Bazy Danych
projekt

Janusz Długosz — 235746
Jakub Dorda — 235013
Marcin Kotas — 235098

Prowadzący:
Dr inż. Dariusz Jankowski

Wrocław, 9 czerwca 2018

Spis treści

1 Wstęp teoretyczny

1.1 Podstawy relacyjnych baz danych

Relacyjna baza danych jest zbiorem schematów tabel i relacji między nimi. Służą do przechowywania przeważnie dużych zbiorów danych w ściśle określony sposób. Relacyjne bazy danych są jednym z trzech istniejących komercyjnie modelów, oprócz niej istnieją również bazy hierarchiczne i obiektowe, jednak nie cieszą się one tak dużą popularnością. Model relacyjny został opracowany przez Edgara Frank Codda w latach 70 ubiegłego wieku. Model organizuje dane w jedną lub więcej tabel składającą się z kolumn i rekordów (wierszy). Każdy rekord musi posiadać unikalny klucz, który go identyfikuje. Generalnie jedna tabela reprezentuje jeden typ encji (np. budynek). Rekordy przedstawiają instancje encji (np. ratusz), a w kolumnach przechowywane są atrybuty i informacje (np. rozmiar, rok budowy).

Tabela
To struktura przechowująca dane ściśle określonego typu. Tabela zawiera rekordy, które posiadają swoje atrybuty. Struktury te można łączyć relacjami.

Klucze
Każdy rekord posiada swój unikalny klucz. Pozwalają one na jednoznaczną identyfikację wiersza. Dzięki tym kluczom można łączyć rekordy między różnymi tabelami. To umożliwia modelowanie relacji.

- **KLUCZ PODSTAWOWY (PRIMARY KEY)**
  To klucz identyfikujący jednoznacznie wiersz. Tabela może mieć tylko jeden taki klucz (nie może się powtarzać). Klucze te dzielą się na naturalne oraz sztuczne. Naturalne takie jak np. numer PESEL lub e-mail (koniecznie musi być unikalny) jest z punktu widzenia systemu tak samo jak inne atrybuty jak np. nazwa firmy. Jeśli nie istnieje rzeczywisty identyfikator to nadawany jest klucz sztuczny, który znajduje się w dodatkowo stworzonej kolumnie z możliwie jak najkrótszym kluczem. Zazwyczaj będą to kolejne liczby naturalne.
- **KLUCZ OBCY (FOREIGN KEY)**
  Są to atrybuty, które wskazują na klucz podstawowy w innej tabeli, jest to po prostu relacja między dwoma tabelami. W tabeli, która jest powiązana kluczem obcym należy powielić całą strukturę, aby możliwe było wiązanie rekordów z dwóch różnych tabel. Z definicji pilnowane jest, aby w wartościach klucza obcego, mogły się znaleźć tylko wartości rzeczywiście istniejące jako klucz główny w innej tabeli. Klucz obcy może oczywiście dotyczyć również tej samej tabeli.

Relacje
Relacje opisują związki między tabelami. Dobrze zaprojektowane relacje znacznie upraszczają prawidłowe działanie bazy danych, poziom skomplikowania i czytelność kwerend. Za ich pomocą projektowana jest odpowiednia logika struktury bazy.

- **RELACJA 1:1**
  Każdy rekord w pierwszej tabeli może mieć tylko jednego odpowiednika w drugiej tabeli i vice versa. Taką sytuację można rozpatrywać jako jedną dużą tabelę podzieloną na dwie mniejsze. Stosowana gdy część atrybutów tabeli można oddzielić, ponieważ może zostać użyta jako część innej tabli lub dotyczy tylko części atrybutów całej tabeli. Działanie takie poprawia czytelność oraz funkcjonalność bazy danych. Można też wydzielić część atrybutów, które są rzadko odpytywane.

- **RELACJA 1:N**
  Jest to relacja występująca najczęściej w relacyjnym modelu bazy danych. Występuje kiedy jeden element pierwszego zbioru może zostać powiązany z wieloma elementami zbioru drugiego.

- **RELACJA N:N**
  Realizowany jest jako dwie relacje 1:N. Jeśli między tabelą pierwszą, a drugą ma zostać zaprojektowana taka relacja to potrzebna jest jeszcze trzecia tabela, która będzie pełniła funkcję łącznika.

Operacje na tabelach

Są to operacje, które można wykonać w relacyjnych bazach danych. Pierwsze cztery bazują na matematycznej teorii mnogości.

- `UNION` - suma zbiorów, zwraca wszystkie rekordy pierwszej i drugiej tabeli bez duplikatów.
- `INTERSECTION` - iloczyn zbiorów, zwraca tylko rekordy będące częścią wspólną pierwszej i drugiej tabeli.
- `EXCEPT` - różnica zbiorów, zwraca rekordy z pierwszej tabeli bez części wspólnej pierwszej i drugiej tabeli.
- `CROSS JOIN` - iloczyn kartezjański, zwraca iloczyn kartezjański dwóch tabel, czyli wszystkie możliwe kombinacje połączenia rekordów tych tabel.
- `SELECT` + `WHERE` - zwraca wybrane rekordy z tabeli, które spełniają określony warunek.
- `JOIN` - zwraca połączone tabele, które łączy relacja.
- **PROJECTION OPERATION** - zwraca tylko wybrane atrybuty z rekordów.
- **DIVISION** - jest operacją przeciwną do ilorazu kartezjańskiego.

1.2 Normalizacja

Normalizacja to bezstratny proces organizowania danych w tabelach mający na celu zmniejszenie ilości danych składowanych w bazie oraz wyeliminowanie potencjalnych anomalii. Określa się następujące postacie normalne baz:

1. Pierwsza postać normalna 1NF:
   Dane w tablicach są w postaci atomowej (jedna zmienna na krotkę) oraz każda encja posiada klucz główny.
2. Druga postać normalna 2NF:
   Każda tabela powinna przechowywać dane dotyczące tylko konkretnej klasy obiektów.
3. Trzecia postać normalna 3NF:
   Czyli każdy argument nie będący kluczem jest bezpośrednio zależny tylko od klucza głównego, a nie od innej kolumny.

