Державний вищий навчальний заклад
„Прикарпатський національний університет імені Василя Стефаника”
Кафедра інформаційних технологій

КУРСОВИЙ ПРОЕКТ

З дисципліни БАЗИ ДАНИХ

на тему: Проектування та розробка бази даних “Медична картотека”

Студента 3 курсу, групи ІПЗ-31
напряму підготовки (спеціальності)
“Інженерія програмного забезпечення”
Дороша Р.В
Керівник старший викладач
Козич О. В.

Національна шкала: _____
Університетська шкала:_____
Оцінка ECTS: _____

Члени комісії: _____ _____
(підпис) (прізвище та ініціали)
_____ _____
(підпис) (прізвище та ініціали)
_____ _____
(підпис) (прізвище та ініціали)

м. Івано-Франківськ – 2019 рік

Державний вищий навчальний заклад

„Прикарпатський національний університет імені Василя Стефаника”

ЗАВДАННЯ НА КУРСОВУ РОБОТУ

_____

(прізвище, ім’я, по батькові студента)

Кафедра _____ Дисципліна _____
Спеціальність _____ Курс ____ Група _____ Семестр _____
1. Тема роботи (проекту) _____
_____
_____
2. Рекомендована література _____
_____
_____
_____
_____
_____
_____
_____
3. Перелік питань, які підлягають розробці _____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
4. Дата видачі завдання _____
Термін подачі до захисту _____
5. Студент _____ Керівник _____

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів підготовки курсової роботи (проекту)	Термін виконання	Форма контролю

РЕФЕРАТ

Пояснювальна записка: 29 сторінок(без додатків), 11 рисунків, 5 джерел, 1 додаток.

Ключові слова:SQL, Бази даних, PostgreSQL, проектування, розробка.

Об`єктом дослідження є ефективне проектування та розробка баз даних.

Мета роботи – створити базу даних на основі медичної картотеки.

Стислий опис тексту пояснювальної записки

У даній курсовій роботі представлено класифікацію баз даних, їх фундаментальні поняття. Також розглянуто функціональність, переваги і недоліки реляційних баз, а саме PostgreSQL. У другій та третій частині описано процес проектування, розробки та тестування створеного продукту.

ABSTRACT

Explanatory note: 29 pages (without appendixes), 11 figures, 5 sources, 1 application.

Keywords: SQL, databases, PostgreSQL, modelling, developing.

The object of the study is an effective database modelling and developing.

The purpose of the work is to create database based on medical card.

Brief description of the text of the explanatory note:

This project presents the databases classification, their fundamental issues. It also discusses about functionality, relational databases advantages and disadvantages in scope of PostgreSQL. The modelling, developing and testing processes are described in the second and third part.

ЗМІСТ

ВСТУП

База даних - це впорядкований набір структурованої інформації, або даних, які зазвичай зберігаються в електронному вигляді в комп'ютерній системі. База даних зазвичай управляється системою управління базами даних (СКБД). Дані разом з СУБД, а також додатки, які з ними пов'язані, називаються системою баз даних, або, для стислості, просто базою даних.

Дані в найбільш поширених типах сучасних баз даних зазвичай формуються у вигляді рядків і стовпців в ряді таблиць, щоб забезпечити ефективність обробки і запитів даних. Потім можна легко отримувати доступ до даних, управляти ними, змінювати, оновлювати, контролювати та організовувати. У більшості баз даних для запису і запитів даних використовується мова структурованих запитів (SQL).

SQL - це мова програмування, що використовується в більшості реляційних баз даних для запиту, обробки і визначення даних, а також контролю доступу. SQL був вперше розроблений в IBM в 1970-х роках, і Oracle виступив в якості основного учасника, що призвело до впровадження стандарту SQL ANSI. SQL дав поштовх випуску численних розширень від таких компаній, як IBM, Oracle і Microsoft. Хоча в даний час SQL все ще широко використовується, почали з'являтися нові мови програмування.

В даному курсовому проекті за предметну область взято базу даних, яка буде зберігати усю необхідну інформацію про медичну картотеку. Щоб користувачі мали змогу одержати для себе більш повну інформацію про свої прийоми та про лікарів необхідно створити базу даних, яка зберігатиме усі необхідні дані.

База даних повинна бути зручною у використанні та задовольняти усім стандартам розробки баз даних. Однією з найважливіших функцій бази даних є швидкість обробки запитів від користувача та швидке видання інформації для нього.

1 ЗАГАЛЬНИЙ ОПИС

У цьому розділі розгядаються фундаментальні поняття реляційних баз даних, принципи їх нормалізації. Також тут детально описується функціональність вибраної технології, її переваги та недоліки.

1.1 Еволюція баз даних

Бази даних значно змінилися з моменту їх появи на початку 1960-х років. Вихідними системами, які використовувалися для зберігання і обробки даних, були навігаційні бази даних - наприклад, ієрархічні бази даних (які спиралися на деревоподібну модель і допускали тільки ставлення «один-до-багатьох») і бази даних з мережевою структурою (більш гнучка модель, допускає множинні відносини). Незважаючи на простоту, ці ранні системи були негнучкими. У 1980-х роках стали популярними реляційні бази даних, в 1990-х роках за ними послідували об'єктно-орієнтовані бази даних. Зовсім недавно внаслідок зростання Інтернету і виникнення необхідності швидшої обробки неструктурованих даних з'явилися бази даних NoSQL. В даний час хмарні бази даних і автономні бази даних відкривають нові можливості щодо способів збору, зберігання, використання даних і управління ними.

