Projekt – Bazy danych

System zarządzania kolekcją gier planszowych

Krystian Kościelecki (169427)

Spis treści

Zajęcia 2

1.Określenie tematyki i zakresu projektu

- Temat: System zarządzania kolekcją gier planszowych
- Zakres:
  1. Katalogowanie gier według mechanik (deck-building, worker placment, dice-rolling, trading)
  2. Opis mechanik gry
  3. Filtrowanie gier według wybranych mechanik
  4. Śledzenie historii wydań
  5. Zbieranie recenzji użytkowników

2. Przedstawienie, zagadnień związanych z tematem

Struktura danych

1. **Encje**:
   a. Gry
   b. Mechaniki
   c. Historia wydań
   d. Recenzje
2. **Relacje**:
   a. Gry-mechaniki
   b. Gry – Wydania
   c. Gry – Recenzje
3. **Funkcje**:
   a. Filtrowanie gier
   b. Śledzenie historii wydań
   c. Zbieranie recenzji użytkowników

3. Określenie funkcji bazy danych i ich priorytetu

1. Katalogowanie gier, mechanik i historii wydań:

   Opis:

   System powinien umożliwiać katalogowanie gier, mechanik i ich historii wydań, aby użytkownicy mogli w łatwy sposób znaleźć i poznać szczegółowe informacje o interesujących ich grach.

Priorytet:

Wysoki. Katalogowanie stanowi podstawową funkcję systemu, zapewniając dostęp do wszystkich informacji o grach.

2. Zbieranie i wyświetlanie recenzji:

   Opis:

   Funkcja ta umożliwia użytkownikom dodawanie recenzji gier oraz przeglądanie opinii innych graczy.

   Priorytet:

   Wysoki. Recenzje mają znaczący wpływ na ocenę i wybór gry, dlatego powinny być dostępne dla każdej gry w systemie.

3. Podstawowe operacje CRUD (Create, Read, Update, Delete) dla zarządzania danymi gier:
   Opis:

   System powinien umożliwiać tworzenie, odczytywanie, aktualizowanie i usuwanie danych dotyczących gier, mechanik, historii wydań i recenzji. Te operacje są podstawą zarządzania i utrzymywania aktualnej bazy danych.

   Priorytet:

   Wysoki. Operacje CRUD są niezbędne do zarządzania wszystkimi elementami bazy danych.

4. Filtrowanie gier
   Opis:
   Funkcja umożliwiająca łatwiejsze poruszanie się po spisie gier.

   Priorytet:

   Niski. Funkcja pomagająca użytkownikom poruszanie się po systemie.

4. wybór technologii i rodzaju bazy danych wybór narzędzi

- **Rodzaj Baz Danych:**
  Relacyjna baza danych (RDBMS) ponieważ zawiera uporządkowane dane.

- **Technologia:**
  MySQL jako baza danych, a MySQL Workbench do projektowania, zarządzania i implementacji.

