Projekt Zaliczeniowy
Komputerowy System Wspomagania Wynajmu Krótkoterminowego

Krystian Duma - Grupa Z501 - Nr. Albumu 7763

Grudzień 2018

Spis treści

1 Krótki opis słowny projektu

Projekt zawiera założenia do bazy danych przechowujacej podstawowe informacje o wybranych funkcjach systemu informatycznego wspierajacego funkcjonowanie agencji wynajmu krótkoterminowego domów, mieszkań lub innych obiektów.

2 Założenia do projektu

Przyjęte zostały następujące założenia do projektu

1. Podstawowe Obiekty
   - `Obiekt` - obiekt najmu - np. konkretny dom lub mieszkanie,
   - `Użytkownik` - osoba wynajmująca mieszkanie lub dom,
2. Przechowywane zadania (transakcje)
   - `Najem` - transakcja zwiazana z wynajęciem `Obiektu` przez `Użytkownika`.
3. Szczegoły opisu
   - `Użytkownik` - potrzeba przechowania informacji: nazwisko klienta, imie klienta, wiek klienta, adres zamieszkania klienta, telefon klienta, płec klienta oraz login używany do logowania do bazy danych.
   - `Obiekt` - potrzeba przechowania informacji: nazwa własna obiektu, adres obiektu, dzienna stawka najmu obiektu, kategoria obiektu, obecny status najmu obiektu (informacja czy dany obiekt jest obecnie wolny lub zajęty), opis obiektu oraz inne atrybuty odpowiednie dla zgromadzonych obiektów.
     - Każdy obiekt może znajdować się w wielu różnych kategoriach,
     - Dla uproszczenia inne atrybuty będą znajdować się w opisie danego obiektu.
   - `Najem` - potrzeba przechowania informacji: użytkownika-najemca, wynajmowany obiekt, data rozpoczęcia najmu, data zakończenia najmu, koszt najmu.
     - Najem to transakcja tylko jednego `Użytkownika` i tylko jednego `Obiektu`,
     - Dla uproszczenia najem jest liczony od godziny 00:00 do godziny 23:59,
     - Jeden `Obiekt` może być w danym czasie wynajęty tylko jednemu użytkownikowi.
4. Użytkownicy i Uprawnienia
   - Administrator ma dostęp do danych wszystkich użytkowników,
   - Każdy `Użytkownik` ma założone oddzielne konto serwera SQL,
   - Użytkownicy nie widzą danych oraz wypożyczeń innych użytkowników.

3 Środowisko Projektowe

Środowiskiem uruchomieniowym jest baza danych Microsoft SQL Server 2017 uruchomiona w kontenerze Docker'a. Jako obraz bazowy został wybrany obraz mcr.microsoft.com/mssql/server:2017-latest-ubuntu który zawiera najaktualniejszą obecnie wersję Microsoft SQL Server 2017 uruchomioną na systemie Linux - Ubuntu Server. Do obrazu zostały doinstalowane dodatkowe narzędzia umożliwiające przygotowanie plików wyjściowych: tego dokumentu pdf (LaTeX) oraz skryptów tworzących i usuwających obiekty z bazy (PHP).

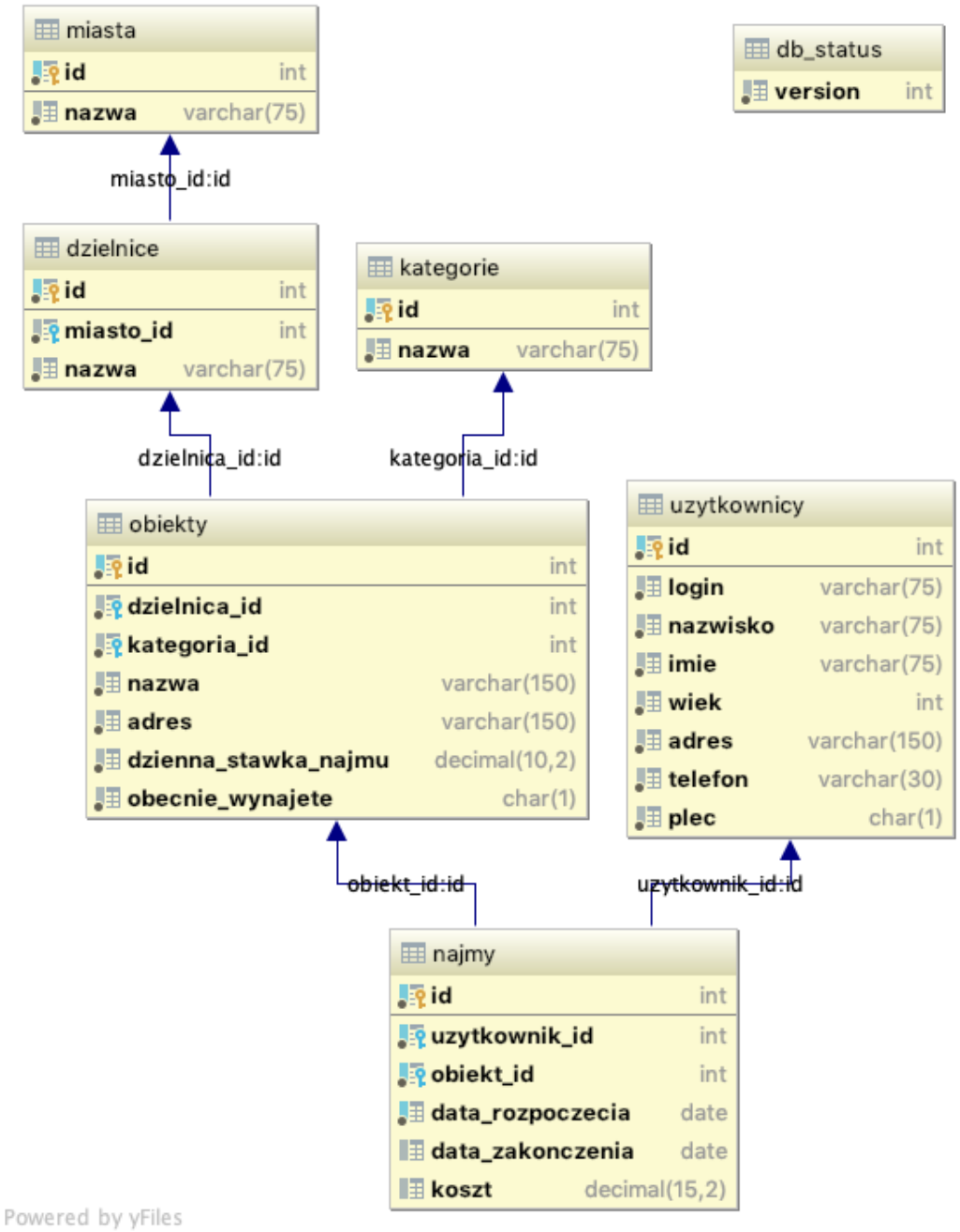
Jako aplikację służącą do łączenia się i wykonywania poleceń wykorzystane zostały aplikacje:

- Dołączona do SQL Server'a aplikacja wiersza poleceń - sqlcmd
- Środowisko IDE od czeskiej firmy JetBrains - DataGrip
- Środowisko IDE od Microsoft'u - SQL Server Management Studio (SSMS)

4 Model fizyczny bazy danych

Na Rysunku 1 znajduje się schemat (diagram tabel) wygenerowanej przez skrypt: skrypt_tworzacy_obiekty_w_bazie_danych.sql.

Rysunek 1: Diagram tabel wygenerowanej bazy danych



5 Skrypt tworzący obiekty w bazie danych

5.1 Model wersjonowania bazy danych

Jak można zauważyć na Rysunku 1, w bazie danych znajduje się jedna dodatkowa tabela `db_status` z jednym polem `version` - służy ona do przechowywania wersji bazy danych. Każda operacja w skrypcie tworzącym sprawdza i porównuje obecną oraz oczekiwaną wersję dla danej operacji. Dzięki temu zabiegowi nie będzie można uruchomić danej operacji dla jednej bazy danych wielokrotnie. Dodatkowo aktualizacja istniejącej bazy danych do najnowszej wersji będzie uproszczona - wystarczy uruchomić najnowszą wersję skryptu, a wykonane zostaną tylko nowe operacje dodane od ostatniego uruchomienia skryptu instalacyjnego. Każda operacja jest opakowana zgodnie z szablonem z listingu 1.

```
PRINT 'Wersja X: ''<<< OPIS OPERACJI >>>'''
IF EXISTS(SELECT * FROM sys.tables WHERE name = N'db_status')
  BEGIN
    IF EXISTS(SELECT * FROM db_status WHERE version = X)
      BEGIN

    <<< MIEJSCE NA KOD >>>

        UPDATE db_status SET version = 1 WHERE version = X;
        PRINT 'Wersja X: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji X';
      END
    ELSE
      BEGIN
        IF EXISTS(SELECT * FROM db_status WHERE version < X)
          BEGIN
            RAISERROR ('Wersja X: Baza danych jest w za niskiej wersji (wymagana jest wersja X) aby zainstalowac migracje', 11, 2);
          END
        ELSE
          BEGIN
            PRINT 'Wersja X: Migracja ju  zostala zainstalowana wczesniej';
          END
      END
  END
ELSE
  BEGIN
    RAISERROR ('Wersja X: Nie znaleziono tabeli wersjonowania bazy danych', 11, 1);
  END
```

Listing 1: Szablon kodu wersjonowanego

Dodatkowo w przypadku wystąpienia jakichkolwiek błędów jest przewidziana procedura ich łapania - na listingu 2 widzimy zawartość bloku `CATCH` skryptu instalacyjnego. Skrypt został przygotowany w taki sposób aby w przypadku wystąpienia błędu przerywał działanie[1] i przechodził od razu do bloku `CATCH`.

```
BEGIN CATCH

  SELECT
    ERROR_NUMBER() AS ErrorNumber,
    ERROR_SEVERITY() AS ErrorSeverity,
    ERROR_STATE() AS ErrorState,
    ERROR_PROCEDURE() AS ErrorProcedure,
    ERROR_LINE() AS ErrorLine,
    ERROR_MESSAGE() AS ErrorMessage;

END CATCH;
```

Listing 2: Blok CATCH w skrypcie tworzącym

5.2 Wynik uruchomienia całego skryptu tworzącego obiekty w trybie wsadowym

Jak widać na listingu 3, skrypt podaje bardzo dokładne informacje na temat aktualnie wykonywanej operacji. W większości przypadków wystąpienia ciągu tekstowego `(1 rows affected )`, następuje zmiana aktualnej wersji bazy danych w tabeli wersjonowania - `db_status`.

```

(1 rows affected)
```

