

Wojskowa Akademia Techniczna im. Jarosława Dąbrowskiego

Bazy danych

Prowadzący przedmiot: dr inż. Robert Waszkowski

Sprawozdanie z projektu przeprowadzonego w ramach ćwiczeń laboratoryjnych
"Przychodnia lekarska"

Michał Cywiński

grupa studencka I2X4S1

Wydział Cybernetyki, Wojskowa Akademia Techniczna ul. Kaliskiego 2, 00-908 Warszawa
Warszawa 2014

STRESZCZENIE: W sprawozdaniu umieszczono zadanie laboratoryjne wraz z jego wykonaniem oraz treścią. Dołączono dodatkowe komentarze i zastrzeżenia dotyczące założeń projektu.

SŁOWA KLUCZOWE: SQL, baza, danych, tabele, procedura składowana, widok, widok zmaterializowany, trigger, check constraint, indeks

Spis treści

1. Treść zadania.....	3
2. Opis tematu	4
3. Implementacja.....	5
3.1. Skrypt główny.....	6
3.2. Tworzenie tabel	7
3.3. Tworzenie indeksów	8
3.4. Tworzenie relacji	9
3.5. Tworzenie procedur składowanych	10
3.6. Tworzenie widoków	15
3.7. Tworzenie widoku zmaterializowanego	16
3.8. Tworzenie triggerów.....	17
3.9. Tworzenie funkcji.....	17
4. Podsumowanie i wnioski	19

1. Treść zadania

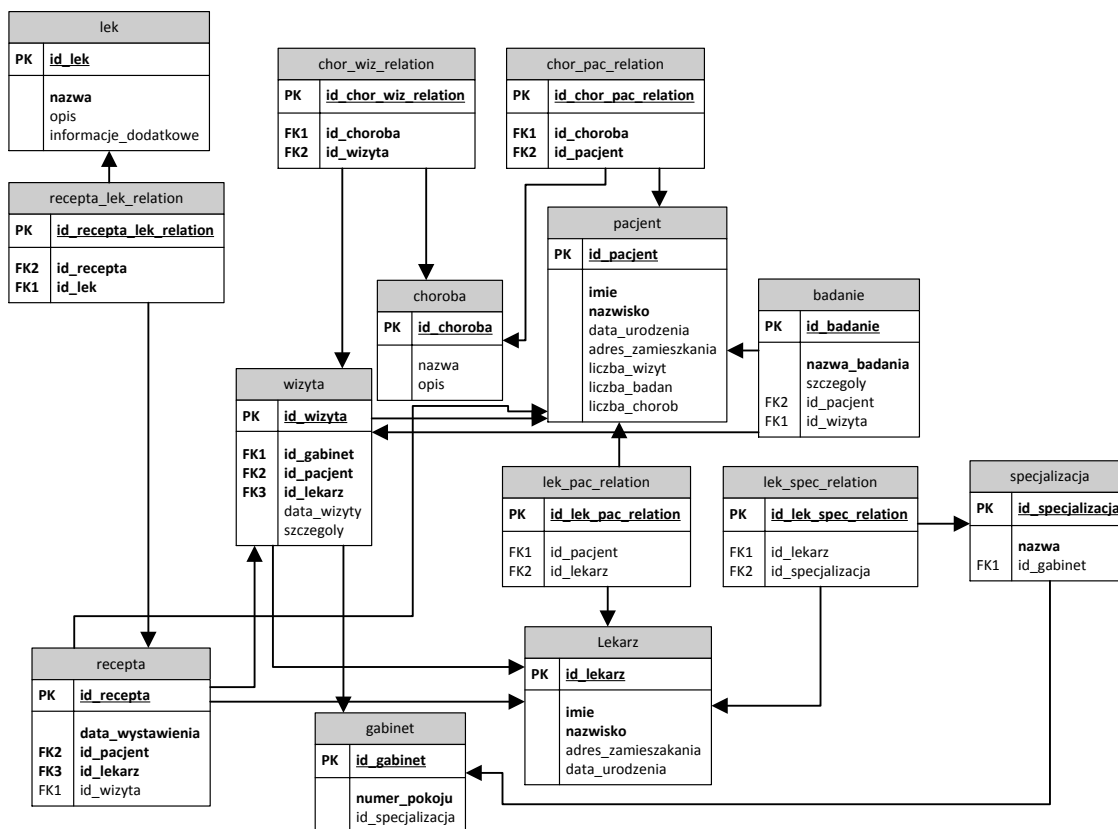
Zadaniem laboratoryjnym było opracowanie koncepcji bazy danych na ćwiczeniach na wybrany przez siebie temat, a następnie zaimplementowanie tej bazy danych (generacja tabel oraz zależności między nimi) w wybranym przez siebie DBMS na podstawie wiedzy i umiejętności wyniesionych z zajęć laboratoryjnych z założeniem wykorzystania następujących elementów:

