

Wypożyczalnia filmów

Zespół: Tomasz Furgała, Konrad Tendaj, Łukasz Zegar

Spis treści

- [Aktorzy](#)
- [Diagram bazy danych](#)
- [Opis tabel](#)
 - [Clients](#)
 - [Reservation](#)
 - [Rental](#)
 - [Copy](#)
 - [Categories](#)
 - [Movies](#)
 - [Actors](#)
 - [Actors_in_movie](#)
- [Widoki](#)
 - [vw_available_copies](#)
 - [vw_currently_borrowed_copies](#)
 - [vw_current_reservations](#)
 - [vw_movie_popularity](#)
 - [vw_clients_delays_sum](#)
 - [vw_actor_rentals](#)
 - [vw_most_popular_actors_per_category](#)
 - [vw_movies_with_category](#)
- [Funkcje](#)
 - [f_get_client_reservations](#)
 - [f_is_copy_available](#)
 - [f_get_movies_by_category](#)
 - [f_check_client_exist](#)
 - [f_check_copy_exist](#)
 - [f_user_has_reservation](#)
 - [f_get_reservation_id](#)
 - [f_get_available_copies_for_movie_id](#)
 - [f_get_available_copies_for_movie_name](#)
- [Procedury](#)
 - [p_add_reservation](#)
 - [p_change_reservation_status](#)
 - [p_add_new_rental](#)
 - [p_return_rental](#)
 - [update_copy_availability](#)
 - [p_add_client](#)
 - [p_delete_client](#)
 - [p_update_client](#)
- [Triggery](#)
 - [t_copy_check_available](#)
 - [t_reservation_add](#)
 - [t_reservation_update](#)
 - [t_rental_add](#)
 - [t_rental_return](#)
 - [t_prevent_delete_client_with_rentals](#)
- [Backend](#)
 - [Połączenie z bazą danych](#)
 - [Główna aplikacja](#)

- Wykonywanie poleceń
- Uruchomienie
- Tabele
- Widoki
- Funkcje
- Procedury
- Działanie procedur
 - Anulowanie rezerwacji
 - Wypożyczenie filmu
 - Zwrot filmu do wypożyczalni
- Operacje CRUD
 - Dodanie nowego klienta
 - Aktualizacja klienta
 - Usunięcie klienta
 - Wyświetlenie pełnej listy klientów
- Pozostały kod użyty w projekcie

Aktorzy

1. Klient:

- klient może założyć konto, które umożliwia mu korzystanie z systemu,
- może składać rezerwacje na wybrany film,
- przegląda listę filmów oferowanych przez wypożyczalnię,
- przeglądanie listy obecnie zarezerwowanych i wypożyczonych przez niego filmów.

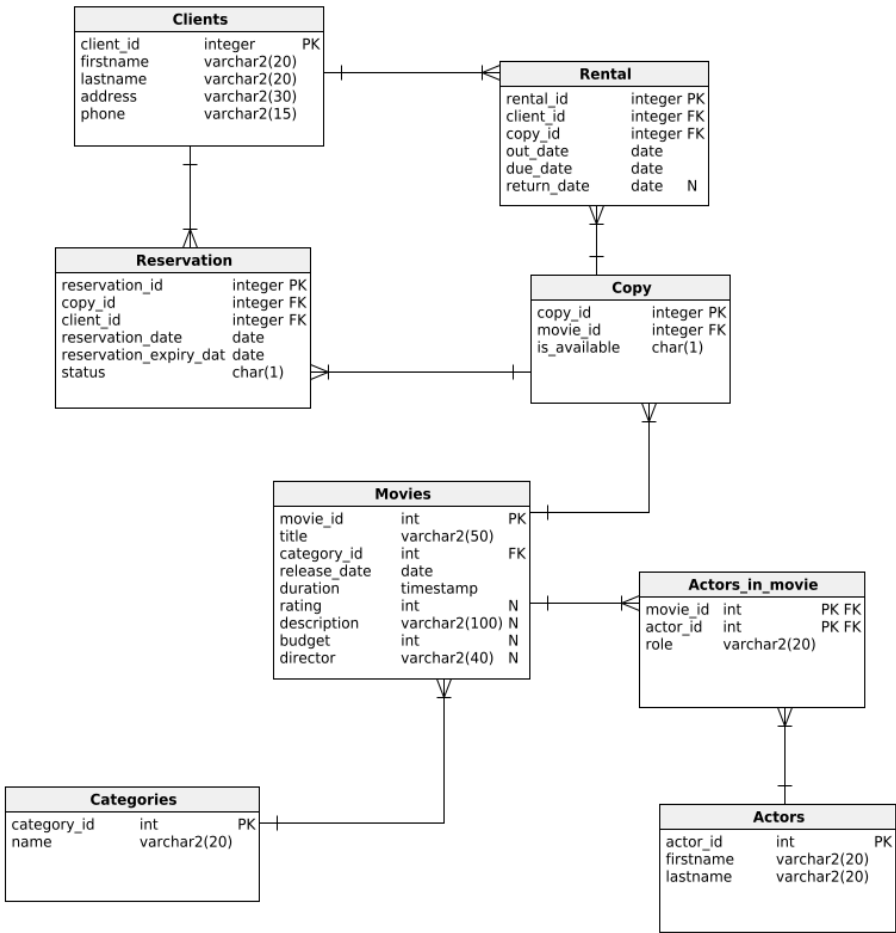
2. Pracownik:

- rejestruje wypożyczenie i zwracanie filmów w systemie,
- dodawanie nowych filmów i kopii do systemu,
- przegląda rezerwacje i wypożyczenia klientów,
- może generować raporty.

3. Administrator

- może edytować, dodawać, usuwać tabele,
- może edytować, dodawać, usuwać dane w tabelach,
- może generować raporty.

Diagram bazy danych



Opis tabel

Clients - tabela zarejestrowanych klientów wypożyczalni

- client_id - id klienta (to samo co login_id),
- firstname - imię,
- lastname - nazwisko,
- address - adres,
- phone - numer telefonu.

```
CREATE TABLE Clients (  
  client_id integer NOT NULL,  
  firstname varchar2(20) NOT NULL,  
  lastname varchar2(20) NOT NULL,  
  address_id integer NOT NULL,  
  phone varchar2(15) NOT NULL,  
  CONSTRAINT Clients_pk PRIMARY KEY (client_id)  
);
```

	CLIENT_ID	FIRSTNAME	LASTNAME	ADDRESS	PHONE
1	1	Sulibor	Dolata	Zamość 77-969 Jutrzenki 327	515-291-260
2	2	Ewa	Stowik	Katowice 10-093 Sosnowa 160	363-979-677
3	3	Ewaryst	Dworak	Siemianowice Śląskie 81-044 Rzemieniewicka 665	404-725-511
4	4	Felicja	Gruszk	Gorzów Wielkopolski 70-948 Freta 841	743-480-997
5	5	Amira	Piskorz	Sosnowiec 43-743 Cukiernicza 234	208-619-313
6	6	Stefania	Łukaszewicz	Bielsko-Biała Bielsko-Biała Lipowa 432	287-333-193
7	7	Judyta	Doliński	Wrocław 02-295 Akacyja 291	415-425-630
8	8	Bernadeta	Kulesza	Gorzów Wielkopolski 45-608 Żytnia 231	604-813-038
9	9	Józefina	Perkowski	Sopot 25-345 Stowackiego 69	583-420-921

Reservation - tabela z rezerwacjami filmów do przyszłego wypożyczenia

- reservation_id - id rezerwacji,
- copy_id - id egzemplarza filmu,
- client_id - id klienta,
- reservation_date - data rezerwacji,
- reservation_expiry_date - data wygaśnięcia rezerwacji,
- status - aktualny status rezerwacji,
 - N - nowa rezerwacja,
 - C - rezerwacja anulowana,
 - R - rezerwacja zrealizowana.

```
CREATE TABLE Reservation (  
  reservation_id integer NOT NULL,  
  copy_id integer NOT NULL,  
  client_id integer NOT NULL,  
  reservation_date date NOT NULL,  
  reservation_expiry_date date NOT NULL,  
  status char(1) NOT NULL,  
  CONSTRAINT Reservation_pk PRIMARY KEY (reservation_id)  
);  
ALTER TABLE Reservation ADD CONSTRAINT Reservation_Clients  
  FOREIGN KEY (client_id)  
  REFERENCES Clients (client_id);  
ALTER TABLE Reservation ADD CONSTRAINT Reservation_Copy  
  FOREIGN KEY (copy_id)  
  REFERENCES Copy (copy_id);
```

	RESERVATION_ID	COPY_ID	CLIENT_ID	RESERVATION_DATE	RESERVATION_EXPIRY_DATE	STATUS
1	1	1	1	2024-05-04 19:28:54	2024-05-14 19:28:54	N
2	2	8	2	2024-01-09	2024-01-16	N
3	3	22	15	2024-03-08	2024-03-12	R
4	4	31	11	2024-03-12	2024-03-17	C
5	5	10	20	2022-12-06	2022-12-12	C
6	6	7	6	2024-02-06	2024-02-19	R
7	7	10	6	2022-04-04	2022-04-13	R
8	8	11	20	2021-11-16	2021-11-23	R
9	9	1	4	2022-12-24	2022-12-28	C
10	10	11	1	2024-01-02	2024-01-05	N
11	11	5	14	2023-12-09	2023-12-16	R

Rental - tabela z informacjami o wypożyczeniach filmów przez klientów (aktualne oraz zwrócone)

- rental_id - id wypożyczenia,
- client_id - id klienta,
- copy_id - id wypożyczonego egzemplarza,
- out_date - data wypożyczenia filmu,
- due_date - okres, na który film został wypożyczony,
- return_date - data faktycznego zwrotu filmu (null jeśli nie zwrócony).

```
CREATE TABLE Rental (  
  rental_id integer NOT NULL,  
  client_id integer NOT NULL,  
  copy_id integer NOT NULL,  
  out_date date NOT NULL,  
  due_date date NOT NULL,  
  return_date date NULL default NULL,  
  CONSTRAINT Rental_pk PRIMARY KEY (rental_id)  
);  
ALTER TABLE Rental ADD CONSTRAINT Copy_Rental  
  FOREIGN KEY (copy_id)  
  REFERENCES Copy (copy_id);
```




```
ALTER TABLE Rental ADD CONSTRAINT Rental_Clients
FOREIGN KEY (client_id)
REFERENCES Clients (client_id);
```

	RENTAL_ID	CLIENT_ID	COPY_ID	OUT_DATE	DUE_DATE	RETURN_DATE
1	1	2	38	2024-05-04	2024-05-07	2024-05-14
2	2	13	12	2022-06-02	2022-06-26	2022-06-23
3	3	7	5	2021-10-14	2021-11-09	2021-11-09
4	4	17	26	2024-02-13	2024-02-19	2024-02-26
5	5	13	33	2022-07-24	2022-08-13	2022-08-11
6	6	15	37	2024-05-01	2024-05-04	<null>
7	7	1	21	2024-02-06	2024-02-09	2024-02-08
8	8	8	20	2023-07-05	2023-07-09	2023-07-27
9	9	8	13	2024-05-11	2024-05-25	<null>
10	10	12	21	2023-01-01	2023-01-15	2023-02-03
11	11	7	40	2023-03-08	2023-03-16	<null>
12	12	20	29	2021-09-25	2021-10-12	2021-11-05
13	13	20	15	2022-12-05	2022-12-13	2022-12-08
14	14	2	1	2021-12-18	2022-01-11	2022-02-01
15	15	11	14	2023-05-09	2023-06-04	2023-05-15
16	16	9	35	2021-12-09	2021-12-30	2021-12-30
17	17	12	11	2023-08-02	2023-09-01	2023-08-29
18	18	18	25	2024-04-01	2024-05-01	<null>
19	19	16	30	2024-01-19	2024-02-09	2024-02-07
20	20	15	27	2023-03-08	2023-03-24	2023-03-12

Copy - tabela fizycznych kopii danego filmu

- copy_id - id danej kopii,
- movie_id - id jej filmu,
- is_available - czy wypożyczona lub zarezerwowana ("Y", jeśli jest dostępna, "N" jeśli nie).



```
CREATE TABLE Copy (
  copy_id integer NOT NULL,
  movie_id integer NOT NULL,
  is_available char(1) NOT NULL,
  CONSTRAINT Copy_pk PRIMARY KEY (copy_id)
);
ALTER TABLE Copy ADD CONSTRAINT Copy_Movies
FOREIGN KEY (movie_id)
REFERENCES Movies (movie_id);
```

	 COPY_ID	÷	 MOVIE_ID	÷	 IS_AVAILABLE	÷
1	1		1		N	
2	2		8		Y	
3	3		10		Y	
4	4		5		Y	
5	5		3		N	
6	6		2		Y	
7	7		3		Y	
8	8		9		N	
9	9		5		Y	
10	10		8		Y	
11	11		1		N	
12	12		3		N	
13	13		6		N	
14	14		5		N	
15	15		8		N	
16	16		1		Y	
17	17		9		Y	

Categories - tabela kategorii filmów

- category_id - id kategorii,
- name - nazwa kategorii.

```
CREATE TABLE Categories (  
  category_id int NOT NULL,  
  name varchar2(20) NOT NULL,  
  CONSTRAINT Categories_pk PRIMARY KEY (category_id)  
);
```

	 CATEGORY_ID	÷	 NAME	÷
1	1		akcja	
2	2		animacja	
3	3		anime	
4	4		biograficzny	
5	5		dokumentalny	
6	6		dramat	
7	7		familijny	
8	8		fantasy	
9	9		gangsterski	
10	10		historyczny	

Movies - tabela zawierająca informacje o filmach

- movie_id - id filmu,
- name - nazwa filmu,
- title - tytuł filmu,
- category_id - id głównej kategorii filmu,
- release_date - data globalna wydania filmu,
- duration - czas trwania filmu,
- rating - ocena filmu w skali 1 do 10,

- description - krótki opis filmu,
- production_country - kraj produkcji,
- director - imię i nazwisko reżysera.

```
CREATE TABLE Movies (  
  movie_id int NOT NULL,  
  title varchar2(50) NOT NULL,  
  category_id int NOT NULL,  
  release_date date NOT NULL,  
  duration timestamp NOT NULL,  
  rating int NULL,  
  description varchar2(100) NULL,  
  budget int NULL,  
  director varchar2(40) NULL,  
  CONSTRAINT Movies_pk PRIMARY KEY (movie_id)  
);  
ALTER TABLE Movies ADD CONSTRAINT Movies_Categories  
  FOREIGN KEY (category_id)  
  REFERENCES Categories (category_id);
```

MOVIE_ID	TITLE	CATEGORY_ID	RELEASE_DATE	DURATION	RATING	DESCRIPTION
1	Challengers	29	2024-04-24	131	6.9	Tashi, była gwiazda tenisa, zostaje trenerką swojego męża, który at...
2	Civil War	1	2024-03-14	109	7.4	Przez pogrążone w krwawej wojnie domowej Stany Zjednoczone przedzi...
3	Dune: Part II	26	2024-02-28	166	8.4	Księżę Paul Atryda przyjmuje przydomek Muad'Dib i rozpoczyna duchow...
4	Nasze magiczne Encanto	2	2021-11-24	99	7.2	Dziewczynka pochodzi z kolumbijskiej rodziny, która obdarzona jest...
5	Piraci z Karaibów: Kłątwa Czarnej Perły	21	2003-06-28	143	7.7	Kowal Will Turner sprzymierza się z kapitanem Jackiem Sparrowem, by...
6	Zaplątani	2	2010-11-24	96	7.6	Żyjąca na odludziu Roszpunka, której włosy mają magiczną moc, marzi...
7	Uncharted	21	2022-02-10	116	6.1	Utalentowany złodziej Nathan Drake zostaje zwerbowany przez doświad...
8	To	11	2017-09-05	135	6.7	Opowieść o bestii karmiącej się dziecięcym strachem.
9	Tytanic	13	1997-11-01	194	7.3	Rok 1912, brytyjski statek Titanic wyrusza w swój dziewiczy rejs d...
10	Shrek	7	2001-04-22	90	7.8	By odzyskać swój dom, brzydki ogr z gadatliwym osłem wyruszają uwo...