Występują jeszcze kolejne postacie normalne aczkolwiek są w zasadzie używane wyłącznie przy okazji rozważań teoretycznych.

Do najważniejszych zalet normalizacji można zaliczyć:

- eliminacja powtarzających się kolumn
- zmniejszenie szansy wystąpienia anomalii w danych
- optymalizacja operacji na bazie danych
- czytelność struktury bazy
- efektywne przechowywanie danych na dysku

Korzyści te uzyskuje się kosztem spowolnienia wykonywania niektórych zapytań (np. wymagających złączeń).

Niekiedy w celu zwiększenia szybkości wykonywania zapytań dokonuje się procesu odwrotnego — denormalizacji. Unika się w ten sposób kosztów obliczeniowych związanych z procesem łączenia tabel. Można również pozostawić kolumnę zależną od argumentów niekluczowych w przypadku, gdy wyliczenie wartości jest złożone obliczeniowo lub wymagany jest częsty dostęp do tych danych.

Proces normalizacji realizowanego systemu

Na początku zaprojektowana została tablica `Apartments`, która przechowywała wszystkie informacje o apartamentach. Wydedukowano, iż dane o lokalizacji warto umieścić w osobnej encji, ponieważ dotyczą innej klasy obiektu. Odseparowano również dane o właścicielu do tabeli `Users`, która przechowuje również informacje o użytkownikach będących klientami. Dodatkowo stworzono encję `Agencies`, która zawiera dane o agencjach.

System wymaga przechowywania historii wszystkich rezerwacji. Do tego celu utworzona została tablica `Reservations`. W tabeli nie zostały umieszczone informacje o rezerwującym użytkowniku ani o apartamencie w celu uniknięcia duplikowania danych. Zamiast tego przechowywane są jedynie klucze obce łączące odpowiednie rekordy. Z encji wydzielono dane o płatności do tabeli `Payments`.

Uzyskane relacje przedstawione zostały na diagramie ERD (rozdział 2.4) oraz w tablicy 5.

2 Część praktyczna projektu

2.1 Przedstawienie problemu

Zadanie projektowe ma na celu stworzenie bazy danych do szybkiej, bezpieczniej, bezproblemowej rezerwacji apartamentów oraz możliwości stworzenia aplikacji korzystającej z tej bazy danych. W dzisiejszych czasach ludzie bardzo często dużo pracują i są przemęczeni, notorycznie brakuje im czasu. Planując urlop chcieliby szybko i łatwo znaleźć sobie miejsce na chwilę relaksu. Wielu ludzi nie potrafi samemu sobie wyszukać i zorganizować takiego wypoczynku, a poza tym szukanie apartamentów w różnych miejscach, stronach internetowych jest czasochłonne i nieefektywne.

- Użytkownik powinien być w stanie szybko wyszukać zakwaterowanie na całym świecie i przede wszystkim porównać je za pomocą interesujących go parametrów takich jak cena, lokalizacja, rozmiar.
- Dużym atutem byłaby możliwość sortowania rosnąco lub malejąco według parametrów oraz mechanizm filtracji.
- Kolejnym problemem jaki może napotkać użytkownik jest przebieg płatności. Klientowi zależy, aby płatność była bezpieczna, szybka i uniwersalna, czyli taka sama dla rezerwacji każdego apartamentu.

- Użytkownik powinien mieć możliwość stworzenia konta i śledzenia historii swoich rezerwacji oraz anulowania jej.

Odbiorcą takiego systemu będzie zwykły człowiek, więc system powinien być jak najprostszy i najbardziej intuicyjny w obsłudze, żeby nawet osoba starsza lub w ogóle niezaznajomiona z technologią mogła z niej z łatwością korzystać. Podmiot obsługujący aplikację oraz bazę danych powinien nawiązać współpracę z jak największą ilością agencji zajmujących się wynajmem apartamentów, aby zapewnić jak największy wybór zakwaterowania. Aplikacja powinna być wykonana w technologii webowej. Umożliwiłoby to użytkownikowi dostęp do niej w każdym miejscu bez instalowania dodatkowego oprogramowania.

2.2 Wymagania systemu

System powinien umożliwiać wyszukiwanie apartamentów według wybranych kryteriów. W ogólnym przypadku podane zostanie jedynie państwo oraz ilość łózek, wtedy system powinien zwrócić wszystkie apartamenty danego rozmiaru w wybranym kraju. Dodatkowymi argumentami są maksymalna cena za noc, klimatyzacja na wyposażeniu oraz standard apartamentu (liczba gwiazdek).

Dostępna musi być funkcja sprawdzająca dostępność apartamentu. System zwraca daty wszystkich rezerwacji związanych z wybraną kwaterą. Anulowane rezerwacje nie powinny zostać uwzględnione w wyszukiwaniu.

Złożenie rezerwacji wymaga podania identyfikatora apartamentu i użytkownika oraz dat początkowej i końcowej pobytu. Na podstawie tych danych system tworzy nowy wpis w tablicy rezerwacji ze statusem 'złożona'. Właściciel apartamentów musi mieć możliwość przejrzenia wszystkich rezerwacji związanych z jego apartamentami oraz zaakceptowania oczekujących (złożonych) rezerwacji. Po zaakceptowaniu rezerwacji przez właściciela system powinien zarejestrować nową wpłatę ze statusem 'rozpoczęta' oraz wymaganym typem (zaliczka lub całość). Każda wpłata powiązana jest z identyfikatorem odpowiedniej rezerwacji.

System umożliwia zmianę statusu płatności na wykonaną lub anulowaną w zależności od potrzeby.

