Міністерство освіти і науки України
ДВНЗ «Прикарпатський національний університет імені Василя Стефаника»
Кафедра комп'ютерної інженерії та електроніки
(повна назва кафедри)

Шкандрій Юлія Василівна
Shkandrii Yuliia

УДК 004:681.5

Спеціальність 123 «Комп'ютерна інженерія»
(шифр та назва спеціальності)

Кваліфікаційна робота
на здобуття освітнього ступеня бакалавр
(бакалавр, спеціаліст, магістр)

Система обробки даних діагностування і лікування пацієнтів на основі СУБД PostgreSQL

Patient Diagnostic and Treatment Data Processing System based on PostgreSQL DBMS

Науковий керівник:
доцент Голота В. І.

Рецензент:
к.х.н., проф. каф. фізики і хімії твердого тіла
Горічок І. В.

Івано-Франківськ
2020

Формат	Поз.	Позначення	Найменування	Кільк.	Прим.
		123.УДК:004:681.5	Пояснювальна записка	51	

Змн.	Арк.	№ докум.	Підпис	Дата	123.УДК:004:681.5		
Розробила		Шкандрій Ю. В.			Специфікація	Літ.	Арк. 2 / Аркушів 1
Перевірив		Голота В. І.					
Н. Контр.							
Затвердив							

АНОТАЦІЯ

В бакалаврській роботі розроблено систему для автоматизації обробки даних діагностування і лікування пацієнтів на основі СУБД PostgreSQL.

Система складається з бази даних і інтерфейсу користувача для взаємодії з серверною частиною. Розробка бази даних здійснювалася засобами системи управління базами даних PostgreSQL, а розробка інтерфейсу – мовою C# та API Windows Forms.

Результатом роботи є реалізована база даних для збереження необхідної інформації та побудований інтерфейс користувача для керування системою.

ABSTRACT

In graduation work has developed a system for automating the processing of patient diagnosis and treatment based on PostgreSQL DBMS.

The system consists of a database and user interface to interact with the server part. Database development was done through the PostgreSQL database management system, and interface development in C # and the Windows Forms API.

The result is a database for storing the necessary information and a built-in user interface for managing the system.

Міністерство освіти і науки України

Державний вищий навчальний заклад

«Прикарпатський національний університет імені Василя Стефаника»

Фізико-технічний факультет

Кафедра «Комп’ютерної інженерії та електроніки»

Пояснювальна записка

до кваліфікаційної роботи на тему:

Система обробки даних діагностування і лікування пацієнтів на основі СУБД PostgreSQL

					123.УДК:004:681.5		
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробила		Шкандрій Ю.В.			Пояснювальна записка	*Літ.*	*Арк.* 4 / *Аркушів* 51
Перевірив		Голота В. І.					
Н. Контр.							
Затвердив							

ЗМІСТ

ВСТУП

Автоматизація в наш час відіграє важливу роль у життєдіяльності людини. Це відноситься і до автоматизації обробки даних. Створення баз даних набуло великої популярності на сучасному етапі розвитку комп'ютерних технологій. Зокрема, бази даних створили для полегшення роботи користувача, який працює з великими обсягами даних.

База даних – це сукупність даних, яка відображає стан об'єктів та їх взаємозв'язок і зберігається в пам'яті обчислювальної системи. Дані з бази даних використовують в різних додатках, а способи використання цих даних при потребі можна легко і швидко змінити.

Для обліку пацієнтів у лікарнях є дуже потрібними бази даних. Адже, обсяг інформації, що обробляється в лікарнях є дуже великим. Правильно розроблена система обліку пацієнтів дуже сильно зекономить час при зверненні до необхідних даних. При правильному складанні та внесенні даних в базу швидкість пошуку потрібної інформації зводиться до мінімуму. Створення такої бази даних допоможе з легкістю працювати з даними, які зберігається в ній та дозволить отримати повну інформацію, як про окремого пацієнта, так і про всіх пацієнтів обраного лікаря.

Отже, метою даної дипломної роботи є розробка компонентів системи обробки даних діагностування і лікування пацієнтів – бази даних та інтерфейсу користувача.

1. СУБД POSTGRESQL І ГРАФІЧНИЙ ІНТЕРФЕЙС

PostgreSQL – це потужна об'єктно–реляційна база даних з відкритим кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають та масштабують найскладніші навантаження даних. Витоки PostgreSQL відносяться до 1986 року в рамках проєкту «POSTGRES» в Каліфорнійському університеті в Берклі.