Actors - tabela aktorów

- actor_id - id aktora,
- firstname - imię aktora,
- lastname - nazwisko aktora.

```
CREATE TABLE Actors (  
  actor_id int NOT NULL,  
  firstname varchar2(20) NOT NULL,  
  lastname varchar2(20) NOT NULL,  
  CONSTRAINT Actors_pk PRIMARY KEY (actor_id)  
);
```

ACTOR_ID	FIRSTNAME	LASTNAME
1	Zendaya	Stoermer Coleman
2	Josh	OConnor
3	Mike	Faist
4	Kirsten	Dunst
5	Cailee	Spaeny
6	Wagner	Moura
7	Stephen	Henderson
8	Nick	Offerman
9	Jefferson	White
10	Timothee	Chalamet
11	Rebecca	Ferguson
12	Javier	Bardem

Actors_in_movie - tabela łącząca aktora z filmem (do relacji wiele do wielu)

- movie_id - id filmu,
- actor_id - id aktora,
- role - rola aktora w filmie (jaką postać gra).

```
CREATE TABLE Actors_in_movie (  
    movie_id int NOT NULL,  
    actor_id int NOT NULL,  
    role varchar2(20) NOT NULL,  
    CONSTRAINT Actors_in_movie_pk PRIMARY KEY (actor_id,movie_id)  
);  
ALTER TABLE Actors_in_movie ADD CONSTRAINT Actors_in_movie_Actors  
    FOREIGN KEY (actor_id)  
    REFERENCES Actors (actor_id);  
ALTER TABLE Actors_in_movie ADD CONSTRAINT Actors_in_movie_Movies  
    FOREIGN KEY (movie_id)  
    REFERENCES Movies (movie_id);
```

	MOVIE_ID	÷	ACTOR_ID	÷	ROLE	÷
1			1		1 Tashi	
2			1		2 Patrick	
3			1		3 Faist	
4			2		4 Lee	
5			2		5 Jessie	
6			2		6 Joel	
7			2		7 Sammy	
8			2		8 President	
9			2		9 Dave	
10			3		10 Paul Atryda	
11			3		1 Chani	
12			3		11 Jessica Atryda	







Widoki

vw_available_copies

Widok dostępnych kopii filmów wyświetla listę dostępnych kopii filmów wraz z ich szczegółami, takimi jak id filmu, id kopii, tytuł filmu, kategorię, data wydania oraz czas trwania.

```
create or replace view VW_AVAILABLE_COPIES  
AS  
    SELECT m.MOVIE_ID, c.COPY_ID, m.TITLE,  
    cat.NAME AS category_name, m.RELEASE_DATE, m.DURATION  
    FROM Copy c  
    JOIN Movies m ON c.movie_id = m.movie_id  
    JOIN Categories cat ON m.category_id = cat.category_id  
    WHERE c.is_available = 'Y'
```

```
select * from vw_available_copies;
```



	 COPY_ID	÷	 MOVIE_TITLE	÷	 CATEGORY_NAME	÷	 RELEASE_DATE	÷	 DURATION	÷	 IS_AVAILABLE	÷
1	2		To		horror		2017-09-05		135		Y	
2	3		Shrek		familijny		2001-04-22		90		Y	
3	4		Piraci z Karaibów: Klątwa Czarnej Perły		przygodowy		2003-06-28		143		Y	
4	5		Diune: Part II		science-fiction		2024-02-28		166		Y	
5	6		Civil War		akcja		2024-03-14		109		Y	
6	7		Diune: Part II		science-fiction		2024-02-28		166		Y	
7	8		Tytanic		katastroficzny		1997-11-01		194		Y	
8	9		Piraci z Karaibów: Klątwa Czarnej Perły		przygodowy		2003-06-28		143		Y	
9	10		To		horror		2017-09-05		135		Y	
10	11		Challengers		sportowy		2024-04-24		131		Y	
11	12		Diune: Part II		science-fiction		2024-02-28		166		Y	
12	13		Zapłątani		animacja		2010-11-24		96		Y	
13	14		Piraci z Karaibów: Klątwa Czarnej Perły		przygodowy		2003-06-28		143		Y	

vw_currently_borrowed_copies

Widok aktualnie wypożyczonych kopii z informacjami o kliencie, filmie oraz dniami opóźnienia lub 0 gdy nie minął termin zwrotu. Dane posortowane są w kolejności malejącej liczby dni opóźnienia zwrotu.

```
CREATE OR REPLACE VIEW vw_currently_borrowed_copies AS
SELECT
    r.CLIENT_ID,
    c1.FIRSTNAME || ' ' || c1.LASTNAME AS Name,
    r.COPY_ID,
    m.TITLE,
    r.OUT_DATE,
    r.DUE_DATE,
    GREATEST(TRUNC(SYSDATE) - TRUNC(r.DUE_DATE), 0) AS Days_of_delay
FROM
    RENTAL r
JOIN
    COPY c ON c.COPY_ID = r.COPY_ID
JOIN
    MOVIES m ON m.MOVIE_ID = c.MOVIE_ID
JOIN
    CLIENTS c1 ON c1.CLIENT_ID = r.CLIENT_ID
WHERE
    RETURN_DATE IS NULL
ORDER BY
    Days_of_delay DESC;
```

```
SELECT * FROM vw_currently_borrowed_copies;
```

	 CLIENT_ID	÷	 NAME	÷	 COPY_ID	÷	 TITLE	÷	 OUT_DATE	÷	 DUE_DATE	÷	 DAYS_OF_DELAY	▼
1	7		Judyta Dołiński		40		Zapłątani		2023-03-08		2023-03-16		438	
2	18		Bojana Chmielewski		25		Shrek		2024-04-01		2024-05-01		26	
3	15		Ida Flis		37		Challengers		2024-05-01		2024-05-04		23	
4	8		Bernadeta Kulesza		13		Zapłątani		2024-05-11		2024-05-25		2	

vw_current_reservations

Widok rezerwacji aktualnych klientów pokazuje rezerwacje aktualnych klientów wraz z danymi klientów, filmami, na które zarezerwowali kopie, datami rezerwacji itp.

```
CREATE VIEW vw_current_reservations AS
SELECT r.reservation_id,
    c.client_id,
    c.firstname,
    c.lastname,
```

```
m.title AS movie_title,
r.reservation_date,
r.reservation_expiry_date
FROM Reservation r
JOIN Clients c ON r.client_id = c.client_id
JOIN Copy co ON r.copy_id = co.copy_id
JOIN Movies m ON co.movie_id = m.movie_id
WHERE r.status = 'R';
```

```
select * from vw_current_reservations;
```

	RES...	CLIENT_ID	FIRSTNAME	LASTNAME	MOVIE_TITLE	RESERVATION_DATE	RESERVATION_EXPIRY
1	8	20	Jasława	Ziółkowski	Challengers	2021-11-16	2021-11-23
2	11	14	Paula	Nowakowski	Diune: Part II	2023-12-09	2023-12-16
3	6	6	Stefania	Łukaszewicz	Diune: Part II	2024-02-06	2024-02-19
4	7	6	Stefania	Łukaszewicz	To	2022-04-04	2022-04-13
5	3	15	Ida	Flis	Tytanic	2024-03-08	2024-03-12

vw_movie_popularity

Widok pokazujący filmy od najpopularniejszego do najmniej popularnego względem sumarycznej liczby wypożyczeń jego kopii.

```
CREATE VIEW vw_movie_popularity AS
SELECT m.movie_id,
       m.title AS movie_title,
       COUNT(r.rental_id) AS num_rentals
FROM Movies m
LEFT JOIN Copy c ON m.movie_id = c.movie_id
LEFT JOIN Rental r ON c.copy_id = r.copy_id
GROUP BY m.movie_id, m.title
ORDER BY COUNT(r.rental_id) DESC;
```

```
select * from vw_movie_popularity;
```

	MOVIE_ID	MOVIE_TITLE	NUM_RENTALS
1	10	Shrek	6
2	1	Challengers	4
3	8	To	3
4	3	Diune: Part II	3
5	5	Piraci z Karaibów: Klątwa Czarnej Perły	2
6	6	Zapłątani	2
7	4	Nasze magiczne Encanto	0
8	9	Tytanic	0
9	2	Civil War	0
10	7	Uncharted	0

vw_clients_delays_sum

Widok pokazujący klientów i ich sumę spóźnień w oddawaniu filmów względem aktualnie wypożyczonych oraz tych już oddanych z opóźnieniem. Wyniki posortowane są od tych klientów z największą liczbą dni.

```
CREATE OR REPLACE VIEW vw_clients_delays_sum AS
SELECT
  cl.CLIENT_ID,
  cl.FIRSTNAME || ' ' || c1.LASTNAME AS Name,
  SUM(
    CASE
      WHEN r.RETURN_DATE IS NULL THEN
        GREATEST(TRUNC(SYSDATE) - TRUNC(r.DUE_DATE), 0)
      ELSE
        GREATEST(TRUNC(r.RETURN_DATE) - TRUNC(r.DUE_DATE), 0)
    END
  ) AS Total_days_of_delay
FROM
  RENTAL r
JOIN
  CLIENTS cl ON cl.CLIENT_ID = r.CLIENT_ID
GROUP BY
  cl.CLIENT_ID,
  cl.FIRSTNAME,
  cl.LASTNAME
ORDER BY
  Total_days_of_delay DESC;
```

```
SELECT * FROM vw_clients_delays_sum;
```

	CLIENT_ID	NAME	TOTAL_DAYS_OF_DELAY
1	7	Judyta Dołiński	438
2	2	Ewa Słowik	28
3	18	Bojana Chmielewski	26
4	20	Jasława Ziółkowski	24
5	15	Ida Flis	23
6	8	Bernadeta Kułesza	20
7	12	Baltazar Andrzejczak	19
8	17	Helga Chmiel	7
9	16	Antoni Sidor	0
10	13	Jaromira Kawecki	0
11	9	Józefina Perkowski	0
12	1	Sulibor Dołata	0
13	11	Eugeniusz Stępniews...	0

vw_actor_rentals

Widok przedstawiający listę aktorów występujących w obecnie wypożyczonych filmach oraz liczbę filmów, w których każdy aktor wystąpił.

```
CREATE OR REPLACE VIEW vw_actor_rentals AS
SELECT a.actor_id,
  a.firstname,
  a.lastname,
  COUNT(*) AS num_movies
FROM Actors a
JOIN Actors_in_movie aim ON a.actor_id = aim.actor_id
JOIN Movies m ON aim.movie_id = m.movie_id
JOIN Copy c ON m.movie_id = c.movie_id
```

```
JOIN Rental r ON c.copy_id = r.copy_id
GROUP BY a.actor_id, a.firstname, a.lastname;
```

```
SELECT * FROM VW_ACTOR_RENTALS;
```

	🔍 ACTOR_ID ↕	🔍 FIRSTNAME ↕	🔍 LASTNAME ↕	🔍 NUM_MOVIES ↕
1	10	Timothee	Chalamet	3
2	26	Ron	Perlman	2
3	40	Cameron	Diaz	6
4	33	Jaeden	Martell	3
5	39	Eddie	Murphy	6
6	11	Rebecca	Ferguson	3
7	1	Zendaya	Stoermer Coleman	7
8	3	Mike	Faist	4
9	13	Josh	Brolin	3
10	14	Austin	Butler	3
11	2	Josh	OConnor	4
12	23	Mandy	Moore	2
13	32	Bill	Skarsgard	3

vw_most_popular_actors_per_category

Widok przedstawia najpopularniejszego aktora występującego w filmach danej kategorii.
Widok zawiera kolumny o nazwie kategorii, najpopularniejszym aktorze, liczbie filmów z tym aktorem w danej kategorii.

```
CREATE OR REPLACE VIEW vw_most_popular_actors_per_category AS
SELECT category_name,
       actor_name,
       movie_count
FROM (
  SELECT c.name AS category_name,
         a.firstname || ' ' || a.lastname AS actor_name,
         COUNT(*) AS movie_count,
         ROW_NUMBER() OVER (PARTITION BY c.category_id ORDER BY COUNT(*) DESC) AS actor_rank
  FROM Categories c
  JOIN Movies m ON c.category_id = m.category_id
  JOIN Actors_in_movie aim ON m.movie_id = aim.movie_id
  JOIN Actors a ON aim.actor_id = a.actor_id
  GROUP BY c.category_id, c.name, a.firstname, a.lastname
)
WHERE actor_rank = 1;
```

```
SELECT * FROM vw_most_popular_actors_per_category;
```

	CATEGORY_NAME	ACTOR_NAME	MOVIE_COUNT
1	akcja	Jefferson White	1
2	animacja	Stephanie Beatriz	1
3	familijny	Cameron Diaz	1
4	horror	Jaeden Martell	1
5	katastroficzny	Leonardo DiCaprio	1
6	przygodowy	Johnny Depp	1
7	science-fiction	Javier Bardem	1
8	sportowy	Zendaya Stoermer Coleman	1

vw_movies_with_category

Widok wyświetla informacje o wszystkich filmach, które należą do wypożyczalni. Do tabeli `Movies` zamiast wyświetlać `Category_Id` pokazujemy nazwę tej kategorii.

```
CREATE OR REPLACE VIEW vw_movies_with_category AS
SELECT
    c.name AS category_name,
    m.title AS movie_name,
    m.description AS movie_description,
    m.release_date,
    m.duration,
    m.rating,
    m.director
FROM Movies m
JOIN Categories c ON m.category_id = c.category_id;
```

