Relacyjne systemy zarządzania bazami danych

Sprawozdanie z projektu, Zaawansowane Systemy Baz Danych

Jakub Matłacz
Grupa Czwartkowa 12:15

Data ostatniej zmiany: 29.10.23

Spis treści

Część I

Studium przypadku

1 Temat projektu

Rewolucyjna usługa Cloud Bot Rental: rewolucja w świecie gier, która zdefiniuje wirtualną rozrywkę. Jak to działa: użytkownicy zakładają konto i uzyskują dostęp do wielu botów, z których każdy może grać w jedną z obsługiwanych gier komputerowych. Ale to nie są zwykłe boty. To dynamiczne jednostki, które nieustannie ewoluują i dostosowują się, naśladując zachowanie ludzkich graczy w wirtualnym świecie. To, co wyróżnia naszą usługę, to jej meta charakter. Te boty nie tylko replikują ludzką rozgrywkę, ale także aktywnie wpływają na swoje własne rozwinięcie w grze. Kiedy boty biorą udział w meczach w różnych grach, użytkownicy mogą monitorować ich postępy. Właściciele mają możliwość dostosowywania cech, kształtując umiejętności i strategie bota w czasie rzeczywistym. Im więcej bot gra, tym więcej się uczy, a użytkownik doświadcza satysfakcji - podróż zatopiona w dopaminie. W istocie usługa Cloud Bot Rental nie polega tylko na graniu w gry. Chodzi o kształtowanie przyszłości gier AI. To meta gra, w której celem nie jest tylko wygrana, ale ciągłe ulepszanie umiejętności botów. To podróż, w której użytkownicy doświadczają dreszczyku emocji związanego z ewolucją i postępem w każdym rozegranym meczu.

2 Diagram relacji encji

Diagram ERD widoczny jest na rysunku 1.

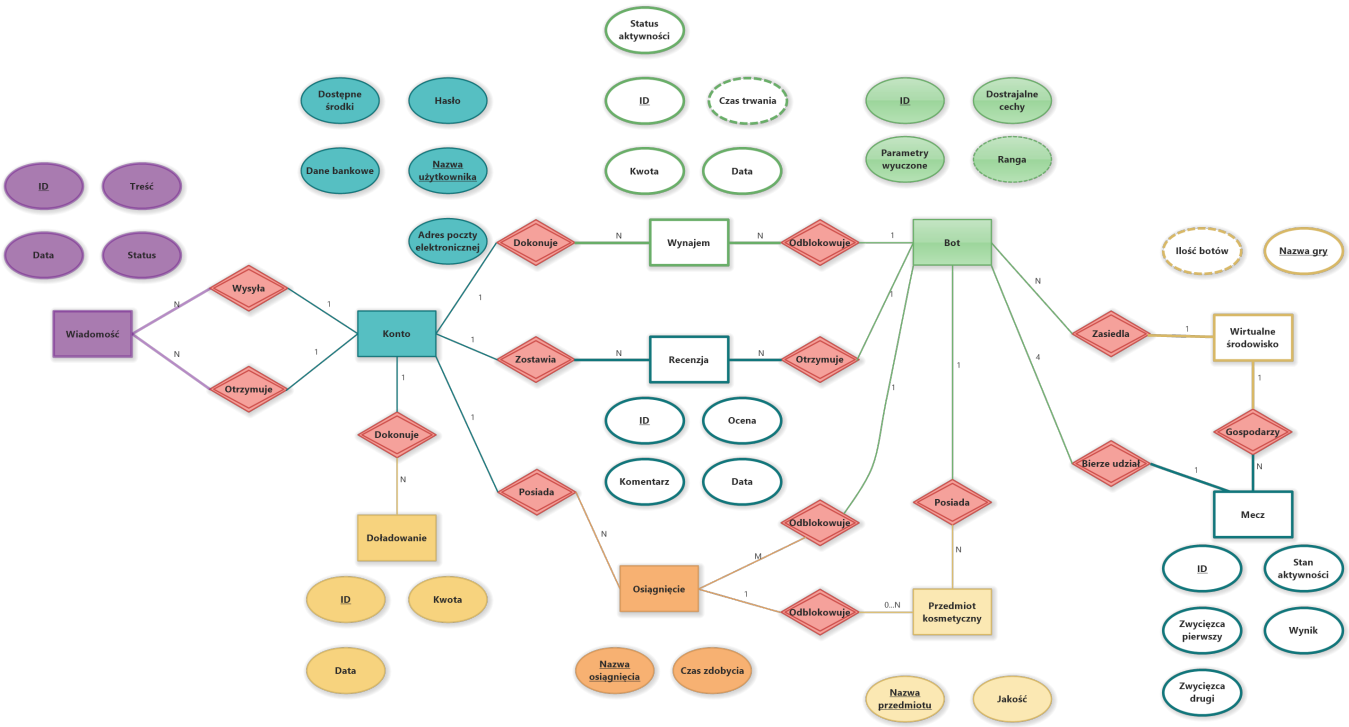
Encje:

1. Konto – użytkownik podaje dane osobiste i bankowe. Na początku nie posiada dostępnych środków.
2. Doładowanie – Konto może zwiększyć swoje środki, które użyje na wynajem bota. Historia doładowań.
3. Wynajem – historyczne i aktywne wynajmy botów za środki z doładowań.
4. Bot – przypisany jest do pojedynczej gry, której się uczy. Użytkownik może zmieniać jego hiperparametry i cieszyć się postępem rangi. Bot nie może zostać zniszczony. W przypadku nieprzedłużenia subskrypcji przechodzi z rąk do rąk bez zmian w jego parametrach. Może zostać utworzony nowy.
5. Wirtualne środowisko – odpowiada światowi danej gry. Hostuje mecze i graczy.
6. Mecz – historyczne i aktywne mecze botów. Wpływają na rangę.
7. Przedmiot kosmetyczny – lista dostępnych meta-przedmiotów dla lepszej prezentacji bota przed znajomymi wynajmującego. Posiada dodatkowe cechy i jakość.
8. Osiągnięcie – lista meta-osiągnięć. Niektórym z nich odpowiada przedmiot ale nie wszystkim.

Relacje:

1. Konto – Dokonuje – Wynajem – jedno konto może mieć wiele wynajmów, czyli wiele botów.
2. Konto – Posiada – Osiągnięcie – jedno konto ma wiele osiągnięć.
3. Konto – Dokonuje – Doładowanie – jedno konto ma wiele doładowań.
4. Wynajem – Odblokowuje – Bot – jeden wynajem to jeden bot.
5. Bot – Odblokowuje – Osiągnięcie – jeden bot ma wiele osiągnięć.

6. Osiągnięcie – Odblokowuje – Przedmiot kosmetyczny – niektóre osiągnięcia, ale nie wszystkie, dlatego tabela niezależna.
7. Bot – Posiada – Przedmiot kosmetyczny – jeden bot ma wiele.
8. Bot – Zasiedla – Wirtualne środowisko – wiele botów jedno środowisko.
9. Bot – Bierze udział – Mecz – cztery boty na jeden mecz.
10. Wirtualne środowisko – Gospodarzy – Mecz – jedno środowisko ma wiele meczy.



Rysunek 1: Diagram relacji encji

3 Wybrana baza, instalacja i konfiguracja

Wybrano bazę SQL Server ze względu na interfejs graficzny, dostępność zasobów, przyzwyczajenie. Pobrano plik instalacyjny ze strony microsoft i przeprowadzono normalną instalację. Następnie połączono się z bazą w terminalu.

Wyświetlono właściwości serwera, korzystając z widoku katalogu sys.configurations. Zmieniano właściwości serwera za pomocą sp_configure. Spośród opcji warto wymienić: user connections, allow updates, remote access, max worker threads, network packet size (B).

4 Użytkownicy i usługi

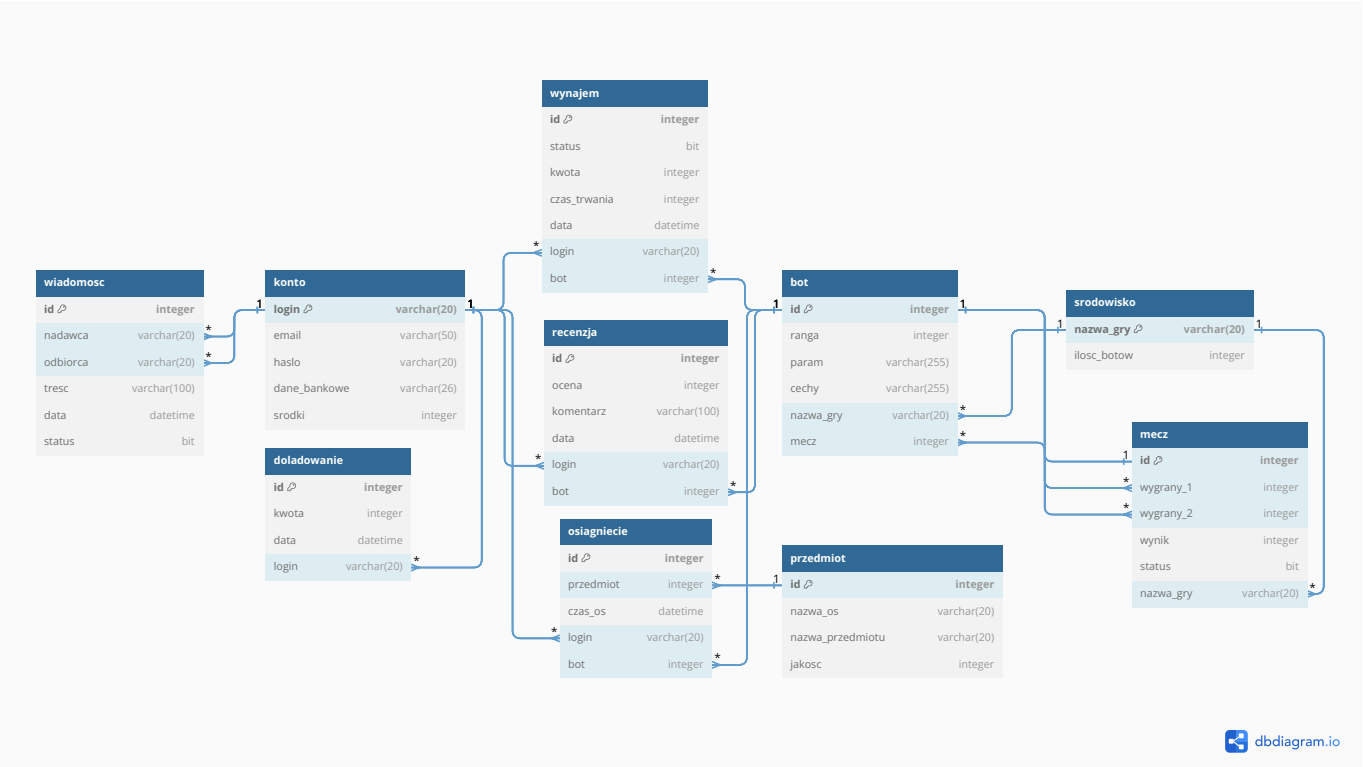
Użytkownikami będzie wynajmujący bota oraz developer. Usługi, którym będą odpowiadać zapytania SQL pobierające potrzebne dane.

1. Utworzenie konta.
2. Doładowywanie konta.
3. Wynajmowanie botów.
4. Monitorowanie wynajmów.

5. Zarządzanie botami.
6. Oglądanie osiągnięć.
7. Oglądanie przedmiotów botów.

5 Schemat logiczny bazy danych

Na rysunku 2 przedstawiono schemat logiczny bazy danych.



Rysunek 2: Schemat logiczny bazy danych

Część II

Etap pierwszy projektu

6 Baza danych w Trzeciej Postaci Normalnej

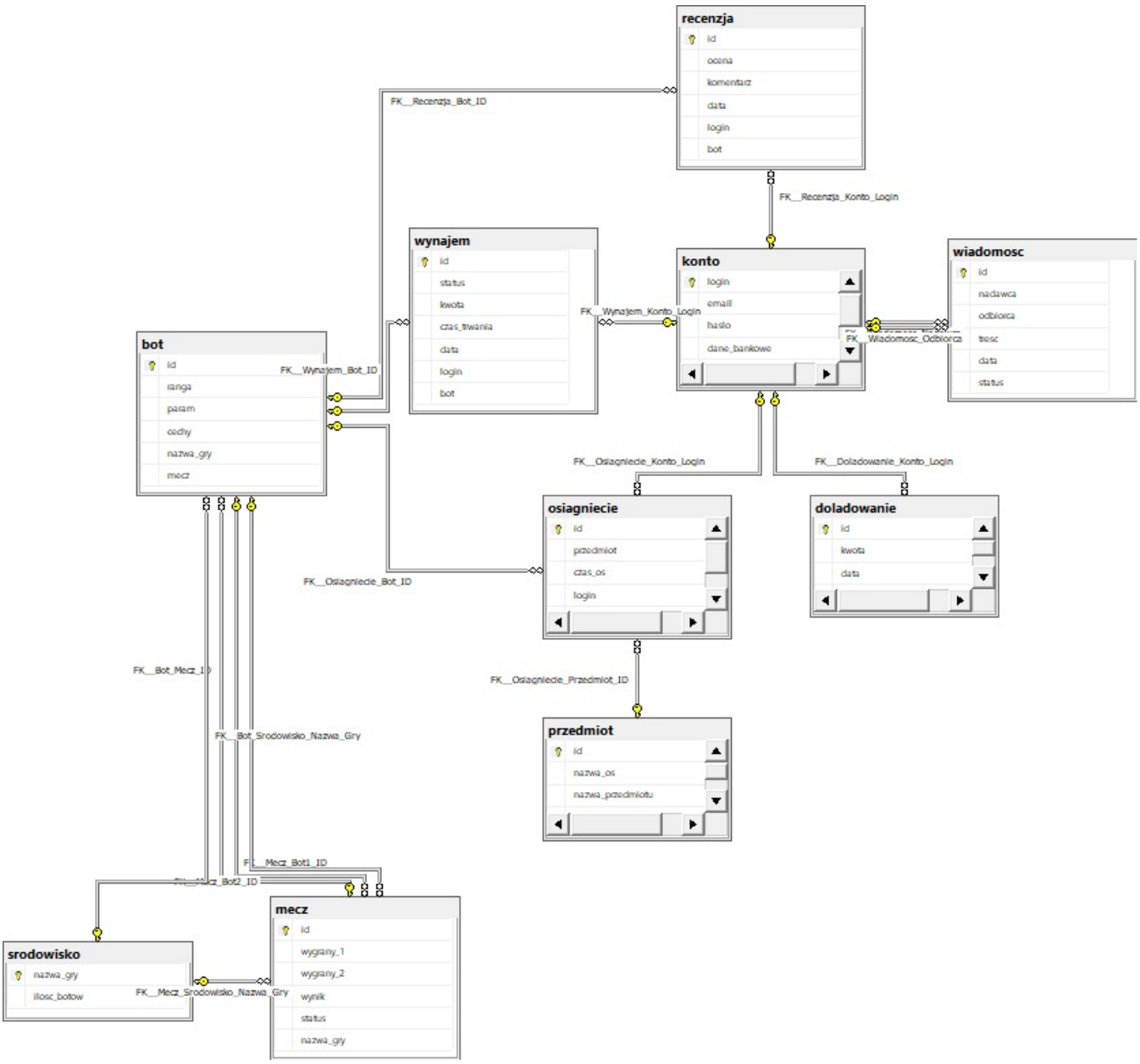
6.1 Struktura bazy danych

Strukturę bazy danych widać na obrazku 2 oraz na obrazku 3.