1.2 Різниця між базою даних і електронною таблицею

Бази даних та електронні таблиці (зокрема, Microsoft Excel) надають зручні способи зберігання інформації. Основні відмінності між ними полягають в наступному.

- Спосіб зберігання і обробки даних
- Повноваження доступу до даних
- Обсяг зберігання даних

Електронні таблиці спочатку розроблялися для одного користувача, і їх властивості відображають це. Вони відмінно підходять для одного користувача або невеликого числа користувачів, яким не потрібно робити надзвичайно складні операції з даними. З іншого боку, бази даних призначені для зберігання набагато більших наборів впорядкованої інформації – іноді величезних обсягів. Бази даних дозволяють безлічі користувачів в один і той же час швидко і безпечно отримувати доступ до даних і запитувати їх.

1.3 Типи баз даних

Існує безліч різних типів баз даних. Вибір найкращої бази даних для конкретної організації залежить від того, як ця організація має намір використовувати дані.

- **Реляційні бази даних**. Реляційні бази даних стали переважати в 1980-х роках. Елементи в реляційній базі даних організовані у вигляді набору таблиць за допомогою стовпців і рядків. Технологія реляційних баз даних забезпечує найбільш ефективний і гнучкий спосіб доступу до інформації.
- **Об'єктно-орієнтовані бази даних**. Інформація в об'єктно-орієнтованої баз даних представлена у формі об'єкта, як в об'єктно-орієнтованому програмуванні.
- **Розподілені бази даних**. Розподілена база даних складається з двох або більше файлів, розташованих на різних вузлах. Така база даних може зберігатися на декількох комп'ютерах, розташованих в одному фізичному місці або розподілених по різних мережах.
- **Сховища даних.** Будучи централізованим репозиторієм для даних, банк даних являє собою тип бази даних, спеціально призначеної для швидкого виконання запитів і аналізу.
- **Бази даних NoSQL.** База даних NoSQL, або нереляційних база даних, дозволяє зберігати і обробляти неструктуровані або слабоструктуровані дані (на відміну від реляційної бази даних, яка задає структуру містяться в ній даних). Популярність баз даних NoSQL зростає в міру поширення і ускладнення веб-додатків.
- **Графові бази даних.** Графова база даних зберігає дані в контексті сутностей і зв'язків між сутностями.
- **Бази даних OLTP.** База даних OLTP - це швидка база даних аналітичного типу, призначена для великого обсягу транзакцій, що виконуються безліччю користувачів.

Це лише деякі з десятків типів баз даних, які використовуються в даний час. Інші, менш поширені бази даних, призначені для дуже специфічних наукових, фінансових та інших завдань. Крім появи нових типів, бази даних розвиваються в абсолютно нових напрямах - змінюються підходи до розробки технологій, відбуваються значні зрушення, такі як впровадження хмарних технологій і автоматизації.

1.4 Реляційна СУБД PostgreSQL

PostgreSQL - це вільно поширювана об'єктно-реляційна система управління базами даних (ORDBMS), найбільш розвинена з відкритих СУБД в світі і є реальною альтернативою комерційних баз даних.
Фундаментальна характеристика об'єктно-реляційної бази даних - це підтримка об'єктів і їх поведінки, включаючи типи даних, функції, операції, домени і індекси. Це робить Постгрес неймовірно гнучким і надійним. Серед іншого, він вміє створювати, зберігати та видавати складні структури даних. У деяких прикладах нижче ви побачите вкладені і складові конструкції, які не підтримуються стандартними РСУБД.

1.5 Переваги PostgreSQL над іншими SQL базами з відкритим вихідним кодом

1.5.1 Структури і типи даних

Існує великий список типів даних, які підтримує Постгрес. Крім числових, з плаваючою точкою, текстових, булевих і інших очікуваних типів даних (а також безлічі їх варіацій), PostgreSQL може похвалитися підтримкою uuid, грошового, перечислень, геометричного, бінарного типів, мережевих адрес, бітових рядків, текстового пошуку, xml, json , масивів, композитних типів і діапазонів, а також деяких внутрішніх типів для ідентифікації об'єктів і розташування логів. Варто сказати, що MySQL, MariaDB і Firebird теж мають деякі з цих типів даних, але тільки Постгрес підтримує їх всіх. Давайте розглянемо докладніше деякі з них:

- **Мережеві адреси**

  PostgreSQL забезпечує зберігання різних типів мережевих адрес. Тип даних CIDR (безкласова маршрутизація інтернет домену, Classless Internet Domain Routing) слідує угоду для мережевих адрес IPv4 і IPv6. Ось кілька прикладів:

- 192.168.100.128/25
- 10.1.2.3/32
- 2001: 4f8: 3: ba: 2e0: 81ff: fe22: d1f1 / 128
- :: ffff: 1.2.3.0/128

  Також для зберігання мережевих адрес доступний тип даних INET, який використовується для IPv4 і IPv6 хостів, де підмережі є необов'язковими.