PostgreSQL заслужив високу репутацію за свою перевірену архітектуру, розширюваність та надійність, цілісність даних, надійний набір функцій. PostgreSQL працює на всіх основних операційних системах, сумісний з ACID ((atomicity (атомарність), consistency (узгодженість), isolation (ізольованість), durability (довговічність)) сукупність властивостей транзакцій бази даних, призначених для дії гарантії, навіть в разі виникнення помилок, збоїв електроживлення і т.д.) з 2001 року і має потужні додатки, такі як розширювач геопросторових баз даних PostGIS [1].

1.1. Проєкт POSTGRES в Берклі

Проєкт POSTGRES, очолюваний професором Майклом Стоунбрейкер, спонсорувало агентство DARPA при Міноборони США, Управління військових досліджень, Національний Науковий Фонд і компанія ESL. Реалізація проєкту розпочалася в 1986 році.

З тих пір POSTGRES пройшов кілька етапів розвитку. Перша «демоверсія» почала працювати в 1987 році і в 1988 році її було показано на конференції «ACM–SIGMOD». Першу версію випустили для декількох зовнішніх користувачів в червні 1989 року. У відповідь на критику першої системи правил, вона була перероблена, і в другій версії, випущеній в червні 1990 році, була вже нова система правил. У 1991 році вийшла третя версія, в якій з'явилася підтримка різних менеджерів сховища, покращений виконувач запитів і переписана система правил. Наступні випуски до Postgres95 в основному були спрямовані на покращення портування і надійності.

POSTGRES застосовувався для реалізації безлічі дослідних і виробничих завдань. У їх числі: система для аналізу фінансових даних, база даних спостережень за астероїдами, пакет моніторингу роботи реактивних двигунів, база даних для медичної інформації, географічні інформаційні системи. POSTGRES використовувався також для навчання в університетах. Компанія IIT скористалася кодом і знайшла йому комерційне застосування. Наприкінці 1992 року POSTGRES став основною СУБД наукового обчислювального проєкту «Sequoia 2000» .

В 1993 році кількість зовнішніх користувачів подвоїлася. Стало очевидно, що обслуговування коду та підтримка займає надто багато часу, і його не вистачає на дослідження. Для зниження цього навантаження проєкт POSTGRES в Берклі був офіційно закритий на версії 4.2.

1.2. Postgres95

В 1994 році в POSTGRES було додано інтерпретатор мови SQL. Вже з новим ім'ям та з відкритим вихідним кодом, Postgres95 був опублікований в Інтернеті і почав свій шлях як нащадок розробленого в Берклі POSTGRES.

Код Postgres95 було приведено у повну відповідність з ANSI C і зменшено на 25%. Завдяки безлічі внутрішніх змін він став швидшим та зручнішим. Postgres95 версії 1.0.x працював приблизно на 30–50% швидше ніж POSTGRES версії 4.2 (за тестами Wisconsin Benchmark). Крім того, що були виправлені помилки, відбулися такі зміни:

- На зміну мови запитів PostQUEL прийшов SQL. (Інтерфейсна бібліотека libpq успадкувала своє ім'я від PostQUEL). До виходу PostgreSQL підзапити не підтримувалися, хоча їх можна було імітувати в Postgres95 за допомогою призначених для користувача функцій SQL . Були заново реалізовані агрегатні функції. Також з'явилася підтримка GROUP BY.
- Для SQL – запитів була розроблена програма (psql), в якій використовувалося GNU Readline . Стара програма «monitor» стала не потрібна.

- З'явилася нова клієнтська бібліотека libpgtcl для підтримки Tcl – клієнтів. Приклад оболонки, pgtclsh, представляв нові команди Tcl для взаємодії програм Tcl з сервером Postgres95 .
- Був вдосконалений інтерфейс для роботи з великими об'єктами. Єдиним механізмом зберігання таких даних стали інверсійні об'єкти.
- З вихідним кодом поширювалася інформація про можливості стандартного SQL, а також самого Postgres95 .
- Для компонування використовувався GNU make. Крім того, стало можливо скомпілювати Postgres95 з немодифікованою версією GCC.

1.3. PostgreSQL

В 1996 році було зрозуміло, що ім'я «Postgres95» не витримає випробування часом. Тому вибрали нове ім'я, PostgreSQL , що відображає зв'язок між оригінальним POSTGRES і більш пізніми версіями з підтримкою SQL . У той же час, було продовжено нумерацію версій з 6.0, повернувшись до послідовності, розпочатої в проєкті Берклі POSTGRES .

Багато хто продовжує називати PostgreSQL ім'ям «Postgres» (тепер уже рідко великими літерами) за традицією або для простоти. Ця назва закріпилася як псевдонім або неформальне позначення.

В процесі розробки Postgres95 основними завданнями були пошук і розуміння існуючих проблем в серверному коді. З переходом до PostgreSQL акценти змістилися до реалізації нових функцій і можливостей, хоча робота триває у всіх напрямках [2].