Używano schematu "dbo" w SQL Server odnoszącego się do domyślnego schematu, który jest używany jako domyślne miejsce dla obiektów baz danych, takich jak tabele, widoki, procedury składowane. Domyślny schemat jest nazywany "dbo", co oznacza "Database Owner" (Właściciel Bazy Danych).

Dzięki spełnieniu poniższych warunków, baza danych jest w trzeciej postaci normalnej, co oznacza, że jest zoptymalizowana pod kątem struktury danych i minimalizuje redundancję, zapewniając efektywne przechowywanie i manipulację danymi.

1. **Wszystkie tabele posiadają klucz główny**: Każda tabela ma zdefiniowany klucz główny, co eliminuje redundancję danych i pozwala na jednoznaczne identyfikowanie rekordów.



Rysunek 3: Diagram wygenerowany programem SSMS

2. **Brak powiązań funkcyjnych na podstawie częściowego klucza**: Wszystkie kolumny w tabelach zależą tylko od klucza głównego, a nie od częściowego klucza. Nie ma sytuacji, gdzie częściowy klucz determinuje wartości innych kolumn.

3. **Brak tranzytywnych zależności funkcyjnych**: Żadna kolumna nie jest zależna od innej kolumny, która nie jest kluczem głównym. Nie ma sytuacji, gdzie jedna kolumna determinuje wartość innej kolumny, która z kolei determinuje wartość innej kolumny.

Tabele Słownikowe:

1. konto
2. srodowisko
3. przedmiot

Tabele Asocjacyjne:

1. bot

2. wynajem
3. mecz
4. osiagniecie
5. doladowanie
6. recenzja
7. wiadomosc

6.2 Uzasadnienie istotnych decyzji

Te decyzje projektowe zostały podjęte w celu stworzenia spójnej, wydajnej i bezpiecznej bazy danych, zdolnej do obsłużenia wymagań aplikacji.

1. **Podział na Tabele:**
   - Każda tabela (Konto, Bot, Mecz itp.) reprezentuje jedną główną encję, co ułatwia zarządzanie danymi i zapewnia integralność danych.
2. **Klucze i Indeksy:**
   - Stosowanie unikalnych kluczy głównych (Primary Keys) w każdej tabeli zapewnia jednoznaczność identyfikacji rekordów.
   - Użycie kluczy obcych (Foreign Keys) umożliwia powiązania między tabelami, ułatwiając zapytania dotyczące danych powiązanych.
3. **Optymalizacja Typów Danych:**
   - Zastosowanie VARCHAR dla pól, takich jak login, email, czy parametry, pomaga w optymalizacji miejsca na dysku i przyspiesza zapytania.
   - INTEGER został użyty do przechowywania liczb całkowitych, co jest efektywne pod względem przechowywania i manipulacji danymi.
4. **Domyślne Wartości i NOT NULL:**
   - Używanie domyślnych wartości (DEFAULT) pomaga unikać pustych pól, zapewniając kompletność danych.
   - NOT NULL jest używane tam, gdzie pole musi mieć wartość, co przyczynia się do spójności danych.
5. **Relacje Many-to-One i One-to-One:**
   - Relacje many-to-one, np. między tabelą "wynajem" a "konto", umożliwiają przypisanie wielu wynajmów do jednego konta użytkownika.
   - Relacje one-to-one, jak w tabeli "recenzja", zapewniają, że każda recenzja jest przypisana do jednego użytkownika (konto) i jednego bota.
6. **Indeksowanie dla Szybkiego Dostępu:**
   - Indeksy mogą być dodane dla pól często używanych w zapytaniach, przyspieszając operacje wyszukiwania i złączania danych.
7. **Normalizacja Danych:**
   - Normalizacja eliminuje nadmiarowość danych i redukuje redundancję, co przyczynia się do spójności danych i wydajności.
8. **Bezpieczeństwo:**

- Hasła są przechowywane jako VARCHAR, ale powinny być odpowiednio zabezpieczone, na przykład przez hashowanie, dla zapewnienia bezpieczeństwa.

9. **Rozszerzalność:**
   - Użycie IDENTITY dla kluczy głównych ułatwia automatyczne przydzielanie unikalnych identyfikatorów, co ułatwia dodawanie nowych rekordów.
10. **Opisujące Nazwy Pól i Tabel:**
    - Starannie dobrane nazwy tabel, pól i kluczy pomagają w zrozumieniu struktury bazy danych nawet osobom, które nie brały udziału w projektowaniu.

7 Wypełnienie bazy danymi

Wypełniono bazę sztucznymi danymi korzystając z systemu LLM. Następnie zweryfikowano ręcznie. Każda tabela ma średnio po około 20 rekordów.

8 Potrzebni użytkownicy

```
CREATE LOGIN AdminUser WITH PASSWORD = 'AdminPassword';
CREATE LOGIN BotUser WITH PASSWORD = 'BotPassword';
CREATE LOGIN RegularUser WITH PASSWORD = 'RegularPassword';
GO

USE wynajem_db;
GO

CREATE USER AdminUser FOR LOGIN AdminUser;
CREATE USER BotUser FOR LOGIN BotUser;
CREATE USER RegularUser FOR LOGIN RegularUser;
GO
```

Listing 1: Utworzenie użytkowników

```
USE wynajem_db;
GO

-- Assign admin privileges to the AdminUser
ALTER ROLE db_owner ADD MEMBER AdminUser;
GO

CREATE ROLE RestrictedExecutorBotUser;
GRANT EXECUTE ON [dbo].[CreateBot] TO RestrictedExecutorBotUser;
GRANT SELECT ON [dbo].[BotDashboard] TO RestrictedExecutorBotUser;
ALTER ROLE RestrictedExecutorBotUser ADD MEMBER BotUser;
GO

CREATE ROLE RestrictedExecutorRegularUser;
GRANT EXECUTE ON [dbo].[RentBot] TO RestrictedExecutorRegularUser;
GRANT EXECUTE ON [dbo].[SendMessage] TO RestrictedExecutorRegularUser;
GRANT EXECUTE ON [dbo].[ViewTransactionHistory] TO RestrictedExecutorRegularUser;
GRANT SELECT ON [dbo].[UserDashboard] TO RestrictedExecutorRegularUser;
ALTER ROLE RestrictedExecutorRegularUser ADD MEMBER RegularUser;
GO
```

Listing 2: Nadanie uprawnień użytkownikom

8.1 Testowanie uprawnień

- "AdminUser" powinien mieć wszelkie uprawnienia.
- "RegularUser" powinien mieć dostęp jedynie do procedur "RentBot" i "SendMessage" oraz do widoku "UserDashboard".
- "BotUser" powinien być w stanie jedynie użyć procedury "CreateBot" i mieć dostęp do widoku "BotDashboard".

8.1.1 Uprawnienia AdminUser

```
DECLARE @login varchar(20) = 'monkeyDluffy'
DECLARE @newEmail varchar(50) = 'funky@lake.pl'
DECLARE @newPassword varchar(20) = 'strawhat'
DECLARE @newDaneBankowe varchar(26) = '12345678901234567890123456'
EXEC UpdateUserInformation @login, @newEmail, @newPassword, @newDaneBankowe
GO

Timestamp Message
[12:38:08 PM] Started executing query at Line 40

  Commands completed successfully.

  Total execution time: 00:00:00.006
```

Listing 3: Uprawnienia AdminUser do procedury UpdateUserInformation

```
ALTER ROLE db_owner ADD MEMBER AdminUser;
GO

Timestamp Message
[12:39:40 PM] Started executing query at Line 5

  Commands completed successfully.

  Total execution time: 00:00:00.001
```

Listing 4: Uprawnienia AdminUser do nadania sobie roli db_owner

8.1.2 Uprawnienia RegularUser

```
DECLARE @sender varchar(20) = 'adventureSeeker'
DECLARE @recipient varchar(20) = 'fitnessFanatic'
DECLARE @messageText varchar(100) = 'Hello, world!'
EXEC SendMessage @sender, @recipient, @messageText
GO

Timestamp Message
[12:35:40 PM] Started executing query at Line 20

  (1 row affected)

  (1 row affected)

  Total execution time: 00:00:00.004
```

Listing 5: Uprawnienia RegularUser do procedury SendMessage

```
SELECT TOP (1000) [RentalID]
      ,[RentalStatus]
      ,[RentalAmount]
      ,[RentalDuration]
      ,[RentalDate]
      ,[BotID]
      ,[GameName]
      ,[BotRank]
      ,[EnvironmentBotCount]
      ,[AchievementID]
      ,[AchievementTime]
      ,[AchievementItemID]
      ,[MessageID]
      ,[Sender]
      ,[MessageContent]
      ,[MessageDate]
      ,[MessageStatus]
  FROM [wynajem_db].[dbo].[UserDashboard]

Timestamp Message
[12:36:06 PM] Started executing query at Line 1

  (42 rows affected)

  Total execution time: 00:00:00.019
```

Listing 6: Uprawnienia RegularUser do widoku UserDashboard

```
DECLARE @login varchar(20) = 'monkeyDluffy'
DECLARE @newEmail varchar(50) = 'funky@lake.pl'
DECLARE @newPassword varchar(20) = 'strawhat'
DECLARE @newDaneBankowe varchar(26) = '12345678901234567890123456'
EXEC UpdateUserInformation @login, @newEmail, @newPassword, @newDaneBankowe
GO

Timestamp Message
[12:36:34 PM] Started executing query at Line 40

  Msg 229, Level 14, State 5, Procedure UpdateUserInformation, Line 1
The EXECUTE permission was denied on the object 'UpdateUserInformation', database
    'wynajem_db', schema 'dbo'.

  Total execution time: 00:00:00
```

Listing 7: Brak uprawnień RegularUser do procedury UpdateUserInformation

8.1.3 Uprawnienia BotUser

```
DECLARE @ranga integer = 1;
DECLARE @param varchar(255) = '1 2 8 9 10 100 43 1';
DECLARE @cechy varchar(255) = '1 82 1 11 1 223 1';
DECLARE @nazwa_gry varchar(20) = 'FoodFiesta';
DECLARE @ilosc_botow integer = 1;
DECLARE @login varchar(20) = 'puzzleSolver';
DECLARE @wygrany_1 integer = 13;
DECLARE @wygrany_2 integer = 15;
EXEC CreateBot @ranga, @param, @cechy, @nazwa_gry, @ilosc_botow, @login,
   @wygrany_1, @wygrany_2;
GO

Timestamp Message
[12:27:39 PM] Started executing query at Line 69
```

```
14
15   Commands completed successfully.
16
17   Total execution time: 00:00:00.008
```

Listing 8: Uprawnienia BotUser do procedury CreateBot

```
1  SELECT TOP (1000) [MatchID]
2        ,[Winner1]
3        ,[Winner2]
4        ,[MatchOutcome]
5        ,[AchievementTime]
6        ,[AchievementID]
7    FROM [wynajem_db].[dbo].[BotDashboard]
8
9  Timestamp Message
10 [12:31:31 PM] Started executing query at Line 1
11
12   (40 rows affected)
13
14   Total execution time: 00:00:00.004
```

Listing 9: Uprawnienia BotUser do widoku BotDashboard

```
1  DECLARE @login varchar(20) = 'monkeyDluffy'
2  DECLARE @newEmail varchar(50) = 'funky@lake.pl'
3  DECLARE @newPassword varchar(20) = 'strawhat'
4  DECLARE @newDaneBankowe varchar(26) = '12345678901234567890123456'
5  EXEC UpdateUserInformation @login, @newEmail, @newPassword, @newDaneBankowe
6  GO
7
8  Timestamp Message
9  [12:33:41 PM] Started executing query at Line 40
10
11   Msg 229, Level 14, State 5, Procedure UpdateUserInformation, Line 1
12 The EXECUTE permission was denied on the object 'UpdateUserInformation', database
       'wynajem_db', schema 'dbo'.
13
14   Total execution time: 00:00:00
```

Listing 10: Brak uprawnień BotUser do procedury UpdateUserInformation

9 Dokumentacja zapytań SQL

Zaimplementowano kilka zapytań SQL w formie procedur. Prawa do ich wykonywania mają określeni użytkownicy. Zapytania te spełniają potrzebne funkcjonalności.