- 5-7 tabel
- Klucze główne, klucze obce
- Widoki
- Widok zmaterializowany
- Użycie procedur składowanych
- Użycie funkcji
- Użycie triggerów
- Użycie indeksów
- Użycie Check Constraints

Jako temat wybrałem opis przychodni lekarskiej z naciskiem na różnego rodzaju akcje dotyczące obsługi pacjenta.

2. Opis tematu

Na poniższym rysunku zawarłem schemat fizycznego modelu bazy danych opartej o wybrany wcześniej temat:



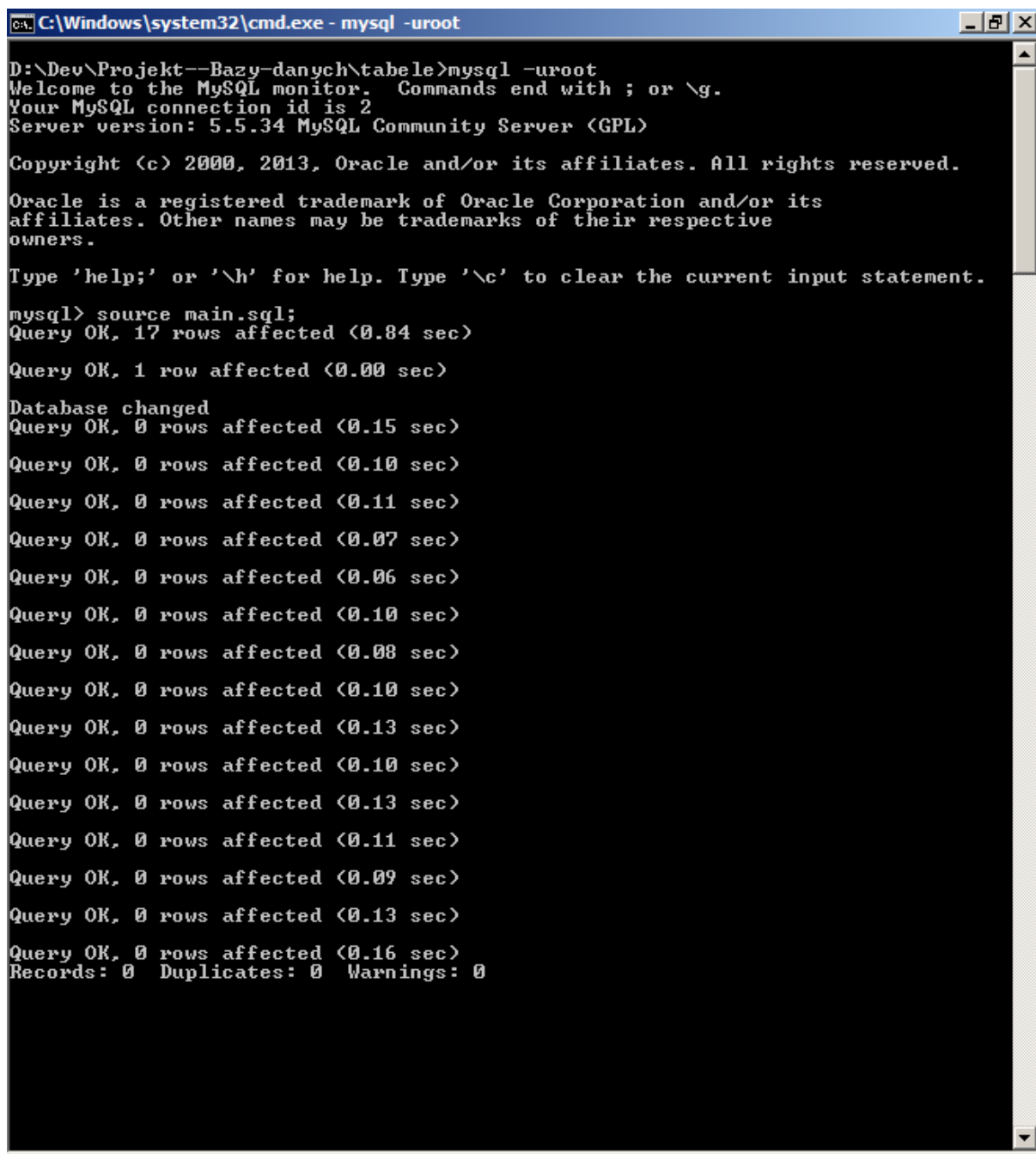
Rysunek 1: Schemat bazy danych

Baza danych składa się ze spisu lekarzy, pacjentów, gabinetów, leków, chorób, przeprowadzonych badań, recept oraz specjalizacji lekarzy. Tabele z końcówką _relation służą do obsługi relacji "wiele do wielu" - na przykład obsługi pozycji w receptce lub przypisania wielu specjalizacji lekarzom.

3. Implementacja

Jako DBMS wybrałem MySQL ze względu na dostępność, szybkość instalacji, łatwość użycia i rozbudowaną społeczność oraz kompatybilność z obecnie zdobywającą coraz większą popularność bazą MariaDB.

Poszczególne elementy bazy danych generuję z pojedynczych plików *.sql, które są uprzednio wywoływane z pliku main.sql za pomocą komendy MySQL "source". Plik main.sql wywołuję jako użytkownik root z poziomu CLI MySQL.



```
C:\Windows\system32\cmd.exe - mysql -uroot
D:\Dev\Projekt--Bazy-danych\tabele>mysql -uroot
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.5.34 MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> source main.sql;
Query OK, 17 rows affected (0.84 sec)

Query OK, 1 row affected (0.00 sec)

Database changed
Query OK, 0 rows affected (0.15 sec)