	CATEGORY_NAME	MOVIE_NAME	MOVIE_DESCRIPTION	RELEASE_DATE	DURATION	RATING	DIRECTOR
1	akcja	Civil War	Przez pogrążone w krwawej wojnie domowej Stany Zjednoczone przedziera...	2024-03-14	109	7.4	Alex Garland
2	animacja	Nasze magiczne Encanto	Dziewczynka pochodzi z kolumbijskiej rodziny, która obdarzona jest ma...	2021-11-24	99	7.2	Jared Bush
3	animacja	Zaplątani	Żyjąca na odludziu Roszpunka, której włosy mają magiczną moc, marzy, ...	2010-11-24	96	7.6	Nathan Greno
4	familijny	Shrek	By odzyskać swój dom, brzydki ogar z gadatliwym osłem wyruszają uwolni...	2001-04-22	90	7.8	Andrew Adamson
5	horror	To	Opowieść o bestii karmiącej się dziecięcym strachem.	2017-09-05	135	6.7	Andy Muschietti
6	katastroficzny	Titanic	Rok 1912, brytyjski statek Titanic wyrusza w swój dziewiczy rejs do U...	1997-11-01	194	7.3	James Cameron
7	przygodowy	Uncharted	Utalentowany złodziej Nathan Drake zostaje zwerbowany przez doświadcz...	2022-02-10	116	6.1	Ruben Fleischer
8	przygodowy	Piraci z Karaibów: Klątwa Czarnej Perty	Kowal Will Turner sprzymierza się z kapitanem Jackiem Sparrowem, by o...	2003-06-28	143	7.7	Gore Verbinski
9	science-fiction	Dune: Part II	Księżę Paul Atryda przyjmuje przydomek Muad'Dib i rozpoczyna duchowo-f...	2024-02-28	166	8.4	Denis Villeneuve
10	sportowy	Challengers	Tashi, była gwiazda tenisa, zostaje trenerką swojego męża, który aby ...	2024-04-24	131	6.9	Luca Guadagnino

Funkcje

f_get_client_reservations

Funkcja ta umożliwia pobranie listy rezerwacji dla określonej osoby na podstawie jej identyfikatora klienta. Zwraca informacje o identyfikatorze rezerwacji, tytule filmu, dacie rezerwacji, dacie wygaśnięcia rezerwacji i statusie rezerwacji, co ułatwia zarządzanie rezerwacjami klientów w systemie wypożyczalni filmów.

```
CREATE OR REPLACE FUNCTION f_get_client_reservations(client_id_input INT)
RETURN SYS_REFCURSOR
IS
    is_person INT;
    reservation_cursor SYS_REFCURSOR;
BEGIN
    -- Sprawdzamy czy istnieje taka osoba w systemie
    SELECT count(*) INTO is_person FROM CLIENTS
    WHERE CLIENT_ID = client_id_input;
    IF is_person < 1
    THEN RAISE_APPLICATION_ERROR(-20000, 'Nie ma takiego użytkownika');
    END IF;

    OPEN reservation_cursor FOR
        SELECT r.reservation_id,
            m.title AS movie_title,
```

```

        r.reservation_date,
        r.reservation_expiry_date,
        r.status
    FROM Reservation r
    JOIN Copy c ON r.copy_id = c.copy_id
    JOIN Movies m ON c.movie_id = m.movie_id
    WHERE r.client_id = client_id_input;

    RETURN reservation_cursor;
END;
```

```
select f_get_client_reservations(1) from dual;
```

	RES...	MOVIE_TITLE	RESERVATION_DATE	RESERVATION_EXPIRY_DATE	STATUS
1	1	Challengers	2024-05-04 19:28:54	2024-05-14 19:28:54	N
2	10	Challengers	2024-01-02	2024-01-05	N

```
select f_get_client_reservations(73) from dual;
```

```

[72000][20000]
ORA-20000: Nie ma takiego użytkownika
ORA-06512: przy "BD_416551.F_GET_CLIENT_RESERVATIONS", linia 11
Position: 7
```

f_is_copy_available

Funkcja umożliwia sprawdzenie statusu konkretnej kopii filmu na podstawie jej identyfikatora. Zwraca informacje o tym, czy kopia jest aktualnie zarezerwowana, wypożyczona, dostępna, lub czy nie istnieje w bazie danych. Jest to przydatne narzędzie do zarządzania dostępnością kopii filmów w systemie wypożyczalni.

```

CREATE OR REPLACE FUNCTION f_is_copy_available(copy_id_input INT)
RETURN BOOLEAN
IS
    is_copy_anvailable CHAR(1);
BEGIN
    -- Sprawdzenie, czy kopia jest dostępna w tabeli Copy
    SELECT IS_AVAILABLE
    INTO is_copy_anvailable
    FROM Copy
    WHERE copy_id = copy_id_input;

    IF is_copy_anvailable = 'Y' THEN RETURN TRUE;
    ELSE RETURN FALSE;
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END;
```

```

BEGIN
    IF f_is_copy_available(1) THEN
        DBMS_OUTPUT.PUT_LINE('Copy is available.');
```

```

    ELSE
        raise_application_error(-20001, 'Copy is not available.');
```

```
END IF;  
END;
```

```
BD_416551> BEGIN  
          IF f_is_copy_available(2) THEN  
              DBMS_OUTPUT.PUT_LINE('Copy is available.');          ELSE  
              raise_application_error(-20001, 'Copy is not available.');          END IF;  
        END;  
[2024-05-14 23:51:21] completed in 44 ms  
Copy is available.
```

Gdy podamy copy_id = -1 to otrzymamy błąd:

```
[72000][20001]  
ORA-20001: Copy is not available.  
ORA-06512: przy linia 5  
Position: 0
```

f_get_movies_by_category

Funkcja zwraca filmy należące do określonej kategorii na podstawie przekazanego identyfikatora kategorii. Zestawienie zawiera nazwę kategorii, tytuł filmu, opis filmu, datę premiery, czas trwania, ocenę, i reżysera.

```
CREATE OR REPLACE FUNCTION f_get_movies_by_category(category_id_input INT)  
RETURN SYS_REFCURSOR  
IS  
  is_category INT;  
  movie_cursor SYS_REFCURSOR;  
BEGIN  
  -- Sprawdzamy czy istnieje taka kategoria  
  SELECT count(*) INTO is_category FROM CATEGORIES  
  WHERE CATEGORY_ID = category_id_input;  
  IF is_category < 1  
  THEN RAISE_APPLICATION_ERROR(-20000, 'Nie ma takiej kategorii');  
  END IF;  
  
  OPEN movie_cursor FOR  
  SELECT  
    c.name AS category_name,  
    m.title AS movie_name,  
    m.description AS movie_description,  
    m.release_date,  
    m.duration,  
    m.rating,  
    m.director  
  FROM Movies m  
  JOIN Categories c ON m.category_id = c.category_id  
  WHERE c.category_id = category_id_input;  
  
  RETURN movie_cursor;  
END;
```

```
select f_get_movies_by_category(2) from dual;
```

	□ CATEGORY_NAME	□ MOVIE_NAME	□ MOVIE_DESCRIPTION	□ RELEASE_DATE	□ DURATION	□ RATING	□ DIRECTOR
1	animacja	Nasze magiczne Encanto	Dziewczynka pochodzi z kolumbijskiej rodziny, która obdarzona jest mag-	2021-11-24	99	7.2	Jared Bush
2	animacja	Zaplątani	Żyjąca na odludziu Roszpunka, której włosy mają magiczną moc, marzy, b-	2010-11-24	96	7.6	Nathan Greno

```
select f_get_movies_by_category(123) from dual;
```

```
[72000][20000]
```

```
ORA-20000: Nie ma takiej kategorii
```

```
ORA-06512: przy "BD_416551.F_GET_MOVIES_BY_CATEGORY", linia 11
```

```
Position: 7
```

f_check_client_exist

Funkcja ta sprawdza czy klient o podanym `client_id` istnieje w bazie danych. Funkcja zwraca odpowiednią wartość logiczną.

```
CREATE OR REPLACE FUNCTION f_check_client_exist(client_id_input INT)
RETURN BOOLEAN
IS
    client_count INT;
BEGIN
    -- Sprawdzenie, czy istnieje osoba o podanym ID w tabeli Clients
    SELECT COUNT(*)
    INTO client_count
    FROM Clients
    WHERE client_id = client_id_input;

    -- Jeśli liczba znalezionych rekordów jest większa od zera, to osoba istnieje
    IF client_count > 0 THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END;
```

```
BEGIN
    IF f_check_client_exist(-1) THEN
        DBMS_OUTPUT.PUT_LINE('Client exists.');
```

```
ELSE
```

```
    raise_application_error(-20001, 'Client does not exist.');
```

```
END IF;
```

```
END;
```

```
[72000][20001]
```

```
ORA-20001: Client does not exist.
```

```
ORA-06512: at line 5
```

```
Position: 0
```

f_check_copy_exist

Funkcja sprawdza czy podana kopia z takim `copy_id` istnieje, zwraca odpowiednią wartość logiczną.

```
CREATE OR REPLACE FUNCTION f_check_copy_exist(copy_id_input INT)
RETURN BOOLEAN
IS
    copy_count INT;
BEGIN
```



```

SELECT COUNT(*)
INTO copy_count
FROM Copy
WHERE copy_id = copy_id_input;

-- Jeśli liczba znalezionych rekordów jest większa od zera, to kopia istnieje
IF copy_count > 0 THEN
    RETURN TRUE;
ELSE
    RETURN FALSE;
END IF;
END;

```

```

BEGIN
    IF f_check_copy_exist(-1) THEN
        DBMS_OUTPUT.PUT_LINE('Copy exists.');
```

```

    ELSE
        raise_application_error(-20001, 'Copy does not exist.');
```

```

    END IF;
END;

```

```

[72000][20001]
ORA-20001: Client does not exist.
ORA-06512: at line 5
Position: 0

```

f_user_has_reservation

Funkcja sprawdza czy podany użytkownik złożył wcześniej rezerwację na konkretną kopię o podanym id. Zwraca odpowiednią wartość logiczną.

```

CREATE OR REPLACE FUNCTION f_user_has_reservation(
    user_id_input INT,
    copy_id_input INT
) RETURN BOOLEAN
IS
    v_count INT;
BEGIN
    SELECT COUNT(*)
    INTO v_count
    FROM Reservation
    WHERE CLIENT_ID = user_id_input
        AND copy_id = copy_id_input
        AND STATUS = 'N';

    RETURN v_count > 0;
EXCEPTION
    WHEN OTHERS THEN
        raise_application_error(-20001, 'Error checking reservation: ' || SQLERRM);
END;

```

	RESERVATION_ID	COPY_ID	CLIENT_ID	RESERVATION_DATE	RESERVATION_EXPIRY_DATE	STATUS
1	1	1	1	2024-05-04 19:28:54	2024-05-14 19:28:54	N

```

BEGIN
    IF f_user_has_reservation(1, 1) THEN
        DBMS_OUTPUT.PUT_LINE('User has reservation.');
```

```

    ELSE
        raise_application_error(-20001, 'Reservation does not exist.');
```

```

    END IF;
END;

```

Przy wywołaniu tej funkcji nie pojawia się żaden błąd więc można wnioskować, że działa poprawnie.

f_get_reservation_id

Funkcja odpowiedzialna za pobranie oraz zwrócenie wartości `reservation_id` istniejącej rezerwacji z tabeli `Reservation` podając przy tym id użytkownika oraz id kopii.

```
CREATE OR REPLACE FUNCTION f_get_reservation_id(
    user_id_input INT,
    copy_id_input INT
) RETURN INT
IS
    v_reservation_id INT;
BEGIN
    IF NOT F_USER_HAS_RESERVATION(user_id_input, copy_id_input) THEN
        raise_application_error(-20002, 'No such reservation.');
```

```
    END IF;

    SELECT reservation_id
    INTO v_reservation_id
    FROM Reservation
    WHERE CLIENT_ID = user_id_input
        AND COPY_ID = copy_id_input
        AND STATUS = 'N';

    RETURN v_reservation_id;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        raise_application_error(-20003, 'No matching reservation found.');
```

```
    WHEN OTHERS THEN
        raise_application_error(-20001, 'Error retrieving reservation_id: ' || SQLERRM);
END;
```

Użycie:

```
BD_416551> DECLARE
    user_id INT := 1;
    copy_id INT := 1;
    reservation_id INT;
    BEGIN
        BEGIN
            reservation_id := f_get_reservation_id(user_id, copy_id);
            DBMS_OUTPUT.PUT_LINE('Reservation ID: ' || reservation_id);
        EXCEPTION
            WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE(SQLERRM);
        END;
    END;
```

```
[2024-05-22 11:31:52] completed in 12 ms
```

f_get_available_copies_for_movie_id

Funkcja zwraca listę wszystkich dostępnych kopii filmu o podanym id. Funkcja zwraca wszystkie kolumny z widoku `vw_available_copies`, czyli kolejno:
id filmu, id kopii, tytuł filmu, kategoria, data wydania, czas trwania.

```
create or replace FUNCTION f_get_available_copies_for_movie_id(movie_id_input INT)
RETURN SYS_REFCURSOR
IS
```

```
is_movie INT;
movies_cursor SYS_REFCURSOR;
BEGIN
  -- Sprawdzamy czy istnieje taki film w systemie
  SELECT count(*) INTO is_movie FROM MOVIES
  WHERE MOVIE_ID = movie_id_input;
  IF is_movie < 1 THEN
    RAISE_APPLICATION_ERROR(-20000, 'Nie ma takiego filmu');
  END IF;

  OPEN movies_cursor FOR
    SELECT *
    FROM VW_AVAILABLE_COPIES c
    WHERE c.MOVIE_ID = movie_id_input;

  RETURN movies_cursor;
END;
```

```
select f_get_available_copies_for_movie_id(5) from dual;
```

	MOVIE_ID	COPY_ID	TITLE	NAME	RELEASE_DATE	DURATION
1	5	9	Piraci z Karaibów: Klątwa Czarnej Perły	przygodowy	2003-06-28	143
2	5	31	Piraci z Karaibów: Klątwa Czarnej Perły	przygodowy	2003-06-28	143
3	5	34	Piraci z Karaibów: Klątwa Czarnej Perły	przygodowy	2003-06-28	143

```
1 ! select f_get_available_copies_for_movie_id( MOVIE_ID_INPUT: -5) from dual;
```

[72000][20000]

ORA-20000: Nie ma takiego filmu

ORA-06512: przy "BD_416551.F_GET_AVAILABLE_COPIES_FOR_MOVIE_ID", linia 11

Position: 7

f_get_available_copies_for_movie_name

Funkcja zwraca listę wszystkich dostępnych kopii filmu o podobnej nazwie. Funkcja zwraca wszystkie kolumny z widoku

`vw_available_copies`, czyli kolejno:

id filmu, id kopii, tytuł filmu, kategoria, data wydania, czas trwania.

```
create or replace FUNCTION f_get_available_copies_for_movie_name(movie_name_input VARCHAR2)
RETURN SYS_REFCURSOR
IS
  is_movie INT;
  movies_cursor SYS_REFCURSOR;
BEGIN
  -- Sprawdzamy czy istnieje taki film w systemie (ignorując wielkość liter)
  SELECT count(*) INTO is_movie FROM MOVIES
  WHERE UPPER(TITLE) LIKE '%' || UPPER(movie_name_input) || '%';
  IF is_movie < 1 THEN
    RAISE_APPLICATION_ERROR(-20000, 'Nie ma takiego filmu o podobnym tytule');
  END IF;

  OPEN movies_cursor FOR
    SELECT *
    FROM VW_AVAILABLE_COPIES c
    WHERE UPPER(c.TITLE) LIKE '%' || UPPER(movie_name_input) || '%';

  RETURN movies_cursor;
END;
```

```
select F_GET_AVAILABLE_COPIES_FOR_MOVIE_NAME('piraci') from dual;
```