Тип даних MACADDR може використовуватися для зберігання MAC-адрес для ідентифікації обладнання, таких як 08-00-2b-01-02-03.

У MySQL і MariaDB теж є INET функції для конвертації мережевих адрес, але вони не надають типи даних для внутрішнього зберігання мережевих адрес. У Firebird теж немає типів для зберігання мережевих адрес.

- **Багатовимірні масиви**

Оскільки Постгрес - це об'єктно-реляційна база даних, масиви значень можуть зберігатися для більшості існуючих типів даних. Зробити це можна шляхом додавання квадратних дужок до специфікації типу даних для стовпця або за допомогою виразу ARRAY. Розмір масиву може бути заданий, але це необов'язково. Давайте розглянемо меню святкового пікніка для демонстрації використання масивів:

```
1. -- створюємо таблицю, у які значенння є масивами
2. CREATE TABLE holiday_picnic (
3.       holiday varchar(50)
4.       sandwich text[], -- масив
5.       side text[] [], -- багатовимірний масив
6.       dessert text ARRAY, -- масив
7.       beverage text ARRAY[4] -- масив з 4-х елементів
8. );
9.
10.  -- вставляєсо значения масивів в таблицю
11. INSERT INTO holiday_picnic VALUES
12.       ('Labor Day',
13.       '{"roast beef","veggie","turkey"}',
14.       '{
15.          {"potato salad","green salad","macaroni salad"},
16.          {"chips","crackers"}
17.       }',
18.       '{"fruit cocktail","berry pie","ice cream"}',
19.       '{"soda","juice","beer","water"}'
20.       );
```

MySQL, MariaDB, і Firebird так не вміють. Щоб зберігати такі масиви значень в традиційних реляційних базах даних, доведеться використовувати обхідний шлях і створювати окрему таблицю з рядками для кожного із значень масиву.

- **Геометричні дані**

Геодані швидко стають основною вимогою для багатьох додатків. PostgreSQL вже давно підтримує безліч геометричних типів даних, таких як точки, лінії, кола і багатокутники. Один з цих типів - PATH, він складається з безлічі послідовно розташованих точок і може бути відкритим (початкова та кінцева точки не пов'язані) або закритим (початкова та кінцева точки пов'язані). Давайте розглянемо як приклад туристичну стежку. В даному випадку туристична стежка - це петля, тому початкова і кінцева точки пов'язані, і, значить, мій шлях є закритим.

Круглі дужки навколо набору координат вказують на закритий шлях, а квадратні - на відкритий.

```
- створюємо таблицю для зберігання шляхів
CREATE TABLE trails (
      trail_name varchar(250),
      trail_path path
);


 -- маршрут визначається координатами в форматі широта-довгота
INSERT INTO trails VALUES
      ('Dool Trail - Creeping Forest Trail Loop',
      ((37.172,-122.22261666667),
      (37.171616666667,-122.22385),
      (37.1735,-122.2236),
      (37.175416666667,-122.223),
      (37.1758,-122.22378333333),
      (37.179466666667,-122.22866666667),
      (37.18395,-122.22675),
      (37.180783333333,-122.22466666667),
      (37.176116666667,-122.2222),
      (37.1753,-122.22293333333),
      (37.173116666667,-122.22281666667)));
```

Зауважте, що в MySQL 5.7.8 і в MariaDB, починаючи з версії 5.3.3, були додані розширення типів даних для підтримки стандарту географічної інформації OpenGIS. Ця версія MySQL і наступні версії MariaDB пропонують зберігання типів даних, аналогічні штатним геоданим Постгреса. Проте, в MySQL і MariaDB значення даних спочатку повинні бути сконвертовані в геометричний формат простими командами перед тим, як будуть вставлені в таблицю. Firebird на даний момент не підтримує геометричні типи даних.

- **Створення нового типу**

  Якщо раптом так трапиться, що великого списку типів даних Постгреса вам виявиться недостатньо, ви можете використовувати команду CREATE TYPE, щоб створити нові типи даних, такі як складові, перечислення, діапазон і базовий. Розглянемо приклад створення і відправлення запитів нового складеного типу:

```
 - створюємо новий тип "wine"
CREATE TYPE wine AS (
      wine_vineyard varchar(50),
      wine_type varchar(50),
      wine_year int
);


CREATE TABLE pairings (
      menu_entree varchar(50),
      wine_pairing wine
);

 -- вставляєсо дані в таблицу за допомогою виразу ROW
INSERT INTO pairings VALUES
      ('Lobster Tail',ROW('Stag''s Leap','Chardonnay', 2012)),
      ('Elk Medallions',ROW('Rombauer','Cabernet Sauvignon',2012));


SELECT (wine_pairing).wine_vineyard, (wine_pairing).wine_type
FROM pairings
WHERE menu_entree = 'Elk Medallions';
```

1.5.2 Розмір даних

PostgreSQL може обробляти багато даних. Поточні опубліковані обмеження перераховані нижче:

Максимальний розмір бази даних	Не обмежений
Максимальний розмір таблиці	32 TB
Максимальна розмір рядку	1.6 TB
Максимальний розмір поля	1 ГБ
Максимальна кількість рядків в таблиці	Не обмежений
Максимальна кількість стовпців в таблиці	250-1600 залежно від типу стовпця
Максимальна кількість індексів в таблиці	Не обмежений

Для порівняння, MySQL і MariaDB сумно відомі обмеженням розміру рядків в 65 535 байт. Firebird також пропонує всього лише 64 Кб як максимального розміру рядка. Зазвичай обсяг даних обмежується максимальним розміром файлів операційної системи. Оскільки PostgreSQL вміє зберігати табличні дані в безлічі файлів меншого розміру, він може обійти це обмеження. Але варто відзначити, що занадто велика кількість файлів може негативно позначитися на продуктивності. MySQL і MariaDB підтримують більшу кількість стовпців в таблиці (до 4,096 в залежності від типу даних) і великі індивідуальні розміри таблиці, ніж PostgreSQL.