[1] Aby wywołanie funkcji `RAISERROR` przekazało kontrolę do bloku `CATCH`, parametr `severity` musi mieć wartość z zakresu od `11` do `19`. Wartości poniżej nie powodują przerwania skryptu, a wartości powyżej terminują połączenie z bazą danych.

```
Tabela wersjonowania zostala utworzona
Wersja 1: 'Utworzenie tabeli z miastami'

(1 rows affected)
Wersja 1: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 1
Wersja 2: 'Utworzenie tabeli z dzielnicami'

(1 rows affected)
Wersja 2: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 2
Wersja 3: 'Utworzenie relacji pomiedzy miastami a dzielnicami'

(1 rows affected)
Wersja 3: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 3
Wersja 4: 'Utworzenie tabeli z kategoriami'

(1 rows affected)
Wersja 4: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 4
Wersja 5: 'Utworzenie tabeli z obiektami'

(1 rows affected)
Wersja 5: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 5
Wersja 6: 'Utworzenie relacji pomiedzy dzielnicami a obiektami'

(1 rows affected)
Wersja 6: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 6
Wersja 7: 'Utworzenie relacji pomiedzy kategoriami a obiektami'

(1 rows affected)
Wersja 7: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 7
Wersja 8: 'Utworzenie tabeli z uzytkownikami'

(1 rows affected)
Wersja 8: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 8
Wersja 9: 'Utworzenie tabeli z najmami'

(1 rows affected)
Wersja 9: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 9
Wersja 10: 'Utworzenie indeksu unikatowego w tabeli z najemcami'

(1 rows affected)
Wersja 10: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 10
Wersja 11: 'Utworzenie relacji pomiedzy uzytkownikami a najmami'

(1 rows affected)
Wersja 11: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 11
Wersja 12: 'Utworzenie relacji pomiedzy obiektami a najmami'

(1 rows affected)
Wersja 12: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 12
Wersja 13: 'Utworzenie indeksu unikatowego w tabeli z uzytkownikami'

(1 rows affected)
Wersja 13: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 13
Wersja 14: 'Utworzenie widoku z lista wszystkich najmow'

(1 rows affected)
Wersja 14: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 14
Wersja 15: 'Utworzenie widoku z lista popularnosci obiektow'

(1 rows affected)
Wersja 15: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 15
Wersja 16: 'Utworzenie widoku z lista niewynajmowanych obiektow'

(1 rows affected)
Wersja 16: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 16
Wersja 17: 'Utworzenie funkcji wyliczajacej koszt najmu obiektu'

(1 rows affected)
Wersja 17: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 17
Wersja 18: 'Utworzenie funkcji wyliczajacej koszt konkretnego najmu'

(1 rows affected)
Wersja 18: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 18
Wersja 19: 'Utworzenie triggeru aktualizujacego koszt najmu'

```

```
(1 rows affected)
Wersja 19: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 19
Wersja 20: 'Utworzenie funkcji wyswietlajacej adresowke uzytkownika'

(1 rows affected)
Wersja 20: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 20
Wersja 21: 'Utworzenie funkcji wyswietlajacej spozniajacych sie uzytkownikow'

(1 rows affected)
Wersja 21: Migracja zostala zainstalowana pomyslnie - teraz baza jest w wersji 21
```

Listing 3: Wynik uruchomienia całego skryptu tworzącego obiekty w trybie wsadowym

5.3 Tabele

Wszystkie tabele są tworzone przez 13 skryptów `SQL`:

- Utworzenie tabeli z miastami
- Utworzenie tabeli z dzielnicami
- Utworzenie relacji pomiedzy miastami a dzielnicami
- Utworzenie tabeli z kategoriami
- Utworzenie tabeli z obiektami
- Utworzenie relacji pomiedzy dzielnicami a obiektami
- Utworzenie relacji pomiedzy kategoriami a obiektami
- Utworzenie tabeli z uzytkownikami
- Utworzenie tabeli z najmami
- Utworzenie indeksu unikatowego w tabeli z najmami
- Utworzenie relacji pomiedzy uzytkownikami a najmami
- Utworzenie relacji pomiedzy obiektami a najmami
- Utworzenie indeksu unikatowego w tabeli z uzytkownikami

Tworzenie relacji pomiędzy tabelami oraz indeksów zostało oddzielone od operacji tworzenia poszczególnych tabel - celem tego działania jest lepsza organizacja skryptów. Dodatkowo oddzielając te operacje, w przypadku wystąpienia jakiegoś błędu jesteśmy w stanie określić co i gdzie się "wysypało".

Ze względu na to aby nie zajmować zbyt dużo miejsca, poniżej zostaną przedstawione tylko najważniejsze z powyższych skryptów.

5.3.1 Utworzenie tabeli z obiektami

Ponieważ polecenia `CREATE DEFAULT` oraz `CREATE RULE` zostały zdeprecjonowane i w kolejnych wersjach SQL Serwera prawdopodobnie zostaną usunięte zdecydowałem się umieścić wartości domyślne oraz reguły sprawdzające w definicjach konkretnych tabel.

W wyniku projektowania zostało dodatkowo ustalone że `dzienna_stawka_najmu` musi być większa od 0.

Status obiektu znajdujący się w polu `obecnie_wynajete` może przyjmować dwie wartości `T` oraz `N` - odpowiednio dla obiekty wynajętego oraz wolnego.

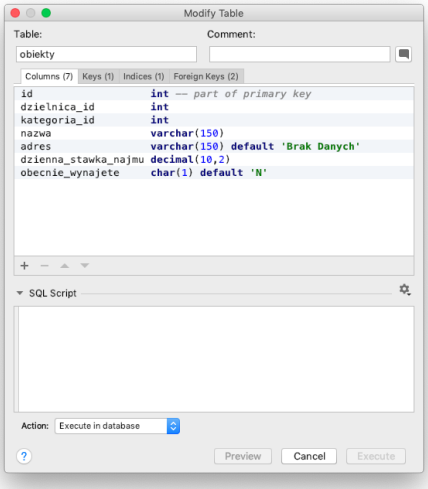
```
CREATE TABLE obiekty (
  id INT PRIMARY KEY NOT NULL IDENTITY (1, 1),

  dzielnica_id INT NOT NULL,
  kategoria_id INT NOT NULL,

  nazwa VARCHAR(150) NOT NULL,
  adres VARCHAR(150) NOT NULL DEFAULT 'Brak Danych',
  dzienna_stawka_najmu DECIMAL(10, 2) NOT NULL CHECK (dzienna_stawka_najmu > 0),

  obecnie_wynajete CHAR(1) NOT NULL DEFAULT 'N' CHECK (obecnie_wynajete IN ('T', 'N')),
);
```