	MOVIE_ID	COPY_ID	TITLE	NAME	RELE...	DURATION
1	5	9	Piraci z Karaibów: Kł...	przygodowy	2003-06-28	143
2	5	31	Piraci z Karaibów: Kł...	przygodowy	2003-06-28	143
3	5	34	Piraci z Karaibów: Kł...	przygodowy	2003-06-28	143

```
1 ! select F_GET_AVAILABLE_COPIES_FOR_MOVIE_NAME( MOVIE_NAME_INPUT: 'Polski') from dual
[72000][20000]
ORA-20000: Nie ma takiego filmu o podobnym tytule
ORA-06512: przy "BD_416551.F_GET_AVAILABLE_COPIES_FOR_MOVIE_NAME", linia 12
Position: 7
```

Procedury

p_add_reservation

Procedura odpowiedzialna za dodanie nowej rezerwacji do tabeli **Reservations**.
Na początku sprawdzamy argument o długości wypożyczenia. Później obliczamy datę wygaśnięcia rezerwacji i wstawiamy te dane do tabeli.

```
CREATE OR REPLACE PROCEDURE p_add_reservation(
    client_id_input INT,
    copy_id_input INT,
    rental_duration_input INT
)
IS
    reservation_date_input DATE := SYSDATE;
    reservation_expiry_date_input DATE;
BEGIN
    -- Sprawdzenie, czy podana długość wypożyczenia jest większa niż 0
    IF rental_duration_input <= 0 THEN
        raise_application_error(-20001, 'Rental duration must be greater than 0.');
```

Użycie:

```
BEGIN
    P_ADD_RESERVATION(1, 1, 10);
END;
```

	RESERVATION_ID	COPY_ID	CLIENT_ID	RESERVATION_DATE	RESERVATION_EXPIRY_DATE	STATUS
1	1	1	1	2024-05-04 19:28:54	2024-05-14 19:28:54	N

p_change_reservation_status

Procedura odpowiedzialna za zmianę statusu rezerwacji. Jeżeli nowy status jest **C** (cancelled) to zmienia pole **is_available** w tabeli **Copy** na **Y** (dostępna), w przeciwnym przypadku ustawia tą wartość na **N** (niedostępna).

```
CREATE OR REPLACE PROCEDURE p_change_reservation_status(
    reservation_id_input INT,
    new_status_input CHAR
)
IS
    reservation_exists INT;
BEGIN
    -- Sprawdzenie, czy podane reservation_id istnieje
    SELECT COUNT(*)
    INTO reservation_exists
    FROM Reservation
    WHERE reservation_id = reservation_id_input;

    IF reservation_exists = 0 THEN
        raise_application_error(-20001, 'Reservation with the given ID does not exist.');
```

```
    END IF;

    -- Aktualizacja statusu rezerwacji
    UPDATE Reservation
    SET status = new_status_input
    WHERE reservation_id = reservation_id_input;

    -- Aktualizacja stanu dostępności kopii w tabeli Copy
    IF new_status_input = 'C' THEN
        UPDATE Copy
        SET is_available = 'Y'
        WHERE copy_id IN (SELECT copy_id FROM Reservation WHERE reservation_id = reservation_id_input);
    ELSE
        UPDATE Copy
        SET is_available = 'N'
        WHERE copy_id IN (SELECT copy_id FROM Reservation WHERE reservation_id = reservation_id_input);
    END IF;

    DBMS_OUTPUT.PUT_LINE('Reservation status updated successfully.');
```

```
EXCEPTION
    WHEN OTHERS THEN
        raise_application_error(-20003, 'Error updating reservation status: ' || SQLERRM);
END;
```

	RESERVATION_ID	COPY_ID	CLIENT_ID	RESERVATION_DATE	RESERVATION_EXPIRY_DATE	STATUS
1	1	1	1	2024-05-04 19:28:54	2024-05-14 19:28:54	N

I teraz spróbujemy anulować rezerwację:

```
begin
    p_change_reservation_status(1, 'C');
end;
```

	RESERVATION_ID	COPY_ID	CLIENT_ID	RESERVATION_DATE	RESERVATION_EXPIRY_DATE	STATUS
1	1	1	1	2024-05-04 19:28:54	2024-05-14 19:28:54	C

I zmiana w tabeli **Copy**:

	COPY_ID	MOVIE_ID	IS_AVAILABLE
1	1	1	Y

p_add_new_rental

Procedura jest odpowiedzialna za dodawanie nowego wypożyczenia. Jeżeli użytkownik złożył wcześniej rezerwację to procedura zmienia status rezerwacji w tabeli *Reservation*. Jeśli użytkownik nie złożył rezerwacji to procedura zmienia status kopii na niedostępną w tabeli *Copy*.

```
CREATE OR REPLACE PROCEDURE p_add_new_rental(
    client_id_input INT,
    copy_id_input INT,
    rental_duration_input INT
)
IS
    reservation_date_input DATE := SYSDATE;
    reservation_expiry_date_input DATE;
    v_reservation_id INT;
BEGIN

    -- Sprawdzenie, czy podana długość wypożyczenia jest większa niż 0
    IF rental_duration_input <= 0 THEN
        raise_application_error(-20001, 'Rental duration must be greater than 0.');
```

```
    END IF;

    -- Obliczenie daty wygaśnięcia rezerwacji poprzez dodanie liczby dni do daty rezerwacji
    reservation_expiry_date_input := reservation_date_input + rental_duration_input;

    IF F_USER_HAS_RESERVATION(client_id_input, copy_id_input) THEN
        -- jeśli użytkownik złożył wcześniej rezerwacje to zmień status w tabeli Reservation
        v_reservation_id := F_GET_RESERVATION_ID(client_id_input, copy_id_input);
        P_CHANGE_RESERVATION_STATUS(v_reservation_id, 'R');
```

```
    ELSIF NOT F_IS_COPY_AVAILABLE(copy_id_input) THEN
        raise_application_error(-20002, 'Error: Cannot rent unavailable copy');
```

```
    ELSE
        -- jeśli użytkownik nie złożył wcześniej rezerwacji to zmień status w tabeli Copy
        UPDATE Copy
        SET is_available = 'N'
        WHERE copy_id = copy_id_input;
    END IF;

    -- Wstawienie nowego wypożyczenia do tabeli Rental
    INSERT INTO Rental (client_id, copy_id, out_date, due_date, return_date)
    VALUES (client_id_input, copy_id_input, reservation_date_input, reservation_expiry_date_input, null);

    DBMS_OUTPUT.PUT_LINE('New rental added successfully.');
```

```
EXCEPTION
    WHEN OTHERS THEN
        raise_application_error(-20003, 'Error adding new rental: ' || SQLERRM);
END;
```

Użycie:

Kopia o id = 2 jest dostępna:

	COPY_ID	MOVIE_ID	IS_AVAILABLE
1	1	1	N
2	2	8	Y

po wykonaniu procedury:

```
begin
  P_ADD_NEW_RENTAL( client_id_input: 1, copy_id_input: 2, rental_duration_input: 10);
end;
```

	COPY_ID	MOVIE_ID	IS_AVAILABLE
1	1	1	N
2	2	8	N

	RENTAL_ID	CLIENT_ID	COPY_ID	OUT_DATE	DUE_DATE
1	21	1	2	2024-05-22 12:05:13	2024-06-01 12:05:13

p_return_rental

Procedura odpowiedzialna za zwrot kopii i uzupełnienie daty zwrotu RETURN_DATE w Rental. Odpowiada fizycznemu zwrotowi filmu do wypożyczalni.

```
create PROCEDURE P_RETURN_RENTAL (rental_id_input INT)
IS
  v_count INT;
BEGIN
  -- Sprawdzamy czy istnieje takie wypożyczenie
  SELECT COUNT(*) INTO v_count
  FROM Rental
  WHERE rental_id = rental_id_input;
  IF v_count = 0 THEN
    raise_application_error(-20002, 'Rental ID does not exist.');
```

Użycie:

Przed:

	RENTAL_ID	CLIENT_ID	COPY_ID	OUT_DATE	DUE_DATE	RETURN_DATE
1	41	1	6	2024-06-06 18:52:46	2024-06-17 18:52:46	2024-06-06 19:03:26
2	61	1	6	2024-06-18 19:30:35	2024-07-02 19:30:35	2024-06-18 19:36:26
3	1	2	38	2024-05-04	2024-05-07	2024-05-14
4	2	13	12	2022-06-02	2022-06-26	2022-06-23
5	3	7	5	2021-10-14	2021-11-09	2021-11-09
6	4	17	26	2024-02-13	2024-02-19	2024-02-26
7	5	13	33	2022-07-24	2022-08-13	2022-08-11
8	6	15	37	2024-05-01	2024-05-04	<null>

25	25	10	N
26	26	8	N
27	27	10	N
28	28	6	Y
29	29	3	N
30	30	1	N
31	31	5	Y
32	32	8	Y
33	33	10	N
34	34	5	Y
35	35	10	N
36	36	4	Y
37	37	1	N
38	38	8	N
39	39	4	Y
40	40	6	N

Po wykonaniu:

```
begin
  p_return_rental(6);
end;
```

Kopia o copy_id = 37 jest teraz dostępna:

25	25	10	N
26	26	8	N
27	27	10	N
28	28	6	Y
29	29	3	N
30	30	1	N
31	31	5	Y
32	32	8	Y
33	33	10	N
34	34	5	Y
35	35	10	N
36	36	4	Y
37	37	1	Y
38	38	8	N
39	39	4	Y
40	40	6	N

I oczywiście uzupełnia się wtedy pole RETURN_DATE.

	RENTAL_ID	CLIENT_ID	COPY_ID	OUT_DATE	DUE_DATE	RETURN_DATE
1	41	1	6	2024-06-06 18:52:46	2024-06-17 18:52:46	2024-06-06 19:03:26
2	61	1	6	2024-06-18 19:30:35	2024-07-02 19:30:35	2024-06-18 19:36:26
3	1	2	38	2024-05-04	2024-05-07	2024-05-14
4	2	13	12	2022-06-02	2022-06-26	2022-06-23
5	3	7	5	2021-10-14	2021-11-09	2021-11-09
6	4	17	26	2024-02-13	2024-02-19	2024-02-26
7	5	13	33	2022-07-24	2022-08-13	2022-08-11
8	6	15	37	2024-05-01	2024-05-04	2024-06-19 01:54:53
9	7	1	21	2024-02-06	2024-02-09	2024-02-08

update_copy_availability

Procedura odpowiedzialna za aktualizację pola `IS_AVAILABLE` w tabeli `Copy` w przypadku, gdy występują pewne nieprawidłowości. Sprawdzamy czy dana kopia jest zarezerwowana lub wypożyczona. Jeśli tak to poprawiamy pole na `Y`. Jeśli nie to ustawiamy na `N`.

```
CREATE OR REPLACE PROCEDURE update_copy_availability IS
BEGIN
  FOR copy_record IN (SELECT COPY_ID FROM COPY) LOOP
    DECLARE
      v_rental_count NUMBER;
      v_reservation_count NUMBER;
    BEGIN
      -- Sprawdzamy, czy kopia jest wypożyczona
      SELECT COUNT(*) INTO v_rental_count
      FROM RENTAL
      WHERE COPY_ID = copy_record.COPY_ID
      AND RETURN_DATE IS NULL;

      -- Sprawdzamy, czy kopia jest zarezerwowana
      SELECT COUNT(*) INTO v_reservation_count
      FROM RESERVATION
      WHERE STATUS = 'N' AND COPY_ID = copy_record.COPY_ID;

      -- Aktualizujemy dostępność kopii
      IF v_rental_count > 0 OR v_reservation_count > 0 THEN
        UPDATE COPY
        SET is_available = 'N'
        WHERE COPY_ID = copy_record.COPY_ID;
      ELSE
        UPDATE COPY
        SET is_available = 'Y'
        WHERE COPY_ID = copy_record.COPY_ID;
      END IF;
    END;
  END LOOP;

  COMMIT;
END;
```

W ten sposób można z niej skorzystać:

```
BEGIN
  update_copy_availability();
END;
```

p_add_client

Procedura `P_ADD_CLIENT` dodaje nowego klienta do tabeli `CLIENTS` z podanymi danymi: imieniem, nazwiskiem, adresem i numerem telefonu. Jeśli podczas dodawania klienta wystąpi błąd, procedura zgłasza komunikat.

```
create PROCEDURE P_ADD_CLIENT (
  p_firstname IN VARCHAR2,
  p_lastname IN VARCHAR2,
  p_address IN VARCHAR2,
  p_phone IN VARCHAR2
) AS
BEGIN
  BEGIN
    INSERT INTO CLIENTS (FIRSTNAME, LASTNAME, ADDRESS, PHONE)
    VALUES (p_firstname, p_lastname, p_address, p_phone);
  EXCEPTION
    WHEN OTHERS THEN
      RAISE_APPLICATION_ERROR(-20001, 'Error adding client: ' || SQLERRM);
  END;
END p_add_client;
```

Przykład użycia:

```
BEGIN
  p_add_client('Jan', 'Krakowski', 'Warszawa, ul. Krakowskie Przedmieście 1', '123456789');
END;
```

Rezultat - nowy klient został dodany:

19	19	Edyta	Dziuba	Leszno 29-480 Hebanowa 14	538-933-400
20	20	Jasława	Ziółkowski	Elbląg 79-419 Moniuszki 43	676-858-116
21	61	Jan	Kowalski	Warszawa, ul. Krakowskie Przedmieście 1	123456789

p_delete_client

Procedura umożliwia usunięcie klienta z tabeli **Clients** znające jego **id**. Operacja jest niemożliwa do wykonania gdy w bazie nie występuje taki klient lub gdy klient ma na wypożyczeniu jakiś film.

```
create PROCEDURE P_DELETE_CLIENT (
  p_client_id IN NUMBER
) AS
  v_exists NUMBER;
BEGIN
  SELECT COUNT(*) INTO v_exists FROM CLIENTS WHERE CLIENT_ID = p_client_id;

  IF v_exists = 0 THEN
    RAISE_APPLICATION_ERROR(-20002, 'Client does not exist.');
```

```
  END IF;

  BEGIN
    DELETE FROM CLIENTS
    WHERE CLIENT_ID = p_client_id;
  EXCEPTION
    WHEN OTHERS THEN
      RAISE_APPLICATION_ERROR(-20003, 'Error deleting client: ' || SQLERRM);
  END;
END P_DELETE_CLIENT;
```

Wykorzystanie procedury:

```
BEGIN
  P_DELETE_CLIENT(61); -- Jan Kowalski
END;
```