1.6 Недоліки PostgreSQL

- **Продуктивність**: В простих операціях читання PostgreSQL може поступатися своїм суперникам.
- **Популярність**: через свою складність інструмент не дуже популярний.
- **Хостинг**: через перерахованих вище факторів проблематично знайти підходящого провайдера.

1.7 Принципи нормалізації баз даних

Нормалізація - це процес організації даних в базі даних, що включає створення таблиць і встановлення відносин між ними відповідно до правил, які забезпечують захист даних і роблять базу даних більш гнучкою, усуваючи надмірність і неузгоджені залежності.

Надмірність даних призводить до непродуктивного витрачання вільного місця на диску і ускладнює обслуговування баз даних. Наприклад, якщо дані, що зберігаються в кількох місцях, буде потрібно змінити, в них доведеться внести одні і ті ж зміни в усіх цих місцях. Зміна адреси клієнта набагато легше реалізувати, якщо в базі даних ці відомості зберігаються тільки в таблиці Customers і ніде більше.

Існує кілька правил нормалізації баз даних. Кожне правило називається «нормальною формою». Якщо виконується перше правило, говорять, що база даних представлена в «першій нормальній формі». Якщо виконуються три перших правила, вважається, що база даних представлена в «третій нормальній формі». Є й інші рівні нормалізації, однак для більшості додатків досить нормалізувати бази даних до третьої нормальної форми.

Як і у випадку з багатьма іншими формальними правилами і специфікаціями, забезпечити повну відповідність реальних ситуацій не завжди можливо. Як правило, для виконання нормалізації доводиться створювати додаткові таблиці, і деякі клієнти вважають це небажаним. Збираючись порушити одне з перших трьох правил нормалізації, переконайтеся в тому, що в додатку враховані всі пов'язані з цим проблеми, такі як надмірність даних і неузгоджені залежності.

В описах нижче наведені відповідні приклади.

1.7.1 Перша нормальна форма

- Усуньте повторювані групи в окремих таблицях.
- Створіть окрему таблицю для кожного набору пов'язаних даних.
- Визначте кожен набір пов'язаних даних за допомогою первинного ключа.

Не використовуйте кілька полів в одній таблиці для зберігання схожих даних. Наприклад, для спостереження за товаром, який закуповується у двох різних постачальників, можна створити запис з полями, що визначають код першого постачальника і код другого постачальника.
Що станеться при додаванні третього постачальника? Додавання третього поля небажано, так як для цього потрібно змінювати програму і таблицю, тому даний спосіб погано адаптується до динамічного зміни числа постачальників. Замість цього можна помістити всі відомості про постачальників в окрему таблицю Vendors (постачальники) і зв'язати товари з постачальниками за допомогою кодів товарів або постачальників з товарами за допомогою кодів постачальників.

1.7.2 Друга нормальна форма

- Створіть окремі таблиці для наборів значень, що відносяться до декількох записів.
- Зв'яжіть ці таблиці за допомогою зовнішнього ключа.

Записи можуть залежати тільки від первинного ключа таблиці (складеного ключа, якщо необхідно). Візьмемо для прикладу адреса клієнта в системі бухгалтерського обліку. Ця електронна адреса необхідний не тільки таблиці Customers, але і таблицями Orders, Shipping, Invoices, Accounts Receivable і Collections. Замість того щоб зберігати адресу клієнта як окремий елемент в кожній з цих таблиць, зберігайте його в одному місці: або в таблиці Customers, або в окремій таблиці Addresses.

1.7.3 Третя нормальна форма

- Усуньте поля, які не залежать від ключа.

Значення, що входять до запису та не є частиною ключа цього запису, не належать таблиці. Якщо вміст групи полів може ставитися більш ніж до однієї записи в таблиці, подумайте про те, чи не помістити ці поля в окрему таблицю.

Наприклад, в таблицю Employee Recruitment (наймання співробітників) можна включити адресу кандидата і назва університету, в якому він отримав освіту. Однак для організації групової поштової розсилки необхідний повний список університетів. Якщо відомості про університети будуть зберігатися в таблиці Candidates, скласти список університетів при відсутності кандидатів не вийде. Таким чином, створіть замість цього окрему таблицю Universities і зв'яжіть її з таблицею Candidates за допомогою ключа - коду університету.

Виконувати нормалізацію до третьої нормальної форми може бути доцільно тільки для часто змінюваних даних. Якщо при цьому збережуться залежні поля, спроектуйте додаток так, щоб при зміні одного з цих полів користувач повинен був перевірити всі пов'язані поля.