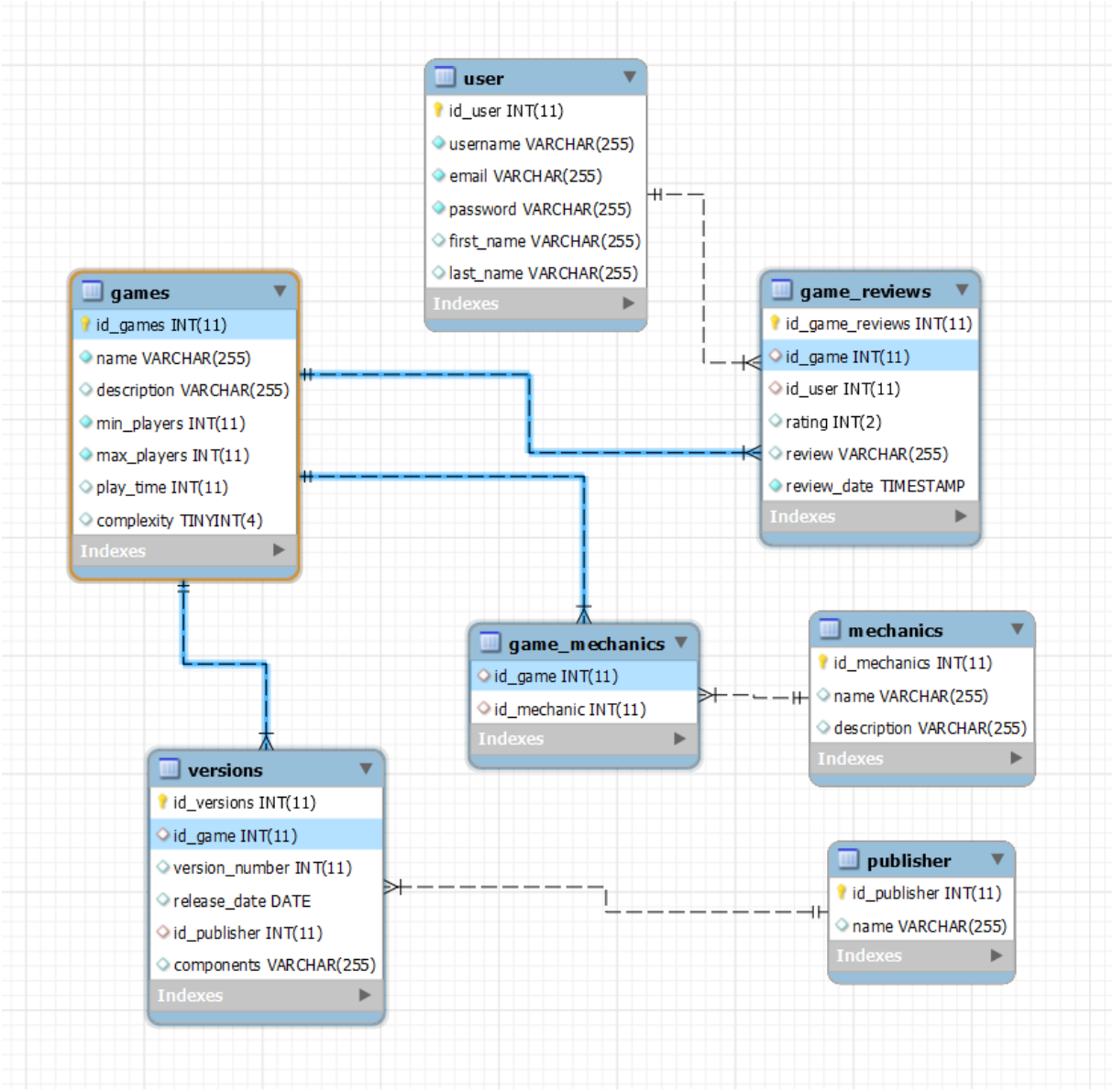
- **Język programowania(opcjonalnie):**
  JavaScript poprzez wykorzystanie frameworku Node.js

5. prezentacja przygotowanego repozytorium z opisem

https://github.com/ermir2001/bazy_danych_projekt/tree/main

Zajęcia 3

6. prezentacja diagramu bazy danych



7. opis tabel bazy danych i ich funkcji

Tabela games

- Funkcja: Przechowuje informacje o grach, w tym ich nazwy, opisy, minimalną i maksymalną liczbę graczy, czas gry oraz stopień złożoności.
- Kolumny:
    - id_games: Unikalny identyfikator gry (klucz główny).
    - name: Nazwa gry.
    - description: Opcjonalny opis gry.
    - min_players: Minimalna liczba graczy.
    - max_players: Maksymalna liczba graczy.
    - play_time: Czas gry w minutach.
    - complexity: Stopień złożoności gry.

Tabela mechanics

- Funkcja: Przechowuje informacje na temat różnych mechanik stosowanych w grach, takich jak zarządzanie zasobami, umieszczanie kafelków itp.
- Kolumny:
    - id_mechanics: Unikalny identyfikator mechaniki (klucz główny).
    - name: Nazwa mechaniki.
    - description: Opcjonalny opis mechaniki.

Tabela game_mechanics

- Funkcja: Jest to tabela łącząca, która tworzy relację wiele-do-wielu między grami a mechanikami. Pozwala to na przypisanie wielu mechanik do jednej gry i na odwrót.
- Kolumny:
    - id_game: Identyfikator gry (klucz obcy).
    - id_mechanic: Identyfikator mechaniki (klucz obcy).
- Relacje:
    - Klucze obce odnoszą się odpowiednio do tabel games i mechanics.