9.1 CreateNewUserAccount

Opis

Procedura składowana 'CreateNewUserAccount' jest odpowiedzialna za tworzenie nowych kont użytkowników w systemie. Przyjmuje jako parametry login użytkownika, adres e-mail, hasło oraz dane bankowe. Procedura sprawdza, czy podany login jest unikalny w tabeli 'konto'. Jeśli jest unikalny, procedura haszuje podane hasło za pomocą algorytmu SHA-256 i dodaje nowego użytkownika do tabeli 'konto' wraz z danymi wejściowymi.

Parametry

- `@login` - Login użytkownika (varchar(20)).

- `@email` - Adres e-mail użytkownika (varchar(50)).
- `@password` - Hasło użytkownika (varchar(20)).
- `@dane_bankowe` - Dane bankowe użytkownika (varchar(26)).

Działanie

1. Sprawdza, czy podany login użytkownika jest unikalny w tabeli ‘konto‘.
2. Jeśli login jest unikalny, procedura haszuje podane hasło przy użyciu algorytmu SHA-256.
3. Dodaje nowego użytkownika do tabeli ‘konto‘ z podanymi danymi oraz zabezpieczonym hasłem.
4. Zwraca status operacji: 1 w przypadku powodzenia, 0 w przypadku niepowodzenia.

Zwracane Wartości

- `Status` - 1, jeśli konto zostało pomyślnie utworzone; 0, jeśli operacja się nie powiodła (int).
- `Message` - Komunikat informujący o statusie operacji (varchar(max)).

Przykład Użycia

```
EXEC CreateNewUserAccount
    @login = 'janedoe',
    @email = 'jane.doe@example.com',
    @password = 'securepassword123',
    @dane_bankowe = '12345678901234567890123456';
```

9.2 UpdateUserInformation

Opis

Procedura składowana ‘UpdateUserInformation‘ umożliwia aktualizację informacji użytkownika w systemie. Przyjmuje jako parametry login użytkownika, nowy adres e-mail, nowe hasło oraz nowe dane bankowe. Procedura sprawdza, czy użytkownik o podanym loginie istnieje w tabeli ‘konto‘. Jeśli istnieje, procedura haszuje nowe hasło za pomocą algorytmu SHA-256 i aktualizuje informacje użytkownika w tabeli ‘konto‘.

Parametry

- `@login` - Login użytkownika, którego informacje mają zostać zaktualizowane (varchar(20)).
- `@newEmail` - Nowy adres e-mail użytkownika (varchar(50)).
- `@newPassword` - Nowe hasło użytkownika (varchar(20)).
- `@newDaneBankowe` - Nowe dane bankowe użytkownika (varchar(26)).

Działanie

1. Sprawdza, czy użytkownik o podanym loginie istnieje w tabeli ‘konto‘.
2. Jeśli użytkownik istnieje, procedura haszuje nowe hasło przy użyciu algorytmu SHA-256.
3. Aktualizuje adres e-mail, hasło i dane bankowe użytkownika w tabeli ‘konto‘.
4. Zwraca status operacji: 1 w przypadku powodzenia, 0 w przypadku niepowodzenia.

Zwracane Wartości

- `Status` - 1, jeśli informacje użytkownika zostały pomyślnie zaktualizowane; 0, jeśli operacja się nie powiodła (int).
- `Message` - Komunikat informujący o statusie operacji (varchar(max)).

Przykład Użycia

```
EXEC UpdateUserInformation
    @login = 'janedoe',
    @newEmail = 'jane.doe@example.com',
    @newPassword = 'newsecurepassword456',
    @newDaneBankowe = '65432109876543210987654321';
```

9.3 CreateBot

Opis

Procedura składowana 'CreateBot' umożliwia tworzenie nowego bota w systemie gier. Przyjmuje jako parametry rangę, parametry, cechy, nazwę gry, ilość botów, login użytkownika, identyfikatory graczy wygrywających (jeśli dotyczy). Procedura tworzy nowego bota, przypisuje go do meczu lub tworzy nowy mecz, zależnie od dostępności meczu dla danej gry, aktualizuje ilość botów w środowisku gry.

Parametry

- `@ranga` - Ranga nowego bota (integer).
- `@param` - Parametry bota (varchar(255)).
- `@cechy` - Cechy bota (varchar(255)).
- `@nazwa_gry` - Nazwa gry, do której bot jest przypisany (varchar(20)).
- `@ilosc_botow` - Ilość botów w meczu (integer).
- `@login` - Login użytkownika, który tworzy bota (varchar(20)).
- `@wygrany_1` - Identyfikator gracza pierwszego wygrywającego (integer).
- `@wygrany_2` - Identyfikator gracza drugiego wygrywającego (integer).

Działanie

1. Rozpoczyna transakcję.
2. Tworzy nowego bota i przypisuje go do istniejącego meczu lub tworzy nowy mecz dla podanej gry.
3. Aktualizuje ilość botów w środowisku gry.
4. Zatwierdza transakcję, jeśli operacje są udane, w przeciwnym razie cofa transakcję.

Zwracane Wartości

- `Status` - 1, jeśli bot został pomyślnie utworzony; 0, jeśli operacja się nie powiodła (int).
- `Message` - Komunikat informujący o statusie operacji (varchar(max)).

Przykład Użycia

```
EXEC CreateBot
    @ranga = 1,
    @param = 'parametry...',
    @cechy = 'cechy...',
    @nazwa_gry = 'nazwa_gry',
    @ilosc_botow = 2,
    @login = 'janedoe',
    @wygrany_1 = 1,
    @wygrany_2 = 2;
```

9.4 RentBot

Opis

Procedura składowana 'RentBot' umożliwia użytkownikowi wynajęcie bota na określony czas. Procedura ta sprawdza, czy użytkownik ma wystarczające środki na koncie, oblicza opłatę za wynajem na podstawie podanej liczby dni oraz aktualizuje saldo użytkownika po dokonaniu płatności.

Parametry

- `@login` - Login użytkownika, który wynajmuje bota (varchar(20)).
- `@botID` - Identyfikator wynajmowanego bota (integer).
- `@rentalDuration` - Okres wynajmu w dniach (integer).

Działanie

1. Oblicza opłatę za wynajem na podstawie podanej liczby dni (10 jednostek za dzień wynajmu).
2. Sprawdza, czy użytkownik ma wystarczające środki na koncie, aby pokryć opłatę wynajmu.
3. Jeśli użytkownik ma wystarczające środki, procedura pobiera opłatę za wynajem z konta użytkownika.
4. Przypisuje bota do użytkownika na określony czas oraz rejestruje transakcję wynajmu w tabeli 'wynajem'.
5. Zwraca odpowiedni komunikat w zależności od wyniku operacji.

Zwracane Wartości

- `Status` - Komunikat informujący o statusie operacji wynajmu bota (varchar(max)).

Przykład Użycia

```
EXEC RentBot
    @login = 'janedoe',
    @botID = 123,
    @rentalDuration = 7;
```

9.5 SendMessage

Opis

Procedura składowana ‘SendMessage‘ pozwala użytkownikowi wysłać wiadomość do innego użytkownika. Procedura ta umożliwia nadawcy wysłanie wiadomości tekstowej do określonego odbiorcy. Po wykonaniu operacji wstawienia nowej wiadomości do tabeli ‘wiadomosc‘, procedura zwraca komunikat potwierdzający wysłanie wiadomości.

Parametry

- `@sender` - Login nadawcy wiadomości (varchar(20)).
- `@recipient` - Login odbiorcy wiadomości (varchar(20)).
- `@messageText` - Treść wiadomości (varchar(100)).

Działanie

1. Wstawia nową wiadomość do tabeli ‘wiadomosc‘ z określonym nadawcą, odbiorcą, treścią, datą wysłania oraz ustawiając status wiadomości na 0 (nieprzeczytana).
2. Zwraca komunikat potwierdzający wysłanie wiadomości.

Zwracane Wartości

- `Status` - Komunikat informujący o statusie operacji wysłania wiadomości (varchar(max)).

Przykład Użycia

```
EXEC SendMessage
    @sender = 'janedoe',
    @recipient = 'johnsmith',
    @messageText = 'Hello, how are you?';
```

9.6 ViewTransactionHistory

Opis

Procedura składowana ‘ViewTransactionHistory‘ umożliwia użytkownikowi wyświetlanie historii transakcji związanych z jego kontem. Procedura ta pozwala użytkownikowi zobaczyć historię wszystkich operacji związanych z doładowaniami i wynajmem botów.

Parametry

- `@login` - Login użytkownika, dla którego wyświetlana jest historia transakcji (varchar(20)).

Działanie

1. Wyświetla historię transakcji dla użytkownika obejmującą doładowania i wynajmy botów.
2. Zwraca identyfikator transakcji (`id`), kwotę (`kwota`), datę (`data`) i typ transakcji (`TransactionType`).

Zwracane Wartości

- `id` - Identyfikator transakcji (integer).
- `kwota` - Kwota transakcji (integer).
- `data` - Data transakcji (datetime).
- `TransactionType` - Typ transakcji: 'Doladowanie' lub 'Wynajem' (varchar(20)).

Przykład Użycia

```
EXEC ViewTransactionHistory
    @login = 'janedoe';
```

10 Perspektywy

```
GRANT SELECT ON [dbo].[BotDashboard] TO RestrictedExecutorBotUser;
GRANT SELECT ON [dbo].[UserDashboard] TO RestrictedExecutorRegularUser;
GO
```

Listing 11: Nadano użytkownikom dostępy do widoków

10.1 AdminDashboard

Perspektywa ta dostarcza kompleksowego widoku danych dla administratora systemu. Obejmuje informacje o kontach użytkowników, wynajmowanych botach, środowisku gry, meczach, osiągnięciach i wiadomościach. Administrator może monitorować aktywność użytkowników, wynajmowane boty, wydarzenia w meczach i komunikację między użytkownikami.

10.2 BotDashboard

To zbiorczy widok dla administratorów, prezentujący wyniki meczów, osiągnięcia botów i ich uzyskany czas. Zawiera identyfikator meczu, graczy wygrywających (Winner1 i Winner2), wynik meczu (MatchOutcome), czas uzyskania osiągnięcia (AchievementTime) i identyfikator osiągnięcia (AchievementID). Pozwala na monitorowanie wyników botów w różnych meczach i zdobywane osiągnięcia.

10.3 UserDashboard

Widok dla użytkowników, prezentujący ich bieżące aktywności i interakcje w systemie. Zawiera informacje o wynajmowanych botach, osiągnięciach, wiadomościach i komunikatach. Użytkownicy mogą śledzić swoje wynajmy, zdobywać osiągnięcia i komunikować się z botami w ich środowisku gry. Zapewnia przegląd aktywności użytkowników w systemie wynajmu botów.

11 Indeksy

Indeksy w bazach danych są kluczowe dla optymalizacji zapytań i przyspieszania dostępu do danych. Poniżej znajduje się wyjaśnienie wyboru typu, liczby i kolumn pokrytych indeksami dla każdej z tabel oraz różnica między indeksami pojedynczymi a kompozytowymi.

Indeks kompozytowy różni się od pojedynczego tym, że może obejmować więcej niż jedną kolumnę. Indeks kompozytowy jest użyteczny, gdy chcemy przyspieszyć zapytania, które zawierają warunki dotyczące więcej niż jednej kolumny. Składające się na nie kolumny są uporządkowane leksykograficznie, co pozwala na skuteczne wyszukiwanie danych na podstawie różnych kombinacji wartości tych kolumn.

```
CREATE NONCLUSTERED INDEX IX_konto_email ON konto(email);
CREATE NONCLUSTERED INDEX IX_doladowanie_login_data ON doladowanie(login, data);
CREATE NONCLUSTERED INDEX IX_wynajem_login_data ON wynajem(login, data);
CREATE NONCLUSTERED INDEX IX_bot_nazwa_gry ON bot(nazwa_gry);
CREATE NONCLUSTERED INDEX IX_mecz_nazwa_gry_status ON mecz(nazwa_gry, status);
GO
```

Listing 12: Tworzenie indeksów

Tabela konto

Tabela `konto` zawiera kolumnę `email`, która jest często używana jako kryterium wyszukiwania. Dlatego został utworzony indeks pojedynczy na kolumnie `email` (`IX_konto_email`). Jest to indeks non-clustered, co oznacza, że dane w tabeli nie są uporządkowane według tego indeksu, co przyspiesza operacje wyszukiwania na tej kolumnie.

Tabela doladowanie

Tabela `doladowanie` zawiera kolumny `login` i `data`, które są często używane razem jako kryterium wyszukiwania. Dlatego utworzono indeks kompozytowy na tych kolumnach (`IX_doladowanie_login_data`). Indeks ten umożliwia efektywne wyszukiwanie na podstawie obu kolumn jednocześnie.

Tabela wynajem

Podobnie jak w przypadku tabeli `doladowanie`, dla tabeli `wynajem` utworzono indeks kompozytowy na kolumnach `login` i `data` (`IX_wynajem_login_data`), ponieważ są one często używane razem w zapytaniach.

Tabela bot

Tabela `bot` posiada indeks non-clustered na kolumnie `nazwa_gry` (`IX_bot_nazwa_gry`). Indeks ten pozwala na szybkie wyszukiwanie botów na podstawie nazwy gry.

Tabela mecz

Indeks kompozytowy (`IX_mecz_nazwa_gry_status`) na kolumnach `nazwa_gry` i `status` został utworzony dla tabeli `mecz`. Ten indeks umożliwia efektywne filtrowanie meczów na podstawie nazwy gry i statusu jednocześnie.

12 Struktura projektu

Wszelkie skrypty dostępne są w załączniku. Kolejność wykonywania skryptów w celu pełnej konfiguracji bazy danych wskazana jest przez przedrostek liczbowy w nazwach plików.