2 МОДЕЛЮВАННЯ

У цьому розділі сформульовується постановка задачі, описуються сутності і відповідні таблиці, які вирішують дану проблему, а також наприкінці представляється ER-діаграма.

2.1 Постановка задачі

Спроектувати та реалізувати базу даних “Медична картотека”.

2.2 Опис сутностей

Можна виділити такі сутності:

- **Patient(Пацієнт) –** оскільки я створюю базу даних “Медична картотека” , ця сутність є центральним елементом у ній.
- **Doctor(Лікар) -** сутність, яка також є головною частиною у базі.
- **History(Історія) –** сутність, яка є з’єднуючим елементом між двома вищими сутностями. Вона містить детальну інформацію про прийом між пацієнтом і лікарем, а також про виписаний рецепт і час зустрічі.
- **Document(Довідка) –** ця сутність являє собою стан здоров’я пацієнта. Наприклад, загальний аналіз крові чи рентген грудної клітки і т.д.
- **Clinic(Лікарня) –** сутність лікарня.
- **Prescription(Рецепт) –** сутність рецепт, у випадку хвороби пацієнта.
- **Medication(Ліки) –** сутність ліки.
- **Desease(Хвороба) –** сутність хвороба.

2.3 Загальний опис таблиць

1. Таблиця **Patiens(Пацієнти)** містить:
   patient_id – унікальний код пацієнта;
   surname, name, patronymic – ПІБ пацієнта;
   gender – стать пацієнта;
   birth_date – дата народження;
   mobile_number – мобільний номер;
   place – місто(село або смт);
   address – адреса проживання пацієнта;
2. Таблиця **Doctors(Лікарі)** містить:
   doctor_id – унікальний код лікаря;
   surname, name, patronymic – ПІБ лікаря;
   doctor_spec – спеціалізація лікаря;
3. Таблиця **History(Історія хвороб)** містить:
   event_id – унікальний код зустрічі між пацієнтом і лікарем;
   event_start – дата і час початку прийому;
   event_end – дата і час кінцю прийому;
   event_description – опис прийому, можливо додаткові замітки;
   patient_id – унікальний код пацієнта;
   doctor_id – унікальний код лікаря;
4. Таблиця **Prescriptions(Рецепти)** містить:
   presc_id – унікальний код рецепту;
   description – опис рецепту;
   doctor_id - унікальний код лікаря, який виписав цей рецепт;
   event_id - унікальний код зустрічі між пацієнтом і лікарем;
5. Таблиця **Medications(Медикаменти)** містить:
   med_id – унікальний код ліків;
   med_type – тип ліків;
   med_name – назва ліків;
   presc_id – унікальний код рецепту, до якого належать ці ліки;
6. Таблиця **Documents(Довідки)** містить:
   doc_id – унікальний код довідки;
   doc_date – дата отримання довідки;
   doc_desciption – опис довідки;
   patient_id - унікальний код пацієнта, який отримує довідку;
   event_id – унікальний код прийому, для якого потрібна ця довідка;
7. Таблиця **Deseases(Хвороби)** містить:
   desease_id – унікальний код хвороби;
   desease_name – назва хвороби;
   desease_category – категорія хвороби;

desease_description – особливі замітки про хворобу;

patient_id – унікальний код пацієнта, який хворій даною хворобою;

8. Таблиця **Clinics(Лікарні)** містить наступні поля:
   clinic_id – унікальний код лікарні;
   clinic_name – назва клініки;
   clinic_description – короткий опис клініки;
   clinic_place – населений пункт лікарні;
   clinic_address – адреса;
9. Проміжна таблиця **Doctor_clinic(Лікар_клініка)** містить:
   doctor_id – унікальний код лікаря;
   clinic_id – унікальний код лікарні чи поліклініки;

2.4 Опис взаємозв'язків між таблицями

Мета правильної розробки таблиць – видалення надлишкових (повторюваних) даних. Для досягнення цієї мети таблицю розділяють на багато тематичних таблиць, щоб кожний факт було представлено тільки один раз.

Потім використовують засоби, за допомогою яких ці розділені дані можна зібрати разом – для цього потрібно вставити в пов'язані між собою таблиці спільні поля.

Модель «сутність - зв'язок» заснована на використанні 3-х основних конструктивних елементах:

- Сутність.
- Атрибут.
- Зв'язок.

Існує три типи зв'язків між таблицями:

- Зв'язок "один-до-багатьох"
  Давайте скористаємося базою даних відстеження замовлень, до якої входять таблиця "Клієнти" й таблиця "Замовлення". Клієнт може розмістити будь-яку кількість замовлень. Таким чином, для будь-якого клієнта, представленого в таблиці "Клієнти", у таблиці "Замовлення" може міститися багато замовлень. Взаємозв'язок між таблицями "Клієнти" та "Замовлення" – це зв'язок "один-до-багатьох".
- Зв'язок "багато-до-багатьох"
  Тепер давайте розглянемо зв'язок між таблицями "Товари" та "Замовлення". В одному замовленні може бути вказано кілька товарів. З іншого боку, один товар може зустрічатися в багатьох замовленнях. Отже, кожному запису в таблиці "Замовлення" може відповідати багато записів у "Товари".