Listing 4: Skrypt tworzący tabelę `obiekty`



Rysunek 2: Tabela `obiekty` wyświetlona w programie DataGrip

5.3.2 Utworzenie tabeli z najmami i utworzenie indeksu unikatowego w tabeli z najmami

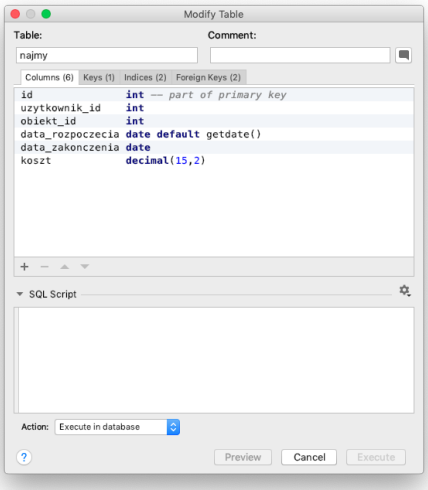
Przyjmujemy że domyślną datą rozpoczęcia najmu jest data jego dodania do bazy.

```
CREATE TABLE najmy (
  id INT PRIMARY KEY NOT NULL IDENTITY (1, 1),

  uzytkownik_id INT NOT NULL,
  obiekt_id INT NOT NULL,

  data_rozpoczecia DATE NOT NULL DEFAULT getdate(),
  data_zakonczenia DATE NULL,
  koszt DECIMAL(15, 2) NULL,
);
```

Listing 5: Skrypt tworzący tabelę `najmy`



Rysunek 3: Tabela `najmy` wyświetlona w programie DataGrip

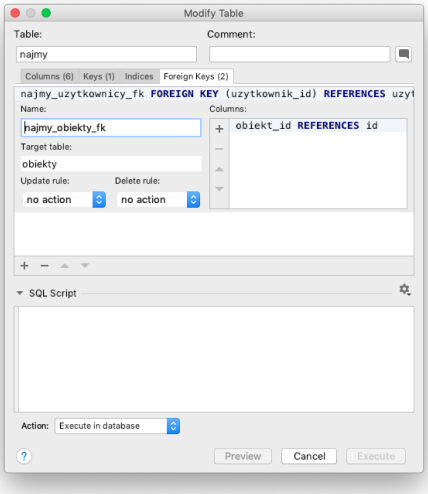
Ponieważ w założeniach projektowych przyjeliśmy że najem następuje od północy do godziny 23:59, to zostało wykorzystane pole typu `DATE`. Więc na przykład jeśli klient wynajmie dany obiekt tylko na jeden dzień to w polach `data_rozpoczecia` oraz `data_zakonczenia` wartość będzie ta sama. Dzięki temu możemy również utworzyć indeks unikatowy na pola `uzytkownik_id`, `obiekt_id` oraz `data_rozpoczecia`.

```
CREATE UNIQUE INDEX najemcy_ui
   ON najmy (uzytkownik_id, obiekt_id, data_rozpoczecia);
```

Listing 6: Skrypt tworzący indeks unikatowy w tabeli `obiekty`

5.4 Utworzenie relacji pomiędzy użytkownikami a najmami

Wszystkie tabele zostały połączone relacją (CONSTRAINT) typu klucz obcy (FOREIGN KEY) tak jak na przykładzie z listingu 7.



Rysunek 4: Relacja `najmy_obiekty_fk` wyświetlona w programie DataGrip

```
ALTER TABLE najmy
  ADD CONSTRAINT najmy_obiekty_fk
  FOREIGN KEY (obiekt_id)
  REFERENCES obiekty(id);
GO;
```

Listing 7: Skrypt tworzący relację `najmy_obiekty_fk`

5.5 Widoki

Zostało stworzone kilka widoków:

- `lista_najmow` - Lista wszystkich najmów
- `lista_popularnosci_obiektow` - Lista obiektów wraz z ilością (popularnoscią) ich najmów
- `lista_niepopularnych_obiektow` - Lista obiektów które nie zostały nigdy wynajęte

5.5.1 `lista_najmow` - Lista wszystkich najmów

Widok ten zwraca listę wszystkich najmów, wraz z następującymi polami:

- nazwisko
- imie
- datę rozpoczecia najmu
- datę zakonczenia najmu
- nazwę wynajmowanego obiektu
- całkowity koszt najmu

	nazwisko	imie	data_rozpoczecia	data_zakonczenia	nazwa_obiektu	calkowity_koszt
22	Nazwisko 2	Imie 1	2018-12-11	2018-12-21	Nazwa 3	704.55
23	Nazwisko 3	Imie 3	2018-11-28	2018-12-25	Nazwa 4	1793.40
24	Nazwisko 2	Imie 4	2018-12-06	2018-12-23	Nazwa 3	2160.00
25	Nazwisko 3	Imie 3	2018-12-12	2018-12-21	Nazwa 4	1200.00
26	Nazwisko 2	Imie 2	2018-12-11	2018-12-25	Nazwa 4	1800.00
27	Nazwisko 2	Imie 4	2018-11-30	2018-12-24	Nazwa 3	633.75
28	Nazwisko 2	Imie 1	2018-12-04	2018-12-22	Nazwa 2	2280.00
29	Nazwisko 2	Imie 1	2018-12-03	2018-12-25	Nazwa 3	1473.15
30	Nazwisko 3	Imie 3	2018-12-08	2018-12-22	Nazwa 1	380.25
31	Nazwisko 2	Imie 4	2018-12-08	2018-12-19	Nazwa 3	768.60
32	Nazwisko 3	Imie 3	2018-12-04	2018-12-24	Nazwa 2	1345.05
33	Nazwisko 2	Imie 1	2018-12-25	<null>	Nazwa 2	<null>
34	Nazwisko 3	Imie 3	2018-12-23	<null>	Nazwa 1	<null>
35	Nazwisko 2	Imie 4	2018-12-25	<null>	Nazwa 4	<null>
36	Nazwisko 2	Imie 2	2018-12-24	<null>	Nazwa 2	<null>
37	Nazwisko 2	Imie 4	2018-12-23	<null>	Nazwa 2	<null>
38	Nazwisko 3	Imie 3	2018-12-26	<null>	Nazwa 3	<null>
39	Nazwisko 2	Imie 4	2018-12-27	<null>	Nazwa 2	<null>
40	Nazwisko 2	Imie 1	2018-12-23	<null>	Nazwa 1	<null>