Query OK, 0 rows affected (0.10 sec)

Query OK, 0 rows affected (0.11 sec)

Query OK, 0 rows affected (0.07 sec)

Query OK, 0 rows affected (0.06 sec)

Query OK, 0 rows affected (0.10 sec)

Query OK, 0 rows affected (0.08 sec)

Query OK, 0 rows affected (0.10 sec)

Query OK, 0 rows affected (0.13 sec)

Query OK, 0 rows affected (0.10 sec)

Query OK, 0 rows affected (0.13 sec)

Query OK, 0 rows affected (0.11 sec)

Query OK, 0 rows affected (0.09 sec)

Query OK, 0 rows affected (0.13 sec)

Query OK, 0 rows affected (0.16 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Rysunek 2: Działanie skryptu main.sql

W pierwszej kolejności skrypt tworzy wszystkie tabele i odpowiednie dla nich klucze główne oraz indeksy i relacje.

Aby ułatwić przyszłe pisanie aplikacji korzystających z bazy danych stworzyłem zestaw procedur składowanych, które wykonują operację INSERT do poszczególnych tabeli bazy danych.

Następnie skrypt główny wywołuje tworzenie trzech widoków, w tym widoku zmaterializowanego. Dwa pierwsze widoki pokazują lekarzy i ich listę kwalifikacji oraz historię wizyt pacjentów. Trzeci widok pokazuje szczegóły dotyczące danej recepty (należy podać odpowiednie ID recepty).

Przedostatnim z tworzonych elementów są triggery. Inkrementują one odpowiednio liczbę odbytych badań, przebytych chorób i zliczają wizyty w przychodni lekarskiej - wszystkie informacje są aktualizowane w odpowiednich kolumnach tabeli pacjentów.

Ostatnim elementem jest stworzenie funkcji zliczającej średnią chorób przebytych przez wszystkich pacjentów.

3.1. Skrypt główny