Tabela user

- Funkcja: Przechowuje informacje o użytkownikach, którzy mogą recenzować gry.
- Kolumny:
    - id_user: Unikalny identyfikator użytkownika (klucz główny).
    - username: Nazwa użytkownika.
    - email: Adres email użytkownika.
    - password: Hasło użytkownika.
    - first_name: Imię użytkownika.
    - last_name: Nazwisko użytkownika.

Tabela game_reviews

- Funkcja: Przechowuje recenzje gier dodane przez użytkowników, wraz z ocenami.
- Kolumny:
    - id_game_reviews: Unikalny identyfikator recenzji (klucz główny).
    - id_game: Identyfikator gry, której dotyczy recenzja (klucz obcy).
    - id_user: Identyfikator użytkownika, który dodał recenzję (klucz obcy).
    - rating: Ocena gry.
    - review: Tekst recenzji.
    - review_date: Data dodania recenzji, domyślnie ustawiona na aktualny czas.
- Relacje:
    - Klucze obce odnoszą się odpowiednio do tabel games i user.

Tabela publisher

- Funkcja: Przechowuje informacje o wydawcach gier.
- Kolumny:
    - id_publisher: Unikalny identyfikator wydawcy (klucz główny).
    - name: Nazwa wydawcy.

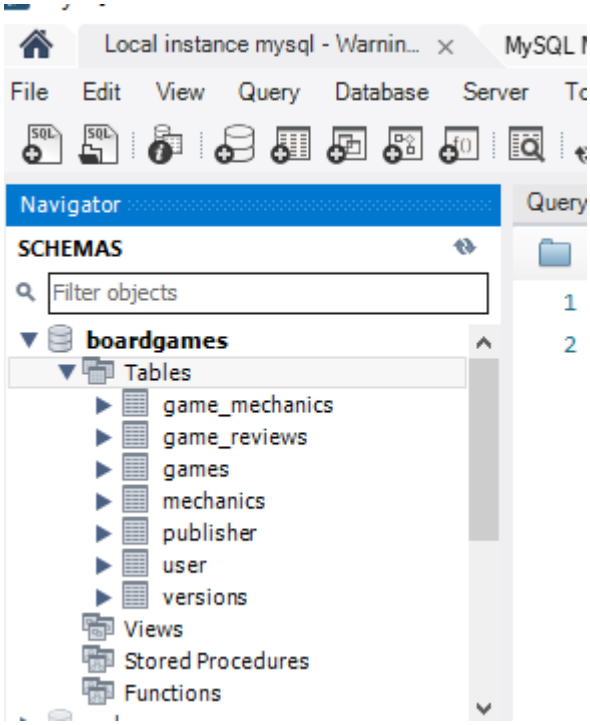
Tabela versions

- Funkcja: Przechowuje informacje o różnych wersjach gier, w tym o numerach wersji, datach wydania i wydawcach.
- Kolumny:

- id_versions: Unikalny identyfikator wersji (klucz główny).
- id_game: Identyfikator gry, której dotyczy wersja (klucz obcy).
- version_number: Numer wersji.
- release_date: Data wydania wersji.
- id_publisher: Identyfikator wydawcy, który wydał wersję (klucz obcy).
- components: Opcjonalny opis komponentów wersji.

- Relacje:
  - Klucze obce odnoszą się odpowiednio do tabel games i publisher.