- `1_drop_constraints.sql`
- `2_drop_tables.sql`
- `3_create_tables.sql`
- `4_create_constraints.sql`
- `5_create_records.sql`
- `6_stored_procedures`

  - admin_procedures
    * create_new_user_account.sql
    * update_user_information.sql
  - bot_procedures
    * create_bot.sql
  - regular_user_procedures
    * rent_bot.sql
    * send_message.sql
    * view_transaction_history.sql
- 7_views
  - admin_dashboard_view.sql
  - bot_dashboard_view.sql
  - user_dashboard_view.sql
- 8_drop_users.sql
- 9_create_users.sql
- 10_manage_users.sql
- 11_drop_indexes.sql
- 12_create_indexes.sql

13 Napotkane problemy

13.1 Błędy Składniowe i Strukturalne:

- **Problem:** Błędy w zapytaniach SQL powodowały nieprawidłowe wyniki.
- **Rozwiązanie:** Poprawiłem składnię i zweryfikowałem zapytania, eliminując błędy.
- **Problem:** Niewłaściwe indeksowanie spowalniało wykonywanie zapytań.
- **Rozwiązanie:** Zoptymalizowałem indeksy, poprawiając czas odpowiedzi zapytań.

13.2 Problemy z Relacjami i Strukturą Tabel:

- **Problem:** Nieprawidłowe ustawienie kluczy obcych i relacji między tabelami.
- **Rozwiązanie:** Zrewidowałem i poprawiłem klucze obce, zapewniając poprawność relacji między tabelami.
- **Problem:** Niewłaściwe normalizowanie danych prowadziło do redundancji danych.
- **Rozwiązanie:** Przeorganizowałem strukturę bazy danych, eliminując zbędne powiązania między danymi.

13.3 Problemy z Optymalizacją i Wydajnością:

- **Problem:** Nieefektywne zapytania spowalniały pracę systemu.
- **Rozwiązanie:** Zoptymalizowałem zapytania, eliminując skomplikowane operacje i poprawiając ich efektywność.
- **Problem:** Błędne użycie funkcji agregujących prowadziło do niepoprawnych wyników.
- **Rozwiązanie:** Poprawiłem logikę funkcji agregujących, uzyskując dokładniejsze i spójniejsze wyniki.

13.4 Problemy z Bezpieczeństwem i Kontrolą Dostępu:

- **Problem:** Nieprawidłowe zarządzanie uprawnieniami użytkowników.
- **Rozwiązanie:** Skonfigurowałem odpowiednie role i uprawnienia, zabezpieczając dane przed nieautoryzowanym dostępem.

14 Wnioski

14.1 Optymalizacja Zapytań i Struktury Bazy Danych:

1. Skuteczne zarządzanie indeksami oraz optymalizacja zapytań SQL znacząco przyspieszyła operacje na bazie danych.
2. Poprawa struktury baz danych, włączając w to prawidłowe ustawienia kluczy obcych i normalizację danych, przyczyniła się do zwiększenia spójności i łatwości obsługi danych.

14.2 Bezpieczeństwo Danych i Kontrola Dostępu:

1. Staranne skonfigurowanie ról i uprawnień użytkowników znacznie ograniczyło ryzyko nieautoryzowanego dostępu do poufnych informacji.

15 Podsumowanie

Podczas prac nad bazą danych, przezwyciężyłem różnorodne wyzwania, które pozwoliły mi na rozwinięcie umiejętności zarządzania danymi oraz lepszego zrozumienia pracy z SQL Server. Projekt nauczył mnie, jak ważne jest podejście systematyczne i analityczne podczas pracy nad bazami danych.

Część III

Etap drugi projektu

16 Różnice pomiędzy elementami programowalnymi

- **Funkcje:**
  - Funkcje zwracają wartość skalarną lub tabelę jako wynik.
  - Mogą przyjmować parametry wejściowe, ale nie mogą modyfikować danych w bazie.
  - Zwracają wartość, która może być wykorzystana w zapytaniach SELECT, WHERE i innych operacjach na danych.
  - Przykład:

```
CREATE FUNCTION dbo.GetTotalSales (@Year INT)
RETURNS INT
AS
BEGIN
    DECLARE @TotalSales INT;
    -- Logika obliczeń
    RETURN @TotalSales;
END;
```

- **Procedury:**

- Procedury są zbiorami instrukcji SQL, które mogą przyjmować parametry wejściowe i wyjściowe oraz modyfikować dane w bazie.
- Procedury nie zwracają wartości, chyba że mają parametr wyjściowy.
- Procedury są często używane do przetwarzania danych, manipulacji tabelami oraz wykonywania operacji na bazie danych.
- Przykład:

```
CREATE PROCEDURE dbo.UpdateEmployeeSalary
@EmployeeID INT, @NewSalary MONEY OUTPUT
AS
BEGIN
    -- Logika aktualizacji
END;
```

- **Wyzwalacze (Triggers):**
  - Wyzwalacze są blokami kodu SQL, które są automatycznie uruchamiane w odpowiedzi na określone zdarzenia na tabeli (np. INSERT, UPDATE, DELETE).
  - Mogą być używane do automatycznego wykonywania określonych działań po zmianach w tabeli, takich jak aktualizacja powiązanych danych w innych tabelach.
  - Wyzwalacze mogą odwoływać się do kolumn zmienianych danych oraz używać ich wartości w logice zapytania.
  - Przykład:

```
CREATE TRIGGER dbo.AfterInsertTrigger
ON dbo.Orders
AFTER INSERT
AS
BEGIN
    -- Logika działania po INSERT
END;
```

17 Funkcja użytkownika

Funkcja umożliwia obliczanie całkowitego wydatku użytkownika na wynajem botów. Poniżej znajduje się implementacja funkcji `CalculateTotalExpenditure`:

```
CREATE FUNCTION CalculateTotalExpenditure (@login varchar(20))
RETURNS INT
AS
BEGIN
    DECLARE @totalExpenditure INT;
    SELECT @totalExpenditure = SUM(kwota)
    FROM wynajem
    WHERE [status] = 1 AND login = @login;
    RETURN @totalExpenditure;
END;
GO
```

Listing 13: Definicja funkcji CalculateTotalExpenditure

Funkcja ta działa na tabeli `wynajem`, gdzie dla danego użytkownika (`@login`) oblicza sumę kwoty (`kwota`) z wynajmowanych botów, których status wynajmu (`[status]`) wynosi 1 (oznaczający udany wynajem).

Poniżej znajduje się przykład użycia funkcji `CalculateTotalExpenditure`, gdzie obliczamy całkowity wydatek dla użytkownika o loginie `adventureSeeker`:

```
DECLARE @userLogin varchar(20);
SET @userLogin = 'adventureSeeker';
SELECT dbo.CalculateTotalExpenditure(@userLogin) AS 'Total Expenditure for
    ExampleUser';
GO
```

Listing 14: Przykład użycia funkcji CalculateTotalExpenditure

18 Procedury dodające elementy do tabel danych

```
DECLARE @login varchar(20)
DECLARE @email varchar(50)
DECLARE @haslo varchar(20)
DECLARE @dane_bankowe varchar(26)
DECLARE @srodki integer
SET @login = 'nowy_uzytkownik'
SET @email = 'nowy_uzytkownik@example.com'
SET @haslo = 'haslo123'
SET @dane_bankowe = '1234567890'
SET @srodki = 1000
EXEC DodajKonto @login, @email, @haslo, @dane_bankowe, @srodki
GO

Timestamp Message
[4:22:21 PM]  Started executing query at Line 1
  Commands completed successfully.
[4:22:21 PM]  Started executing query at Line 3
  Commands completed successfully.
[4:22:21 PM]  Started executing query at Line 18
  (1 row affected)
[4:22:21 PM]  Started executing query at Line 31
  Commands completed successfully.
  Total execution time: 00:00:00.006
```

Listing 15: Test poprawnego działa procedury wstawiającej dane o nazwie DodajKonto

```
DECLARE @nadawca varchar(20)
DECLARE @odbiorca varchar(20)
DECLARE @tresc varchar(100)
DECLARE @data datetime
DECLARE @status bit
SET @nadawca = 'gameMaster'
SET @odbiorca = 'musicLover'
SET @tresc = 'To jest testowa wiadomosc.'
SET @data = GETDATE()
SET @status = 0
EXEC DodajWiadomosc @nadawca, @odbiorca, @tresc, @data, @status
GO

Timestamp Message
[4:23:41 PM]  Started executing query at Line 1
  Commands completed successfully.
[4:23:41 PM]  Started executing query at Line 3
  Commands completed successfully.
[4:23:41 PM]  Started executing query at Line 18
  (1 row affected)
[4:23:41 PM]  Started executing query at Line 31
  Commands completed successfully.
  Total execution time: 00:00:00.008
```

Listing 16: Test poprawnego działa procedury wstawiającej dane o nazwie DodajWiadomosc

```
DECLARE @kwota integer
DECLARE @data datetime
DECLARE @login varchar(20)
SET @kwota = 100
SET @data = GETDATE()
SET @login = 'strategyGuru'
EXEC DodajDoladowanie @kwota, @data, @login
GO

Timestamp Message
[4:26:38 PM]  Started executing query at Line 1
  Commands completed successfully.
[4:26:38 PM]  Started executing query at Line 3
  Commands completed successfully.
[4:26:38 PM]  Started executing query at Line 16
  (1 row affected)
[4:26:38 PM]  Started executing query at Line 25
  Commands completed successfully.
  Total execution time: 00:00:00.012
```

Listing 17: Test poprawnego działa procedury wstawiającej dane o nazwie DodajDoladowanie

```
DECLARE @status bit
DECLARE @kwota integer
DECLARE @czas_trwania integer
DECLARE @data datetime
DECLARE @login varchar(20)
DECLARE @bot integer
SET @status = 1
SET @kwota = 500
SET @czas_trwania = 30
SET @data = GETDATE()
SET @login = 'triviaExpert'
SET @bot = 1
EXEC DodajWynajem @status, @kwota, @czas_trwania, @data, @login, @bot
GO

Timestamp Message
[4:28:30 PM]  Started executing query at Line 1
  Commands completed successfully.
[4:28:30 PM]  Started executing query at Line 3
  Commands completed successfully.
[4:28:30 PM]  Started executing query at Line 19
  (1 row affected)
[4:28:30 PM]  Started executing query at Line 39
  Commands completed successfully.
  Total execution time: 00:00:00.008
```

Listing 18: Test poprawnego działa procedury wstawiającej dane o nazwie DodajWynajem

```
DECLARE @ocena integer
DECLARE @komentarz varchar(100)
DECLARE @data datetime
DECLARE @login varchar(20)
DECLARE @bot integer
SET @ocena = 4
SET @komentarz = 'Bardzo dobra obsluga. Polecam!'
SET @data = GETDATE()
SET @login = 'wordsmith'
SET @bot = 10
EXEC DodajRecenzje @ocena, @komentarz, @data, @login, @bot
GO

```

```
Timestamp Message
[4:30:37 PM]  Started executing query at Line 1
  Commands completed successfully.
[4:30:37 PM]  Started executing query at Line 3
  Commands completed successfully.
[4:30:37 PM]  Started executing query at Line 18
  (1 row affected)
[4:30:37 PM]  Started executing query at Line 31
  Commands completed successfully.
  Total execution time: 00:00:00.009
```

Listing 19: Test poprawnego działa procedury wstawiającej dane o nazwie DodajRecenzje

```
DECLARE @przedmiot integer
DECLARE @czas_os datetime
DECLARE @login varchar(20)
DECLARE @bot integer
SET @przedmiot = 1
SET @czas_os = GETDATE()
SET @login = 'puzzleSolver'
SET @bot = 1
EXEC DodajOsiagniecie @przedmiot, @czas_os, @login, @bot
GO

Timestamp Message
[4:32:17 PM]  Started executing query at Line 1
  Commands completed successfully.
[4:32:17 PM]  Started executing query at Line 3
  Commands completed successfully.
[4:32:17 PM]  Started executing query at Line 17
  (1 row affected)
[4:32:17 PM]  Started executing query at Line 28
  Commands completed successfully.
  Total execution time: 00:00:00.009
```

Listing 20: Test poprawnego działa procedury wstawiającej dane o nazwie DodajOsiagniecie

```
DECLARE @ranga integer
DECLARE @param varchar(255)
DECLARE @cechy varchar(255)
DECLARE @nazwa_gry varchar(20)
DECLARE @mecz integer
SET @ranga = 1
SET @param = 'Parametry testowe'
SET @cechy = 'Cechy testowe'
SET @nazwa_gry = 'FoodFiesta'
SET @mecz = 1
EXEC DodajBota @ranga, @param, @cechy, @nazwa_gry, @mecz
GO

Timestamp Message
[4:34:53 PM]  Started executing query at Line 1
  Commands completed successfully.
[4:34:53 PM]  Started executing query at Line 3
  Commands completed successfully.
[4:34:53 PM]  Started executing query at Line 18
  (1 row affected)
[4:34:53 PM]  Started executing query at Line 31
  Commands completed successfully.
  Total execution time: 00:00:00.007
```

Listing 21: Test poprawnego działa procedury wstawiającej dane o nazwie DodajBota

```
DECLARE @nazwa_os varchar(20)
```

```
DECLARE @nazwa_przedmiotu varchar(20)
DECLARE @jakosc integer
SET @nazwa_os = 'FoodFiesta'
SET @nazwa_przedmiotu = 'Bookworms Collection'
SET @jakosc = 1
EXEC DodajPrzedmiot @nazwa_os, @nazwa_przedmiotu, @jakosc
GO

Timestamp Message
[4:36:44 PM]  Started executing query at Line 1
  Commands completed successfully.
[4:36:44 PM]  Started executing query at Line 3
  Commands completed successfully.
[4:36:44 PM]  Started executing query at Line 16
  (1 row affected)
[4:36:44 PM]  Started executing query at Line 25
  Commands completed successfully.
  Total execution time: 00:00:00.007
```

Listing 22: Test poprawnego działa procedury wstawiającej dane o nazwie DodajPrzedmiot

```
DECLARE @nazwa_gry varchar(20)
DECLARE @ilosc_botow integer
SET @nazwa_gry = 'Gra_testowa'
SET @ilosc_botow = 10
EXEC DodajSrodowisko @nazwa_gry, @ilosc_botow
GO

Timestamp Message
[4:37:58 PM]  Started executing query at Line 1
  Commands completed successfully.
[4:37:58 PM]  Started executing query at Line 3
  Commands completed successfully.
[4:37:58 PM]  Started executing query at Line 15
  (1 row affected)
[4:37:58 PM]  Started executing query at Line 22
  Commands completed successfully.
  Total execution time: 00:00:00.007
```

Listing 23: Test poprawnego działa procedury wstawiającej dane o nazwie DodajSrodowisko

```
DECLARE @wygrany_1 integer
DECLARE @wygrany_2 integer
DECLARE @wynik integer
DECLARE @status bit
DECLARE @nazwa_gry varchar(20)
SET @wygrany_1 = 1
SET @wygrany_2 = 2
SET @wynik = 3
SET @status = 1
SET @nazwa_gry = 'BookLand'
EXEC DodajMecz @wygrany_1, @wygrany_2, @wynik, @status, @nazwa_gry
GO

Timestamp Message
[4:39:28 PM]  Started executing query at Line 1
  Commands completed successfully.
[4:39:28 PM]  Started executing query at Line 3
  Commands completed successfully.
[4:39:28 PM]  Started executing query at Line 18
  (1 row affected)
[4:39:28 PM]  Started executing query at Line 31
  Commands completed successfully.
  Total execution time: 00:00:00.008
```

Listing 24: Test poprawnego działa procedury wstawiającej dane o nazwie DodajMecz

19 Złożona procedura

Opis

Procedura składowana 'WynajmijBota' pozwala na wynajmowanie bota na określony czas oraz dodawanie nowego meczu do systemu gier. Procedura ta przyjmuje jako parametry login użytkownika, identyfikator bota, czas trwania wynajmu, kwotę, nazwę gry oraz identyfikatory botów wygrywających i wynik meczu (jeśli dotyczy). Procedura ta rozpoczyna transakcję, aby zapewnić spójność danych. Następnie próbuje wynająć bota na podany czas, dodać nowy mecz i zatwierdzić transakcję. W przypadku błędu, transakcja zostanie cofnięta, a odpowiedni komunikat o błędzie zostanie zwrócony.