```
DROP DATABASE IF EXISTS przychodnia_lekarska;
CREATE DATABASE przychodnia_lekarska;
use przychodnia_lekarska;
source badanie.sql;
source chor_pac_relation.sql;
source chor_wiz_relation.sql;
source choroba.sql;
source gabinet.sql;
source lek.sql;
source lek_pac_relation.sql;
source lek_spec_relation.sql;
source lekarz.sql;
source pacjent.sql;
source recepta.sql;
source recepta_lek_relation.sql;
source specjalizacja.sql;
source wizyta.sql;
source create_indexes.sql;
source klucze_obce.sql;
source insert_choroba.sql;
source insert_pacjent.sql;
source insert_lekarz.sql;
source insert_gabinet.sql;
source insert_specjalizacja.sql;
source insert_lek.sql;
source insert_badanie.sql;
source insert_wizyta.sql;
source insert_recepta.sql;
source insert_pozycja_recepty.sql;
```

```
source insert_lekarz_pacjent_relation.sql;
source insert_lekarz_specjalizacja_relation.sql;
source insert_zdiagnozowana_choroba.sql;
source create_views.sql;
source create_triggers.sql;
source func_srednia_chorob.sql;
```

3.2. Tworzenie tabel

```
CREATE TABLE lek_spec_relation(
    id_lek_spec_relation int primary key not null auto_increment,
    id_lekarz int not null,
    id_specjalizacja int not null
);
CREATE TABLE lekarz(
    id_lekarz int primary key not null auto_increment,
    imie varchar(255),
    nazwisko varchar(255),
    adres_zamieszkania text,
    data_urodzenia datetime,
    CONSTRAINT chc_birth_date CHECK (data_urodzenia < NOW())
);
CREATE TABLE lek(
    id_lek int primary key not null auto_increment,
    nazwa varchar(255) not null,
    opis text,
    informacje_dodatkowe text
);
CREATE TABLE recepta_lek_relation(
    id_recepta_lek_relation int primary key not null auto_increment,
    id_recepta int not null,
    id_lek int not null
);
CREATE TABLE specjalizacja(
    id_specjalizacja int primary key not null auto_increment,
    nazwa varchar(255),
    id_gabinet int
);
CREATE TABLE lek_pac_relation(
    id_lek_pac_relation int primary key not null auto_increment,
    id_pacjent int not null,
    id_lekarz int not null
);
CREATE TABLE chor_wiz_relation(
    id_chor_wiz_relation int primary key not null auto_increment,
    id_choroba int not null,
    id_wizyta int not null
);
CREATE TABLE chor_pac_relation(
    id_chor_pac_relation int primary key not null auto_increment,
    id_choroba int not null,
    id_pacjent int not null
```