8. prezentacja wykonania bazy danych (SQL,diagram)



9. proste zapytania SQL (dodanie danych, selekcja)

```
INSERT INTO `boardgames`.`games` (`id_games`, `name`, `description`)
VALUES (1, 'Catan', 'Strategiczna gra planszowa, w której gracze
zdobywają zasoby i budują osady');
```

```sql
-- Zapytanie zwraca nazwy gier, które używają mechaniki "Deck
Building"
SELECT g.name
FROM boardgames.games g
INNER JOIN boardgames.game_mechanics gm ON g.id_games = gm.id_game
INNER JOIN boardgames.mechanics m ON gm.id_mechanic = m.id_mechanics
WHERE m.name = 'Deck Building';
```

```sql
-- Zapytanie zwraca nazwy gier oraz średnią ocen i liczbę ocen,
gdzie średnia ocena jest równa lub wyższa niż 4
SELECT g.name, AVG(gr.rating) AS average_rating, COUNT(gr.rating) AS
number_of_ratings
FROM boardgames.games g
LEFT JOIN boardgames.game_reviews gr ON g.id_games = gr.id_game
GROUP BY g.id_games
HAVING AVG(gr.rating) >= 4;
```

```sql
-- Zapytanie zwraca nazwy użytkowników i liczbę napisanych przez
nich recenzji, gdzie liczba recenzji przekracza 2
SELECT u.username, COUNT(gr.id_game_reviews) AS review_count
FROM boardgames.user u
LEFT JOIN boardgames.game_reviews gr ON u.id_user = gr.id_user
GROUP BY u.id_user
HAVING COUNT(gr.id_game_reviews) > 2;
```

```sql
-- Zapytanie zwraca nazwy gier, których najnowsza wersja została
wydana po 1 stycznia 2010 roku
SELECT g.name, MAX(v.release_date) AS latest_release
FROM boardgames.games g
JOIN boardgames.versions v ON g.id_games = v.id_game
GROUP BY g.id_games
HAVING latest_release > '2010-01-01';
```

Zajęcia 4

10. zaawansowane zapytania SQL (dodawanie, aktualizacja, selekcja danych)

```sql
--Pobranie wszystkich gier wraz z ich recenzjami i użytkownikami,
którzy je ocenili
SELECT g.name AS game_name, r.rating, r.review, u.username
FROM boardgames.games g
JOIN boardgames.game_reviews r ON g.id_games = r.id_game
JOIN boardgames.user u ON r.id_user = u.id_user;
```

```sql
--Pobranie gier, które mają najwyższą średnią ocenę
SELECT g.name, g.description, g.min_players, g.max_players,
g.play_time, g.complexity,
       (SELECT AVG(rating)
        FROM boardgames.game_reviews gr
        WHERE gr.id_game = g.id_games) AS average_rating
```

```
FROM boardgames.games g
ORDER BY average_rating DESC;
```

```
--Lista mechanik wraz z ilością gier które posiadają taką mechanikę
SELECT m.id_mechanics, m.name AS mechanic_name,
       (SELECT COUNT(*)
        FROM boardgames.game_mechanics gm
        WHERE gm.id_mechanic = m.id_mechanics) AS game_count
FROM boardgames.mechanics m
ORDER BY game_count DESC;
```

```
--Wprowadzenie danych do tabeli publisher
INSERT INTO `boardgames`.`publisher` (`id_publisher`, `name`) VALUES
(1, 'Column2')
```

```
--Wprowadzenie danych do tabeli versions
INSERT INTO `boardgames`.`versions` (`id_versions`, `id_game`,
`version_number`, `release_date`, `id_publisher`, `components`)
VALUES
(1, 1, 1, '2020-01-01', 2, 'Board, resource cards, settlements,
roads, cities, dice');
```

```
--Zmiana opisu komponentów gry
UPDATE `boardgames`.`versions` SET `components` = 'Plansza, karty
zasobów, osady, drogi, miasta, kostki' WHERE `id_versions` = 1;
```

```
--Modyfikacja tabeli games poprzez dodanie min i max
players
ALTER TABLE `boardgames`.`games`
ADD COLUMN `min_players` INT(11) NOT NULL DEFAULT 1,
ADD COLUMN `max_players` INT(11) NOT NULL DEFAULT 1;
```

```
--Modyfikacja tabeli games poprzez dodanie min_players, max_players,
play_time i complexity
ALTER TABLE `boardgames`.`games`
ADD COLUMN `min_players` INT(11) NOT NULL DEFAULT 1,
ADD COLUMN `max_players` INT(11) NOT NULL DEFAULT 1,
ADD COLUMN `play_time` INT(11) NULL DEFAULT NULL,
ADD COLUMN `complexity` DECIMAL(3,2) NULL DEFAULT NULL;
```

```
--Modyfikacja tabeli versions poprzez dodanie components
ALTER TABLE `boardgames`.`versions`
ADD COLUMN `components` VARCHAR(255) NULL DEFAULT NULL;
```