Możliwe musi być również anulowanie rezerwacji. W takiej sytuacji system zmienia status rezerwacji na anulowaną, oraz jeżeli wpłacona była cała opłata to zmienia stan wpłaty na zwróconą (następuje zwrot 80% wartości, zaliczka nie jest zwracana).

Użytkownik zarejestrowany jako właściciel musi mieć możliwość dodawania nowych apartamentów.

Klient może wyświetlić wszystkie swoje rezerwacje.

2.3 Model danych ERD

a) Identyfikacja zbioru encji wraz z ich atrybutami kluczowymi

W systemie wyodrębnione zostały następujące encje oraz identyfikujące je atrybuty:

Tablica 1: Encje i identyfikatory

Encja	Atrybut
Locations	Location_ID
Agencies	Agency_ID
Users	User_ID
Apartments	Apartment_ID
Reservations	Reservation_ID
Payments	Payment_ID

b) Identyfikacja bezpośrednich zależności między encjami

Badane jest bezpośrednie powiązanie pomiędzy wszystkimi parami obiektów w systemie. Jeżeli któreś dane obiektu A powiązane są z obiektem B, to w tabeli (macierzy) wstawiany jest X w miejscu $table[A][B]$. Uzyskane są w ten sposób relacje pomiędzy encjami. Wszystkie połączenia przedstawione zostały w postaci tabeli krzyżowej (tablica 2), jest ona symetryczna względem głównej przekątnej. Dokładna specyfika relacji przedstawiona została w tablicy 5.

Tabela 3 zawiera opisy wszystkich atrybutów używanych w encjach, natomiast ich przynależność do odpowiednich encji przedstawiona jest w tabeli 4.

Na podstawie uzyskanych wyników sporządzony został diagram ERD przedstawiony w rozdziale 2.4.

Tablica 2: Tabela krzyżowa, zależności bezpośrednie pomiędzy encjami

	Locations	Agencies	Users	Apartments	Reservations	Payments
Locations		X	X	X		
Agencies	X			X		
Users	X			X	X	
Apartments	X	X	X		X	
Reservations			X	X		X
Payments					X	

Tablica 3: Opis atrybutów

Atrybut	Opis
Location_ID	Identyfikator lokalizacji
Country_Name	Nazwa państwa
City	Nazwa miasta
Longitude	Długość geograficzna
Latitude	Szerokość geograficzna
Street	Adres
Street2	Adres c.d.
Apartment_Number	Nr mieszkania
Zip_Code	Kod pocztowy
Agency_ID	Identyfikator agencji
Agencies.Name	Nazwa agencji
Agencies.Info	Informacje o agencji
Phone	Nr telefonu
Web	Strona internetowa
User_ID	Identyfikator użytkownika
Users.Type	Typ użytkownika (właściciel | klient)
Email	Adres email
Password	Hasło użytkownika
Users.Info	Informacje o użytkowniku
Users.Name	Imię użytkownika
Last_Name	Nazwisko użytkownika
Apartment_ID	Identyfikator apartamentu
Cost_Per_Night	Koszt za jedną noc
Bed_Count	Liczba łóżek w apartamencie
Air_Cond	Klimatyzacja w apartamencie na wyposażeniu
Standard	Standard apartamentu (1 — 5)
Owner_ID	Identyfikator właściciela apartamentu
Reservation_ID	Identyfikator rezerwacji
Date_Begin	Data rozpoczęcia pobytu
Date_End	Data zakończenia pobytu
Reservations.Status	Status rezerwacji (rozpoczęta | zaakceptowana | anulowana)
Payment_ID	Identyfikator wpłaty
Payments.Type	Typ wpłaty (zaliczka | całość)
Value	Wartość wpłaty
Payments.Status	Status wpłaty (rozpoczęta | zakończona | anulowana | zwrócona)

Tablica 4: Wykaz encji i powiązanych z nimi atrybutów

Encja	Atrybut
Locations	Location_ID, Country_Name, City, Longitude, Latitude, Street, Street2, Apartment_Number, Zip_Code
Agencies	Agency_ID, Location_ID, Name, Info, Phone, Web
Users	User_ID, Type, Email, Password, Info, Phone, Name, Last_Name, Location_ID
Apartments	Apartment_ID, Cost_Per_Night, Bed_Count, Location_ID, Agency_ID, Air_Cond, Standard, Owner_ID
Reservations	Reservation_ID, Date_Begin, Date_End, Apartment_ID, User_ID, Status
Payments	Payment_ID, Reservation_ID, Type, Value, Status

Tablica 5: Opis związków pomiędzy encjami

Związek	Opis
Posiada	Agencies — Locations; 1:1
Posiada	Users — Locations; 1:1
Posiada	Apartments — Locations; 1:1
Ma	Apartments — Agencies; N:1
Ma	Apartments — Users; N:1
Dotyczy	Reservations — Apartments; N:1
Dotyczy	Reservations — Users; N:1
Dotyczy	Payments — Reservations; 1:1