```

);
CREATE TABLE pacjent(
    id_pacjent int not null auto_increment primary key,
    imie varchar(255) not null,
    nazwisko varchar(255) not null,
    data_urodzenia datetime,
    adres_zamieszkania text,
    liczba_wizyt int default 0,
    liczba_badan int default 0,
    liczba_chorob int default 0,
    CONSTRAINT chk_counters CHECK (liczba_wizyt > -1 AND liczba_badan > -1 AND
liczba_chorob > -1),
    CONSTRAINT chc_birth_date CHECK (data_urodzenia < NOW())
);
CREATE TABLE gabinet(
    id_gabinet int primary key not null auto_increment,
    nazwa varchar(255)
);
CREATE TABLE recepta(
    id_recepta int primary key not null auto_increment,
    data_wystawienia datetime,
    id_pacjent int not null,
    id_lekarz int not null,
    id_wizyta int
);
CREATE TABLE badanie(
    id_badanie int primary key not null auto_increment,
    nazwa_badania varchar(255),
    szczegoly text,
    id_pacjent int,
    id_wizyta int
);
CREATE TABLE choroba(
    id_choroba int primary key not null auto_increment,
    nazwa varchar(255),
    opis text
);
CREATE TABLE wizyta(
    id_wizyta int primary key not null auto_increment,
    id_gabinet int not null,
    id_pacjent int not null,
    id_lekarz int not null,
    data_wizyty datetime,
    szczegoly text
);

```

3.3. Tworzenie indeksów

```

CREATE INDEX pacjent_imienazwisko
ON pacjent (imie, nazwisko);

```

```

CREATE INDEX lekarz_imienazwisko

```



```

ON lekarz (imie, nazwisko);

CREATE INDEX specjalizacja_nazwa
ON specjalizacja (nazwa);

CREATE INDEX choroba_nazwa
ON choroba (nazwa);

CREATE INDEX lek_nazwa
ON lek(nazwa);

```

3.4. Tworzenie relacji

```

ALTER TABLE recepta_lek_relation
    add foreign key (id_recepta) references recepta(id_recepta) on delete
    cascade;
ALTER TABLE recepta_lek_relation
    add foreign key (id_lek) references lek(id_lek) on delete cascade;
ALTER TABLE chor_wiz_relation
    add foreign key (id_choroba) references choroba(id_choroba) on delete
    cascade;
ALTER TABLE chor_wiz_relation
    add foreign key (id_wizyta) references wizyta(id_wizyta) on delete cascade;
ALTER TABLE chor_pac_relation
    add foreign key (id_choroba) references choroba(id_choroba) on delete
    cascade;
ALTER TABLE chor_pac_relation
    add foreign key (id_pacjent) references pacjent(id_pacjent) on delete
    cascade;
ALTER TABLE lek_pac_relation
    add foreign key (id_pacjent) references pacjent(id_pacjent) on delete
    cascade;
ALTER TABLE lek_pac_relation
    add foreign key (id_lekarz) references lekarz(id_lekarz) on delete cascade;
ALTER TABLE lek_spec_relation
    add foreign key (id_specjalizacja) references
    specjalizacja(id_specjalizacja) on delete cascade;
ALTER TABLE lek_spec_relation
    add foreign key (id_lekarz) references lekarz(id_lekarz) on delete cascade;
ALTER TABLE recepta
    add foreign key (id_pacjent) references pacjent(id_pacjent) on delete
    cascade;
ALTER TABLE recepta
    add foreign key (id_lekarz) references lekarz(id_lekarz) on delete cascade;
ALTER TABLE recepta
    add foreign key (id_wizyta) references wizyta(id_wizyta) on delete cascade;
ALTER TABLE wizyta
    add foreign key (id_gabinet) references gabinet(id_gabinet) on delete
    cascade;
ALTER TABLE wizyta
    add foreign key (id_pacjent) references pacjent(id_pacjent) on delete
    cascade;

```

```

ALTER TABLE wizyta
    add foreign key (id_lekarz) references lekarz(id_lekarz) on delete cascade;
ALTER TABLE badanie
    add foreign key (id_pacjent) references pacjent(id_pacjent) on delete
    cascade;
ALTER TABLE badanie
    add foreign key (id_wizyta) references wizyta(id_wizyta) on delete cascade;
ALTER TABLE specjalizacja
    add foreign key (id_gabinet) references gabinet(id_gabinet) on delete
    cascade;

```

3.5. Tworzenie procedur składowanych