Klient został usunięty

	CLIENT_ID	FIRSTNAME	LASTNAME	ADDRESS	PHONE
1	1	Sulibor	Dolata	Zamość 77-969 Jutrzenki 327	515-291-260
2	2	Ewa	Słowik	Katowice 10-093 Sosnowa 160	363-979-677
3	3	Ewaryst	Dworak	Siemianowice Śląskie 81-044 Rzemieniewicka 665	404-725-511
4	4	Felicja	Gruszka	Gorzów Wielkopolski 70-948 Freta 841	743-480-997
5	5	Amira	Piskorz	Sosnowiec 43-743 Cukiernicza 234	208-619-313
6	6	Stefania	Łukaszewicz	Bielsko-Biała Bielsko-Biała Lipowa 432	287-333-193
7	7	Judyta	Doliński	Wrocław 02-295 Akacyjowa 291	415-425-630
8	8	Bernadeta	Kulesza	Gorzów Wielkopolski 45-608 Żytnia 231	604-813-038
9	9	Józefina	Perkowski	Sopot 25-345 Słowackiego 69	583-420-921
10	10	Miłostawa	Kiełbasa	Tarnobrzeg 51-634 Leśna 81a	563-508-811
11	11	Eugeniusz	Stępniewski	Świnoujście 29-318 Przedmieście 621b	365-347-928
12	12	Baltazar	Andrzejczak	Olsztyn 04-519 Kilińskiego 98	608-948-284
13	13	Jaromira	Kawecki	Wrocław 17-701 Klasztorna 135a	940-871-026
14	14	Paula	Nowakowski	Tarnobrzeg 39-600 Leśna 709	203-382-722
15	15	Ida	Flis	Przemyśl 15-399 Herbaciana 55c	236-462-162
16	16	Antoni	Sidor	Białystok 76-676 Kujawska 36a	331-305-493
17	17	Helga	Chmiel	Słupsk 87-559 Żytnia 70	215-103-396
18	18	Bojana	Chmielewski	Łomża 94-962 Szkolna 774	238-976-688
19	19	Edyta	Dziuba	Leszno 29-480 Hebanowa 14	538-933-400
20	20	Jasława	Ziółkowski	Elbląg 79-419 Moniuszki 43	676-858-116

```
BEGIN
  P_DELETE_CLIENT(62); -- Nie ma takiego klienta
END;
```

```
[72000][20002]
ORA-20002: Client does not exist.
ORA-06512: przy "BD_416551.P_DELETE_CLIENT", linia 9
ORA-06512: przy linia 2
Position: 0
```

p_update_client

Procedura umożliwia modyfikacje danych klienta z tabeli `Clients`. Jeśli klient o podanym identyfikatorze nie istnieje, procedura zgłasza wyjątek z komunikatem.

```
create PROCEDURE P_UPDATE_CLIENT (
  p_client_id IN NUMBER,
  p_firstname IN VARCHAR2,
  p_lastname IN VARCHAR2,
  p_address IN VARCHAR2,
  p_phone IN VARCHAR2
) AS
  v_exists NUMBER;
BEGIN
  SELECT COUNT(*) INTO v_exists FROM CLIENTS WHERE CLIENT_ID = p_client_id;

  IF v_exists = 0 THEN
    RAISE_APPLICATION_ERROR(-20004, 'Client does not exist.');
```

```
  END IF;

  BEGIN
    UPDATE CLIENTS
      SET FIRSTNAME = p_firstname, LASTNAME = p_lastname, ADDRESS = p_address, PHONE = p_phone
      WHERE CLIENT_ID = p_client_id;
  EXCEPTION
    WHEN OTHERS THEN
      RAISE_APPLICATION_ERROR(-20005, 'Error updating client: ' || SQLERRM);
  END;
END P_UPDATE_CLIENT;
```

Wykorzystanie procedury:

```
BEGIN
  p_update_client(1, 'Janusz', 'Kowalski', 'Kraków, ul. Floriańska 2', '987654321');
END;
```

I dane klienta o Id = 1 zostały zaaktualizowane

	CLIENT_ID	FIRSTNAME	LASTNAME	ADDRESS	PHONE
1	1	Janusz	Kowalski	Kraków, ul. Floriańska 2	987654321
2	2	Ewa	Słowik	Katowice 10-093 Sosnowa 160	363-979-677

```
BEGIN
  p_update_client(100, 'Dariusz', 'Michalski', 'Kraków, ul. Floriańska 2', '987654321'); -- Nie ma takiego
klienta
END;
```

```
[72000][20004]
ORA-20004: Client does not exist.
ORA-06512: at "BD_416551.P_UPDATE_CLIENT", line 13
ORA-06512: at line 2
Position: 0
```

Triggery

t_copy_check_available

Triger sprawdza przed dodaniem lub aktualizacją rekordu w tabeli **Copy**, czy pole **is_available** ma wartość **N** albo **Y**.

```
CREATE OR REPLACE TRIGGER t_copy_check_available
BEFORE INSERT OR UPDATE OF is_available ON Copy
FOR EACH ROW
BEGIN
  IF :NEW.is_available NOT IN ('Y', 'N') THEN
    raise_application_error(-20001, 'Invalid value for is_available column. Value must be "Y" or "N".');
  END IF;
END;
```

```
248 update COPY
249 set IS_AVAILABLE = 'A'
250 where COPY_ID = 1;
```

```
[72000][20001]
ORA-20001: Invalid value for is_available column. Value must be "Y" or "N".
ORA-06512: at "BD_416551.T_COPY_CHECK_AVAILABLE", line 3
ORA-04088: error during execution of trigger 'BD_416551.T_COPY_CHECK_AVAILABLE'
Position: 7
```

t_reservation_add

Triger odpowiada za walidację danych przy dodawaniu nowej rezerwacji oraz za zmianę statusu w tabeli **Copy**.
Sprawdzamy czy istnieje taki użytkownik i kopia oraz sprawdzamy czy można zarezerwować tę kopię.

```
CREATE OR REPLACE TRIGGER t_reservation_add
BEFORE INSERT ON reservation
FOR EACH ROW
DECLARE
```

```

copy_available BOOLEAN;
BEGIN
  IF NOT f_check_client_exist(:NEW.client_id) THEN
    raise_application_error(-20001, 'Client with the given ID does not exist.');
```

END IF;

```

  IF NOT f_check_copy_exist(:NEW.copy_id) THEN
    raise_application_error(-20002, 'Copy with the given ID does not exist.');
```

END IF;

```

  -- Wywołanie funkcji F_IS_COPY_AVAILABLE i przypisanie wyniku do zmiennej copy_available
  copy_available := f_is_copy_available(:NEW.copy_id);

  -- Sprawdzenie, czy kopia jest już zarezerwowana lub wypożyczona
  IF NOT copy_available THEN
    raise_application_error(-20003, 'Copy is already reserved or rented.');
```

END IF;

```

  -- Sprawdzenie, czy status rezerwacji jest poprawny
  IF :NEW.status <> 'N' THEN
    raise_application_error(-20004, 'Invalid reservation status. Status must be "N".');
```

END IF;

```

  -- Aktualizacja pola is_available na wartość 'N' w tabeli Copy
  UPDATE copy SET is_available = 'N' WHERE copy_id = :NEW.copy_id;
END;
```

Test trigera - na początku osoba o id=1 rezerwuje film o copy_id=1, a następnei osoba o id=2 próbuje zarezerwować ten sam film: Pierwsza operacja - wszystko działa:

```

BEGIN
  P_ADD_RESERVATION(1, 1, 10);
END;
```

Drugie polecenie - pojawia się błąd:

```

BEGIN
  P_ADD_RESERVATION(2, 1, 10);
END;
```

```

[72000][20002]
ORA-20002: Error adding reservation: ORA-20003: Copy is already reserved or rented.
ORA-06512: at "BD_416551.T_RESERVATION_ADD", line 17
ORA-04088: error during execution of trigger 'BD_416551.T_RESERVATION_ADD'
ORA-06512: at "BD_416551.P_ADD_RESERVATION", line ...
```

t_reservation_update

Triger sprawdza czy nowy status składa się ze znaku C, N czy R oraz aktualizuje status dostępności w tabeli Copy.

```

CREATE OR REPLACE TRIGGER t_reservation_update
BEFORE UPDATE ON Reservation
FOR EACH ROW
DECLARE
  new_status_input CHAR(1);
BEGIN
  -- Przypisanie nowego statusu do zmiennej
  new_status_input := :NEW.status;

  -- Sprawdzenie, czy nowy status jest prawidłowy
  IF new_status_input NOT IN ('N', 'C', 'R') THEN
```

```
        raise_application_error(-20001, 'Invalid status. Status must be "N", "C", or "R".');
    END IF;

    -- Aktualizacja stanu dostępności kopii w tabeli Copy
    IF new_status_input = 'C' THEN
        UPDATE Copy
        SET is_available = 'Y'
        WHERE copy_id = :NEW.copy_id;
    ELSE
        UPDATE Copy
        SET is_available = 'N'
        WHERE copy_id = :NEW.copy_id;
    END IF;
END;
```

```
248 ❗ update RESERVATION
249    set STATUS = 'A'
250    where RESERVATION_ID = 1;

[72000][20001]
ORA-20001: Invalid status. Status must be "N", "C", or "R".
ORA-06512: at "BD_416551.T_RESERVATION_UPDATE", line 9
ORA-04088: error during execution of trigger 'BD_416551.T_RESERVATION_UPDATE'
Position: 7
```

t_rental_add

Triger odpowiada za zmianę statusu kopii w tabeli **Copy** przy dodawaniu nowego wypożyczenia.

```
CREATE OR REPLACE TRIGGER t_rental_add
BEFORE INSERT ON Rental
FOR EACH ROW
BEGIN
    -- Aktualizacja stanu dostępności kopii w tabeli Copy
    UPDATE Copy
    SET is_available = 'N'
    WHERE copy_id = :NEW.copy_id;

    DBMS_OUTPUT.PUT_LINE('Rental added successfully.');
```

```
EXCEPTION
    WHEN OTHERS THEN
        raise_application_error(-20001, 'Error updating rental: ' || SQLERRM);
END;
```

t_rental_return

Triger odpowiada za zmianę statusu kopii w tabeli **Copy** przy zwracaniu wypożyczenia.

```
CREATE OR REPLACE TRIGGER t_rental_return
AFTER UPDATE ON Rental
FOR EACH ROW
BEGIN
    -- Sprawdamy czy zmiana dotyczyła zwrotu filmu
    IF :OLD.return_date IS NULL AND :NEW.return_date IS NOT NULL THEN
        -- Aktualizacja stanu dostępności kopii w tabeli Copy
        UPDATE Copy
        SET is_available = 'Y'
        WHERE copy_id = :OLD.copy_id;

        DBMS_OUTPUT.PUT_LINE('Rental returned successfully.');
```

```
END IF;
EXCEPTION
  WHEN OTHERS THEN
    raise_application_error(-20001, 'Error returning rental: ' || SQLERRM);
END;
```

t_prevent_delete_client_with_rentals

Trigger `T_PREVENT_DELETE_CLIENT_WITH_RENTALS` uniemożliwia usunięcie klienta z tabeli `CLIENTS`, jeśli ma on wypożyczone filmy z nieuzupełnioną datą zwrotu w tabeli `RENTAL`, w takim przypadku trigger zgłasza błąd z komunikatem "Nie można usunąć klienta, który ma wypożyczone filmy"

```
create trigger T_PREVENT_DELETE_CLIENT_WITH_RENTALS
  before delete
  on CLIENTS
  for each row
DECLARE
  v_rentals_count NUMBER;
BEGIN
  -- Sprawdzenie, czy klient ma wypożyczone filmy
  SELECT COUNT(*)
  INTO v_rentals_count
  FROM RENTAL
  WHERE CLIENT_ID = :OLD.CLIENT_ID
     AND RETURN_DATE IS NULL;

  IF v_rentals_count > 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Nie można usunąć klienta, który ma wypożyczone filmy.');
```

Przykład działania:

RENTAL_ID	CLIENT_ID	COPY_ID	OUT_DATE	DUE_DATE	RETURN_DATE
6	15	37	2024-05-01	2024-05-04	<null>

Klient o `id=6` nie oddał jeszcze filmu. Trigger zablokuje usunięcie danego klienta

```
[72000][20003]
ORA-20003: Error deleting client: ORA-02292: naruszono więzy spójności (BD_416551.RESERVATION_CLIENTS) - znaleziono rekord podrzędny
ORA-06512: przy "BD_416551.P_DELETE_CLIENT", linia 17
ORA-06512: przy linia 2
Position: 0
```

Backend

Aplikacja została zrealizowana w `Pythonie`, przy użyciu frameworka `Flask`.

Połączenie z bazą danych

W pliku `base.py` łączymy się z bazą danych przy pomocy modułu `cx_Oracle`, który umożliwia interakcję z bazą danych `Oracle` za pomocą języka `Python`. Funkcja odczytuje plik `config.txt` z nazwą użytkownika i hasłem do bazy danych by móc nawiązać połączenie.

```
import cx_Oracle

global conn
```

```
def connect_to_data_base():
    global conn
    try:
        with open('MiniProjekt/backend/config.txt', 'r') as file:
            lines = file.readlines()

        config_data = {}

        for line in lines:
            key, value = line.strip().split(' = ')
            config_data[key] = value

        lib_dir = "C:\\instantclient_21_13"
        user = config_data['user']
        password = config_data['password']
        dsn = cx_Oracle.makedsn("dbmanage.lab.ii.agh.edu.pl", 1521, sid="DBMANAGE")

        cx_Oracle.init_oracle_client(lib_dir=lib_dir)
        conn = cx_Oracle.connect(user=user, password=password, dsn=dsn, encoding="UTF-8")

        return conn

    except cx_Oracle.Error as error:
        print("Błąd podczas łączenia z bazą danych:", error)
        return None
```

Główna aplikacja

Ten kod konfiguruje aplikację webową za pomocą frameworka **Flask**, która umożliwia interakcję z bazą danych poprzez różne komponenty, takie jak tabele, widoki, procedury składowane i funkcje. Po uruchomieniu aplikacji, główna strona (<http://localhost:5000>) wyświetla dostępne tabele, widoki i funkcje oraz domyślnie pokazuje dane klienta o identyfikatorze 1.

```
from flask import Flask, render_template
from base import connect_to_data_base
from tables import tables, tables_blueprint
from views import views, views_blueprint
from procedures import procedures_blueprint
from functions import functions, functions_blueprint

app = Flask(__name__)
app.config['JSON_AS_ASCII'] = False

conn = connect_to_data_base()
app.register_blueprint(tables_blueprint)
app.register_blueprint(views_blueprint)
app.register_blueprint(procedures_blueprint)
app.register_blueprint(functions_blueprint)

# http://localhost:5000
@app.route('/')
def index():
    return render_template(
        'main.html',
        tables=tables,
        views=views,
        functions=functions,
        default_client_id=1 # Domyślnie wyświetlany klient
    )

if __name__ == '__main__':
    app.run(debug=True)
```

Wykonywanie poleceń:

Funkcja `execute_query` wykonuje podane zapytanie SQL do bazy danych przy użyciu biblioteki `cx_Oracle`. Zwraca listę wyników zapytania lub słownik z komunikatem błędu, obsługując również możliwość cofnięcia transakcji w przypadku wystąpienia problemu.

```
def execute_query(sql: str) -> list[any] | dict[str, str]:
    if conn is None:
        return {'error': 'Błąd podczas łączenia z bazą danych'}

    try:
        cursor = conn.cursor()
        cursor.execute(sql)
        rows = cursor.fetchall() if cursor.description else []
        cursor.close()
        return rows

    except cx_Oracle.Error as error:
        if conn:
            conn.rollback()
        print("Błąd podczas wykonania zapytania:", error)
        return {'error': str(error)}
```

Funkcja `call_function` wykonuje wywołanie funkcji przechowywanej w bazie danych Oracle, przekazując jej argumenty, a następnie zwraca wynik jako listę wierszy wynikowych. Obsługuje również błędy związane z połączeniem z bazą danych i wyjątki z biblioteki `cx_Oracle`, umożliwiając roll-back transakcji w przypadku wystąpienia problemu.

```
def call_function(func_name: str, args: list[int | str]) -> dict[str, any]:
    if conn is None:
        return {'error': 'Błąd podczas łączenia z bazą danych'}

    try:
        cursor = conn.cursor()
        result_cursor = cursor.var(cx_Oracle.CURSOR)
        cursor.callfunc(func_name, result_cursor, args)
        result_cursor = result_cursor.getvalue()
        rows = result_cursor.fetchall() if result_cursor else []
        cursor.close()
        return rows

    except cx_Oracle.Error as error:
        if conn:
            conn.rollback()
        print("Błąd podczas wykonania funkcji:", error)
        return {'error': str(error)}
```