2.4 Schemat diagramu ERD

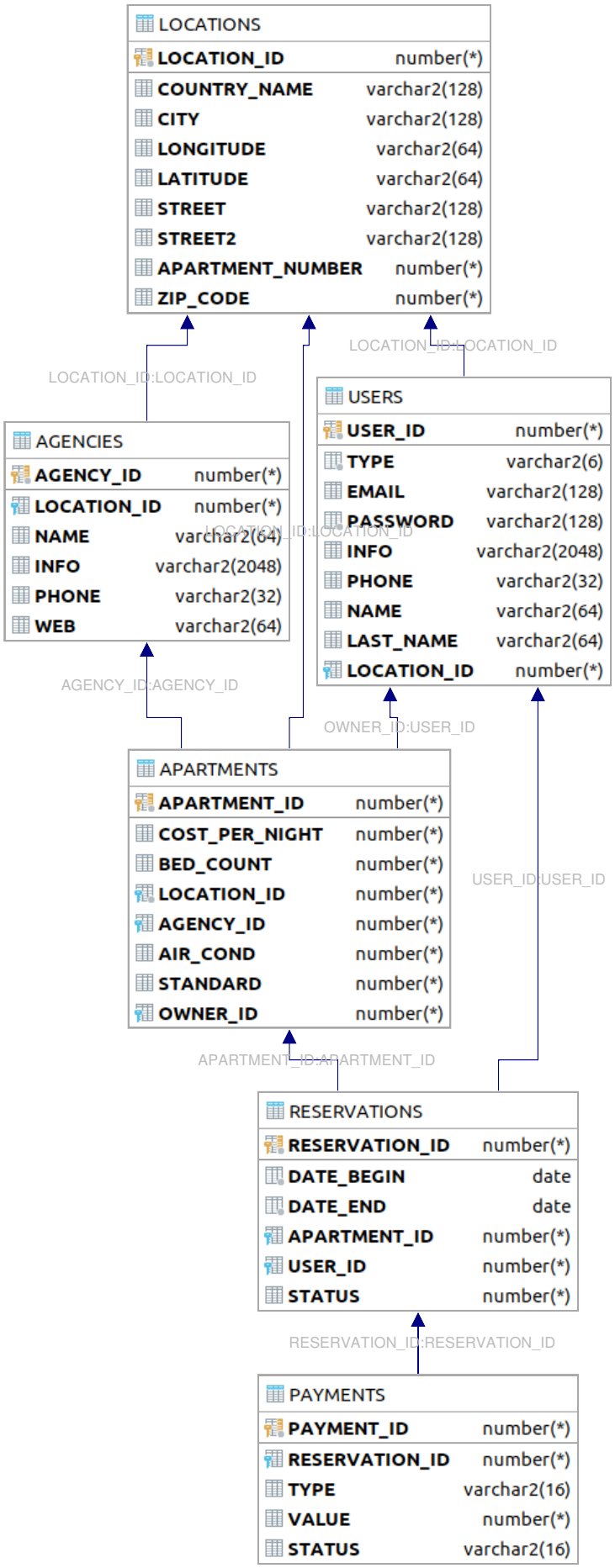
a) Opis aplikacji w której modelowano schemat

W celu wygenerowania diagramu ERD użyto aplikacji **DataGrip** oferowanej przez firmę JetBrains. Jest to IDE obsługujące większość popularnych odmian SQL oraz związanych z nimi systemów bazodanowych.

Aby wygenerować diagram ERD należy nawiązać połączenie z wybranym serverem hostującym system bazodanowy. Po zalogowaniu kliknąć prawym przyciskiem myszy na obecne połączenie i z menu kontekstowego wybrać opcję **'Diagrams' -> 'Show visualization'**, po kliknięciu w nowej zakładce wygeneruje się wizualizacja ERD. Diagram jest automatycznie generowany z encji dostępnych dla zalogowanego użytkownika. Widok pozwala jedynie na odczyt struktury bazy danych w formie graficznym, bez możliwości edycji. Użytkownik ma możliwość eksportu widoku do pliku w formacie graficznym, diagramu UML oraz PostScript lub wydrukowania diagramu na drukarce. Zmiany uwzględniane są na bieżąco po każdej operacji ingerującej w strukturę bazy danych.

Wizualizacja została zaimplementowana z wykorzystaniem biblioteki **yFiles**

b) Prezentacja schematu ERD bazy danych


LOCATIONS
LOCATION_ID number(*)
COUNTRY_NAME varchar2(128)
CITY varchar2(128)
LONGITUDE varchar2(64)
LATITUDE varchar2(64)
STREET varchar2(128)
STREET2 varchar2(128)
APARTMENT_NUMBER number(*)
ZIP_CODE number(*)
LOCATION_ID:LOCATION_ID
LOCATION_ID:LOCATION_ID
USERS
USER_ID number(*)
TYPE varchar2(6)
EMAIL varchar2(128)
PASSWORD varchar2(128)
INFO varchar2(2048)
PHONE varchar2(32)
NAME varchar2(64)
LAST_NAME varchar2(64)
LOCATION_ID number(*)
AGENCIES
AGENCY_ID number(*)
LOCATION_ID number(*)
NAME varchar2(64)
INFO varchar2(2048)
PHONE varchar2(32)
WEB varchar2(64)
AGENCY_ID:AGENCY_ID
OWNER_ID:USER_ID
APARTMENTS
APARTMENT_ID number(*)
COST_PER_NIGHT number(*)
BED_COUNT number(*)
LOCATION_ID number(*)
AGENCY_ID number(*)
AIR_COND number(*)
STANDARD number(*)
OWNER_ID number(*)
USER_ID:USER_ID
APARTMENT_ID:APARTMENT_ID
RESERVATIONS
RESERVATION_ID number(*)
DATE_BEGIN date
DATE_END date
APARTMENT_ID number(*)
USER_ID number(*)
STATUS number(*)
RESERVATION_ID:RESERVATION_ID
PAYMENTS
PAYMENT_ID number(*)
RESERVATION_ID number(*)
TYPE varchar2(16)
VALUE number(*)
STATUS varchar2(16)

2.5 Rozwiązanie problemu

a) System bazodanowy

- **Utworzenie bazy danych**

Zgodnie z założeniami projektu wykorzystano system bazodanowy oferowany przez firmę Oracle - **Oracle Database 11g Express Edition Release 2** w wersji na system Linux z architekturą 64 bitową. Firma Oracle wspiera jedynie dystrybucje Linuxa wykorzystujące **RPM Package Manager** jako system dystrybucji paczek. Oracle Database jest produktem komercyjnym z zamkniętym źródłem, więc nie istnieje możliwość samodzielnej kompilacji na dowolnej dystrybucji Linuxa.

W celu usprawnienia pracy nad projektem oraz umożliwieniem dostępu do bazy danych dla każdego członka grupy projektowej, za pośrednictwem strony **digitalocean.com** utworzono maszynę wirtualną Linuxa z dystrybucją **CentOS 7.5**, który wspiera paczki RPM. Po utworzeniu maszyny i podaniu publicznego klucza RSA można zalogować się na nowo utworzony serwer przy pomocy SSH. Każda maszyna wirtualna otrzymuje bezpośredni, zewnętrzny adres IP.