Rysunek 5: Wyświetlony widok `lista_najmow`

```
CREATE VIEW lista_najmow
  AS
    SELECT nazwisko, imie, data_rozpoczecia, data_zakonczenia, nazwa nazwa_obiektu, koszt
    calkowity_koszt
        FROM najmy n
        JOIN uzytkownicy u on n.uzytkownik_id = u.id
        JOIN obiekty o on n.obiekt_id = o.id;
```

Listing 8: Skrypt tworzący widok `lista_najmow`

5.5.2 `lista_niepopularnych_obiektow` - Lista obiektów które nie zostały nigdy wynajęte

Widok ten zwraca listę obiektów które nie zostały nigdy, prze nikogo, wynajęte, wraz z następującymi polami:

- nazwę obiektu
- adres obiektu

Widok ten korzysta z innego (nieopisanego w tym dokumencie) widoku - `lista_popularnosci_obiektow`.

	id	nazwa	adres
1	4	Nazwa 4	Adres 2
2	6	Nazwa 1	Adres 3
3	7	Nazwa 3	Adres 2
4	9	Nazwa 4	Adres 2
5	10	Nazwa 4	Adres 1
6	14	Nazwa 4	Adres 3
7	19	Nazwa 1	Adres 4
8	24	Nazwa 2	Adres 4
9	25	Nazwa 2	Adres 4
10	28	Nazwa 1	Adres 4
11	32	Nazwa 2	Adres 1
12	38	Nazwa 3	Adres 4
13	40	Nazwa 4	Adres 3
14	43	Nazwa 2	Adres 4
15	47	Nazwa 1	Adres 1
16	51	Nazwa 4	Adres 2
17	53	Nazwa 4	Adres 4
18	56	Nazwa 2	Adres 4
19	58	Nazwa 4	Adres 1
20	59	Nazwa 3	Adres 2
21	60	Nazwa 2	Adres 3
22	62	Nazwa 4	Adres 3

Rysunek 6: Wyświetlony widok `lista_niepopularnych_obiektow`

```
CREATE VIEW lista_niepopularnych_obiektow
  AS
    SELECT id, nazwa, adres
      FROM lista_popularnosci_obiektow
      GROUP BY id, nazwa, adres
      HAVING SUM(liczba_najmow) = 0;
```

Listing 9: Skrypt tworzący widok `lista_niepopularnych_obiektow`

5.6 Funkcje Skalarne i Tabelarne

Zostały stworzone następujące funkcje:

- `koszt_najmu_obiektu` - Funkcja obliczająca koszt najmu wskazanego obiektu
- `koszt_najmu` - Funkcja obliczająca koszt konkretnego najmu
- `adresowka` - Funkcja generująca etykietę adresową dla użytkownika
- `opoznieni` - Funkcja generująca etykiety dla użytkowników którzy coś wynajęli ale jeszcze nie oddali

5.6.1 `koszt_najmu_obiektu` - Funkcja obliczająca koszt najmu wskazanego obiektu

`koszt_najmu_obiektu (@obiekt_id INT, @liczba_dni INT)` - Funkcja ta oblicza koszt najmu obiektu wskazanego w parametrze `@obiekt_id` przez liczbę dni określoną w parametrze `@liczba_dni`, zwracana wartość ma typ `DECIMAL(15, 2)` który jest kompatybilny z typem kolumny `koszt` w tabeli `najmy`. Funkcja ta została wydzielona w celu uniknięcia duplikacji kodu w funkcji skalarnej `koszt_najmu` z listingu 11 oraz w funkcji tabelarnej `opoznieni` z listingu 13 (strona 11).

```
CREATE FUNCTION koszt_najmu_obiektu (@obiekt_id INT, @liczba_dni INT)
  RETURNS DECIMAL(15, 2)
  AS
  BEGIN
    DECLARE @dzienna_stawka DECIMAL(10,2);

    SELECT @dzienna_stawka = dzienna_stawka_najmu
      FROM obiekty o
      WHERE id = @obiekt_id;

    RETURN @liczba_dni*@dzienna_stawka;
  END
```

Listing 10: Skrypt tworzący funkcję skalarną `koszt_najmu_obiektu`

5.6.2 `koszt_najmu` - Funkcja obliczająca koszt danego najmu

`koszt_najmu (@najem_id INT)` - Funkcja ta oblicza koszt konkretnego najmu wskazanego w parametrze `@najem_id`, zwracana wartość ma taki sam typ jak funkcja `koszt_najmu_obiektu` która jest wywoływana - czyli `DECIMAL(15, 2)` który jest kompatybilny z typem kolumny `koszt` w tabeli `najmy`. Przykład wykorzystania tej funkcji jest w triggerze z listingu 14 (strona 12).

```
CREATE FUNCTION koszt_najmu (@najem_id INT)
  RETURNS DECIMAL(15, 2)
  AS
  BEGIN
    DECLARE @koszt DECIMAL(15, 2);
    DECLARE @liczba_dni INT;
    DECLARE @obiekt_id INT;
    DECLARE @data_rozpoczecia DATE;
    DECLARE @data_zakonczenia DATE;

    SELECT @data_rozpoczecia = n.data_rozpoczecia,
           @data_zakonczenia = n.data_zakonczenia,
           @obiekt_id = o.id
      FROM najmy n
      JOIN obiekty o on n.obiekt_id = o.id
      WHERE n.id = @najem_id;

    IF @data_zakonczenia IS NULL
      RETURN NULL;

    SELECT @liczba_dni = (1+DATEDIFF(DAY, @data_rozpoczecia, @data_zakonczenia));

    SELECT @koszt = dbo.koszt_najmu_obiektu(@obiekt_id, @liczba_dni);

    RETURN @koszt;
  END
```