Крім того, кожному запису в таблиці "Товари" також може відповідати багато записів у таблиці "Замовлення". Такий тип зв’язку називається зв’язком "багато-до-багатьох". Зверніть увагу, що для виявлення наявних зв’язків "багато-до-багатьох" між таблицями важливо розглянути обидва кінці зв’язку.

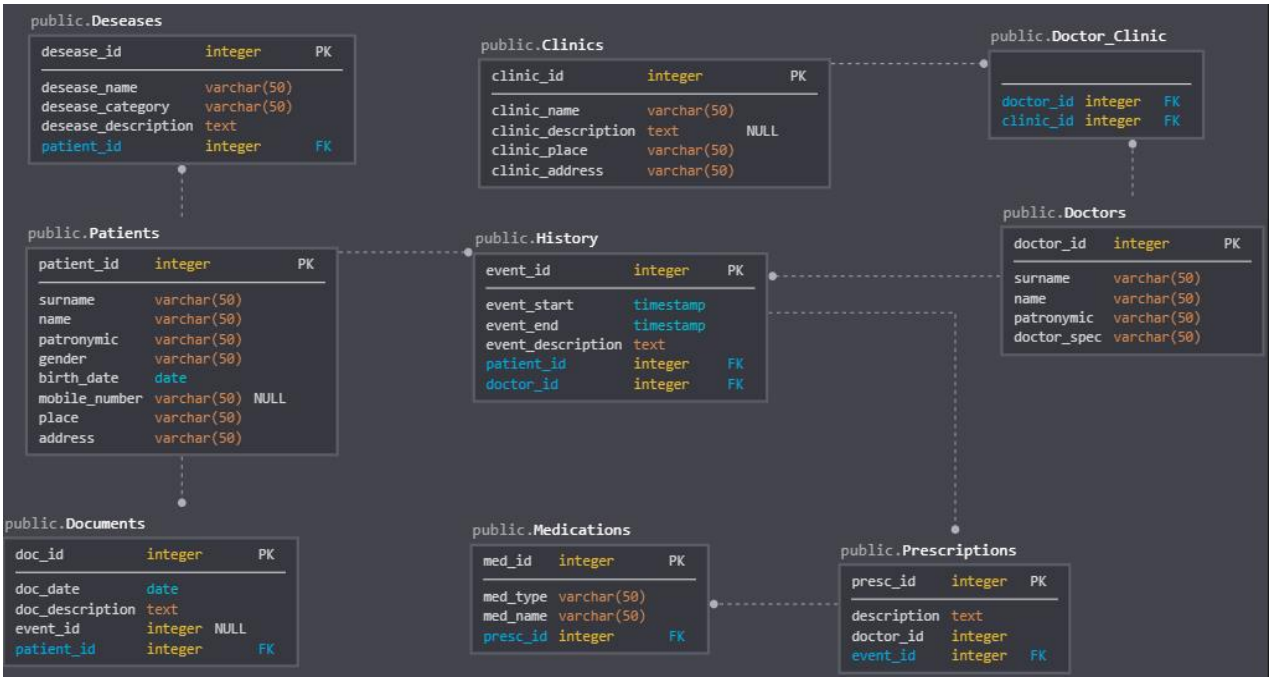
- Зв’язок "один-до-одного"
  У зв’язку "один-до-одного" кожному запису в першій таблиці може відповідати лише один запис у другій таблиці, а кожному запису у другій таблиці може відповідати лише один запис у першій таблиці. Цей зв’язок не дуже поширений, оскільки зазвичай відомості, пов’язані між собою в такий спосіб, зберігаються в одній таблиці. Зв’язок "один-до-одного" можна використовувати для розділення таблиці з великою кількістю полів, для відокремлення частини таблиці з міркувань безпеки або для зберігання даних, які застосовуються лише до підмножини головної таблиці. У разі визначення такого зв’язку в обох таблицях мають бути спільні поля.

В курсовому проекті були визначені наступні типи зв’язків (Таблиця 1):
Таблиця 1 – Класифікація зв’язків

Номер зв’язку	Батьківська таблиця	Дочірня таблиця	Тип зв’язку
1	Patients	Documents	1:M
2	Patients	Deseases	1:M
3	Patients	History	1:1
4	Doctors	Clinics	M:M
5	Doctors	History	1:M
6	History	Prescriptions	1:1
7	Prescriptions	Medications	1:M

2.5 ER-діаграма

Рисунок 1 – ER-діаграма спроектованої бази



3 РЕАЛІЗАЦІЯ

У даному розділі розглядається процес створення таблиць з використанням PostgreSQL і написання взаємозв'язків між ними. Оскільки після проведення попереднього етапу ми отримаємо готовий приклад робочої бази , то ми будемо здатні заповнити її даними і перевірити базові і складні запити на одерження очікуваного результату і ефективності її функціонування.