W celu instalacji produktu Oracle należy najpierw zarejestrować się na stronie korporacji. Po zalogowaniu otrzymujemy link z indywidualnym tokenem autoryzacyjnym, po skopiowaniu linku możemy pobrać spakowaną paczkę przy pomocy programu **curl** lub **wget** bezpośrednio na nasz serwer. Następnie trzeba ją rozpakować używając komendy **unzip**. Przed instalacją należy upewnić się, że nasz system spełnia wymagania określone przez Oracle. Należy doinstalować wymagane paczki na których bazuje system bazodanowy Oracle - **libaio, bc, oraz flex**. Można to wykonać automatycznie korzystając z managera paczek **YUM**, który jest domyślnym systemem zarządzania paczkami w systemie CentOS. Składnia jest następująca: **yum install libaio bc flex**. Oprócz tego, Oracle wymaga by w systemie znajdowała się aktywna partycja swap lub swapfile o minimalnym rozmiarze 2GB. Po wypakowaniu pliku instalujemy paczkę przy pomocy **rpm -i <nazwa paczki>**. Po zakończeniu instalacji, przed pierwszym użyciem należy skonfigurować serwis hostujący bazę Oracle. W tym celu musimy wywołać konfigurator jako root poleceniem: **/etc/init.d/oracle-xe configure**. Konfiguracja prosi o nadanie hasła dla użytkowników **SYS/SYSTEM**, będących administratorami bazy Oracle, określenie portów dla serwera HTTP oraz listenera bazy danych, oraz czy chcemy by serwis załączał się automatycznie przy starcie hosta.

Po wykonaniu tych operacji możemy zalogować się po IP serwera na użytkownika **SYSTEM** w celu dalszej konfiguracji bazy danych, którą można wykonać przy pomocy zapytań SQL. W pierwszej kolejności utworzono użytkownika do którego będą należeć encje projektu. Oprócz tego stworzono użytkownika i nadano odpowiednie uprawnienia dla każdego członka zespołu projektowego. Dzięki zastosowaniu synonimów/aliasów publicznych - **CREATE PUBLIC SYNONYM...** encje oraz funkcje projektu są dostępne dla każdego członka zespołu bez konieczności adresowania z uwzględnieniem właściciela.

Takie rozwiązanie umożliwia równoległą pracę nad jedną bazą danych dla wielu użytkowników fizycznych, w dowolnym miejscu i czasie.

- **Implementacja obiektów**

W pierwszej kolejności utworzono wymagane encje, bez modelowania relacji, używając polecenia: **CREATE TABLE**

```
CREATE TABLE RESERVATIONS(
  RESERVATION_ID int NOT NULL PRIMARY KEY,
  DATE_BEGIN date NOT NULL,
  DATE_END date NOT NULL,
  APARTMENT_ID int,
  USER_ID int,
  STATUS int
);

CREATE TABLE PAYMENTS(
  PAYMENT_ID int NOT NULL PRIMARY KEY,
  RESERVATION_ID int,
  TYPE VARCHAR2(16) CHECK(TYPE IN ('advance', 'final')),
  VALUE int,
  STATUS VARCHAR2(16) CHECK(STATUS IN ('started', 'completed',
    'canceled', 'refunded'))
);

CREATE TABLE LOCATIONS(
  LOCATION_ID int NOT NULL PRIMARY KEY,
  COUNTRY_NAME VARCHAR2(128),
  CITY VARCHAR2(128),
  LONGITUDE VARCHAR2(64),
  LATITUDE VARCHAR2(64),
  STREET VARCHAR2(128),
  STREET2 VARCHAR2(128),
  APARTMENT_NUMBER int,
  ZIP_CODE int
);

CREATE TABLE AGENCIES(
  AGENCY_ID INT NOT NULL PRIMARY KEY,
  LOCATION_ID INT,
  NAME VARCHAR2(64),
  INFO VARCHAR2(2048),
  PHONE VARCHAR2(32),
  WEB VARCHAR2(64)
);

CREATE TABLE USERS(
  USER_ID INT NOT NULL PRIMARY KEY,
  TYPE VARCHAR2(6) CHECK(TYPE IN('owner','agency','client')) NOT NULL,
  EMAIL VARCHAR2(128),
  PASSWORD VARCHAR(128) NOT NULL,
  INFO VARCHAR2(2048),
  PHONE VARCHAR2(32),
  NAME VARCHAR2(64),
  LAST_NAME VARCHAR2(64),
  LOCATION_ID INT
);
```

```
51
52 CREATE TABLE APARTMENTS(
53     APARTMENT_ID int NOT NULL PRIMARY KEY,
54     COST_PER_NIGHT int,
55     BED_COUNT int,
56     LOCATION_ID int NOT NULL,
57     AGENCY_ID int,
58     AIR_COND number CHECK(AIR_COND IN (1,0)),
59     STANDARD number CHECK(STANDARD IN (1,2,3,4,5)),
60     OWNER_ID int
61 );
```