Parametry

- `@login` - Login użytkownika, który wynajmuje bota (varchar(20)).
- `@bot_id` - Identyfikator bota, który ma zostać wynajęty (integer).
- `@czas_trwania` - Czas wynajmu w minutach (integer).
- `@kwota` - Kwota wynajmu (integer).
- `@nazwa_gry` - Nazwa gry, dla której odbywa się mecz (varchar(20)).
- `@wygrany_1` - Identyfikator pierwszego wygranego bota (integer).
- `@wygrany_2` - Identyfikator drugiego wygranego bota (integer).
- `@wynik` - Wynik meczu (integer).

Działanie

1. Rozpoczyna transakcję.
2. Wynajmuje bota na podany czas i dodaje nowy mecz do systemu gier.
3. Zatwierdza transakcję, jeśli operacje są udane, w przeciwnym razie cofa transakcję.

Zwracane Wartości

- `Status` - 1, jeśli operacja zakończyła się pomyślnie; 0, jeśli operacja się nie powiodła (integer).
- `Message` - Komunikat informujący o statusie operacji (varchar(max)).

Przykład Użycia

```
DECLARE @login varchar(20)
DECLARE @bot_id integer
DECLARE @czas_trwania integer
DECLARE @kwota integer
DECLARE @nazwa_gry varchar(20)
DECLARE @wygrany_1 integer
DECLARE @wygrany_2 integer
DECLARE @wynik integer
```

```
SET @login = 'triviaExpert'
SET @bot_id = 1
SET @czas_trwania = 30 -- Czas wynajmu w minutach
SET @kwota = 500 -- Kwota wynajmu
SET @nazwa_gry = 'BookLand'
SET @wygrany_1 = 2 -- Identyfikator pierwszego wygranego bota
SET @wygrany_2 = 3 -- Identyfikator drugiego wygranego bota
SET @wynik = 2 -- Wynik meczu

EXEC WynajmijBota @login, @bot_id, @czas_trwania, @kwota, @nazwa_gry,
                  @wygrany_1, @wygrany_2, @wynik

Timestamp Message
[4:51:32 PM] Started executing query at Line 1
Commands completed successfully.
[4:51:32 PM] Started executing query at Line 3
Commands completed successfully.
[4:51:32 PM] Started executing query at Line 33
(1 row affected)
(1 row affected)
Operacja zakonczona pomyslnie.
[4:51:32 PM] Started executing query at Line 52
Commands completed successfully.
Total execution time: 00:00:00.007
```

20 Wyzwalacze

Po wstawieniu nowego rekordu do tabeli doladowanie, zwiększa wartość pola srodki w tabeli konto dla określonego użytkownika (login) o wartość nowo dodanego doladowania (kwota).

```
SELECT srodki AS przed_doladowaniem
FROM konto
WHERE login = 'fitnessFanatic';
GO

INSERT INTO doladowanie
    (kwota, data, login)
VALUES
    (100, GETDATE(), 'fitnessFanatic');
GO

SELECT srodki AS po_doladowaniu
FROM konto
WHERE login = 'fitnessFanatic';
GO

przed_doladowaniem
290
po_doladowaniu
390
```

Listing 25: Test poprawnego działania wyzwalacza o nazwie tr_doladowanie_konta.

Po wstawieniu nowego rekordu do tabeli wynajem, zmniejsza wartość pola srodki w tabeli konto dla określonego użytkownika (login) o wartość nowego wynajmu (kwota), jeśli status wynajmu wynosi 1 (co oznacza, że wynajem został zakończony).

```
DECLARE @BeforeUpdate TABLE (
```

```
    login varchar(20),
    srodki_before int
);

DECLARE @AfterUpdate TABLE (
    login varchar(20),
    srodki_after int
);

INSERT INTO @BeforeUpdate
    (login, srodki_before)
SELECT login, srodki
FROM konto
WHERE login = 'fitnessFanatic';

INSERT INTO wynajem
    (status, kwota, czas_trwania, data, login, bot)
VALUES
    (1, 100, 30, GETDATE(), 'fitnessFanatic', 1);

INSERT INTO @AfterUpdate
    (login, srodki_after)
SELECT login, srodki
FROM konto
WHERE login = 'fitnessFanatic';

SELECT 'Before Update' AS operation, *
    FROM @BeforeUpdate
UNION ALL
    SELECT 'After Update' AS operation, *
    FROM @AfterUpdate;
GO

operation login srodki_before
Before Update fitnessFanatic  190
After Update  fitnessFanatic  90
```

Listing 26: Test poprawnego działania wyzwalacza o nazwie tr_dodawanie_wynajmu.

Po aktualizacji pola status w tabeli mecz, oblicza sumę wyników (wygrany_1 i wygrany_2) dla meczu i aktualizuje pole wynik w tabeli mecz na podstawie tej sumy.

```
INSERT INTO mecz
    (wygrany_1, wygrany_2, wynik, status, nazwa_gry)
VALUES
    (1, 2, 0, 0, 'GymGalaxy');
GO

SELECT id, wygrany_1, wygrany_2, wynik
FROM mecz
WHERE nazwa_gry = 'GymGalaxy' AND status = 0;
GO

UPDATE mecz
SET wygrany_1 = 3, wygrany_2 = 2
WHERE nazwa_gry = 'GymGalaxy' AND status = 0;
GO

SELECT id, wygrany_1, wygrany_2, wynik
FROM mecz
WHERE nazwa_gry = 'GymGalaxy' AND status = 0;
GO

id  wygrany_1 wygrany_2 wynik
```

```
30  1 2 0
id  wygrany_1 wygrany_2 wynik
30  3 2 5
```

Listing 27: Test poprawnego działania wyzwalacza o nazwie tr_zmiana_statusu_meczu.

Po wstawieniu nowego rekordu do tabeli recenzja, oblicza średnią ocenę dla określonego bota (bot) i aktualizuje pole ranga w tabeli bot na podstawie tej średniej oceny.

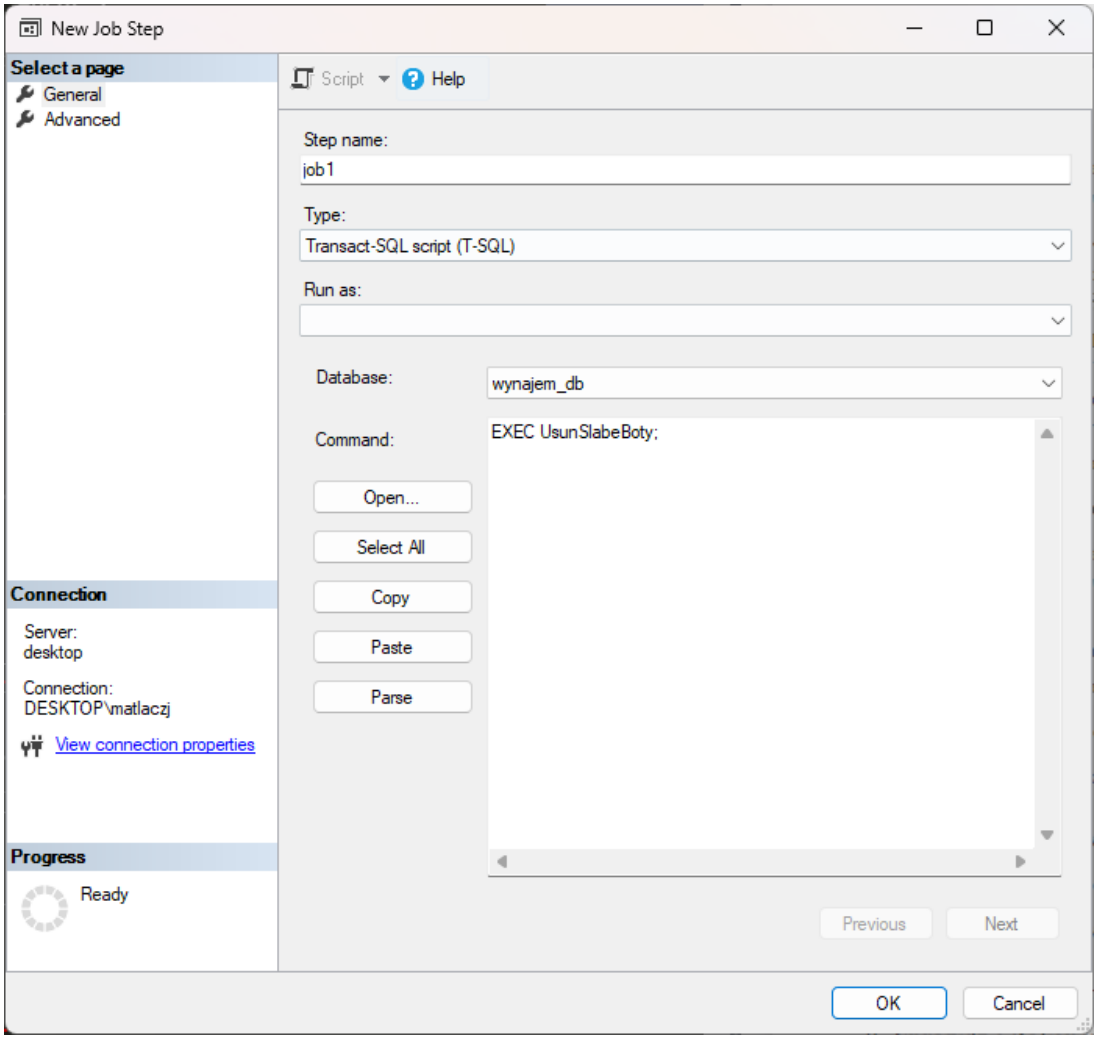
```
SELECT ranga
FROM bot
WHERE id = 1;
GO

INSERT INTO recenzja
    (ocena, komentarz, data, login, bot)
VALUES
    (10000, 'Bardzo dobry bot!', GETDATE(), 'cardPlayer', 1);

INSERT INTO recenzja
    (ocena, komentarz, data, login, bot)
VALUES
    (4, 'Dobry bot, ale jeszcze ma miejsce na poprawe.', GETDATE(), 'gameMaster',
        1);
GO

SELECT ranga
FROM bot
WHERE id = 1;
GO

ranga
3
ranga
225
```

Listing 28: Test poprawnego działania wyzwalacza o nazwie tr_obliczanie_sredniej_oceny.

21 Automatyzacja zadania

Co tydzień, automatycznie identyfikuj boty z najniższą oceną w grze i usuwaj je z systemu. Poniżej znajduje się lista kroków, które wykonano, aby osiągnąć cel. Zautomatyzowane zadanie jest dostępne w formie procedury. Na obrazku 4 widać jak podpięto komendę wykonującą procedurę.

1. **Otwórz SQL Server Management Studio (SSMS):**
   - Uruchom SQL Server Management Studio i połącz się z instancją serwera SQL.
2. **Rozwiń SQL Server Agent:**
   - W oknie "Object Explorer", rozwinąć węzeł "SQL Server Agent".
3. **Kliknij prawym przyciskiem myszy na Jobs (Joby):**
   - Kliknij prawym przyciskiem myszy na węzeł "Jobs" i wybierz opcję "New Job...".
4. **Podaj Nazwę Joba i Opis:**
   - W oknie "New Job" wprowadź nazwę joba w polu "Name".
   - Opcjonalnie, dodaj opis joba w polu "Description".
5. **Zdefiniuj Kroki Joba:**



Rysunek 4: Automatyzacja zadania w GUI SSMS.

- Kliknij na zakładkę "Steps" po lewej stronie.
- Kliknij "New...", aby dodać nowy krok do joba.
- W oknie "New Job Step" wprowadź nazwę kroku w polu "Step name".
- Wybierz typ kroku. Na przykład, jeśli chcesz uruchomić skrypt Transact-SQL, wybierz "Transact-SQL script (T-SQL)".
- Wprowadź skrypt T-SQL lub polecenie w oknie "Command". W moim przypadku to było wywołanie procedury: "EXEC UsunSlabeBoty;".
- Skonfiguruj inne opcje kroku, takie jak kontekst bazy danych, plik wynikowy lub zaawansowane opcje, jeśli są wymagane.
- Kliknij "OK", aby zapisać krok.