11. prezentacja użytkowników BD i ich ról

```
                    --Tworzenie użytkownika
    CREATE USER 'tester'@'localhost' IDENTIFIED BY 'tester';
```

```
            --Pełne uprawnienia do konkretnej bazy danych
  GRANT ALL PRIVILEGES ON boardgames.* TO 'tester'@'localhost';
```

```
--Ograniczone uprawnienia tylko do odczytu
GRANT SELECT ON boardgames.* TO 'tester'@'localhost';
```

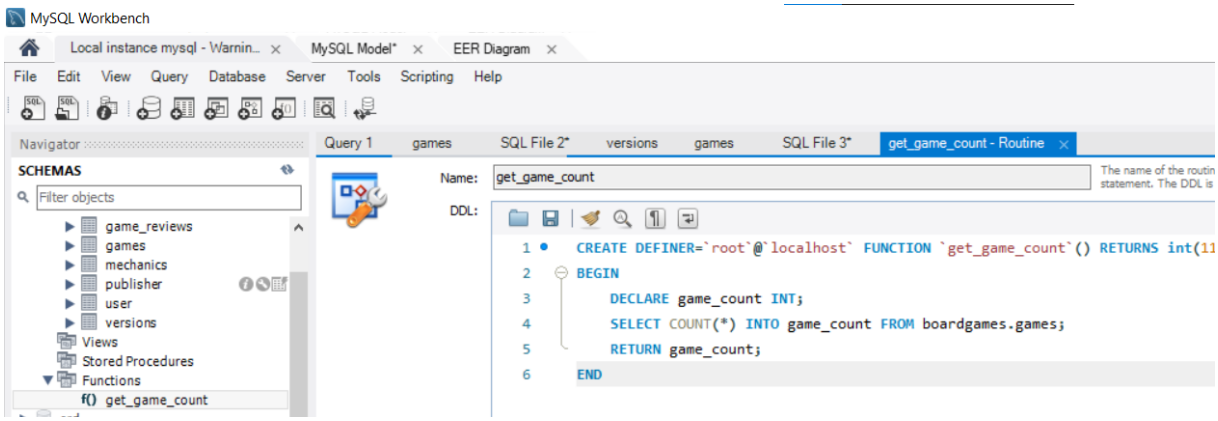
```
--Zastosowanie zmian
FLUSH PRIVILEGES;
```

```
--Wyświetlanie użytkowników i ich uprawnień
SELECT user, host FROM mysql.user;
```

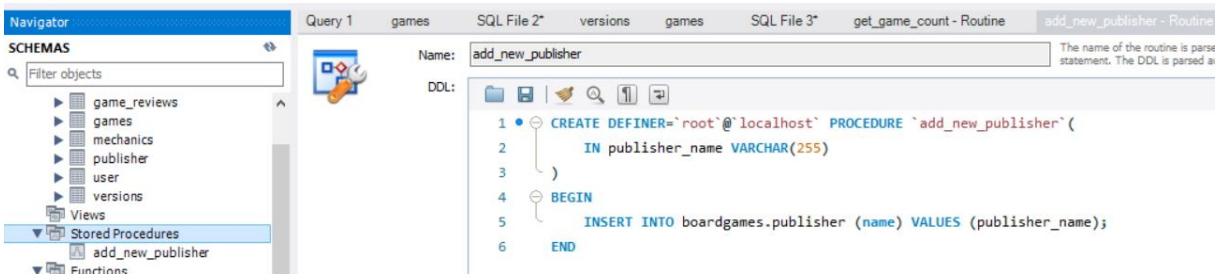
```
--Wyświetlanie uprawnień dla konkretnego użytkownika
SHOW GRANTS FOR 'new_user'@'localhost';
```