Funkcja `call_procedure` wykonuje procedurę przechowywaną w bazie danych Oracle, przekazując jej argumenty i zatwierdzając zmiany w bazie danych po jej wykonaniu. Obsługuje błędy związane z połączeniem z bazą danych oraz wyjątki z biblioteki `cx_Oracle`, umożliwiając roll-back transakcji w przypadku wystąpienia problemu podczas wykonywania procedury. Funkcja zwraca komunikat o pomyślnym wykonaniu procedury lub informację o błędzie.

```
def call_procedure(proc_name: str, args: list[int | str]) -> dict[str, any]:
    if conn is None:
        return {'error': 'Błąd podczas łączenia z bazą danych'}

    try:
        cursor = conn.cursor()
        cursor.callproc(proc_name, args)
        conn.commit()
        cursor.close()
        return {'message': f'Procedure {proc_name} executed successfully'}

    except cx_Oracle.Error as error:
        if conn:
            conn.rollback()
```

```
print("Błąd podczas wykonania procedury:", error)
return {'error': str(error)}
```

Funkcja `get_table_data` pobiera nazwy kolumn oraz dane z określonej tabeli w bazie danych, a następnie renderuje je w szablonie HTML. Parametry opcjonalne `display_name` i `comment` służą do dodatkowego dostosowania wyglądu wyrenderowanej tabeli.

```
def get_table_data(table_name: str, display_name: str = "", comment: str = "") -> str:
    column_names_packed = execute_query(
        f"SELECT column_name FROM USER_TAB_COLUMNS WHERE table_name = '{table_name.upper()}'"
    )
    column_names = [item for sublist in column_names_packed for item in sublist]
    data = execute_query(f"SELECT * FROM {table_name}")
    return render_template('table.html',
                           table_name=table_name,
                           column_names=column_names,
                           data=data,
                           comment=comment,
                           display_name=display_name,
                           )
```

Funkcja `execute_and_render` wykonuje podane zapytanie do bazy danych za pomocą funkcji `execute_query`, a następnie renderuje wynikowy szablon HTML przy użyciu `render_template`. Jeśli wykonanie zapytania zakończy się błędem, funkcja zwraca komunikat o błędzie. W przeciwnym razie zwraca szablon HTML z danymi zapytania, które są przekazane pod nazwą określoną przez parametr `value_name`.

```
def execute_and_render(query: str, template_url: str, value_name: str = 'data') -> str:
    result = execute_query(query)
    if 'error' in result:
        return f"Wystąpił błąd: {result['error']}", 500
    else:
        return render_template(template_url, **{value_name: result})
```

Uruchomienie

Po uruchomieniu pliku `main.py` (zawartość jest pokazana wyżej) i wpisaniu w przeglądarce `http://localhost:5000` pokazuje się wybór możliwych endpointów.

Lista tabeli w bazie:	
Clients	
Reservation	
Rental	
Copy	
Categories	
Movies	
Actors	
Actors_in_movie	
Lista widoków:	
Movie Popularity	
Current Reservations	
Available Copies (filtrowanie, rezerwacja)	
Actor Rentals	
Actors Per Category	
Clients Delays Summary	
Currently Borrowed Copies	
Lista funkcji:	
Search movies (po kategoriach)	
Client Reservations (po id klienta)	
Rent movie (formularz do wypożyczenia)	
Return movie (formularz do zwrotów)	
Zarządzanie klientami	
Dodaj klienta	
Usuń klienta	
Zaktualizuj klienta	
Lista klientów	

Szablon strony internetowej, który wyświetla naszą stronę startową:

```
<!doctype html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <title>Elementy z bazy danych</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #eeeeee;
      color: #333;
      margin: 0;
      padding: 0;
      display: flex;
      flex-direction: column;
      align-items: center;
    }
    h2 {
      color: #757575;
      margin-top: 20px;
      text-align: center;
    }
    ul {
      list-style-type: none;
      padding: 0;
    }
    li {
      background: #fdfdfd;
      margin: 10px 0;
      padding: 0;
      border-radius: 5px;
      box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
      overflow: hidden;
    }
    li a {
      text-decoration: none;
      color: #606060;
      font-weight: bold;
      display: block;
      padding: 10px;
      width: 100%;
      height: 100%;
    }
    li a:hover {
      color: #fff;
      background-color: #757575;
    }
    .container {
      width: 80%;
      max-width: 800px;
      margin: 20px auto;
      padding: 50px;
      background: #f7f7f7;
      border-radius: 10px;
      box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    }
  </style>
</head>
<body>

  <div class="container">
    <h2>Lista tabeli w bazie:</h2>
    <ul>
      {% for table in tables %}
        <li><a href="{ { url_for(table[0]) } }">{ { table[1] } }</a></li>
      {% endfor %}
    </ul>
  </div>
</body>
```

```

    </ul>

    <br>

    <h2>Lista widoków:</h2>
    <ul>
        {% for view in views %}
            <li><a href="{ { url_for(view[0]) } }">{{ view[1] }}</a></li>
        {% endfor %}
    </ul>

    <br>

    <h2>Lista funkcji:</h2>
    <ul>
        {% for func in functions %}
            <li><a href="{ { url_for(func[0], client_id=default_client_id) } }">{{ func[1] }}</a></li>
        {% endfor %}
    </ul>

    <br>

    <h2>Zarządzanie klientami</h2>
    <ul>
        <li><a href="{ { url_for('procedures.add_client_form') } }">Dodaj klienta</a></li>
        <li><a href="{ { url_for('procedures.delete_client_form') } }">Usuń klienta</a></li>
        <li><a href="{ { url_for('procedures.update_client_form') } }">Zaktualizuj klienta</a></li>
        <li><a href="{ { url_for('tables.get_Clients') } }">Lista klientów</a></li>
    </ul>
</div>

</body>
</html>

```

Tabele

Do wyświetlania tabel stworzyliśmy osobny plik `tables.py` z endpoint'ami do każdej tabeli, które następnie wywołują funkcję `get_table_data()` odpowiedzialną za wygenerowanie odpowiedniej tabeli.

```

from flask import Blueprint
from base import get_table_data

tables_blueprint = Blueprint('tables', __name__)

tables = (
    ('tables.get_Clients', 'Clients'),
    ('tables.get_Reservation', 'Reservation'),
    ('tables.get_Rental', 'Rental'),
    ('tables.get_Copy', 'Copy'),
    ('tables.get_Categories', 'Categories'),
    ('tables.get_Movies', 'Movies'),
    ('tables.get_Actors', 'Actors'),
    ('tables.get_Actors_in_movie', 'Actors_in_movie'),
)

@tables_blueprint.route('/tables/Clients')
def get_Clients():
    return get_table_data('Clients')

@tables_blueprint.route('/tables/Reservation')
def get_Reservation():
    return get_table_data('Reservation')

@tables_blueprint.route('/tables/Rental')

```

```
def get_Rental():
    return get_table_data('Rental')

@tables_blueprint.route('/tables/Copy')
def get_Copy():
    return get_table_data('Copy')

@tables_blueprint.route('/tables/Categories')
def get_Categories():
    return get_table_data('Categories')

@tables_blueprint.route('/tables/Movies')
def get_Movies():
    return get_table_data('Movies')

@tables_blueprint.route('/tables/Actors')
def get_Actors():
    return get_table_data('Actors')

@tables_blueprint.route('/tables/Actors_in_movie')
def get_Actors_in_movie():
    return get_table_data('Actors_in_movie')
```

Wszystkie dane z Clients:

CLIENT_ID	FIRSTNAME	LASTNAME	ADDRESS	PHONE
1	Sulibor	Dolata	Zamość 77-969 Jutrzenki 327	515-291-260
2	Ewa	Słowik	Katowice 10-093 Sosnowa 160	363-979-677
3	Ewaryst	Dworak	Siemianowice Śląskie 81-044 Rzemieniewicka 665	404-725-511
4	Felicja	Gruszka	Gorzów Wielkopolski 70-948 Freta 841	743-480-997
5	Amira	Piskorz	Sosnowiec 43-743 Cukiernicza 234	208-619-313
6	Stefania	Łukaszewicz	Bielsko-Biała Bielsko-Biała Lipowa 432	287-333-193
7	Judyta	Doliński	Wrocław 02-295 Akacyjowa 291	415-425-630
8	Bernadeta	Kulesza	Gorzów Wielkopolski 45-608 Żytnia 231	604-813-038
9	Józefina	Perkowski	Sopot 25-345 Słowackiego 69	583-420-921
10	Miłoslawa	Kielbasa	Tarnobrzeg 51-634 Leśna 81a	563-508-811
11	Eugeniusz	Stępniewski	Świnoujście 29-318 Przedmieście 621b	365-347-928
12	Baltazar	Andrzejczak	Olsztyn 04-519 Kilińskiego 98	608-948-284
13	Jaromira	Kawecki	Wrocław 17-701 Klasztorna 135a	940-871-026
14	Paula	Nowakowski	Tarnobrzeg 39-600 Leśna 709	203-382-722
15	Ida	Flis	Przemyśl 15-399 Herbaciana 55c	236-462-162
16	Antoni	Sidor	Białystok 76-676 Kujawska 36a	331-305-493
17	Helga	Chmiel	Słupsk 87-559 Żytnia 70	215-103-396
18	Bojana	Chmielewski	Łomża 94-962 Szkolna 774	238-976-688
19	Edyta	Dziuba	Leszno 29-480 Hebanowa 14	538-933-400
20	Jasława	Ziółkowski	Elbląg 79-419 Moniuszki 43	676-858-116

Dane są wyświetlane w tabelce dzięki plikowi `table.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>All Rentals</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #eeeeee;
      color: #333;
```

```

    }
    table {
        width: 70%;
        border-collapse: collapse;
        margin: auto;
    }
    th, td {
        border: 1px solid rgb(125, 125, 125);
        padding: 8px;
        text-align: left;
    }
    th {
        text-align: center;
    }
    th {
        background-color: #d5d5d5;
    }
    h2, p {
        text-align: center;
    }
}
</style>
</head>
<body>
    <br>
    <a href="..">< Powrót do strony głównej</a>

    {% if display_name == "" %}
        <h2>Wszystkie dane z {{ table_name }}:</h2>
    {% else %}
        <h2>Wszystkie dane z {{ display_name }}:</h2>
    {% endif %}

    {% if comment != "" %}
        <p>{{comment}}</p>
    {% endif %}
    <br>
    <table border="1">
        <thead>
            <tr>
                {% for column in column_names %}
                    <th>{{ column }}</th>
                {% endfor %}
            </tr>
        </thead>
        <tbody>
            {% for row in data %}
                <tr>
                    {% for cell in row %}
                        <td>{{ cell }}</td>
                    {% endfor %}
                </tr>
            {% endfor %}
        </tbody>
    </table>
</body>
</html>

```

Widoki

Lista widoków:

Movie Popularity

Current Reservations

Available Copies (filtrowanie, rezerwacja)

Actor Rentals

Actors Per Category

Clients Delays Summary

Currently Borrowed Copies

Stworzyliśmy osobny plik zawierający endpointy dla widoków:

```
from flask import Blueprint, request, render_template
from base import execute_and_render, get_table_data, call_function

views_blueprint = Blueprint('views', __name__)

views = (
    ('views.VW_MOVIE_POPULARITY', 'Movie Popularity'),
    ('views.VW_CURRENT_RESERVATIONS', 'Current Reservations'),
    ('views.VW_AVAILABLE_COPIES', 'Available Copies (filtrowanie, rezerwacja)'),
    ('views.VW_ACTOR_RENTALS', 'Actor Rentals'),
    ('views.VW_MOST_POPULAR_ACTORS_PER_CATEGORY', 'Actors Per Category'),
    ('views.VW_CLIENTS_DELAYS_SUM', 'Clients Delays Summary'),
    ('views.VW_CURRENTLY_BORROWED_COPIES', 'Currently Borrowed Copies'),
)

@views_blueprint.route('/VW_MOVIE_POPULARITY')
def VW_MOVIE_POPULARITY():
    return get_table_data('VW_MOVIE_POPULARITY', display_name='Movie Popularity')

@views_blueprint.route('/VW_CURRENT_RESERVATIONS')
def VW_CURRENT_RESERVATIONS():
    return get_table_data('VW_CURRENT_RESERVATIONS', display_name='Current Reservations')

@views_blueprint.route('/VW_AVAILABLE_COPIES', methods=['GET', 'POST']) # dostępne kopie wyszukiwaniem i
funkcją rezerwacji
def VW_AVAILABLE_COPIES():
    if request.method == 'GET':
        return execute_and_render("select * from VW_AVAILABLE_COPIES", 'views/available_copies.html',
'copies')

    movie_id = request.form.get('movie_id')
    movie_name = request.form.get('movie_name')
    print("Arguments:", movie_id, movie_name)

    if movie_id:
        print("Movie ID:", movie_id)
        result = call_function('f_get_available_copies_for_movie_id', [int(movie_id)])
        print("Result:", result)
        if 'error' in result:
            result = []
        return render_template('views/available_copies.html', copies=result)
```



```

elif movie_name:
    print("Movie Name:", movie_name)
    result = call_function('f_get_available_copies_for_movie_name', [movie_name])
    print("Result:", result)
    if 'error' in result:
        result = []
    return render_template('views/available_copies.html', copies=result)
else:
    return execute_and_render("select * from VW_AVAILABLE_COPIES", 'views/available_copies.html',
'copies')

@views_blueprint.route('/VW_ACTOR_RENTALS')
def VW_ACTOR_RENTALS():
    return get_table_data('VW_ACTOR_RENTALS', display_name='Actor Rentals')

@views_blueprint.route('/VW_MOST_POPULAR_ACTORS_PER_CATEGORY')
def VW_MOST_POPULAR_ACTORS_PER_CATEGORY():
    return get_table_data('VW_MOST_POPULAR_ACTORS_PER_CATEGORY', display_name='Most Popular Actors Per
Category')

@views_blueprint.route('/VW_CLIENTS_DELAYS_SUM')
def VW_CLIENTS_DELAYS_SUM():
    return get_table_data('VW_CLIENTS_DELAYS_SUM',
                           display_name='Clients Delays Summary',
                           comment='This view displays the total days of delay for each client.')