6. **Zdefiniuj Harmonogram Joba:**

- Kliknij na zakładkę "Schedules" po lewej stronie.
- Kliknij "New...", aby dodać nowy harmonogram do joba.
- W oknie "New Job Schedule" skonfiguruj opcje harmonogramu, takie jak częstotliwość, dzienne powtarzanie, datę rozpoczęcia i datę zakończenia.
- Kliknij "OK", aby zapisać harmonogram.

7. **Ustaw Powiadomienia (Opcjonalnie):**

- Jeśli chcesz otrzymywać powiadomienia o stanie joba, kliknij na zakładkę "Notifications" po lewej stronie.

- Skonfiguruj powiadomienia e-mailowe lub pagerowe, jeśli są wymagane.

8. **Ustaw Alerty (Opcjonalnie):**
   - Jeśli chcesz, aby job reagował na określone zdarzenia SQL Server, kliknij na zakładkę "Alerts" po lewej stronie.
   - Skonfiguruj alerty, jeśli są wymagane.
9. **Sprawdź i Zakończ:**
   - Przejrzyj wszystkie ustawienia na różnych zakładkach, aby upewnić się, że są one poprawnie skonfigurowane.
   - Kliknij "OK", aby utworzyć joba.

22 Kopia zapasowa bazy

Na obrazku 5 oraz na liście poniżej opisano i pokazano kroki tworzenia kopii oraz jej przywracania.

1. **Utwórz Pełną Kopię Oryginalnej Bazy Danych:**
   - Uruchom SQL Server Management Studio (SSMS).
   - Połącz się z instancją serwera SQL, która zawiera oryginalną bazę danych.
   - Kliknij prawym przyciskiem myszy na bazie danych, którą chcesz skopiować.
   - Wybierz "Zadania" -> "Kopia zapasowa...".
   - W oknie kopiowania wybierz "Pełna" jako typ kopii zapasowej.
   - Wskaż miejsce docelowe dla pliku kopii zapasowej (np. lokalny dysk lub lokalizację sieciową).
   - Kliknij "OK", aby utworzyć kopię zapasową.
2. **Przywróć Kopię Zapasową, aby Utworzyć Kopię Bazy Danych:**
   - Połącz się z instancją serwera SQL, do której chcesz utworzyć kopię bazy danych.
   - Kliknij prawym przyciskiem myszy na "Bazy danych" w oknie "Object Explorer".
   - Wybierz "Przywróć bazę danych...".
   - W oknie przywracania bazy danych:
     - Wprowadź nową nazwę dla kopii bazy danych w polu "Do bazy danych".
     - W sekcji "Z urządzenia" kliknij "..." i wybierz plik kopii zapasowej.
     - Kliknij "OK", aby rozpocząć proces przywracania.

23 Struktura projektu

Wszelkie skrypty dostępne są w załączniku. Kolejność wykonywania skryptów w celu pełnej konfiguracji bazy danych wskazana jest przez przedrostek liczbowy w nazwach plików. Baza skryptów znacznie się rozrosła od etapu pierwszego.

- `1_drop_constraints.sql`
- `2_drop_tables.sql`
- `3_create_tables.sql`
- `4_create_constraints.sql`
- `5_create_records.sql`

- 6_stored_procedures
  - admin_procedures
    - create_new_user_account.sql
    - update_user_information.sql
    - usun_slabe_boty.sql
  - bot_procedures
    - create_bot.sql
  - new_record_procedures
    - dodaj_bota.sql
    - dodaj_doladowanie.sql
    - dodaj_konto.sql
    - dodaj_mecz.sql
    - dodaj_osiagniecie.sql
    - dodaj_przedmiot.sql
    - dodaj_recenzje.sql
    - dodaj_srodowisko.sql
    - dodaj_wiadomosc.sql
    - dodaj_wynajem.sql
  - regular_user_procedures
    - rent_bot.sql
    - send_message.sql
    - view_transaction_history.sql
    - wynajmij_bota.sql
- 7_views
  - admin_dashboard_view.sql
  - bot_dashboard_view.sql
  - user_dashboard_view.sql
- 8_drop_users.sql
- 9_create_users.sql
- 10_manage_users.sql
- 11_drop_indexes.sql
- 12_create_indexes.sql
- 13_UDFs
  - calculate_total_expenditure.sql
- 14_triggers
  - tr_dodawanie_wynajmu.sql
  - tr_doladowanie_konta.sql
  - tr_obliczanie_sredniej_oceny.sql
  - tr_zmiana_statusu_meczu.sql

Część IV

Alternatywny system bazodanowy

24 Instalacja i Konfiguracja MySQL

Wybranie MySQL jako systemu zarządzania bazą danych może być uzasadnione z poniższych powodów.

Open Source: MySQL jest open source, co oznacza, że jest dostępny za darmo i jest transparentny.

Popularność i Wsparcie Społeczności: MySQL jest jednym z najbardziej popularnych systemów zarządzania bazą danych na świecie, co oznacza, że istnieje szeroka społeczność użytkowników i dostępność wielu źródeł wsparcia i dokumentacji.

Łatwość Użycia: MySQL jest znany ze swojej łatwości użycia, zarówno dla początkujących, jak i dla zaawansowanych użytkowników. Ma również intuicyjny interfejs graficzny (MySQL Workbench), który ułatwia zarządzanie bazą danych.

24.1 Kroki Instalacji:

1. Pobierz MySQL Installer z oficjalnej strony internetowej MySQL.
2. Uruchom pobrany plik instalatora MySQL.
3. Wybierz opcję instalacji MySQL Server i postępuj zgodnie z instrukcjami.
4. Wybierz również MySQL Workbench jako jeden z komponentów do zainstalowania.

24.2 Konfiguracja MySQL Server:

1. Podczas instalacji MySQL Server, ustal hasło dla użytkownika root (administratora) bazy danych MySQL.
2. Uruchom MySQL Workbench.
3. Utwórz nowe połączenie, podając adres IP, port, nazwę użytkownika (zwykle "root") oraz hasło, które ustawiłeś podczas instalacji MySQL Server.

24.3 Zainstalowane Komponenty:

- MySQL Server: Odpowiedzialny za przechowywanie danych oraz obsługę zapytań.
- MySQL Workbench: Narzędzie do zarządzania bazą danych MySQL za pomocą intuicyjnego interfejsu graficznego.

25 Analiza różnic w dialektach pomiędzy SQL Server i MySQL

- Należy uwzględnić różnice w typach danych, takich jak unikod, data i liczby całkowite.
- Indeksy, w tym klucze główne, klucze obce i indeksy tabel, powinny zostać odpowiednio dostosowane.
- Procedury składowane i funkcje powinny zostać zaktualizowane pod kątem składni i obsługi błędów.
- Triggery wymagają modyfikacji z uwagi na różnice w składni i funkcjonalności.
- Widoki powinny być poprawione pod względem składni i uprawnień.
- Kolumny identyczności należy przestawić na autoinkrementację.

- Składnia SQL, takie jak korzystanie z LIMIT zamiast TOP w zapytaniach, wymaga dostosowania.
- Formaty danych, takie jak konkatenacja ciągów czy funkcje daty, powinny być zmienione.
- Utrzymuj spójność w wielkości liter w nazwach tabel i kolumn.
- Narzędzia do migracji danych, przy eksporcie/importowaniu, należy dostosować do odpowiednich formatów danych.

26 Narzędzia do Migracji Danych między SQL Server a MySQL

Migracja danych między SQL Server a MySQL może być wyzwaniem, ale istnieją różne narzędzia, które ułatwiają ten proces. Poniżej znajduje się lista narzędzi, które można wykorzystać do migracji danych między tymi dwoma systemami baz danych. Wybrano opcję MySQL Workbench.

1. **MySQL Workbench:**
   - *Opis:* MySQL Workbench to oficjalne narzędzie administracyjne MySQL, które zawiera funkcję migracji danych. Pozwala na import danych z SQL Server do MySQL za pomocą intuicyjnego interfejsu graficznego.
   - *Ocena:* Proste w użyciu i odpowiednie dla mniejszych baz danych. Oferuje podstawowe funkcje migracji.
2. **SQL Server Integration Services (SSIS):**
   - *Opis:* SSIS to usługa ETL (Extract, Transform, Load) w SQL Server, która umożliwia tworzenie zaawansowanych pakietów do migracji danych do różnych źródeł, w tym MySQL.
   - *Ocena:* Potężne narzędzie do skomplikowanych procesów migracji. Wymaga znajomości środowiska SSIS.
3. **Microsoft Data Migration Assistant:**
   - *Opis:* Jest to narzędzie opracowane przez Microsoft, które pomaga w ocenie kompatybilności aplikacji, baz danych i migracji schematów oraz danych do docelowego systemu, w tym MySQL.
   - *Ocena:* Pomocne do oceny gotowości do migracji i identyfikacji potencjalnych problemów.
4. **Command-Line Tools:**
   - *Opis:* Zarówno SQL Server, jak i MySQL oferują narzędzia wiersza poleceń (sqlcmd dla SQL Server i mysql dla MySQL), które umożliwiają eksport i import danych.
   - *Ocena:* Wymaga znajomości składni poleceń, ale jest skuteczne dla dużych zbiorów danych i automatyzacji procesu migracji.

27 Migracja Danych za pomocą ODBC i MySQL Workbench Migration Wizard

Poniżej przedstawiono kroki do utworzenia połączenia za pomocą narzędzia ODBC Data Sources (DSN) oraz proces migracji danych przy użyciu MySQL Workbench Migration Wizard:

1. **Tworzenie Połączenia za pomocą ODBC Data Sources (DSN):**
   (a) *Otwórz ODBC Data Sources:*
      - Na systemach Windows, przejdź do "Panel sterowania" -> "Administracyjne narzędzia" -> "Źródła danych (ODBC)".

   (b) *Dodaj Nowe Źródło Danych:*

- W zakładce "DSN użytkownika" lub "DSN systemowe", wybierz opcję "Dodaj".

(c) *Wybierz ODBC Driver:*

- Wybierz odpowiedni sterownik ODBC dla Twojego źródła danych (np. SQL Server ODBC Driver).

(d) *Skonfiguruj Połączenie:*

- Podaj niezbędne informacje, takie jak adres serwera, nazwa bazy danych, login i hasło do połączenia z bazą danych. Przetestuj połączenie, aby upewnić się, że jest prawidłowo skonfigurowane.

2. **Użycie Połączenia w MySQL Workbench Migration Wizard:**

(a) *Otwórz MySQL Workbench:*

- Uruchom MySQL Workbench na swoim komputerze.

(b) *Przejdź do MySQL Workbench Migration Wizard:*

- Wybierz opcję "Database" z górnego menu, a następnie "Migration Wizard".

(c) *Rozpocznij Nowy Projekt Migracji:*

- Kliknij "Start Migration" i wybierz "ODBC" jako źródło danych.

(d) *Konfiguruj Połączenie ODBC:*

- Kliknij "Configure ODBC Data Source". Wybierz wcześniej utworzone źródło danych ODBC z listy i wprowadź odpowiednie dane logowania.

(e) *Wybierz Tabele do Migracji:*

- Po nawiązaniu połączenia, MySQL Workbench pokaże dostępne tabele w źródłowej bazie danych SQL Server. Wybierz tabele, które chcesz migrować do MySQL.

(f) *Skonfiguruj Opcje Migracji:*

- Wybierz docelowy typ obiektu (np. tabelę, widok) dla każdej tabeli źródłowej. Skonfiguruj również opcje takie jak migracja danych, indeksów, kluczy obcych itp.

(g) *Uruchom Migrację:*

- Kliknij "Start Migration" lub podobny przycisk, aby uruchomić proces migracji. MySQL Workbench przeprowadzi migrację danych ze źródła (SQL Server) do docelowego systemu (MySQL).

(h) *Sprawdź i Weryfikuj Wyniki:*

- Po zakończeniu migracji, sprawdź wyniki, aby upewnić się, że dane zostały poprawnie przeniesione. Przejrzyj logi migracji, aby zweryfikować, czy nie wystąpiły żadne błędy.

28 Ręczne dostosowanie reszty obiektów w bazie

Niestety podejście w sekcji powyżej pozwoliło jedynie na przeniesienie części elementów, a pozostałe trzeba było ręcznie przetłumaczyć i uruchomić na nowej bazie. Podsumowanie dostępne jest w załączniku w pliku "MySQL Workbench Migration Wizard Report.txt". Udało się dodać wszystkie elementy do nowej bazy i osiągnąć pełną migrację, co widać na obrazku 6. Wszystko działało jak należy, a nawet szybciej niż na SQL Server.