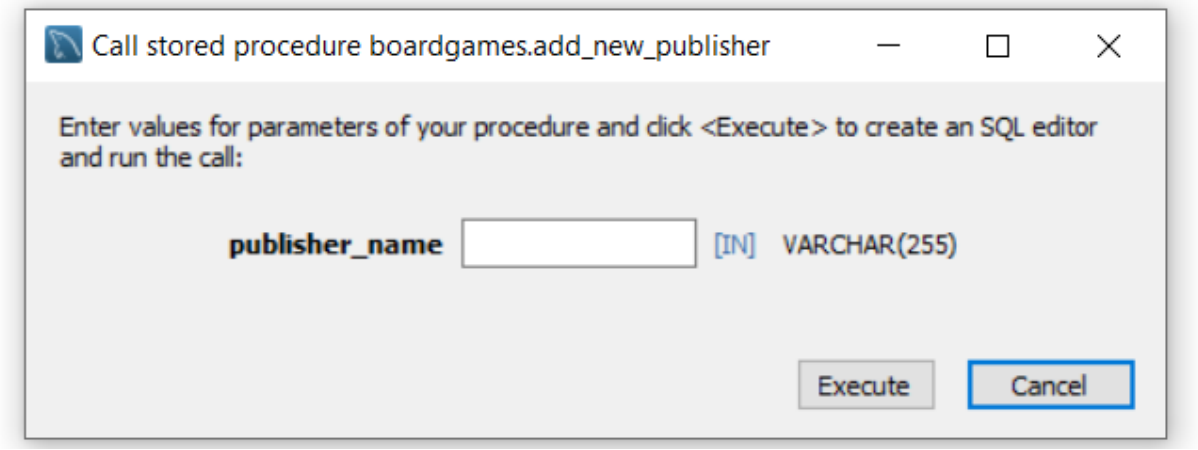
12. funkcje i procedury obsługujące BD

```
--Przykładowa Funkcja liczy wszystkie rekordy w tabeli games i
zapisuje wynik w zmiennej game_count

CREATE FUNCTION boardgames.get_game_count()
RETURNS INT
BEGIN
DECLARE game_count INT;
SELECT COUNT(*) INTO game_count FROM boardgames.games;
RETURN game_count;
END
```



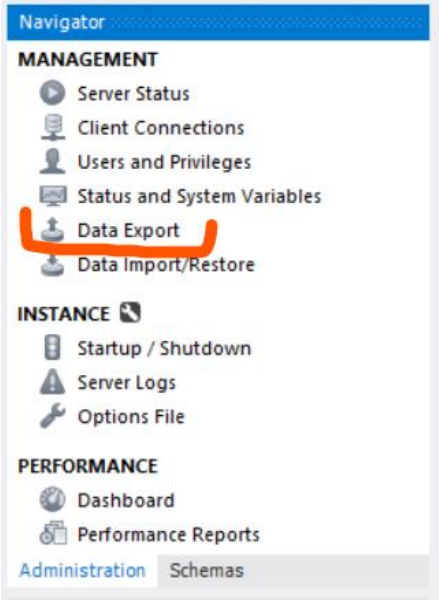

```sql
--Przykładowa Procedura

--Deklaracja parametru wejściowego

--Wstawienie nowego rekordu do tabeli publisher z wartością

publisher_name.

CREATE DEFINER=`root`@`localhost` PROCEDURE `add_new_publisher`(

    IN publisher_name VARCHAR(255)

)

BEGIN

    INSERT INTO boardgames.publisher (name) VALUES (publisher_name);

END
```