1.4. Архітектура PostgreSQL та основні можливості

Однією з сильних сторін PostgreSQL є її архітектура. Як і багато комерційних СУБД, PostgreSQL може застосовуватися в середовищі клієнт–сервер, що дає безліч переваг як користувачам, так і розробникам.

Основу PostgreSQL складає серверний процес бази даних. Він виконується на одному сервері. Доступ з додатків до даних бази здійснюється за допомогою

процесу бази даних. Клієнтські програми не можуть отримати доступ до даних самостійно.

Такий поділ клієнтів і сервера дозволяє побудувати розподілену систему. Можна відокремити клієнтів від сервера за допомогою мережі і розробляти клієнтські програми в середовищі, зручній для користувача. Наприклад, можна реалізувати базу даних під UNIX і створити клієнтські програми, які будуть працювати в системі Microsoft Windows.

Наведена нижче схема (рис. 1.1) показує типову модель розподіленого додатку PostgreSQL:

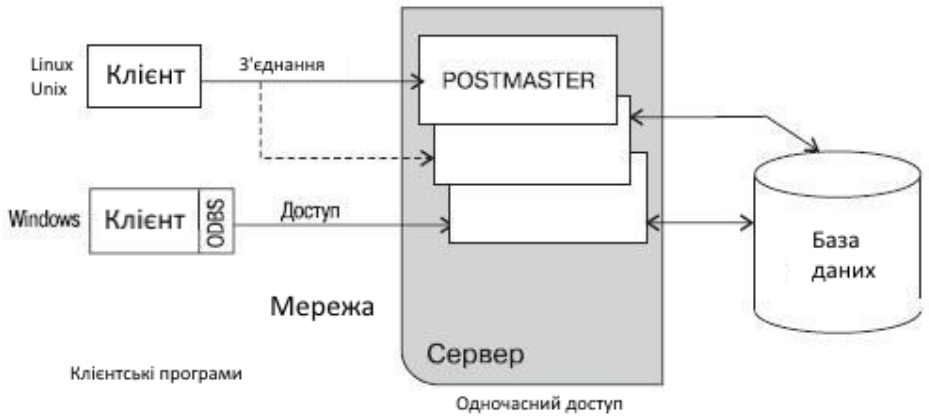

Рисунок 1.1. – Робота типового додатку PostgreSQL

Кілька клієнтів приєднуються до сервера по мережі. PostgreSQL орієнтований на протокол TCP / IP – це може бути локальна мережа або Інтернет. Кожен клієнт з'єднується з основним серверним процесом бази даних (на схемі – Postmaster), який створює новий серверний процес спеціально для обслуговування запитів на доступ до даних конкретного клієнта.

Завдяки тому, що маніпулювання даними зосереджено на сервері, СУБД не доводиться контролювати численних клієнтів, які отримують доступ в спільно використовуваний каталог сервера, і PostgreSQL підтримує цілісність даних при одночасному доступі великої кількості користувачів.

Клієнтські програми з'єднуються з базою зі спеціального протоколу PostgreSQL. Однак можна встановити на стороні клієнта програмне забезпечення, яке надає стандартний інтерфейс для роботи потрібної програми, наприклад, за

стандартом ODBC або JDBC. Доступність ODBC–драйвера дозволяє застосовувати PostgreSQL як базу даних для багатьох існуючих додатків, включаючи такі продукти Microsoft Office, як Excel і Access.

Архітектура клієнт–сервер робить можливим розподіл праці. Машина–сервер добре підходить для зберігання і управління доступом до великих обсягів даних, вона може використовуватися як надійний репозитарій. Для клієнтів можуть бути розроблені складні графічні додатки. Як альтернативу можна створити зовнішній інтерфейс на основі Інтернету, який надавав би доступ до даних і повертав результат у вигляді веб–сторінок в стандартний веб–браузер, при цьому не потрібно було б ніякого додаткового клієнтського програмного забезпечення.

PostgreSQL працює у всіх основних операційних системах, включаючи Linux, UNIX та Windows. Він підтримує текст, зображення, звуки та відео та включає інтерфейси програмування для C/C++, Java, Perl, Python, Ruby, Tcl та Open Database Connectivity (ODBC).