Następnie utworzono relację pomiędzy nowo utworzonymi encjami, używając składni:
ALTER TABLE <TABELA> ADD CONSTRAINT <NAZWA> FOREIGN KEY...

```
1  ALTER TABLE RESERVATIONS
2  ADD CONSTRAINT FK_APARTMENTS
3  FOREIGN KEY (APARTMENT_ID)
4    REFERENCES APARTMENTS(APARTMENT_ID);
5
6  ALTER TABLE PAYMENTS
7  ADD CONSTRAINT FK_RESERVATIONS
8  FOREIGN KEY (RESERVATION_ID)
9    REFERENCES RESERVATIONS(RESERVATION_ID);
10
11 ALTER TABLE RESERVATIONS
12 ADD CONSTRAINT FK_USERS
13 FOREIGN KEY (USER_ID)
14   REFERENCES USERS(USER_ID);
15
16 ALTER TABLE APARTMENTS
17 ADD CONSTRAINT FK_AGENCIES
18 FOREIGN KEY (AGENCY_ID)
19   REFERENCES AGENCIES(AGENCY_ID);
20
21 ALTER TABLE APARTMENTS
22 ADD CONSTRAINT FK_LOCATIONS
23 FOREIGN KEY (LOCATION_ID)
24   REFERENCES LOCATIONS(LOCATION_ID);
25
26 ALTER TABLE AGENCIES
27 ADD CONSTRAINT FK_LOCATIONS_AGENCIES
28 FOREIGN KEY (LOCATION_ID)
29   REFERENCES LOCATIONS(LOCATION_ID);
30
31 ALTER TABLE APARTMENTS
32 ADD CONSTRAINT FK_USERS_APARTMENTS
33 FOREIGN KEY (OWNER_ID)
34   REFERENCES USERS(USER_ID);
35
36 ALTER TABLE USERS
37 ADD CONSTRAINT FK_LOCATIONS_USERS
```

```
38 FOREIGN KEY (LOCATION_ID)
39   REFERENCES LOCATIONS(LOCATION_ID);
```

- **Wprowadzenie danych**

Do wygenerowania przykładowych danych użyto narzędzia **mockaroo.com** pozwalającego na tworzenie zbiorów z danymi na potrzeby testów. Określamy strukturę tabeli, typy danych oraz jakie dane nas interesują. Narzędzie umożliwia utworzenie gotowych poleceń **INSERT**.

Dla tabeli **RESERVATIONS** utworzono 100 obiektów, przykład:

```
1 insert into RESERVATIONS
2     (RESERVATION_ID, DATE_BEGIN, DATE_END, APARTMENT_ID, USER_ID, STATUS)
3     values (1, '2018-08-11', '2018-08-18', 1, 653, 3);
4 insert into RESERVATIONS
5     (RESERVATION_ID, DATE_BEGIN, DATE_END, APARTMENT_ID, USER_ID, STATUS)
6      values (2, '2018-07-20', '2018-07-27', 2, 320, 1);
7 insert into RESERVATIONS
8     (RESERVATION_ID, DATE_BEGIN, DATE_END, APARTMENT_ID, USER_ID, STATUS)
9      values (3, '2018-07-05', '2018-07-12', 3, 253, 2);
```

Dla tabeli **PAYMENTS** utworzono 100 obiektów, przykład:

```
1 insert into PAYMENTS
2     (PAYMENT_ID, RESERVATION_ID, TYPE, VALUE, STATUS)
3     values (1, 1, 'final', 314084, 'started');
4 insert into PAYMENTS
5     (PAYMENT_ID, RESERVATION_ID, TYPE, VALUE, STATUS)
6     values (2, 2, 'advance', 459124, 'canceled');
7 insert into PAYMENTS
8     (PAYMENT_ID, RESERVATION_ID, TYPE, VALUE, STATUS)
9     values (3, 3, 'advance', 163346, 'started');
```

Dla tabeli **LOCATIONS** utworzono 1000 obiektów, przykład:

```
1 insert into LOCATIONS (LOCATION_ID, COUNTRY_NAME, CITY, LONGITUDE,
2     LATITUDE, STREET, STREET2, APARTMENT_NUMBER, ZIP_CODE)
3     values (1, 'Russia', 'Novodvinsk', 40.8058276, 64.3443188,
4     '13573␣Loeprich␣Avenue', '7', '5483', '164903');
5 insert into LOCATIONS (LOCATION_ID, COUNTRY_NAME, CITY, LONGITUDE,
6     LATITUDE, STREET, STREET2, APARTMENT_NUMBER, ZIP_CODE)
7     values (2, 'Colombia', 'Argelia', -76.121514, 4.727773,
8     '10839␣Claremont␣Circle', '8', '0302', '761517');
```

Dla tabeli **AGENCIES** utworzono 50 obiektów, przykład:

```
1 insert into AGENCIES
2     (AGENCY_ID, LOCATION_ID, NAME, INFO, PHONE, WEB)
3     values (1, 1, 'Jamia', 'luctus␣et␣ultrices␣posuere␣cubilia␣curae
4 ␣␣␣␣duis␣faucibus␣accumsan␣odio', '410-258-5854', 'www.zzxuswefx.com');
5 insert into AGENCIES (AGENCY_ID, LOCATION_ID, NAME, INFO, PHONE, WEB)
6     values (2, 2, 'Meedoo', 'ante␣vel␣ipsum␣praesent␣blandit␣lacinia
```

```
    erat vestibulum sed magna at nunc commodo placerat praesent blandit
    nam nulla integer pede justo lacinia eget tincidunt eget tempus vel
    pede morbi', '773-399-9081', 'www.rtcnakspl.com');
```

Dla tabeli **USERS** utworzono 548 obiektów, przykład:

```
insert into USERS
    (USER_ID, TYPE, EMAIL, PASSWORD, INFO, PHONE, NAME,
    LAST_NAME, LOCATION_ID)
    values (201, 'client', 'mgealle0@dedecms.com',
    '1d2351adc06cfeef9a29f0989eae5e130adda76f',
    'odio cras mi pede malesuada in imperdiet et commodo
    vulputate justo in blandit ultrices enim lorem ipsum
    dolor sit amet consectetuer adipiscing elit proin interdum',
    '379-642-0501', 'Marylou', 'Gealle', 453);
```