@views_blueprint.route('/VW_CURRENTLY_BORROWED_COPIES')
def VW_CURRENTLY_BORROWED_COPIES():
    return get_table_data('VW_CURRENTLY_BORROWED_COPIES',
                           display_name='Currently Borrowed Copies',
                           comment='This view displays all copies that are currently borrowed.')

```

Widok `VW_AVAILABLE_COPIES` przedstawiający możliwość złożenia rezerwacji na dostępne filmy. Możliwe jest wyszukanie filmu jaki nas interesuje po nazwie.

Endpoint dla tego widoku:

```

@views_blueprint.route('/VW_AVAILABLE_COPIES', methods=['GET', 'POST']) # dostępne kopie wyszukiwaniem i
funkcją rezerwacji
def VW_AVAILABLE_COPIES():
    if request.method == 'GET':
        return execute_and_render("select * from VW_AVAILABLE_COPIES", 'views/available_copies.html',
'copies')

    movie_id = request.form.get('movie_id')
    movie_name = request.form.get('movie_name')
    print("Arguments:", movie_id, movie_name)

    if movie_id:
        print("Movie ID:", movie_id)
        result = call_function('f_get_available_copies_for_movie_id', [int(movie_id)])
        print("Result:", result)
        if 'error' in result:
            result = []
        return render_template('views/available_copies.html', copies=result)
    elif movie_name:
        print("Movie Name:", movie_name)
        result = call_function('f_get_available_copies_for_movie_name', [movie_name])
        print("Result:", result)
        if 'error' in result:
            result = []

```

```
        return render_template('views/available_copies.html', copies=result)
    else:
        return execute_and_render("select * from VW_AVAILABLE_COPIES", 'views/available_copies.html',
        'copies')
```

< Powrót do strony głównej

Available Copies

Movie ID:

Movie Name:

Search

Reset

Movie ID	Copy ID	Movie Title	Category	Release Date	Duration	Action
2	6	Civil War	akcja	2024-03-14	109	<button>Reserve</button>
4	18	Nasze magiczne Encanto	animacja	2021-11-24	99	<button>Reserve</button>
4	36	Nasze magiczne Encanto	animacja	2021-11-24	99	<button>Reserve</button>
4	39	Nasze magiczne Encanto	animacja	2021-11-24	99	<button>Reserve</button>
6	28	Zapłątani	animacja	2010-11-24	96	<button>Reserve</button>
8	10	To	horror	2017-09-05	135	<button>Reserve</button>
8	32	To	horror	2017-09-05	135	<button>Reserve</button>
9	22	Tytanic	katastroficzny	1997-11-01	194	<button>Reserve</button>
9	17	Tytanic	katastroficzny	1997-11-01	194	<button>Reserve</button>
5	9	Piraci z Karaibów: Klątwa Czarnej Perły	przygodowy	2003-06-28	143	<button>Reserve</button>
5	31	Piraci z Karaibów: Klątwa Czarnej Perły	przygodowy	2003-06-28	143	<button>Reserve</button>

Możemy oczywiście przeszukiwać zbiór dostępnych kopii:

< Powrót do strony głównej

Available Copies

Movie ID:

Movie Name:

Search

Reset

Movie ID	Copy ID	Movie Title	Category	Release Date	Duration	Action
5	9	Piraci z Karaibów: Klątwa Czarnej Perły	przygodowy	2003-06-28	143	<button>Reserve</button>
5	31	Piraci z Karaibów: Klątwa Czarnej Perły	przygodowy	2003-06-28	143	<button>Reserve</button>
5	34	Piraci z Karaibów: Klątwa Czarnej Perły	przygodowy	2003-06-28	143	<button>Reserve</button>

Po kliknięciu przycisku **Reserve** otwiera się formularz z danymi do wypełnienia

Reservation Form

Copy ID:

6

Client ID:

Rental Duration:

Submit Reservation

Reservation Form

Copy ID:

6

Client ID:

2

Rental Duration:

14

Submit Reservation

Po wypełnieniu danych możemy zobaczyć, że wykonana została procedura odpowiedzialna za dodanie nowej rezerwacji. I pojawiła się nowa rezerwacja na film o nazwie "Civil War":

< Powrót do strony głównej

Client Reservations

Client ID:

Search

Client ID: 2

Reservation ID	Movie Title	Reservation Date	Reservation Expiry Date	Status	Cancel
102	Civil War	2024-06-19 01:06:54	2024-07-03 01:06:54	N	<div>Cancel</div>
2	Tytanic	2024-01-09 00:00:00	2024-01-16 00:00:00	N	<div>Cancel</div>

Funkcje

Plik `functions.py` odpowiedzialny za tworzenie endpointów dla funkcji z bazy danych, które zwracają widoki tabel.

```
from flask import Blueprint, request, render_template, redirect, url_for
from base import call_function, execute_query

functions_blueprint = Blueprint('functions', __name__)
```

```

functions = (
    ('functions.filter_movies', 'Search movies (po kategoriach)'),
    ('functions.client_reservations', 'Client Reservations (po id klienta)'),
    ('procedures.rent_movie_form', 'Rent movie (formularz do wypożyczenia)'),
    ('procedures.return_movie_form', 'Return movie (formularz do zwrotów)'),
)

@functions_blueprint.route('/filter_movies', methods=['GET', 'POST'])
def filter_movies():
    if request.method == 'POST':
        category_id = request.form['category_id']
        result = call_function('f_get_movies_by_category', [int(category_id)])

        if 'error' in result:
            return f"Error: {result['error']}", 500
        else:
            movies = result
    else:
        movies = get_all_movies()

    return render_template('views/movie_filter_form.html', categories=get_categories(), movies=movies)

def get_all_movies():
    query = "SELECT * FROM vw_movies_with_category"
    movies = execute_query(query)
    return movies

def get_categories():
    query = "SELECT category_id, name FROM Categories"
    categories = execute_query(query)
    return categories

@functions_blueprint.route('/client_reservations/<int:client_id>')
def client_reservations(client_id):
    result = call_function("f_get_client_reservations", [int(client_id)])

    if 'error' in result:
        return f"Wystąpił błąd: {result['error']}", 500
    else:
        return render_template('functions/get_client_reservations.html', reservations=result,
                               client_id=client_id)

@functions_blueprint.route('/client_reservations', methods=['POST'])
def redirect_client_reservations():
    client_id = request.form.get('client_id')
    return redirect(url_for('functions.client_reservations', client_id=client_id))

```

Przykład funkcji wyświetlającej listę wszystkich filmów w wypożyczalni z możliwością filtrowania po kategorii.

Enpoint dla tej funkcji:

```

@functions_blueprint.route('/filter_movies', methods=['GET', 'POST'])
def filter_movies():
    if request.method == 'POST':
        category_id = request.form['category_id']
        result = call_function('f_get_movies_by_category', [int(category_id)])

        if 'error' in result:
            return f"Error: {result['error']}", 500
        else:
            movies = result
    else:
        movies = get_all_movies()

```

```
return render_template('views/movie_filter_form.html', categories=get_categories(), movies=movies)
```

Filter Movies by Category

Select Category:

akcja

Filter

Reset

Movies in Selected Category

Category Name	Movie Title	Description	Release Date	Duration	Rating	Director
akcja	Civil War	Przez pogrążone w krwawej wojnie domowej Stany Zjednoczone przedziera się grupa dziennikarzy, by w Waszyngtonie przeprowadzić ostatni wywiad z prezydentem.	2024-03-14	109	7.4	Alex Garland
animacja	Nasze magiczne Encanto	Dziewczynka pochodzi z kolumbijskiej rodziny, która obdarzona jest magicznymi mocami. Niestety ona sama nie posiada tego daru.	2021-11-24	99	7.2	Jared Bush
animacja	Zapłątani	Żyjąca na odludziu Roszpunka, której włosy mają magiczną moc, marzy, by poznać świat. Jej przepustką do wolności jest czarujący łotrzyk Flynn.	2010-11-24	96	7.6	Nathan Greno
familijny	Shrek	By odzyskać swój dom, brzydki ogr z gadatliwym osłem wyruszają uwolnić piękną księżniczkę.	2001-04-22	90	7.8	Andrew Adamson
horror	To	Opowieść o bestii karmiącej się dziecięcym strachem.	2017-09-05	135	6.7	Andy Muschietti
katastroficzny	Titanic	Rok 1912, brytyjski statek Titanic wyrusza w swój dziewiczy rejs do USA. Na pokładzie emigrant Jack przypadkowo spotyka arystokratkę Rose.	1997-11-01	194	7.3	James Cameron
przygodowy	Uncharted	Utalentowany złodziej Nathan Drake zostaje zwerbowany przez doświadczonego poszukiwacza skarbów Victora Sullivana. Razem chcą odnaleźć zaginione złoto Ferdynanda Magellana.	2022-02-10	116	6.1	Ruben Fleischer

Filter Movies by Category

Select Category:

animacja

Filter

Reset

Movies in Selected Category

Category Name	Movie Title	Description	Release Date	Duration	Rating	Director
animacja	Nasze magiczne Encanto	Dziewczynka pochodzi z kolumbijskiej rodziny, która obdarzona jest magicznymi mocami. Niestety ona sama nie posiada tego daru.	2021-11-24	99	7.2	Jared Bush
animacja	Zapłątani	Żyjąca na odludziu Roszpunka, której włosy mają magiczną moc, marzy, by poznać świat. Jej przepustką do wolności jest czarujący łotrzyk Flynn.	2010-11-24	96	7.6	Nathan Greno

Procedury

Plik `procedures.py` zawiera endpointy dla procedur z naszej bazy danych

```
from flask import Blueprint, request, render_template, redirect, url_for
from base import call_procedure, get_table_data

procedures_blueprint = Blueprint('procedures', __name__)

@procedures_blueprint.route('/reserve', methods=['POST'])
def reserve():
    copy_id = request.form['copy_id']
    return render_template('procedures/reservation_form.html', copy_id=copy_id)

@procedures_blueprint.route('/add_reservation', methods=['POST'])
def add_reservation():
    client_id = request.form['client_id']
    copy_id = request.form['copy_id']
    rental_duration = request.form['rental_duration']

    result = call_procedure('p_add_reservation', [int(client_id), int(copy_id), int(rental_duration)])

    if 'error' in result:
        return "Error: " + result['error'], 500
    else:
        return redirect(url_for('views.VW_AVAILABLE_COPIES'))

@procedures_blueprint.route('/cancel_reservation/<int:reservation_id>', methods=['POST'])
def cancel_reservation(reservation_id):
    client_id = request.form.get('data-client-id') # Odczytaj client_id z atrybutu data-client-id

    if client_id is None:
        return "Error: Client ID is missing", 400

    new_status = 'C'

    result = call_procedure('p_change_reservation_status', [int(reservation_id), new_status])
    if 'error' in result:
        return "Error: " + result['error'], 500
    else:
        return redirect(url_for('functions.client_reservations', client_id=client_id))

@procedures_blueprint.route('/rent_movie_form', methods=['GET'])
def rent_movie_form():
    return render_template('functions/rent_movie_form.html')

@procedures_blueprint.route('/rental', methods=['POST'])
def rental():
    client_id = request.form['client_id']
    copy_id = request.form['copy_id']
    rental_duration = request.form['rental_duration']

    result = call_procedure('p_add_new_rental', [int(client_id), int(copy_id), int(rental_duration)])

    if 'error' in result:
        return "Error: " + result['error'], 500
    else:
        return redirect('http://localhost:5000')

@procedures_blueprint.route('/return_movie_form', methods=['GET'])
def return_movie_form():
    return render_template('procedures/return_movie_form.html')

@procedures_blueprint.route('/return_movie', methods=['POST'])
def return_movie():
    rental_id = request.form['rental_id']

    result = call_procedure('P_RETURN_RENTAL', [int(rental_id)])

    if 'error' in result:
```

```
        return "Error: " + result['error'], 500
    else:
        return redirect('http://localhost:5000')

@procedures_blueprint.route('/add_client_form', methods=['GET'])
def add_client_form():
    return render_template('procedures/add_client_form.html')

@procedures_blueprint.route('/add_client', methods=['POST'])
def add_client():
    firstname = request.form['firstname']
    lastname = request.form['lastname']
    address = request.form['address']
    phone = request.form['phone']

    result = call_procedure('p_add_client', [firstname, lastname, address, phone])

    if 'error' in result:
        return "Error: " + result['error'], 500
    else:
        return redirect(url_for('index'))

@procedures_blueprint.route('/delete_client_form', methods=['GET'])
def delete_client_form():
    return render_template('procedures/delete_client_form.html')

@procedures_blueprint.route('/delete_client', methods=['POST'])
def delete_client():
    client_id = request.form['client_id']

    result = call_procedure('p_delete_client', [int(client_id)])

    if 'error' in result:
        return "Error: " + result['error'], 500
    else:
        return redirect(url_for('index'))

@procedures_blueprint.route('/update_client_form', methods=['GET'])
def update_client_form():
    return render_template('procedures/update_client_form.html')

@procedures_blueprint.route('/update_client', methods=['POST'])
def update_client():
    client_id = request.form['client_id']
    firstname = request.form['firstname']
    lastname = request.form['lastname']
    address = request.form['address']
    phone = request.form['phone']

    result = call_procedure('p_update_client', [int(client_id), firstname, lastname, address, phone])

    if 'error' in result:
        return "Error: " + result['error'], 500
    else:
        return redirect(url_for('index'))
```

Działanie procedur

Anulowanie rezerwacji

Wcześniej pokazaliśmy tworzenie rezerwacji przez klienta, a teraz anulowanie rezerwacji - po kliknięciu **cancel** rezerwacja zmienia swój status na **C(canceled)**:

Client Reservations

Client ID:

Search

Client ID: 2

Reservation ID	Movie Title	Reservation Date	Reservation Expiry Date	Status	Cancel
102	Civil War	2024-06-19 01:06:54	2024-07-03 01:06:54	N	<div>Cancel</div>
2	Tytanic	2024-01-09 00:00:00	2024-01-16 00:00:00	N	<div>Cancel</div>

Client Reservations

Client ID:

Search

Client ID: 2

Reservation ID	Movie Title	Reservation Date	Reservation Expiry Date	Status	Cancel
102	Civil War	2024-06-19 01:06:54	2024-07-03 01:06:54	N	<div>Cancel</div>
2	Tytanic	2024-01-09 00:00:00	2024-01-16 00:00:00	C	<div>Cancel</div>

Wypożyczenie filmu

Po kliknięciu linku możemy wypełnić formularz do wypożyczenia filmu. Musimy podać **CopyId** i **ClientId** oraz możemy ustawić czas trwania wypożyczenia.

Rent a Movie

Client ID:

Copy ID:

Rental Duration (days):

Submit Rental

```
@procedures_blueprint.route('/return_movie', methods=['POST'])
def return_movie():
    rental_id = request.form['rental_id']

    result = call_procedure('P_RETURN_RENTAL', [int(rental_id)])

    if 'error' in result:
        return "Error: " + result['error'], 500
    else:
        return redirect('http://localhost:5000')
```

Przykładowe dane:

Rent a Movie

Client ID:

Copy ID:

Rental Duration (days):

Submit Rental

I w tabeli rental na samym końcu pojawiło się nowe wypożyczenie:

Wszystkie dane z Rental:

RENTAL_ID	CLIENT_ID	COPY_ID	OUT_DATE	DUE_DATE	RETURN_DATE
41	1	6	2024-06-06 18:52:46	2024-06-17 18:52:46	2024-06-06 19:03:26
61	1	6	2024-06-18 19:30:35	2024-07-02 19:30:35	None

Zwrot filmu do wypożyczalni

Spróbujemy zwrócić przedchwilą wypożyczony film. Po kliknięciu w link do zwrotu ukazyuje nam się formularz do zwrotu filmu.

Return Movie Form

Rental ID:

Return Movie

```
@procedures_blueprint.route('/return_movie_form', methods=['GET'])
def return_movie_form():
    return render_template('procedures/return_movie_form.html')