3.1 Створення таблиць і зв'язків між ними

Оскільки у медичній картці головне місце посідає пацієнт, то почнемо з створення цієї таблиці:

Рисунок 2 – Створення таблиці **Patients(Пацієнти)**

```
1.  -- *************** SqlDBM: PostgreSQL ****************;
2.  -- ***************************************************;
3.
4.
5.  -- ************************************ "public"."Patients"
6.
7.  CREATE TABLE "public"."Patients"
8.  (
9.   "patient_id"    integer NOT NULL,
10.  "surname"       varchar(50) NOT NULL,
11.  "name"          varchar(50) NOT NULL,
12.  "patronymic"    varchar(50) NOT NULL,
13.  "gender"        varchar(50) NOT NULL DEFAULT male,
14.  "birth_date"    date NOT NULL,
15.  "mobile_number" varchar(50) NULL,
16.  "place"         varchar(50) NOT NULL,
17.  "address"       varchar(50) NOT NULL
18.
19. );
20.
21. CREATE UNIQUE INDEX "PK_Patients" ON "public"."Patients"
22. (
23.  "patient_id"
24. );
```

Наступний крок – це створення історії хвороб, яка буде містити в собі дані про прийом пацієнта у відповідного лікаря і призначення йому відповідних ліків чи направлень. Ця інформація зберігається у вигляді таблиці **History(**Рисунок 3**).**

Рисунок 3 – Створення таблиці “Історія хвороб”

```
-- *************** SqlDBM: PostgreSQL ****************;
-- ***************************************************;


-- ************************************ "public"."History"

CREATE TABLE "public"."History"
(
 "event_id"          integer NOT NULL,
 "event_start"       timestamp NOT NULL,
 "event_end"         timestamp NOT NULL,
 "event_description" text NOT NULL,
 "patient_id"        integer NOT NULL,
 "doctor_id"         integer NOT NULL,
 CONSTRAINT "FK_65" FOREIGN KEY ( "patient_id" ) REFERENCES "public"."Patients"
    ( "patient_id" ),
 CONSTRAINT "FK_68" FOREIGN KEY ( "doctor_id" ) REFERENCES "public"."Doctors" (
    "doctor_id" )
);

CREATE UNIQUE INDEX "PK_History" ON "public"."History"
(
 "event_id"
);

CREATE INDEX "fkIdx_65" ON "public"."History"
(
 "patient_id"
);

CREATE INDEX "fkIdx_68" ON "public"."History"
(
 "doctor_id"
);
```

Тепер створюємо одну з головних таблиць(Рисунок 4)

Рисунок 4 – Створення таблиці “Лікарі”

```
CREATE TABLE "public"."Doctors"
(
 "doctor_id"   integer NOT NULL,
 "surname"     varchar(50) NOT NULL,
 "name"        varchar(50) NOT NULL,
 "patronymic"  varchar(50) NOT NULL,
 "doctor_spec" varchar(50) NOT NULL

);

CREATE UNIQUE INDEX "PK_Doctors" ON "public"."Doctors"
(
 "doctor_id"
);
```

Оскільки лікарі можуть працювати в багатьох клініках, а з іншого боку в одній клініці може працювати багато лікарів, то ми повинні реалізувати відношення “Багато-до-Багатьох” між ними. Створення цього зв’язку приведено у Додатку.

Рисунок 5 – Створення таблиці “Clinics”

```
CREATE TABLE "public"."Clinics"
(
 "clinic_id"          integer NOT NULL,
 "clinic_name"        varchar(50) NOT NULL,
 "clinic_description" text NULL,
 "clinic_place"       varchar(50) NOT NULL,
 "clinic_address"     varchar(50) NOT NULL

);

CREATE UNIQUE INDEX "PK_Clinics" ON "public"."Clinics"
(
 "clinic_id"
);
```

Після прийому лікаря часто виникала ситуація, коли було потрібно отримати довідку чи результат якогось обстеження. Отже, потрібно створити окрему таблицю “Довідки”, яка буде містити дану інформацію.

Рисунок 6 – Створення таблиці “Довідки”

```
CREATE TABLE "public"."Documents"
(
 "doc_id"          integer NOT NULL,
 "doc_date"        date NOT NULL,
 "doc_description" text NOT NULL,
 "event_id"        integer NULL,
 "patient_id"      integer NOT NULL,
 CONSTRAINT "FK_75" FOREIGN KEY ( "patient_id" ) REFERENCES "public"."Patients"
 ( "patient_id" )
);

CREATE UNIQUE INDEX "PK_Documents" ON "public"."Documents"
(
 "doc_id"
);

CREATE INDEX "fkIdx_75" ON "public"."Documents"
(
 "patient_id"
);
```

Також для лікаря буде важлива інформація про наявність алергії чи інших хронічних захворювань у пацієнта для подальшого успішного лікування. Тому було б зручно оформити це у вигляді окремої таблиці.

Рисунок 7 – створення таблиці “Хвороби”

```
CREATE TABLE "public"."Deseases"
(
 "desease_id"          integer NOT NULL,
 "desease_name"        varchar(50) NOT NULL,
 "desease_category"    varchar(50) NOT NULL,
 "desease_description" text NOT NULL,
 "patient_id"          integer NOT NULL,
 CONSTRAINT "FK_72" FOREIGN KEY ( "patient_id" ) REFERENCES "public"."Patients"
 ( "patient_id" )
);

CREATE UNIQUE INDEX "PK_Deseases" ON "public"."Deseases"
(
 "desease_id"
);

CREATE INDEX "fkIdx_72" ON "public"."Deseases"
(
 "patient_id"
);
```

База даних повинна містити також таблицю про рецепт і його застосування та виходячи з цього таблицю ліків. Створення цих таблиць описується в Додатку.