```

delimiter //
CREATE PROCEDURE insert_specjalizacja_lekarza(
    IN in_id_lekarz int,
    IN in_id_specjalizacja int
)
BEGIN
    INSERT INTO lek_spec_relation(
        id_lekarz,
        id_specjalizacja
    )
    VALUES(
        in_id_lekarz,
        in_id_specjalizacja
    );
END
//
delimiter ;
delimiter //
CREATE PROCEDURE insert_gabinet(
    IN in_numer_pokoju varchar(255)
)
BEGIN

    INSERT INTO gabinet(
        nazwa
    )
    VALUES(
        in_numer_pokoju
    );

END
//
delimiter ;
delimiter //
CREATE PROCEDURE insert_recepta(
    IN in_data_wystawienia datetime,
    IN in_id_pacjent int,
    IN in_id_lekarz int,
    IN in_id_wizyta int

```

```

)

BEGIN
    INSERT INTO recepta(
        data_wystawienia,
        id_pacjent,
        id_lekarz,
        id_wizyta
    )
    VALUES(
        in_data_wystawienia,
        in_id_pacjent,
        in_id_lekarz,
        in_id_wizyta
    );
END
//
delimiter ;
delimiter //
CREATE PROCEDURE insert_specjalizacja(
    IN in_nazwa varchar(255),
    IN id_gabinet int
)

BEGIN

INSERT INTO specjalizacja(
    nazwa,
    id_gabinet)
VALUES(
    in_nazwa,
    in_gabinet
);

END
//
delimiter ; delimiter //
CREATE PROCEDURE insert_pozycja_recepty(
    IN in_id_recepta int,
    IN in_id_lek int
)

BEGIN

    INSERT INTO recepta_lek_relation(
        id_recepta,
        id_lek
    )
    VALUES(
        in_id_recepta,
        in_id_lek
    );

```

```

END
//
delimiter ;
delimiter //
CREATE PROCEDURE insert_choroba(
    IN in_nazwa varchar(255),
    IN in_opis text
)

BEGIN

INSERT INTO choroba(
    nazwa,
    opis)
VALUES(
    in_nazwa,
    in_opis
);

END
//
delimiter ;
delimiter //
CREATE PROCEDURE insert_pacjent(
    IN in_imie varchar(255),
    IN in_nazwisko varchar(255),
    IN in_data_urodzenia datetime,
    IN in_adres_zamieszkania text,
    IN in_liczba_wizyt int,
    IN in_liczba_badan int,
    IN in_liczba_chorob int
)

BEGIN

INSERT INTO pacjent(
    imie,
    nazwisko,
    data_urodzenia,
    adres_zamieszkania,
    liczba_wizyt,
    liczba_badan,
    liczba_chorob)
VALUES(
    in_imie,
    in_nazwisko,
    in_data_urodzenia,
    in_adres_zamieszkania,
    in_liczba_wizyt,
    in_liczba_badan,
    in_liczba_chorob
);

```

```

END
//
delimiter ;
delimiter //
CREATE PROCEDURE insert_badanie(
    IN in_nazwa_badania varchar(255),
    IN in_szczegoly text,
    IN in_id_pacjent int,
    IN in_id_wizyta int
)
BEGIN

INSERT INTO badanie(
    nazwa_badania,
    szczegoly,
    id_pacjent,
    id_wizyta
)
VALUES(
    in_nazwa_badania,
    in_szczegoly,
    in_id_pacjent,
    in_id_wizyta
);