Dla tabeli **APARTMENTS** utworzono 200 obiektów, przykład:

```
insert into APARTMENTS (APARTMENT_ID, COST_PER_NIGHT, BED_COUNT,
    LOCATION_ID, AGENCY_ID, AIR_COND, STANDARD, OWNER_ID)
    values (1, 37118, 10, 51, 11, 1, 5, 252);
insert into APARTMENTS (APARTMENT_ID, COST_PER_NIGHT, BED_COUNT,
    LOCATION_ID, AGENCY_ID, AIR_COND, STANDARD, OWNER_ID)
    values (2, 2376, 6, 52, 39, 0, 3, 253);
insert into APARTMENTS (APARTMENT_ID, COST_PER_NIGHT, BED_COUNT,
    LOCATION_ID, AGENCY_ID, AIR_COND, STANDARD, OWNER_ID)
    values (3, 34275, 7, 53, 31, 1, 2, 254);
```

- **Zdefiniowanie typowych operacji SQL**

Utworzono przykładowe funkcje w języku PL/SQL potrzebne do obsługi przedstawionego problemu. W celu systematyzacji zapytania podzielono na typowe Usercase'y. Dla pierwszej funkcji (get_apartments) przedstawiono również funkcję w PL/SQL drukującą wynik funkcji. Pod każdą funkcją znajduje się także przykład czystego zapytania SQL, który byłby użyty np. w backendzie aplikacji, po podmianie przykładowych wartości na zmienne.

```
/*
Usercase: user-client chce wyszukac apartament wedug parametrow:
    kraj (obowiazkowe),
    cena za noc,
    ilosc lozek (obowiazkowe),
    klimatyzacja,
    standard
*/
CREATE OR REPLACE FUNCTION get_apartments (
    v_country_name      IN VARCHAR2,
    v_bed_count         IN INT,
    v_cost_per_night   IN INT DEFAULT 1000000,
    v_air_cond        IN INT DEFAULT 0,
    v_standard        IN INT DEFAULT 1
) RETURN SYS_REFCURSOR AS
    my_cursor   SYS_REFCURSOR;
```

```
17 BEGIN
18     OPEN my_cursor FOR SELECT
19             apartment_id
20         FROM apartments
21         JOIN locations USING ( location_id )
22         WHERE
23             country_name = v_country_name
24             AND   bed_count = v_bed_count
25             AND   cost_per_night <= v_cost_per_night
26             AND   air_cond >= v_air_cond
27             AND   standard >= v_standard
28     ORDER BY cost_per_night;
29
30     RETURN my_cursor;
31 END;
32
33 --przyklad drukujacy wynik funkcji get_apartments:
34 DECLARE
35   res int;
36   x SYS_REFCURSOR;
37 BEGIN
38     x := GET_APARTMENTS('Indonesia', 7, 100000);
39     LOOP
40       FETCH x INTO res;
41       EXIT WHEN x%NOTFOUND;
42       dbms_output.put_line(res);
43     END LOOP;
44 end;
45
46 --przykladowe
47 SELECT
48     apartment_id
49 FROM
50     apartments
51     JOIN locations USING ( location_id )
52 WHERE
53     country_name = 'Indonesia'
54     AND   bed_count = 7
55     AND   cost_per_night <= 50000
56     AND   air_cond >= 0
57     AND   standard >= 3;
58
59
60 /*
61 Usercase: user-client chce zarezerwowac apartament:
62     apartment_id (obowiazkowe),
63     data_poczatek (obowiazkowe),
64     data_koniec (obowiazkowe)
65 */
66 CREATE OR REPLACE FUNCTION reserve_apartment (
67     v_apartment_id   IN INT,
```

```
68      v_date_begin      IN DATE,
69      v_date_end        IN DATE,
70      v_user_id         IN INT
71  ) RETURN INT AS
72      id    INT := v_apartment_id;
73  BEGIN
74      INSERT INTO reservations VALUES (
75          (
76              SELECT
77                  MAX(r.reservation_id)
78              FROM
79                  reservations r
80          ) + 1,
81          v_date_begin,
82          v_date_end,
83          v_apartment_id,
84          v_user_id,
85          1
86      );
87
88      RETURN id;
89  END;
90
91  --przykladowe
92  INSERT INTO reservations VALUES (
93      (
94          SELECT
95              MAX(r.reservation_id)
96          FROM
97              reservations r
98      ) + 1,
99      '2018-08-10',
100     '2018-08-17',
101     50,
102     400,
103     1
104 );
105
106 /*
107 Usercase: user-owner chce wyswietlic rezerwacje jego apartamentu:
108     apartment_id (obowiazkowe)
109 */
110 CREATE OR REPLACE FUNCTION get_reservations (
111     v_apartment_id IN INT
112 ) RETURN SYS_REFCURSOR AS
113     my_cursor    SYS_REFCURSOR;
114 BEGIN
115     OPEN my_cursor FOR SELECT
116             reservation_id, status
117         FROM reservations
118         WHERE
```

```
119             apartment_id = v_apartment_id;
120
121     RETURN my_cursor;
122 END;
123
124 --przykladowe
125 select reservation_id, status
126 from reservations
127 where apartment_id = 50;
128
129 /*
130 Usercase: user-owner chce zmienic status wybranej rezerwacji:
131     reservation_id (wymagane),
132     nowy status (wymagane)
133  */
134 CREATE FUNCTION RESERVATION_UPDATE (reservation IN int, status IN int)
135 RETURN int
136 IS id int := reservation;
137 BEGIN
138   UPDATE RESERVATIONS
139   SET STATUS = status
140   WHERE RESERVATION_ID = reservation;
141 RETURN id;
142 end;
143
144 --przykladowe
145 UPDATE RESERVATIONS
146 SET STATUS = 1
147 WHERE RESERVATION_ID = 1;
148
149 /*
150 Usercase: system otrzymal potwierdzenie platnosci:
151     payment_id (wymagane)
152     nowy status (wymagane)
153  */
154 CREATE FUNCTION PAYMENT_COMPLETED (payment IN int, status IN varchar2)
155 RETURN int
156 IS id int := payment;
157 BEGIN
158   UPDATE PAYMENTS
159   SET STATUS = status