29 Testy poprawnego działania nowej bazy

Przetestowano poprawne działanie wszystkich elementów i poniżej przedstawiono przykłady.

```
SELECT * FROM wynajem_db.admindashboard;
# AccountLogin, AccountEmail, RentalID, RentalStatus, RentalAmount,
   RentalDuration, RentalDate, BotID, GameName, BotRank, EnvironmentBotCount,
   MatchID, Winner1, Winner2, MatchResult, MatchStatus, MessageID, Sender,
   Receiver, MessageContent, MessageDate, MessageStatus
```

```
'adventureSeeker', 'adventurer@example.com', '2', '1', '70', '180', '2023-10-28
    12:30:00', '2', 'AdventureQuest', '15', '30', '1', '1', '2', '3', '1', '1', '
    gamerX', 'adventureSeeker', 'Hey there! Ready for a gaming adventure?', '
    2023-10-27 15:30:00', '1'
...
```

Listing 29: Test poprawnego działania admindashboard.

```
SELECT * FROM wynajem_db.botdashboard;
# MatchID, Winner1, Winner2, MatchOutcome, AchievementTime, AchievementID
'1', '1', '2', 'Loss', '2023-10-27 12:00:00', '1'
...
```

Listing 30: Test poprawnego działania botdashboard.

```
SELECT * FROM wynajem_db.userdashboard;
# RentalID, RentalStatus, RentalAmount, RentalDuration, RentalDate, BotID,
    GameName, BotRank, EnvironmentBotCount, AchievementID, AchievementTime,
    AchievementItemID, MessageID, Sender, MessageContent, MessageDate,
    MessageStatus
'1', '1', '50', '120', '2023-10-27 10:00:00', '1', 'GameWorld 1', '10', '50', '1'
    , '2023-10-27 12:00:00', '10', '1', 'gamerX', 'Hey there! Ready for a gaming
    adventure?', '2023-10-27 15:30:00', '1'
...
```

Listing 31: Test poprawnego działania userdashboard.

```
-- Create temporary tables
CREATE TEMPORARY TABLE BeforeUpdate (
    operation VARCHAR(20),
    login VARCHAR(20),
    srodki_before INT
);

CREATE TEMPORARY TABLE AfterUpdate (
    operation VARCHAR(20),
    login VARCHAR(20),
    srodki_after INT
);

-- Populate BeforeUpdate table
INSERT INTO BeforeUpdate
    (operation, login, srodki_before)
SELECT 'Before Update', login, srodki
FROM konto
WHERE login = 'fitnessFanatic';

-- Insert into wynajem table
INSERT INTO wynajem
    (status, kwota, czas_trwania, data, login, bot)
VALUES
    (1, 100, 30, NOW(), 'fitnessFanatic', 1);

-- Populate AfterUpdate table
INSERT INTO AfterUpdate
    (operation, login, srodki_after)
SELECT 'After Update', login, srodki
FROM konto
WHERE login = 'fitnessFanatic';

-- Retrieve data from temporary tables
SELECT * FROM BeforeUpdate
UNION ALL
SELECT * FROM AfterUpdate;
```

```
38
39 -- Drop temporary tables (optional, they will be automatically dropped when the
       session ends)
40 DROP TEMPORARY TABLE IF EXISTS BeforeUpdate;
41 DROP TEMPORARY TABLE IF EXISTS AfterUpdate;
42
43 # operation, login, srodki_before
44 'Before Update', 'fitnessFanatic', '190'
45 'After Update', 'fitnessFanatic', '90'
```

Listing 32: Test poprawnego działania tr_dodawanie_wynajmu

```
1  -- Retrieve srodki before doladowanie
2  SELECT srodki AS przed_doladowaniem
3  FROM konto
4  WHERE login = 'fitnessFanatic';
5
6  -- Perform doladowanie
7  INSERT INTO doladowanie
8      (kwota, data, login)
9  VALUES
10     (100, NOW(), 'fitnessFanatic');
11
12 -- Retrieve srodki after doladowanie
13 SELECT srodki AS po_doladowaniu
14 FROM konto
15 WHERE login = 'fitnessFanatic';
16
17 # po_doladowaniu
18 '290'
19 # po_doladowaniu
20 '390'
```

Listing 33: Test poprawnego działania tr_doladowanie_konta.

```
1  -- Retrieve ranga before recenzja insertion
2  SELECT ranga
3  FROM bot
4  WHERE id = 1;
5
6  -- Insert first recenzja
7  INSERT INTO recenzja
8      (ocena, komentarz, data, login, bot)
9  VALUES
10     (10000, 'Bardzo dobry bot!', NOW(), 'cardPlayer', 1);
11
12 -- Insert second recenzja
13 INSERT INTO recenzja
14     (ocena, komentarz, data, login, bot)
15 VALUES
16     (4, 'Dobry bot, ale jeszcze ma miejsce na poprawe.', NOW(), 'gameMaster', 1);
17
18 -- Retrieve ranga after recenzja insertion
19 SELECT ranga
20 FROM bot
21 WHERE id = 1;
22
23 # ranga
24 '3336'
25 # ranga
26 '4002'
```

Listing 34: Test poprawnego działania tr_obliczanie_sredniej_oceny

```
-- Insert new mecz
INSERT INTO mecz
    (wygrany_1, wygrany_2, wynik, status, nazwa_gry)
VALUES
    (1, 2, 0, 0, 'GymGalaxy');

-- Select mecz information before update
SELECT id, wygrany_1, wygrany_2, wynik
FROM mecz
WHERE nazwa_gry = 'GymGalaxy' AND status = 0;

-- Update mecz information
UPDATE mecz
SET wygrany_1 = 3, wygrany_2 = 2
WHERE nazwa_gry = 'GymGalaxy' AND status = 0;

-- Select mecz information after update
SELECT id, wygrany_1, wygrany_2, wynik
FROM mecz
WHERE nazwa_gry = 'GymGalaxy' AND status = 0;

# id, wygrany_1, wygrany_2, wynik
'21', '1', '2', '0'
# id, wygrany_1, wygrany_2, wynik
'21', '3', '2', '5'
```

Listing 35: Test poprawnego działania tr_zmiana_statusu_meczu

```
SET @userLogin = 'adventureSeeker';
SELECT CalculateTotalExpenditure(@userLogin) AS 'Total Expenditure for
   ExampleUser';

# Total Expenditure for ExampleUser
'70'
```

Listing 36: Test poprawnego działania CalculateTotalExpenditure

```
SET @ranga = 1;
SET @param = 'Parametry testowe';
SET @cechy = 'Cechy testowe';
SET @nazwa_gry = 'FoodFiesta';
SET @mecz = 1;
CALL DodajBota(@ranga, @param, @cechy, @nazwa_gry, @mecz);

09:48:36  CALL DodajBota(@ranga, @param, @cechy, @nazwa_gry, @mecz) 1 row(s)
   affected 0.000 sec
```

Listing 37: Test poprawnego działania jednej z new_record_procedures: DodajBota

```
SET @login = 'adventureSeeker';
SET @botID = 1;
SET @rentalDuration = 1;
CALL RentBot(@login, @botID, @rentalDuration);

# Status
'Bot rented successfully'
```

Listing 38: Test poprawnego działania jednej z regular_user_procedures: RentBot

```
SET @login = 'monkeyDluffy';
SET @newEmail = 'funky@lake.pl';
SET @newPassword = 'strawhat';
SET @newDaneBankowe = '12345678901234567890123456';
```

```
CALL UpdateUserInformation(@login, @newEmail, @newPassword, @newDaneBankowe);

# Status, Message
'0', 'User not found. Information update failed.'
```

Listing 39: Test poprawnego działania jednej z admin_procedures: UpdateUserInformation

```
SET @ranga = 1;
SET @param = '1 2 8 9 10 100 43 1';
SET @cechy = '1 82 1 11 1 223 1';
SET @nazwa_gry = 'FoodFiesta';
SET @ilosc_botow = 1;
SET @login = 'puzzleSolver';
SET @wygrany_1 = 13;
SET @wygrany_2 = 15;
CALL CreateBot(@ranga, @param, @cechy, @nazwa_gry, @ilosc_botow, @login,
    @wygrany_1, @wygrany_2);

09:52:42  CALL CreateBot(@ranga, @param, @cechy, @nazwa_gry, @ilosc_botow, @login
    , @wygrany_1, @wygrany_2)  1 row(s) affected 0.000 sec
```

Listing 40: Test poprawnego działania jednej z bot_procedures: CreateBot

30 Napotkane problemy

30.1 Problemy ze Składnią

1. **Różnice w Składni SQL:** Zauważyliśmy różnice w składni między SQL Server a MySQL, co wymagało dostosowania zapytań SQL.

2. **Brak Dostępnych Funkcji:** Pewne funkcje, które były dostępne w SQL Server, nie są obsługiwane w MySQL, co wymagało przepisania tych fragmentów kodu.

30.2 Problemy Związane z Typami Danych

1. **Inne Typy Danych:** Niektóre typy danych w SQL Server nie mają bezpośrednich odpowiedników w MySQL, co wymagało konwersji typów danych.

2. **Inne Maksymalne Długości dla Typów Tekstowych:** Maksymalne długości dla typów danych tekstowych (np. VARCHAR) różnią się między SQL Server a MySQL, co wymagało dostosowania schematu bazy danych.

30.3 Problemy Związane z Transakcjami

1. **Różnice w Transakcjach:** MySQL może mieć inne zachowanie podczas transakcji w porównaniu z SQL Server, co wymagało analizy i dostosowania kodu transakcji.

30.4 Problemy Związane z Trigerami i Procedurami

1. **Różnice w Składni Trigerów:** Trigery mają różne składnie w SQL Server i MySQL, co wymagało przepisania i przetestowania wszystkich trigerów.

2. **Problemy z Procedurami Składowanymi:** Niektóre procedury składowane napisane dla SQL Server mogą wymagać zmian w składni, aby działały poprawnie w MySQL.

30.5 Problemy Związane z Wydajnością

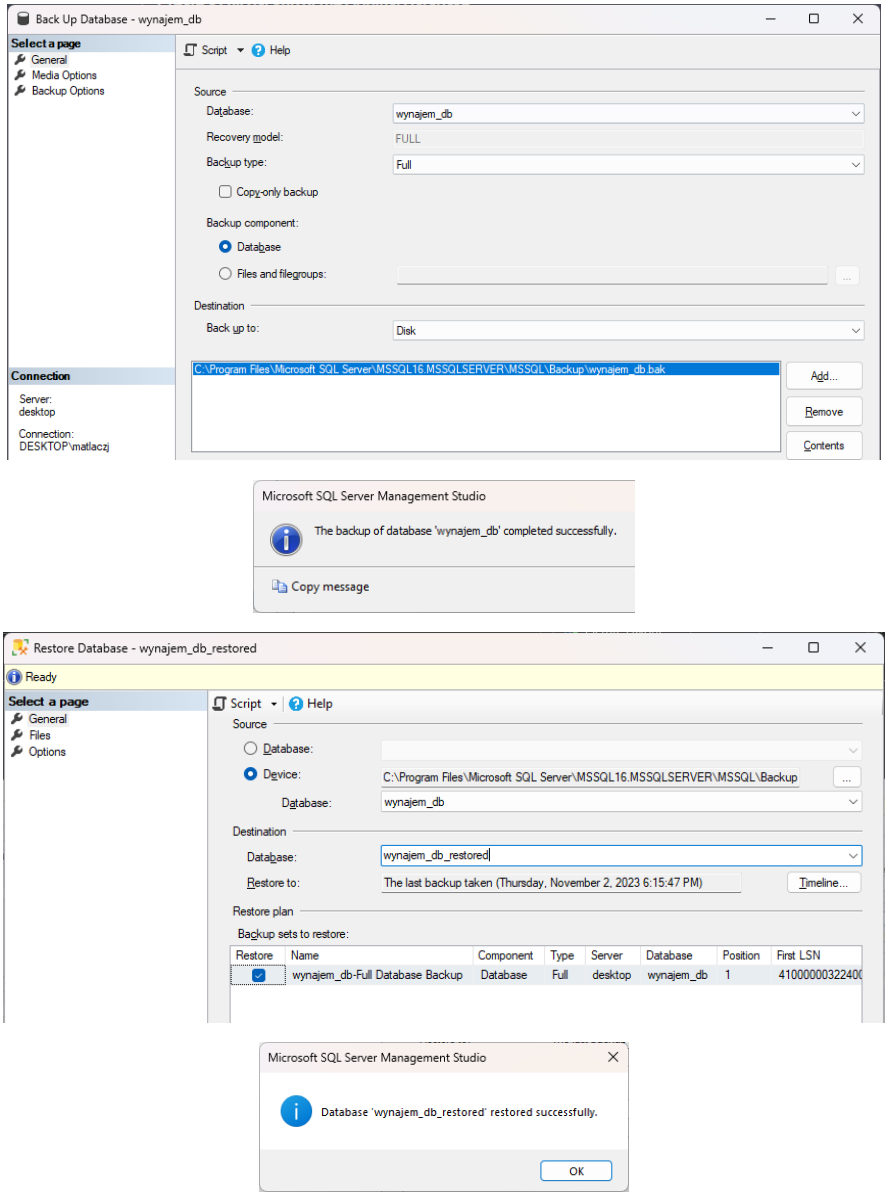
1. **Różnice w Optymalizacji Zapytań:** Optymalizacja zapytań może być inna w MySQL, co wymagało ponownego przemyślenia i zoptymalizowania niektórych zapytań.

30.6 Problemy Związane z Bezpieczeństwem

1. **Różnice w Modelu Bezpieczeństwa:** MySQL i SQL Server mają różne modele bezpieczeństwa, co może wymagać dostosowania uprawnień i kontroli dostępu.
2. **Zarządzanie Uwierzytelnianiem:** Różnice w zarządzaniu uwierzytelnianiem i hasłami mogą wprowadzać problemy podczas migracji kont użytkowników.

