPLACE FUNCTION update_movie_rating_function()
2  RETURNS TRIGGER AS $$
3  BEGIN
4      UPDATE movies
5      SET rating = (SELECT AVG(rating) FROM ratings WHERE movie_id = NEW.movie_id)
6      WHERE movie_id = NEW.movie_id;
7      RETURN NEW;
8  END;
9  $$ LANGUAGE plpgsql;
10
11 CREATE TRIGGER update_movie_rating
12 AFTER INSERT ON ratings
13 FOR EACH ROW
14 EXECUTE FUNCTION update_movie_rating_function();
```

3. Przykład wyzwalacza, który automatycznie zmienia status członkostwa po wygaśnięciu subskrypcji

```
1  CREATE OR REPLACE FUNCTION update_membership_status_function()
2  RETURNS TRIGGER AS $$
3  BEGIN
4      IF NEW.end_date < CURRENT_DATE THEN
5          UPDATE memberships
6          SET status_id = (SELECT status_id FROM membershipstatus WHERE code = 'Expired')
7          WHERE membership_id = NEW.membership_id;
8      END IF;
9      RETURN NEW;
10 END;
11 $$ LANGUAGE plpgsql;
12
13 CREATE TRIGGER update_membership_status
14 AFTER UPDATE ON memberships
15 FOR EACH ROW
16 EXECUTE FUNCTION update_membership_status_function();
```