3.2 Заповнення таблиць даними

Спочатку заповнюємо таблицю “Patients”. Для цього я буду використовувати сайт **Mockaroo.com**, який дозволяє генерувати тестові дані для перевірки.
З інтерфейсом сайту можна ознайомитися в Додатку.
З використанням його функціональності я отримав файл з розширення SQL, який можем виконати так:

```
postgres=# \i C:/Users/T440/Desktop/course/Patients.sql
```

Рисунок 8 – Виконання SQL команд з даного файлу

Також процес вставки даних у інші таблиці не відрізняється від попердньо проведений дій, тому немає сенсу його описувати. Краще перейдімо до найголовнішого елементу розробки баз даних – виконання запитів і перевірки результатів.

3.3 Написання запитів

Одним з найбільш ефективних і універсальних способів вибірки даних з таблиць бази даних є використання запитів SQL.

У розробленій базі даних передбачені наступні запити:

1. Проста вибірка

```
1. SELECT * FROM “Clinics”;
2. SELECT * FROM “Deseases”;
3. SELECT * FROM “Doctor_Clinic”;
4. SELECT * FROM “Doctors”;
5. SELECT * FROM “Documents”;
6. SELECT * FROM “History”;
7. SELECT * FROM “Medications”;
8. SELECT * FROM “Patients”;
9. SELECT * FROM “Prescriptions”;
```

2. Вибірка обчислювальних значень

```
1. select count("Deseases"."patient_id") from "Patients" join "Deseases"
2. on "Patients"."patient_id" = "Deseases"."patient_id"
3. where "Deseases"."desease_category" = 'Allantoin' GROUP BY "Deseases"."patient_
   id";
```

3. Вибірка значень з певного діапазону

```
select "surname", "name" from "Patients" where "birth_date" between '2019-01-
01' and '2019-12-31'
```

4. Запити з сортуванням

```
1. select "doctor_spec" from "Doctors" order by "surname";
2. select "Deseases"."desease_name" from "Documents"
3. join "Patients" on "Documents"."patient_id" = "Patients"."patient_id"
4. join "Deseases" on "Patients"."patient_id" = "Deseases"."patient_id" order by "
   Patients"."name";
```

5. Вибірка з пов’язаних таблиць

```
1. select "Clinics"."clinic_name", "Doctors"."surname" from "Clinics", "Doctor_Cli
   nic", "Doctors"
2. where "Doctor_Clinic"."doctor_id" = "Doctor_Clinic"."clinic_id";
```

ВИСНОВКИ

Метою курсової роботи було проектування бази даних медичної картотеки. Для виконання курсової роботи були проведені всі необхідні дослідження, що стосуються розробки стратегії автоматизації, в результаті яких була надана відповідь на принципові запитання, що стосуються автоматизації поліклініки чи лікарні.

Дана БД забезпечує надійне зберігання інформації, а також істотну економію часу, що витрачається на пошук, редагування існуючих даних. При створенні даної бази даних була, проаналізувала предметна область.

Були описані технології функціонування ІС, побудовані концептуальна і логічна моделі БД, виконано фізичне проектування БД, розроблена функціональної моделі СКБД, розроблені запити для існуючої БД.

При розробці бази даних проект було розділено на три розділи :

Розділ 1:Було розгянуто фундаментальні поняття реляційних баз даних, принципи їх нормалізації. Також тут детально описувалися функціональність вибраної технології, її переваги та недоліки.

Розділ 2: Була сформульована постановка задачі, описані сутності і відповідні таблиці, які вирішують дану проблему.

Розділ 3: Було розглянуто мотивований вибір СКБД для реалізації проекту, реалізація БД, результати, одержувані при роботі з БД, розроблені уявлення для відображення результатів вибірки.

Результатом роботи над КП є створена працездатна база даних, перевагами якої є зручність і швидкість знаходження серед великої кількості інформації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Електронні ресурси	1. https://tproger.ru/translations/sqlite-mysql-postgresql-comparison/ 2. shorturl.at/EGSTW 3. https://habr.com/ru/post/282764/ 4. https://mockaroo.com/ 5. https://sqldbm.com/Home/

ДОДАТОК

Створення проміжної таблиці “Doctor_Clinic”

```
1.  -- *************** SqlDBM: PostgreSQL ***************;
2.  -- ************************************************;
3.
4.
5.  -- ********************************** "public"."Doctor_Clinic"
6.
7.  CREATE TABLE "public"."Doctor_Clinic"
8.  (
9.   "doctor_id" integer NOT NULL,
10.  "clinic_id" integer NOT NULL,
11.  CONSTRAINT "FK_85" FOREIGN KEY ( "doctor_id" ) REFERENCES "public"."Doctors" (
     "doctor_id" ),
12.  CONSTRAINT "FK_88" FOREIGN KEY ( "clinic_id" ) REFERENCES "public"."Clinics" (
     "clinic_id" )
13.);
14.
15.CREATE INDEX "fkIdx_85" ON "public"."Doctor_Clinic"
16.(
17.  "doctor_id"
18.);
19.
20.CREATE INDEX "fkIdx_88" ON "public"."Doctor_Clinic"
21.(
22.  "clinic_id"
23.);
```

Приклад графічного інтерфейсу сайту **Mockaroo.com**