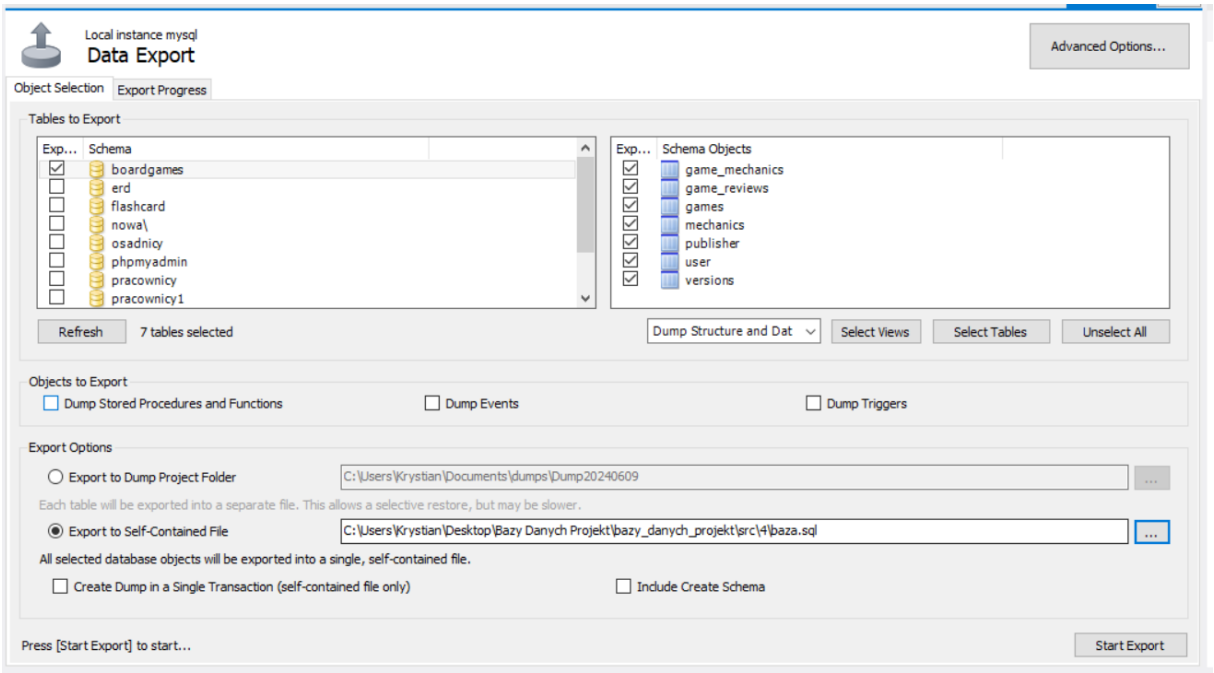




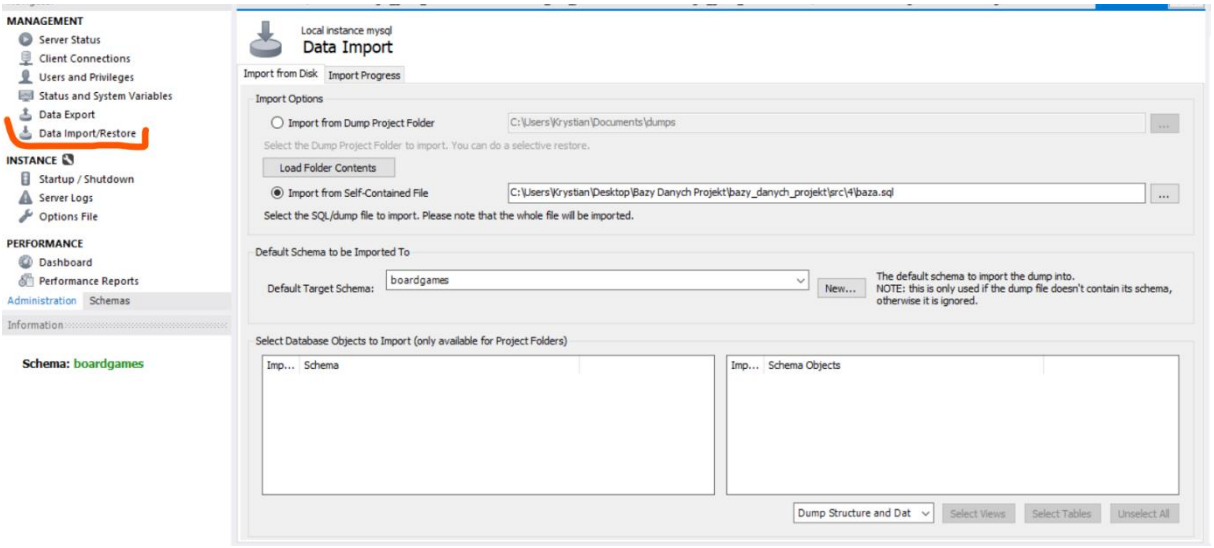
13. zarządzanie bazą danych (backup, restore)