160   WHERE PAYMENT_ID = payment;
161 RETURN id;
162 end;
163
164 --przykladowe
165 UPDATE PAYMENTS
166 SET STATUS = 'started'
167 WHERE PAYMENT_ID = 5;
168
169 /*
```

```
170 Usercase: user-client chce anulowac rezerwacje
171     reservation_id (wymagane)
172  */
173 CREATE FUNCTION RESERVATION_CANCEL (reservation IN int)
174 RETURN int
175 IS refund_amount int := 0;
176   payment_type varchar2(16);
177 BEGIN
178   UPDATE RESERVATIONS
179   SET STATUS = 3
180   WHERE RESERVATION_ID = 1;
181   COMMIT;
182
183   SELECT 'TYPE' into payment_type
184   FROM PAYMENTS
185   WHERE RESERVATION_ID = reservation;
186
187   IF payment_type = 'advance' THEN
188     RETURN refund_amount;
189   ELSE
190     SELECT VALUE*0.8 into refund_amount
191     FROM PAYMENTS
192     WHERE RESERVATION_ID = reservation;
193     UPDATE PAYMENTS
194     SET STATUS = 'refunded'
195     WHERE RESERVATION_ID = reservation;
196     COMMIT;
197   end if;
198
199 RETURN refund_amount;
200 end;
201
202 --przykladowe
203 UPDATE RESERVATIONS
204 SET STATUS = 3
205 WHERE RESERVATION_ID = 1;
206
207 SELECT 'TYPE'
208 FROM PAYMENTS
209 WHERE RESERVATION_ID = 1;
210
211 SELECT VALUE*0.8
212 FROM PAYMENTS
213 WHERE RESERVATION_ID = 1;
214
215 UPDATE PAYMENTS
216 SET STATUS = 'refunded'
217 WHERE RESERVATION_ID = 1;
218
219 /*
220 Usercase: user-client chce wyswietlic swoje rezerwacje
```

```
221     user_id clienta (wymagane)
222  */
223 CREATE FUNCTION GET_USER_RESERVATIONS (client_id IN int)
224 RETURN SYS_REFCURSOR
225 IS my_cursor SYS_REFCURSOR;
226 BEGIN
227     OPEN my_cursor FOR SELECT RESERVATION_ID, STATUS
228     FROM RESERVATIONS
229     WHERE USER_ID = client_id;
230 RETURN my_cursor;
231 END;
232
233 --przykladowe
234 SELECT *
235 FROM RESERVATIONS
236 WHERE USER_ID = 499;
237
238 /*
239 Usercase: user-client chce sprawdzic dostepno?c apartamentu:
240     apartament_id (obowiszkowe)
241 */
242 CREATE OR REPLACE FUNCTION apartment_busy (
243     v_apartment_id IN INT
244 ) RETURN SYS_REFCURSOR AS
245     my_cursor   SYS_REFCURSOR;
246 BEGIN
247     OPEN my_cursor FOR SELECT
248             date_begin,
249             date_end
250         FROM reservations
251         WHERE
252             apartment_id = v_apartment_id
253     AND STATUS != 3;
254
255     RETURN my_cursor;
256 END;
257
258 --przykladowe
259 SELECT
260     date_begin,
261     date_end
262 FROM
263     reservations
264 WHERE
265     apartment_id = 1
266 AND status != 3;
267
268 /*
269 Usercase: user-owner chce dodac apartament:
270         cost_per_night (obowiazkowe)
271         bed_count (obowiazkowe)
```

```
272         location_id (obowiazkowe)
273         agency_id (obowiazkowe)
274         air_cond (obowiazkowe)
275         standard (obowiazkowe)
276         owner_id (obowiazkowe)
277 */
278 CREATE OR REPLACE FUNCTION insert_apartment (
279     v_cost_per_night   IN INT,
280     v_bed_count        IN INT,
281     v_location_id      IN INT,
282     v_agency_id        IN INT,
283     v_air_cond         IN INT,
284     v_standard         IN INT,
285     v_owner_id         IN INT
286 ) RETURN INT IS
287     ret   INT := 0;
288 BEGIN
289     INSERT INTO apartments (
290         apartment_id,
291         cost_per_night,
292         bed_count,
293         location_id,
294         agency_id,
295         air_cond,
296         standard,
297         owner_id
298     ) VALUES (
299         (
300             SELECT
301                 MAX(a.apartment_id)
302             FROM
303                 apartments a
304         ) + 1,
305         v_cost_per_night,
306         v_bed_count,
307         v_location_id,
308         v_agency_id,
309         v_air_cond,
310         v_standard,
311         v_owner_id
312     );
313
314     RETURN ret;
315 END;
316
317 --przykladowe
318 INSERT INTO apartments (
319     apartment_id,
320     cost_per_night,
321     bed_count,
322     location_id,
```

```
323     agency_id,
324     air_cond,
325     standard,
326     owner_id
327 ) VALUES (
328     (
329         SELECT
330             MAX(a.apartment_id)
331         FROM
332             apartments a
333     ) + 1,
334     50000,
335     2,
336     66,
337     11,
338     1,
339     2,
340     400
341 );
```

2.6 Podsumowanie

a) Ocena realizacji tematu

Temat został w pełni zrealizowany, zaimplementowano przykładowe operacje na bazie danych realizujące założenia problemu. Baza danych została oparta o model relacyjny.

Ze względu na brak czasu, aplikacja wykorzystująca stworzoną bazę danych nie została napisana. W zamian za to, do kwerend zostały zrobione funkcje w języku PL/SQL.

b) Wnioski

Projekt został zrealizowany bez większych problemów. Dzięki projektowi udało się przećwiczyć instalację i konfigurację środowiska bazodanowego Oracle na systemie CentOS. Członkowie grupy projektowej przyswoili teorię relacyjnych baz danych, nabyli cennego doświadczenia w tworzeniu i optymalizacji baz danych, generowania zbiorów testowych oraz poznali zaawansowane funkcje języka SQL pozwalające na konfigurację bazy, oraz tworzenie funkcji.