END
//
delimiter ;
delimiter //
CREATE PROCEDURE insert_zdiagnozowana_choroba(
    IN in_id_choroba int,
    IN in_id_wizyta int
)
BEGIN
    INSERT INTO chor_wiz_relation(
        id_choroba,
        id_wizyta
    )
    VALUES(
        in_id_choroba,
        in_id_wizyta
    );
END
//
delimiter ;
delimiter //
CREATE PROCEDURE insert_pacjent_lekarza(
    IN in_id_pacjent int,
    IN in_id_lekarz int
)
BEGIN

```

```

        INSERT INTO lek_pac_relation(
            id_pacjent,
            id_lekarz
        )
        VALUES(
            in_id_pacjent,
            in_id_lekarz
        );
END
//
delimiter ;
delimiter //

CREATE PROCEDURE insert_wizyta(
    IN in_id_gabinet int,
    IN in_id_pacjent int,
    IN in_id_lekarz int,
    IN in_data_wizyty datetime,
    IN in_szczegoly text
)

BEGIN
    INSERT INTO wizyta(
        id_gabinet,
        id_pacjent,
        id_lekarz,
        data_wizyty,
        szczegoly
    )
    VALUES (
        in_id_gabinet,
        in_id_pacjent,
        in_id_lekarz,
        in_data_wizyty,
        in_szczegoly
    );
END
//
delimiter ;
delimiter //

CREATE PROCEDURE insert_lekarz(
    IN in_imie varchar(255),
    IN in_nazwisko varchar(255),
    IN in_data_urodzenia datetime,
    IN in_adres_zamieszkania text
)

BEGIN

INSERT INTO lekarz(
    imie,
    nazwisko,

```

```

        data_urodzenia,
        adres_zamieszkania)
VALUES(
    in_imie,
    in_nazwisko,
    in_data_urodzenia,
    in_adres_zamieszkania
);

END
//
delimiter ;
delimiter //
CREATE PROCEDURE insert_lek(
    IN in_nazwa varchar(255),
    IN in_opis text,
    IN in_informacje_dodatkowe text
)

BEGIN
    INSERT INTO lek(
        nazwa,
        opis,
        informacje_dodatkowe
    )
    VALUES(
        in_nazwa,
        in_opis,
        in_informacje_dodatkowe
    );
END
//
delimiter ;
delimiter //
CREATE PROCEDURE insert_gabinet(
    IN in_numer_pokoju varchar(255)
)
BEGIN

INSERT INTO gabinet(
    nazwa
)
VALUES(
    in_numer_pokoju
);

END
//
delimiter ;

```

3.6. Tworzenie widoków

```

CREATE VIEW view_lista_kwalifikacji AS
SELECT CONCAT(l.imie, " ", l.nazwisko) "Lekarz", s.nazwa "Specjalność", g.nazwa
"Pokój"
FROM specjalizacja s, gabinet g, lekarz l, lek_spec_relation rel
WHERE (rel.id_lekarz = l.id_lekarz) AND (rel.id_specjalizacja =
s.id_specjalizacja) AND (s.id_gabinet = g.id_gabinet)
ORDER BY l.id_lekarz ASC;

```

```

CREATE VIEW view_historia_wizyt AS
SELECT CONCAT(l.imie, " ", l.nazwisko) "Lekarz", w.data_wizyty "Data wizyty",
CONCAT(p.imie, " ", p.nazwisko) "Pacjent"
FROM lekarz l, wizyta w, pacjent p
WHERE (w.id_lekarz = l.id_lekarz) AND (w.id_pacjent = p.id_pacjent)
ORDER BY p.id_pacjent, w.data_wizyty;

```

3.7. Tworzenie widoku zmaterializowanego