@procedures_blueprint.route('/return_movie', methods=['POST'])
def return_movie():
    rental_id = request.form['rental_id']

    result = call_procedure('P_RETURN_RENTAL', [int(rental_id)])

    if 'error' in result:
        return "Error: " + result['error'], 500
    else:
        return redirect('http://localhost:5000')
```

I widzimy, że film został zwrócony.

Wszystkie dane z Rental:

RENTAL_ID	CLIENT_ID	COPY_ID	OUT_DATE	DUE_DATE	RETURN_DATE
41	1	6	2024-06-06 18:52:46	2024-06-17 18:52:46	2024-06-06 19:03:26
61	1	6	2024-06-18 19:30:35	2024-07-02 19:30:35	2024-06-18 19:36:26

Operacje CRUD

Zrealizowaliśmy opracje CRUD na tabeli `Clients`

Zarządzanie klientami

Dodaj klienta

Usuń klienta

Zaktualizuj klienta

Lista klientów

Dodanie nowego klienta

Add a New Client

First Name:

Jan

Last Name:

Nowak

Address:

Kawiry 3 Kraków

Phone:

112 999 999

Add Client

```
@procedures_blueprint.route('/add_client_form', methods=['GET'])
def add_client_form():
    return render_template('procedures/add_client_form.html')

@procedures_blueprint.route('/add_client', methods=['POST'])
def add_client():
    firstname = request.form['firstname']
    lastname = request.form['lastname']
    address = request.form['address']
    phone = request.form['phone']

    result = call_procedure('p_add_client', [firstname, lastname, address, phone])
```

```
if 'error' in result:
    return "Error: " + result['error'], 500
else:
    return redirect(url_for('index'))
```

Wszystkie dane z Clients:

CLIENT_ID	FIRSTNAME	LASTNAME	ADDRESS	PHONE
51	Jan	Nowak	Kawiory 3 Kraków	112 999 999
1	Sulibor	Dolata	Zamość 77-969 Jutrzenki 327	515-291-260

Aktualizacja klienta

Update Client

Client ID:

First Name:

Last Name:

Address:

Phone:

```
@procedures_blueprint.route('/update_client_form', methods=['GET'])
def update_client_form():
    return render_template('procedures/update_client_form.html')

@procedures_blueprint.route('/update_client', methods=['POST'])
def update_client():
    client_id = request.form['client_id']
    firstname = request.form['firstname']
    lastname = request.form['lastname']
    address = request.form['address']
    phone = request.form['phone']

    result = call_procedure('p_update_client', [int(client_id), firstname, lastname, address, phone])

    if 'error' in result:
        return "Error: " + result['error'], 500
    else:
        return redirect(url_for('index'))
```

Wszystkie dane z Clients:

CLIENT_ID	FIRSTNAME	LASTNAME	ADDRESS	PHONE
51	Filip	Kasprzyk	Kawiory 5 Kraków	123
1	Sulibor	Dolata	Zamość 77-969 Jutrzenki 327	515-291-260

Usunięcie klienta

Delete Client

Client ID:

51

Delete Client

```
@procedures_blueprint.route('/delete_client_form', methods=['GET'])
def delete_client_form():
    return render_template('procedures/delete_client_form.html')

@procedures_blueprint.route('/delete_client', methods=['POST'])
def delete_client():
    client_id = request.form['client_id']

    result = call_procedure('p_delete_client', [int(client_id)])

    if 'error' in result:
        return "Error: " + result['error'], 500
    else:
        return redirect(url_for('index'))
```

I klient został usunięty

Wszystkie dane z Clients:				
CLIENT_ID	FIRSTNAME	LASTNAME	ADDRESS	PHONE
1	Sulibor	Dolata	Zamość 77-969 Jutrzenki 327	515-291-260
2	Ewa	Słowik	Katowice 10-093 Sosnowa 160	363-979-677
3	Ewaryst	Dworak	Siemianowice Śląskie 81 044 Przemianowicka 665	404 725 511

Wyświetlenie pełnej listy klientów

Wszystkie dane z Clients:

CLIENT_ID	FIRSTNAME	LASTNAME	ADDRESS	PHONE
1	Sulibor	Dolata	Zamość 77-969 Jutrzenki 327	515-291-260
2	Ewa	Słowik	Katowice 10-093 Sosnowa 160	363-979-677
3	Ewaryst	Dworak	Siemianowice Śląskie 81-044 Rzemieniewicka 665	404-725-511
4	Felicja	Gruszka	Gorzów Wielkopolski 70-948 Freta 841	743-480-997
5	Amira	Piskorz	Sosnowiec 43-743 Cukiernicza 234	208-619-313
6	Stefania	Łukaszewicz	Bielsko-Biała Bielsko-Biała Lipowa 432	287-333-193
7	Judyta	Doliński	Wrocław 02-295 Akacyjowa 291	415-425-630
8	Bernadeta	Kulesza	Gorzów Wielkopolski 45-608 Żytnia 231	604-813-038
9	Józefina	Perkowski	Sopot 25-345 Słowackiego 69	583-420-921
10	Miłoslawa	Kielbasa	Tarnobrzeg 51-634 Leśna 81a	563-508-811
11	Eugeniusz	Stępniewski	Świnoujście 29-318 Przedmieście 621b	365-347-928
12	Baltazar	Andrzejczak	Olsztyn 04-519 Kilińskiego 98	608-948-284
13	Jaromira	Kawecki	Wrocław 17-701 Klasztorna 135a	940-871-026
14	Paula	Nowakowski	Tarnobrzeg 39-600 Leśna 709	203-382-722
15	Ida	Flis	Przemysł 15-399 Herbaciana 55c	236-462-162
16	Antoni	Sidor	Białystok 76-676 Kujawska 36a	331-305-493
17	Helga	Chmiel	Słupsk 87-559 Żytnia 70	215-103-396
18	Bojana	Chmielewski	Łomża 94-962 Szkolna 774	238-976-688
19	Edyta	Dziuba	Leszno 29-480 Hebanowa 14	538-933-400
20	Jasława	Ziółkowski	Elbląg 79-419 Moniuszki 43	676-858-116

Pozostały kod użyty w projekcie

Szkielet do pokazania rezerwacji klienta - `get_client_reservations.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
  <title>Client Reservations</title>
</head>
<body>
  <div class="container mt-5">
    <a href=".." class="btn btn-primary mb-4">< Powrót do strony głównej</a>
    <h1 class="mb-4">Client Reservations</h1>

    <form method="post" action="/client_reservations" class="form-inline mb-4">
      <div class="form-group mr-2">
        <label for="client_id" class="mr-2">Client ID:</label>
        <input type="number" id="client_id" name="client_id" class="form-control" required>
      </div>
      <button type="submit" class="btn btn-primary">Search</button>
    </form>

    <p><strong>Client ID:</strong> {{ client_id }}</p>

    <table class="table table-striped">
      <thead class="thead-dark">
        <tr>
          <th>Reservation ID</th>
          <th>Movie Title</th>
          <th>Reservation Date</th>
          <th>Reservation Expiry Date</th>
          <th>Status</th>
          <th>Cancel</th>
        </tr>
      </thead>
    </table>
  </div>
```

```

</thead>
<tbody>
  {% for row in reservations %}
    <tr>
      <td>{{ row[0] }}</td>
      <td>{{ row[1] }}</td>
      <td>{{ row[2] }}</td>
      <td>{{ row[3] }}</td>
      <td>{{ row[4] }}</td>
      <td>
        <form action="/cancel_reservation/{{ row[0] }}" method="POST" data-client-id="{{ client_id }}" class="d-inline">
          <input type="hidden" name="reservation_id" value="{{ row[0] }}">
          <input type="hidden" name="data-client-id" value="{{ client_id }}">
          <button type="submit" class="btn btn-danger btn-sm">Cancel</button>
        </form>
      </td>
    </tr>
  {% endfor %}
</tbody>
</table>
</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>

```

Szkielet formularza potrzebnego do wyporzyczenia copii - `rent_movie_form.html`

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
  <title>Rent a Movie</title>
</head>
<body>
  <div class="container mt-5">
    <h1>Rent a Movie</h1>
    <form action="/rental" method="post">
      <div class="form-group">
        <label for="client_id">Client ID:</label>
        <input type="text" class="form-control" id="client_id" name="client_id" required>
      </div>
      <div class="form-group">
        <label for="copy_id">Copy ID:</label>
        <input type="text" class="form-control" id="copy_id" name="copy_id" required>
      </div>
      <div class="form-group">
        <label for="rental_duration">Rental Duration (days):</label>
        <input type="text" class="form-control" id="rental_duration" name="rental_duration"
value="14" required>
      </div>
      <button type="submit" class="btn btn-secondary">Submit Rental</button> <!-- Szary przycisk -->
    </form>
  </div>

  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>

```

Skielet do wyświetlania coppi możliwych do zarezerwoania wraz z przyciskami do rezerwacji - `available_copiers.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
  <title>Available Copies</title>
</head>
<body>
  <div class="container mt-5">
    <a href=".." class="btn btn-primary mb-4">< Powrót do strony głównej</a>
    <h1 class="mb-4">Available Copies</h1>

    <form method="post" class="form-inline mb-4">
      <div class="form-group mr-2">
        <label for="movie_id" class="mr-2">Movie ID:</label>
        <input type="text" id="movie_id" name="movie_id" class="form-control">
      </div>
      <div class="form-group mr-2">
        <label for="movie_name" class="mr-2">Movie Name:</label>
        <input type="text" id="movie_name" name="movie_name" class="form-control">
      </div>
      <button type="submit" class="btn btn-primary mr-2">Search</button>
      <button type="button" class="btn btn-secondary"
onclick="window.location.href='/VW_AVAILABLE_COPIES'">Reset</button>
    </form>

    <table class="table table-striped">
      <thead class="thead-dark">
        <tr>
          <th>Movie ID</th>
          <th>Copy ID</th>
          <th>Movie Title</th>
          <th>Category</th>
          <th>Release Date</th>
          <th>Duration</th>
          <th>Action</th>
        </tr>
      </thead>
      <tbody>
        {% for copy in copies %}
          <tr>
            <td>{{ copy[0] }}</td>
            <td>{{ copy[1] }}</td>
            <td>{{ copy[2] }}</td>
            <td>{{ copy[3] }}</td>
            <td>{{ copy[4].strftime('%Y-%m-%d') }}</td>
            <td>{{ copy[5] }}</td>
            <td>
              <form action="/reserve" method="post" class="d-inline">
                <input type="hidden" name="copy_id" value="{{ copy[1] }}">
                <button type="submit" class="btn btn-success btn-sm">Reserve</button>
              </form>
            </td>
          </tr>
        {% endfor %}
      </tbody>
    </table>
  </div>

  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```

Skielet do wyświetlania filmów z możliwością filtrowania po kategorii - `movie_filter_form.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
  <title>Filter Movies by Category</title>
</head>
<body>
  <div class="container mt-5">
    <a href=".." class="btn btn-primary mb-4">< Powrót do strony głównej</a>
    <h1 class="mb-4">Filter Movies by Category</h1>

    <form action="/filter_movies" method="POST" class="form-inline mb-4">
      <div class="form-group mr-2">
        <label for="category_id" class="mr-2">Select Category:</label>
        <select name="category_id" id="category_id" class="form-control">
          {% for category in categories %}
            <option value="{{ category[0] }}">{{ category[1] }}</option>
          {% endfor %}
        </select>
      </div>
      <button type="submit" class="btn btn-primary mr-2">Filter</button>
      <button type="button" class="btn btn-secondary"
onclick="window.location.href='/filter_movies'">Reset</button>
    </form>

    {% if movies %}
    <h2 class="mb-4">Movies in Selected Category</h2>
    <table class="table table-striped">
      <thead class="thead-dark">
        <tr>
          <th>Category Name</th>
          <th>Movie Title</th>
          <th>Description</th>
          <th>Release Date</th>
          <th>Duration</th>
          <th>Rating</th>
          <th>Director</th>
        </tr>
      </thead>
      <tbody>
        {% for movie in movies %}
        <tr>
          <td>{{ movie[0] }}</td>
          <td>{{ movie[1] }}</td>
          <td>{{ movie[2] }}</td>
          <td>{{ movie[3].strftime('%Y-%m-%d') }}</td>
          <td>{{ movie[4] }}</td>
          <td>{{ movie[5] }}</td>
          <td>{{ movie[6] }}</td>
        </tr>
        {% endfor %}
      </tbody>
    </table>
    {% endif %}
  </div>

  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```


Formularze do operacji CRUD na tabeli `Clients` - `add_client_form.html`, `delete_client_form.html`, `update_client_form.html`

```
<!doctype html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
  <title>Add Client</title>
</head>
<body>
  <a href="..">< Powrót do strony głównej</a>
  <div class="container mt-5">

    <h1 class="mb-4">Add a New Client</h1>
    <form action="{{ url_for('procedures.add_client') }}" method="post">
      <div class="form-group">
        <label for="firstname">First Name:</label>
        <input type="text" class="form-control" id="firstname" name="firstname" required>
      </div>
      <div class="form-group">
        <label for="lastname">Last Name:</label>
        <input type="text" class="form-control" id="lastname" name="lastname" required>
      </div>
      <div class="form-group">
        <label for="address">Address:</label>
        <input type="text" class="form-control" id="address" name="address" required>
      </div>
      <div class="form-group">
        <label for="phone">Phone:</label>
        <input type="text" class="form-control" id="phone" name="phone" required>
      </div>
      <button type="submit" class="btn btn-secondary">Add Client</button> <!-- Szary przycisk -->
    </form>
  </div>

  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```

```
<!doctype html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
  <title>Delete Client</title>
</head>
<body>
  <div class="container mt-5">
    <a href="..">< Powrót do strony głównej</a>
    <h1 class="mb-4">Delete Client</h1>
    <form action="{{ url_for('procedures.delete_client') }}" method="post">
      <div class="form-group">
        <label for="client_id">Client ID:</label>
        <input type="text" class="form-control" id="client_id" name="client_id" required>
      </div>
      <button type="submit" class="btn btn-secondary">Delete Client</button> <!-- Szary przycisk -->
    </form>
  </div>

  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
```

```
</html>

<!doctype html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
  <title>Update Client</title>
</head>
<body>
  <a href="..">< Powrót do strony głównej</a>
  <div class="container mt-5">
    <h1 class="mb-4">Update Client</h1>
    <form action="{ url_for('procedures.update_client') }" method="post">
      <div class="form-group">
        <label for="client_id">Client ID:</label>
        <input type="text" class="form-control" id="client_id" name="client_id" required>
      </div>
      <div class="form-group">
        <label for="firstname">First Name:</label>
        <input type="text" class="form-control" id="firstname" name="firstname" required>
      </div>
      <div class="form-group">
        <label for="lastname">Last Name:</label>
        <input type="text" class="form-control" id="lastname" name="lastname" required>
      </div>
      <div class="form-group">
        <label for="address">Address:</label>
        <input type="text" class="form-control" id="address" name="address" required>
      </div>
      <div class="form-group">
        <label for="phone">Phone:</label>
        <input type="text" class="form-control" id="phone" name="phone" required>
      </div>
      <button type="submit" class="btn btn-secondary">Update Client</button> <!-- Szary przycisk -->
    </form>
  </div>

  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```