Listing 11: Skrypt tworzący funkcję skalarną `koszt_najmu`

5.6.3 `adresowka` - Funkcja generująca etykietę adresową dla użytkownika

`adresowka (@uzytkownik_id INT)` - Funkcja ta generuje zawartość etykiety adresowej dla użytkownika wskazanego w parametrze `@uzytkownik_id`, zwracana wartość ma typ VARCHAR(333)[2]. Przykład wykorzystania tej funkcji jest w funkcji tabelarnej z listingu 13.

```
CREATE FUNCTION adresowka (@uzytkownik_id INT)
  RETURNS VARCHAR(333) -- 75+75+150+30+3 = 333 - suma długości czonych pól
  AS
  BEGIN
    DECLARE @adresowka VARCHAR(330);

    SELECT @adresowka = nazwisko + ' ' + imie + CHAR(13) + telefon + CHAR(13) + adres
      FROM uzytkownicy
      WHERE id = @uzytkownik_id;

    RETURN @adresowka;
  END
```

Listing 12: Skrypt tworzący funkcję skalarną `adresowka`

5.6.4 `opoznieni` - Funkcja generująca etykiety dla użytkowników którzy coś wynajęli ale jeszcze nie oddali

`opoznieni (@liczba_dni INT)` - Funkcja ta generuje listę obiektów wraz z najemcami, które zostały wynajęte co najmniej liczbę dni wcześniej określoną parametrem `@liczba_dni`. Funkcja ta zwraca tabelę składającą się z identyfikatora najmu, nazwy wynajętego obiektu, liczby dni od rozpoczęcia najmu, szacunkowego kosztu tego najmu na dzień bieżący oraz etykiety adresowej do najemcy.

	najem_id	nazwa_obiektu	dni_od_wynajmu	szacunkowy_koszt_najmu	etykieta
1	33	Nazwa 1	3	599.96	Nazwisko 4 Imie 3✉+48 987 543 123✉Adres 4
2	36	Nazwa 1	4	600.00	Nazwisko 3 Imie 1✉+48 234 567 890✉Adres 2
3	37	Nazwa 4	3	480.00	Nazwisko 4 Imie 2✉+48 987 543 123✉Adres 2
4	38	Nazwa 3	4	320.25	Nazwisko 3 Imie 1✉+48 234 567 890✉Adres 2
5	48	Nazwa 1	4	320.25	Nazwisko 4 Imie 3✉+48 987 543 123✉Adres 4

Rysunek 7: Uruchomiona funkcja `opoznieni` z parametrem `@liczba_dni` równym 3

```
CREATE FUNCTION opoznieni (@liczba_dni INT)
  RETURNS TABLE
  AS
    RETURN (
      SELECT n.id najem_id,
             o.nazwa nazwa_obiektu,
             DATEDIFF(DAY, n.data_rozpoczecia, getdate()) dni_od_wynajmu,
             dbo.koszt_najmu_obiektu(o.id, DATEDIFF(DAY, n.data_rozpoczecia, getdate()) + 1)
    szacunkowy_koszt_najmu,
             dbo.adresowka(u.id) etykieta
        FROM najmy n
        JOIN obiekty o on n.obiekt_id = o.id
        JOIN uzytkownicy u on n.uzytkownik_id = u.id
        WHERE DATEDIFF(DAY, n.data_rozpoczecia, getdate()) >= @liczba_dni
              AND n.data_zakonczenia IS NULL
    )
```

Listing 13: Skrypt tworzący funkcję tabelarną `opoznieni`

5.7 Triggery

Zostały stworzone następujące triggery:

- `wylicz_koszt_najmu` - Funkcja zapisująca koszt najmu

5.7.1 `wylicz_koszt_najmu` - Funkcja zapisująca koszt najmu

Trigger ten jest uruchamiany w momencie dodania nowego wiersza do tabeli `najmu` lub aktualizacji już istniejącego. Klauzulą WHERE ograniczamy obliczenia tylko do wierszy nowych lub wierszy gdzie kolumna `data_zakonczenia` została zaktualizowana. Dzięki wykorzystaniu wcześniej przygotowanej funkcji skalarnej

[2]Rozmiar pola bierze się z sumy długości użytych pól (`nazwisko` i `imie` po 75 znaków, `telefon` 30 znaków,`adres` 150 znaków) oraz znaków dodanych (jedna spacja i dwa znaki nowej lini - `CHAR(13)`) - jest to najdłuższy możliwy wynik tej funkcji.