```

delimiter //
CREATE PROCEDURE informacje_o_recepcie(
    IN in_id_recepta int
)
BEGIN
    DROP TABLE IF EXISTS widok_w_recepcie;
    CREATE TABLE widok_w_recepcie(
        wystawiajacy varchar(255),
        pacjent varchar(255),
        data_wystawienia datetime,
        liczba_lekow int
    );
    INSERT INTO widok_w_recepcie(wystawiajacy, pacjent, data_wystawienia,
liczba_lekow)
VALUES (
    (SELECT CONCAT(l.imie, ' ', l.nazwisko)
    FROM lekarz l
    WHERE id_lekarz = (SELECT id_lekarz
    FROM recepta
    WHERE id_recepta = in_id_recepta)),
    (SELECT CONCAT(p.imie, ' ', p.nazwisko)
    FROM pacjent p
    WHERE id_pacjent = (SELECT id_pacjent
    FROM recepta
    WHERE id_recepta = in_id_recepta)),
    (SELECT data_wystawienia
    FROM recepta
    WHERE id_recepta = in_id_recepta),
    (SELECT COUNT(*)
    FROM recepta_lek_relation
    WHERE id_recepta = in_id_recepta)
    );
END
//

```



```
delimiter ;
```

3.8. Tworzenie triggerów

```
delimiter //
CREATE TRIGGER trigger_aktualizuj_badanie
AFTER INSERT ON badanie
FOR EACH ROW
BEGIN
    UPDATE pacjent p SET p.liczba_badan = p.liczba_badan + 1 WHERE p.id_pacjent
    = new.id_pacjent;
END
//
delimiter ;

delimiter //

CREATE TRIGGER trigger_aktualizuj_choroby
AFTER INSERT ON chor_pac_relation
FOR EACH ROW
BEGIN
    UPDATE pacjent p SET p.liczba_chorob = p.liczba_chorob + 1 WHERE
    p.id_pacjent = new.id_pacjent;
END

//
delimiter ;

delimiter //
CREATE TRIGGER trigger_aktualizuj_wizyty
AFTER INSERT ON wizyta
FOR EACH ROW
BEGIN
    UPDATE pacjent p SET p.liczba_wizyt = p.liczba_wizyt + 1 WHERE p.id_pacjent
    = new.id_pacjent;
END
//
delimiter ;
```

3.9. Tworzenie funkcji

```
delimiter //

CREATE FUNCTION f_srednia_zachorowalnosc ()
RETURNS INT
BEGIN
    DECLARE avarage INT;
    SET avarage = (SELECT AVG(liczba_chorob) FROM pacjenci);
    RETURN avarage;
END;

//
```

delimiter ;

4. Podsumowanie i wnioski

Zadane elementy bazy danych udało się poprawnie zaimplementować. Jedynym problematycznym elementem okazały się CHECK CONSTRAINTS. MySQL pomyślnie parsuje polecenia zawierające te elementy, natomiast ignoruje ich obecność.

Należy także zaznaczyć, że Packages są elementem typowym dla baz danych Oracle, więc nie było możliwe użyć tej funkcjonalności w MySQL.

Aby polecenia procedur składowanych zostały poprawnie zinterpretowane na etapie ich dodawania, użyłem polecenia "*delimiter*" które definiuje znak kończący polecenie - a więc podmienia średnik odpowiednią kombinacją znaków - w moim wypadku była to kombinacja `//`.

Budowa bazy danych odbyła się na podstawie wcześniej stworzonego schematu w programie Microsoft Visio. Stworzenie schematu wymaga odpowiedniego przemyślenia logiki związanej z reprezentacją danych oraz ewentualnym podłączeniem aplikacji pod bazę danych. Pewne koncepcje mogą również ulec zmianie podczas pisania kodu języka SQL i wiąże się to z powrotem do Visio oraz odpowiednią modyfikacją.

Proces konwersji schematu na rzeczywistą bazę danych za pomocą SQL miałem okazję dokładnie poznać interfejs CLI MySQL oraz narzędzia phpMyAdmin i MySQL Workbench. Była to okazja do wprawienia się w szybkim rozwiązywaniu problemów na podstawie komunikatów błędów oraz świetna praktyka w poszukiwaniu rozwiązań problemów związanych z samą logiką własnych skryptów.