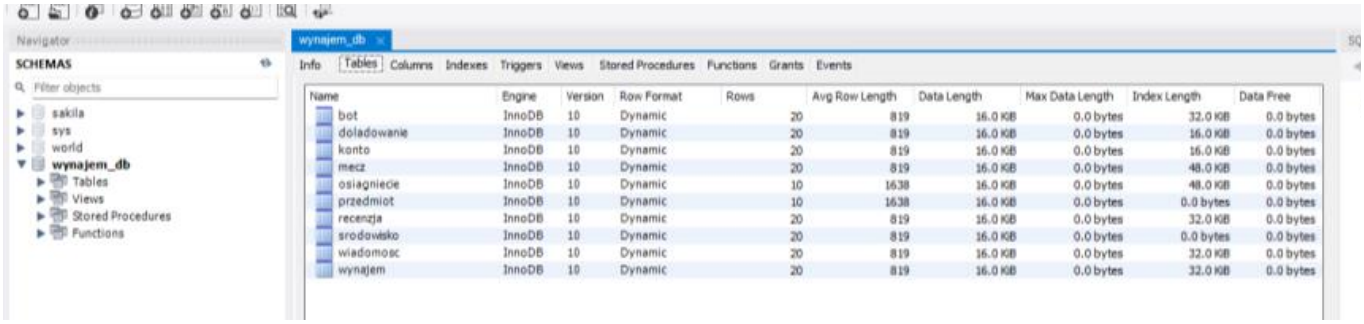
31 Wnioski

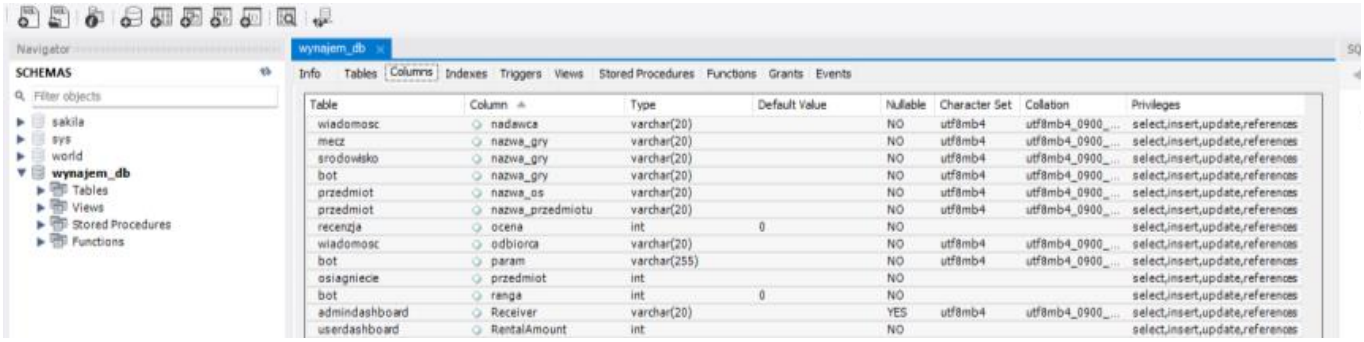
- Doświadczenie w migracji baz danych uczy elastyczności, szybkiego dostosowywania się do nowych sytuacji i skutecznego radzenia sobie z problemami.
- Zrozumienie różnic w elementach programowalnych w SQL Server i MySQL pozwala na lepsze wykorzystanie ich funkcji w obu systemach.
- Praca nad funkcjami użytkownika uświadamia, jak różnice w składni mogą wpływać na implementację, wymagając dokładnego dostosowania.
- Doświadczenie z procedurami uczy, jak dokładnie analizować i dostosowywać składnię SQL, zapobiegając błędom i zapewniając płynne działanie.
- Praca nad złożonymi procedurami pomaga w lepszym planowaniu i optymalizacji operacji w nowym środowisku bazodanowym.
- Zrozumienie różnic w wyzwalaczach pozwala na ich skuteczne dostosowanie do nowego środowiska, zachowując spójność danych.
- Automatyzacja zadań pomaga w efektywnym zarządzaniu rutynowymi operacjami, zwiększając wydajność pracy.
- Tworzenie regularnych kopii zapasowych jest kluczowe dla zachowania integralności danych w przypadku awarii, wymagając odpowiedniego przechowywania.
- Analiza struktury projektu podkreśla wagę odpowiedniego zaprojektowania struktury bazy danych, ułatwiając przyszłe projekty.
- Samodzielna instalacja i konfiguracja nowego środowiska bazodanowego pomaga zyskać niezależność w zarządzaniu danymi.
- Precyzyjne dostosowywanie składni SQL do różnych systemów baz danych jest kluczowe dla prawidłowego funkcjonowania operacji.
- Skuteczne wykorzystanie narzędzi do migracji danych ułatwia przenoszenie danych między różnymi systemami bazodanowymi.
- Szczegółowe sprawdzanie i modyfikacja różnych elementów bazy danych zapewnia spójność i poprawność działania obiektów.
- Przeprowadzanie testów potwierdza, czy wszystkie elementy bazy danych działają zgodnie z oczekiwaniami, zapewniając gotowość do produkcji.



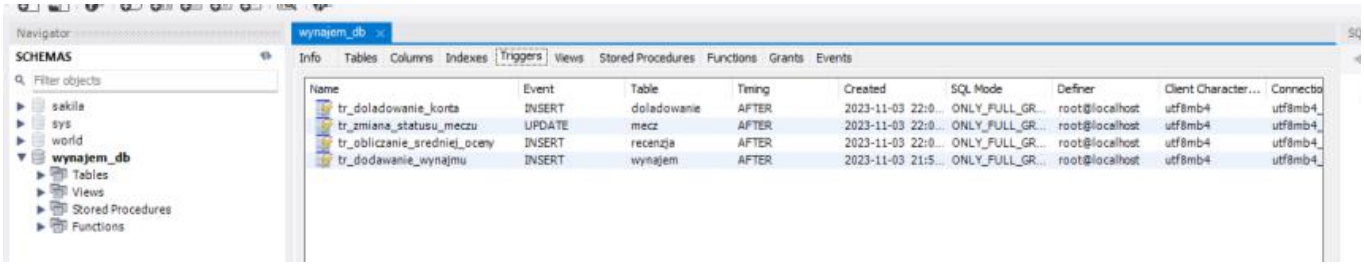
Rysunek 5: Wykonanie kopii oraz przywrócenie bazy danych w GUI SSMS.

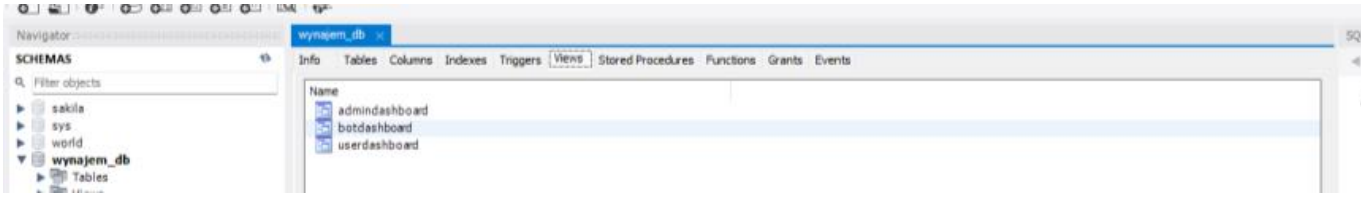

Navigator
SCHEMAS
Filter objects
sakila
sys
world
wynajem_db
Tables
bot
doladowanie
konto
mecz
osiagniecie
Info
Tables
Columns
Indexes
Triggers
Views
Stored Procedures
Functions
Grants
Events
Local instance MySQL80
wynajem_db
Schema Details
Default collation: utf8mb4_0900_ai_ci
Default characterset: utf8mb4
Table count: 13
Database size (rough estimate): 416.0 KiB

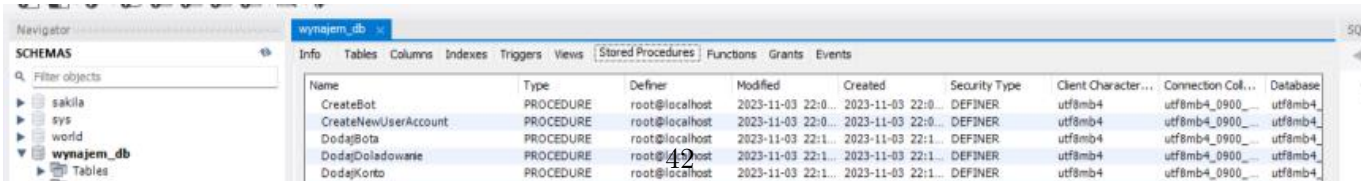

Navigator
SCHEMAS
Filter objects
sakila
sys
world
wynajem_db
Tables
Views
Stored Procedures
Functions
wynajem_db
Info
Tables
Columns
Indexes
Triggers
Views
Stored Procedures
Functions
Grants
Events
Name Engine Version Row Format Rows Avg Row Length Data Length Max Data Length Index Length Data Free
bot InnoDB 10 Dynamic 20 819 16.0 KiB 0.0 bytes 32.0 KiB 0.0 bytes
doladowanie InnoDB 10 Dynamic 20 819 16.0 KiB 0.0 bytes 16.0 KiB 0.0 bytes
konto InnoDB 10 Dynamic 20 819 16.0 KiB 0.0 bytes 16.0 KiB 0.0 bytes
mecz InnoDB 10 Dynamic 20 819 16.0 KiB 0.0 bytes 48.0 KiB 0.0 bytes
osiagniecie InnoDB 10 Dynamic 10 1638 16.0 KiB 0.0 bytes 48.0 KiB 0.0 bytes
przedmiot InnoDB 10 Dynamic 10 1638 16.0 KiB 0.0 bytes 0.0 bytes 0.0 bytes
recenzja InnoDB 10 Dynamic 20 819 16.0 KiB 0.0 bytes 32.0 KiB 0.0 bytes
srodowisko InnoDB 10 Dynamic 20 819 16.0 KiB 0.0 bytes 0.0 bytes 0.0 bytes
wiadomosc InnoDB 10 Dynamic 20 819 16.0 KiB 0.0 bytes 32.0 KiB 0.0 bytes
wynajem InnoDB 10 Dynamic 20 819 16.0 KiB 0.0 bytes 32.0 KiB 0.0 bytes


Navigator
SCHEMAS
Filter objects
sakila
sys
world
wynajem_db
Tables
Views
Stored Procedures
Functions
wynajem_db
Info
Tables
Columns
Indexes
Triggers
Views
Stored Procedures
Functions
Grants
Events
Table Column Type Default Value Nullable Character Set Collation Privileges
wiadomosc nadawca varchar(20) NO utf8mb4 utf8mb4_0900_... select,insert,update,references
mecz nazwa_gry varchar(20) NO utf8mb4 utf8mb4_0900_... select,insert,update,references
srodowisko nazwa_gry varchar(20) NO utf8mb4 utf8mb4_0900_... select,insert,update,references
bot nazwa_gry varchar(20) NO utf8mb4 utf8mb4_0900_... select,insert,update,references
przedmiot nazwa_os varchar(20) NO utf8mb4 utf8mb4_0900_... select,insert,update,references
przedmiot nazwa_przedmiotu varchar(20) NO utf8mb4 utf8mb4_0900_... select,insert,update,references
recenzja ocena int 0 NO select,insert,update,references
wiadomosc odbiorca varchar(20) NO utf8mb4 utf8mb4_0900_... select,insert,update,references
bot param varchar(255) NO utf8mb4 utf8mb4_0900_... select,insert,update,references
osiagniecie przedmiot int NO select,insert,update,references
bot renga int 0 NO select,insert,update,references
admindashboard Receiver varchar(20) YES utf8mb4 utf8mb4_0900_... select,insert,update,references
userdashboard RentalAmount int NO select,insert,update,references


Navigator
SCHEMAS
Filter objects
sakila
sys
world
wynajem_db
Tables
Views
Stored Procedures
Functions
wynajem_db
Info
Tables
Columns
Indexes
Triggers
Views
Stored Procedures
Functions
Grants
Events
Table Name Unique Index... Index Comment Column Seq in Index Packed
bot FK_Bot_Mecz_ID No BTREE mecz 1
mecz FK_Mecz_Bot1_ID No BTREE wygrany_1 1
mecz FK_Mecz_Bot2_ID No BTREE wygrany_2 1
osiagniecie FK_Osiagniecie_Bot_ID No BTREE bot 1
osiagniecie FK_Osiagniecie_Konto_Login No BTREE login 1
osiagniecie FK_Osiagniecie_Przedmiot_ID No BTREE przedmiot 1
recenzja FK_Recenzja_Bot_ID No BTREE bot 1
recenzja FK_Recenzja_Konto_Login No BTREE login 1
wiadomosc FK_Wiadomosc_Nadawca No BTREE nadawca 1
wiadomosc FK_Wiadomosc_Odbiorca No BTREE odbiorca 1
wynajem FK_Wynajem_Bot_ID No BTREE bot 1
bot IX_bot_nazwa_gry No BTREE nazwa_gry 1
doladowanie IX_doladowanie_login_data No BTREE data 2
doladowanie IX_doladowanie_login_data No BTREE login 1


Navigator
SCHEMAS
Filter objects
sakila
sys
world
wynajem_db
Tables
Views
Stored Procedures
Functions
wynajem_db
Info
Tables
Columns
Indexes
Triggers
Views
Stored Procedures
Functions
Grants
Events
Name Event Table Timing Created SQL Mode Definer Client Character... Connecto
tr_doladowanie_konta INSERT doladowanie AFTER 2023-11-03 22:0... ONLY_FULL_GR... root@localhost utf8mb4 utf8mb4_
tr_zmiana_statusu_meczu UPDATE mecz AFTER 2023-11-03 22:0... ONLY_FULL_GR... root@localhost utf8mb4 utf8mb4_
tr_obliczanie_sredniej_oceny INSERT recenzja AFTER 2023-11-03 22:0... ONLY_FULL_GR... root@localhost utf8mb4 utf8mb4_
tr_dodawanie_wynajmu INSERT wynajem AFTER 2023-11-03 21:5... ONLY_FULL_GR... root@localhost utf8mb4 utf8mb4_


Navigator
SCHEMAS
Filter objects
sakila
sys
world
wynajem_db
Tables
wynajem_db
Info
Tables
Columns
Indexes
Triggers
Views
Stored Procedures
Functions
Grants
Events
Name
admindashboard
botdashboard
userdashboard


Navigator
SCHEMAS
Filter objects
sakila
sys
world
wynajem_db
Tables
wynajem_db
Info
Tables
Columns
Indexes
Triggers
Views
Stored Procedures
Functions
Grants
Events
Name Type Definer Modified Created Security Type Client Character... Connection Coll... Database
CreateBot PROCEDURE root@localhost 2023-11-03 22:0... 2023-11-03 22:0... DEFINER utf8mb4 utf8mb4_0900_... utf8mb4_
CreateNewUserAccount PROCEDURE root@localhost 2023-11-03 22:0... 2023-11-03 22:0... DEFINER utf8mb4 utf8mb4_0900_... utf8mb4_
DodajBota PROCEDURE root@localhost 2023-11-03 22:1... 2023-11-03 22:1... DEFINER utf8mb4 utf8mb4_0900_... utf8mb4_
DodajDoladowanie PROCEDURE root@localhost 2023-11-03 22:1... 2023-11-03 22:1... DEFINER utf8mb4 utf8mb4_0900_... utf8mb4_
DodajKonto PROCEDURE root@localhost 2023-11-03 22:1... 2023-11-03 22:1... DEFINER utf8mb4 utf8mb4_0900_... utf8mb4_