zarządzanie bazą danych backup

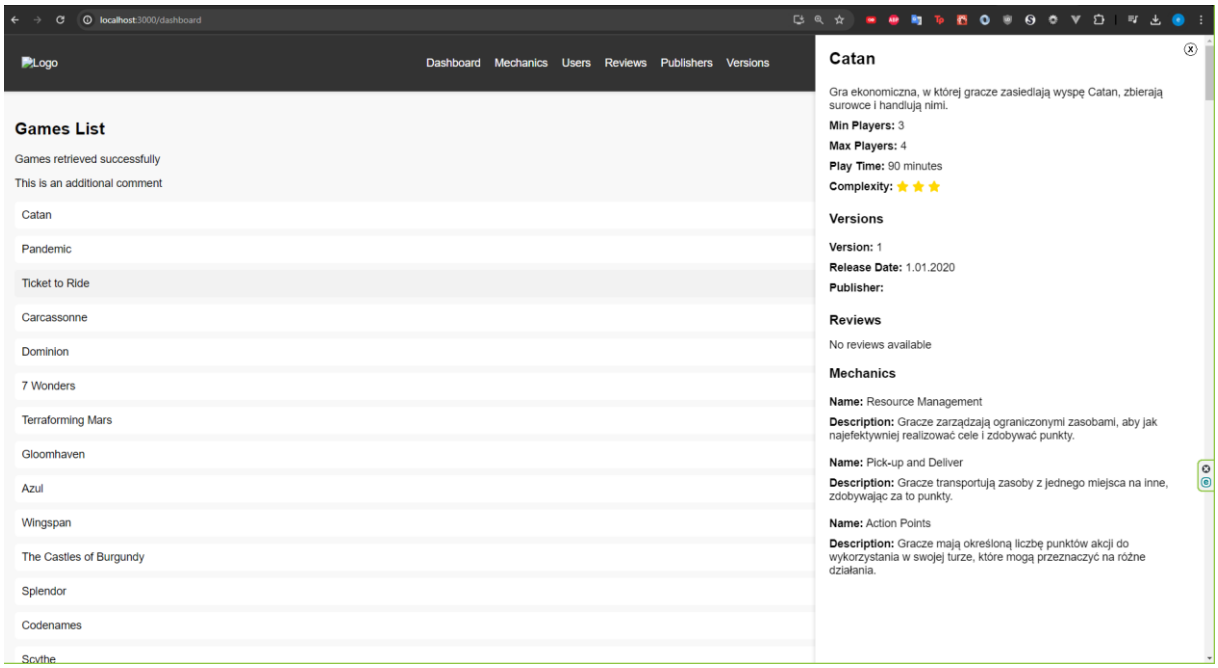




zarządzanie bazą danych restore



14. wariant testowy i na produkcji




AuthModal.js U
ErrorPage.js U
Nav.js U
server.js
Mechanics.js U
Users.js U
Reviews.js U
server > server.js > ...
70
71 // Endpoint do pobierania wszystkich mechanik
72 app.get('/api/mechanics', async (req, res) => {
73 try {
74 const mechanics = await db.Mechanic.findAll();
75 res.json({
76 message: 'Mechanics retrieved successfully',
77 data: mechanics
78 });
79 } catch (err) {
80 res.status(500).send(err);
81 }
82 });
83
84 // Endpoint do pobierania wszystkich użytkowników
85 app.get('/api/users', async (req, res) => {
86 try {
87 const users = await db.User.findAll();
88 res.json({
89 message: 'Users retrieved successfully',
90 data: users
91 });
92 } catch (err) {
93 res.status(500).send(err);
94 }

15. przykład zastosowania ORM

```
module.exports = (sequelize, DataTypes) => {
    const Game = sequelize.define('Game', {
      id_games: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
      },
      name: DataTypes.STRING,
      description: DataTypes.STRING,
      min_players: {
        type: DataTypes.INTEGER,
        defaultValue: 1
      },
      max_players: {
        type: DataTypes.INTEGER,
        defaultValue: 1
      },
      play_time: DataTypes.INTEGER,
      complexity: DataTypes.DECIMAL(3, 2)
    }, {
      tableName: 'games',
      timestamps: false
    });

    return Game;
```