`koszt_najmu` (listing 11), kod tego triggera jest stosunkowo prosty i przejrzysty. Trigger został zaprojektowany w taki sposób aby możliwe było wykonywanie zbiorowych operacji - dzięki temu możemy dodawać/aktualizować wiele wierszy a i tak trigger będzie działać prawidłowo. Przykładowy wynik działania widzimy na rysunku 5 ze strony 9 - po dodaniu losowych danych testowych, koszty najmów zostały automatycznie obliczone.

```
CREATE TRIGGER wylicz_koszt_najmu
  ON najmy
  AFTER INSERT, UPDATE
  AS
  BEGIN
    UPDATE najmy
      SET koszt = dbo.koszt_najmu(n.id)
      FROM najmy n
      JOIN inserted i ON n.id = i.id
      WHERE UPDATE (data_zakonczenia) OR NOT EXISTS(SELECT 1 FROM DELETED)
  END
```

Listing 14: Skrypt tworzący trigger `wylicz_koszt_najmu`

5.8 Procedury Składowane

5.9 Skrypty w oparciu o kursory

5.10 Inne poznane obiekty, własności bazy danych

5.11 Skrypt tworzący użytkowników i nadający uprawnienia

8. SKRYPT TWORZACY UZYTKOWNIKOW BAZY DANYCH ¡KONTA, UPRAWNIENIA itp¿ W FORMIE: a. OPIS UZYTKOWNIKOW ¡ICH ZADANIA¿ b. SKŁADNIA SKRYPTU c. ZRZUTY EKRANU Z: UTWORZENIA UZYTKOWNIKOW, UPRAWNIENIA, TEST MOZLIWOSCI/NIE MOZLIWOSCI KAZDEGO Z NICH

6 Skrypt usuwający obiekty z bazy danych

Ponieważ do każdej operacji tworzącej lub modyfikującej obiekty w bazie danych została napisana również operacja odwrotna, to możliwe jest wygenerowanie skryptu skrypt_usuwajacy_obiekty_z_bazy.sql.

6.1 Wynik uruchomienia całego skryptu usuwającego obiekty w trybie wsadowym

Jak widać na listingu 15, skrypt podaje bardzo dokładne informacje na temat aktualnie wykonywanej operacji. W przypadku tego skryptu, operacje są wykonywane w odwrotnej kolejności niż w skrypcie tworzącym z listingu 3.

```
Wersja 21: 'Utworzenie funkcji wyswietlajacej spozniajacych sie uzytkownikow'

(1 rows affected)
Wersja 21: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 20
Wersja 20: 'Utworzenie funkcji wyswietlajacej adresowke uzytkownika'

(1 rows affected)
Wersja 20: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 19
Wersja 19: 'Utworzenie triggeru aktualizujacego koszt najmu'

(1 rows affected)
Wersja 19: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 18
Wersja 18: 'Utworzenie funkcji wyliczajacej koszt konkretnego najmu'

(1 rows affected)
Wersja 18: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 17
Wersja 17: 'Utworzenie funkcji wyliczajacej koszt najmu obiektu'

(1 rows affected)
Wersja 17: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 16
Wersja 16: 'Utworzenie widoku z lista niewynajmowanych obiektow'

(1 rows affected)
Wersja 16: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 15
Wersja 15: 'Utworzenie widoku z lista popularnosci obiektow'

(1 rows affected)
Wersja 15: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 14
Wersja 14: 'Utworzenie widoku z lista wszystkich najmow'

(1 rows affected)
Wersja 14: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 13
Wersja 13: 'Utworzenie indeksu unikatowego w tabeli z uzytkownikami'

(1 rows affected)
Wersja 13: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 12
Wersja 12: 'Utworzenie relacji pomiedzy obiektami a najmami'

(1 rows affected)
Wersja 12: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 11
Wersja 11: 'Utworzenie relacji pomiedzy uzytkownikami a najmami'

(1 rows affected)
Wersja 11: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 10
Wersja 10: 'Utworzenie indeksu unikatowego w tabeli z najemcami'

(1 rows affected)
Wersja 10: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 9
Wersja 9: 'Utworzenie tabeli z najmami'

(1 rows affected)
Wersja 9: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 8
Wersja 8: 'Utworzenie tabeli z uzytkownikami'

(1 rows affected)
Wersja 8: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 7
Wersja 7: 'Utworzenie relacji pomiedzy kategoriami a obiektami'

(1 rows affected)
Wersja 7: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 6
Wersja 6: 'Utworzenie relacji pomiedzy dzielnicami a obiektami'

```

```
(1 rows affected)
Wersja 6: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 5
Wersja 5: 'Utworzenie tabeli z obiektami'

(1 rows affected)
Wersja 5: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 4
Wersja 4: 'Utworzenie tabeli z kategoriami'

(1 rows affected)
Wersja 4: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 3
Wersja 3: 'Utworzenie relacji pomiedzy miastami a dzielnicami'

(1 rows affected)
Wersja 3: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 2
Wersja 2: 'Utworzenie tabeli z dzielnicami'

(1 rows affected)
Wersja 2: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 1
Wersja 1: 'Utworzenie tabeli z miastami'

(1 rows affected)
Wersja 1: Migracja zostala odinstalowana pomyslnie - teraz baza jest w wersji 0
Tabela wersjonowania zostala skasowana
```

Listing 15: Wynik uruchomienia całego skryptu usuwającego obiekty w trybie wsadowym

7 Dane Testowe

7.1 Skrypt tworzący dane testowe

W skrypcie tworzącym, w pierwszej kolejności definiujemy zbiór danych testowych (na przykład tak jak na listingu 16), a w dalszej części tworzymy i dodajemy do bazy danych, dane testowe (listing 17 lub listing 18).

Część tabel, jak na przykład tabela `uzytkownicy`, jest wypełniana w sposób losowy[3] (Listing 18). Oznacza to że wraz z każdym uruchomieniem skryptu dane dodane do bazy danych będą inne.

```
-- Przygotowanie danych testowych
DECLARE @MIASTA TABLE (nazwa VARCHAR(20));
INSERT INTO @MIASTA VALUES ('Miasto 1'),('Miasto 2'),('Miasto 3'),('Miasto 4');
```

Listing 16: Fragment deklaracji danych testowych

```
-- Wstawianie danych do tabeli 'dzielnice'
INSERT INTO dzielnice (miasto_id, nazwa)
  SELECT m.id miasto_id, d.nazwa
  FROM miasta m
  CROSS JOIN @DZIELNICE d;

SELECT @liczba_wierszy = @@ROWCOUNT;
PRINT 'Do tabeli dzielnice dodano '+CAST(@liczba_wierszy AS VARCHAR)+' wiersz(y).';
```

Listing 17: Fragment prostego tworzenia danych testowych

```
-- Wstawianie danych do tabeli 'uzytkownicy'
INSERT INTO uzytkownicy (login, nazwisko, imie, wiek, adres, telefon, plec)
  SELECT l.login login,
    (SELECT TOP 1 nazwisko from @NAZWISKA WHERE l.login IS NOT NULL ORDER BY NewID())
    nazwisko,
    (SELECT TOP 1 imie from @IMIONA WHERE l.login IS NOT NULL ORDER BY NewID()) imie,
    (SELECT TOP 1 wiek from @WIEKI WHERE l.login IS NOT NULL ORDER BY NewID()) wiek,
    (SELECT TOP 1 adres from @ADRESY WHERE l.login IS NOT NULL ORDER BY NewID()) adres,
    (SELECT TOP 1 telefon from @TELEFONY WHERE l.login IS NOT NULL ORDER BY NewID()) telefon,
    (SELECT TOP 1 plec from @PLCI WHERE l.login IS NOT NULL ORDER BY NewID()) plec
  FROM @LOGINY l;

SELECT @liczba_wierszy = @@ROWCOUNT;
PRINT 'Do tabeli uzytkownicy dodano '+CAST(@liczba_wierszy AS VARCHAR)+' wiersz(y).';
```

Listing 18: Fragment losowego tworzenia danych testowych

7.1.1 Wynik uruchomienia skryptu tworzącego dane testowe w trybie wsadowym

W momencie wykonywania skryptu, podaje on liczbę dodanych wierszy do każdej z tabel, w formacie takim jak na listingu 19.

```
Do tabeli miasta dodano 4 wiersz(y).
Do tabeli dzielnice dodano 16 wiersz(y).
Do tabeli kategorie dodano 4 wiersz(y).
Do tabeli uzytkownicy dodano 4 wiersz(y).
Do tabeli obiekty dodano 64 wiersz(y).
Do tabeli najmy dodano 48 wiersz(y).
```

Listing 19: Wynik uruchomienia całego skryptu tworzącego dane testowe w trybie wsadowym

7.2 Skrypt usuwający dane testowe

Ponieważ nasze tabele posiadają kolumny autonumerowanie (typu `IDENTITY`), to nie możemy skasować danych przy pomocy funkcji `TRUNCATE`. Aby uzyskać podobny wynik (opróżnienie tabeli) wykorzystuję dyrektywę `DBCC CHECKIDENT(<tabela>, RESEED, 0)` która powoduje zresetowanie autonumerowania w tabeli.

```
DELETE FROM najmy WHERE 1=1 DBCC CHECKIDENT (najmy, RESEED, 0);
PRINT 'Tabela ''najmy'' zosta a opr  niona';

DELETE FROM uzytkownicy WHERE 1=1 DBCC CHECKIDENT (uzytkownicy, RESEED, 0);
PRINT 'Tabela ''uzytkownicy'' zosta a opr  niona';
```

[3] Zastanawiający może być warunek `WHERE l.login IS NOT NULL` - jest to pewnego rodzaju "obejście" procesu optymalizacji SQL Server'a. Gdybyśmy nie zastosowali takiej struktury to dane były by wylosowane tylko za pierwszym razem - czyli wszystkie wiersze miały by taką samą wartość danego pola, a w tym przypadku takie zachowanie jest zachowaniem niepożądanym.

```
6
7  DELETE FROM obiekty WHERE 1=1 DBCC CHECKIDENT (obiekty, RESEED, 0);
8  PRINT 'Tabela ''obiekty'' zosta a opr  niona';
9
10 DELETE FROM kategorie WHERE 1=1 DBCC CHECKIDENT (kategorie, RESEED, 0);
11 PRINT 'Tabela ''kategorie'' zosta a opr  niona';
12
13 DELETE FROM dzielnice WHERE 1=1 DBCC CHECKIDENT (dzielnice, RESEED, 0);
14 PRINT 'Tabela ''dzielnice'' zosta a opr  niona';
15
16 DELETE FROM miasta WHERE 1=1 DBCC CHECKIDENT (miasta, RESEED, 0);
17 PRINT 'Tabela ''miasta'' zosta a opr  niona';
```

Listing 20: Skrypt usuwający dane testowe

7.2.1 Wynik uruchomienia skryptu usuwającego dane testowe w trybie wsadowym

```
1
2  (48 rows affected)
3  Checking identity information: current identity value '48'.
4  DBCC execution completed. If DBCC printed error messages, contact your system administrator.
5  Tabela 'najmy' zostala opr zniona
6
7  (4 rows affected)
8  Checking identity information: current identity value '4'.
9  DBCC execution completed. If DBCC printed error messages, contact your system administrator.
10 Tabela 'uzytkownicy' zostala opr zniona
11
12 (64 rows affected)
13 Checking identity information: current identity value '64'.
14 DBCC execution completed. If DBCC printed error messages, contact your system administrator.
15 Tabela 'obiekty' zostala opr zniona
16
17 (4 rows affected)
18 Checking identity information: current identity value '4'.
19 DBCC execution completed. If DBCC printed error messages, contact your system administrator.
20 Tabela 'kategorie' zostala opr zniona
21
22 (16 rows affected)
23 Checking identity information: current identity value '16'.
24 DBCC execution completed. If DBCC printed error messages, contact your system administrator.
25 Tabela 'dzielnice' zostala opr zniona
26
27 (4 rows affected)
28 Checking identity information: current identity value '4'.
29 DBCC execution completed. If DBCC printed error messages, contact your system administrator.
30 Tabela 'miasta' zostala opr zniona
```

Listing 21: Wynik uruchomienia całego skryptu usuwającego dane testowe w trybie wsadowym

Spis listingów

Spis rysunków