PostgreSQL підтримує значну частину стандартного SQL і безліч сучасних функцій, включаючи такі:

- Складні запити SQL
- Підвибір SQL
- Зовнішні ключі
- Тригер
- Перегляди
- Операції
- Мультиверсійний контроль одночасності (MVCC)
- Потокова реплікація (станом на 9,0)
- Гарячий режим очікування (станом на 9,0)

PostgreSQL підтримує чотири стандартні процедурні мови, що дозволяє користувачам писати власний код на будь–якій з мов, і він може бути виконаний сервером баз даних PostgreSQL. Ці процедурні мови – PL / pgSQL, PL / Tcl, PL /

Perl та PL / Python. Крім того, підтримуються й інші нестандартні процедурні мови, такі як PL / PHP, PL / V8, PL / Ruby, PL / Java тощо.[3][4]

1.5. Windows Forms

Windows Forms – це набір керованих бібліотек в .NET Framework, призначений для розробки багатьох клієнтських додатків. Це графічний API для відображення даних та керування взаємодією користувачів з простішим розгортанням та кращою безпекою в клієнтських додатках.

Windows Forms пропонує широку клієнтську бібліотеку, що забезпечує інтерфейс для доступу до рідних елементів графічного інтерфейсу Windows та графіки з керованого коду. Він побудований з керованої архітектури, орієнтованої на події, подібною до клієнтів Windows, і, отже, його програми чекають введення користувачем для його виконання.

Windows Forms подібний до бібліотеки Microsoft Foundation Class (MFC) при розробці клієнтських додатків. Він пропонує обгортку, що складається з набору класів C ++ для розробки програм Windows. Однак він не забезпечує рамки програми за замовчуванням, як MFC.

Кожен елемент керування в додатку Windows Forms є конкретним екземпляром класу. Макетом керування в графічному інтерфейсі та його поведінкою керують за допомогою методів та аксесуарів. Windows Forms надає різноманітні елементи керування, такі як веб–сторінки, текстові поля та кнопки, а також варіанти створення спеціальних елементів керування. Він також містить класи для створення пензлів, шрифтів, піктограм та інших графічних об'єктів (наприклад, лінії та кола).

Windows Forms Designer – це інструмент у Visual Studio.NET, який використовується для вставлення елементів керування у форму та упорядкування їх відповідно до потрібного макета з можливістю додавання коду для обробки подій, що реалізують взаємодію користувачів. Табличні дані, які пов'язані з XML, базою даних тощо, можуть відображатися за допомогою керування DataGrid View у вигляді рядків та комірок.

Налаштування програми – це ще одна особливість Windows Forms для створення, зберігання та підтримки інформації про стан виконання у формі XML, яка може бути використана для отримання бажаних налаштувань користувача, таких як позиції панелі інструментів та останні використовувані списки. Ці параметри можна повторно використовувати в майбутньому додатку.

Деякі найкращі практики створення програм Windows Forms включають:

Класи Windows Forms можна розширити, використовуючи успадкування, щоб створити рамку програми, яка може забезпечити високий рівень абстрагування та використання коду.

Форми повинні бути компактними, при цьому елементи керування обмежені розміром, який може запропонувати мінімальну функціональність. Крім того, створення та вилучення елементів керування динамічно може зменшити кількість статичних елементів керування.

Форми можуть бути розбиті на частини, упаковані у склади, які можуть автоматично оновлюватись і з ними можна легко управляти з мінімальними зусиллями.

Проектування програми без стану забезпечує масштабність та гнучкість із легкістю для налагодження та обслуговування.

Програми Windows Forms повинні бути розроблені виходячи з необхідного рівня довіри, потреби запитувати дозволи та обробляти винятки з безпеки, де це необхідно.

Форму Windows не можна передавати через межі домену додатків, оскільки вони не розроблені для того, щоб їх можна було перемістити через домени додатків.

Система презентації Windows (WPF) – це найновіша технологія візуалізації користувальницьких інтерфейсів у програмах Windows GUI з такими функціями, як підтримка 2D / 3D, інтерактивна візуалізація даних та читаність вмісту. Він покладається на DirectX, а не на GDI (графічний інтерфейс пристрою) для надання моделі програмування, де інтерфейс користувача відокремлений від бізнес–логіки. Однак, маючи можливість взаємодіяти з WPF (де це потрібно),

Windows Forms є хорошим вибором для програм, які не потребують графічного інтерфейсу та інших функцій WPF, таких як шаблони даних / керування, типографічні та функції передачі тексту.[4][8][9]

2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ

2.1. Вимоги до програмного продукту

Вимоги до бази даних:

1) потрібно розробити базу даних, яка зможе повністю задовольнити всі потреби користувача під час роботи з програмою;
2) база даних повинна бути організована таким чином, щоб наступні вдосконалення не викликали глобальних змін в її структурі;
3) база даних повинна мати можливість оновлення, поповнення і розширення;
4) потрібно забезпечити високу надійність зберігання даних;
5) дані повинні зберігатися таким чином, щоб існувала можливість обміну ними;
6) база даних повинна видавати повну і вірогідну інформацію на запити;
7) база даних повинна мати засоби, які будуть забезпечувати її захист від несанкціонованого доступу.

Вимоги до інтерфейсу програми:

1) інтерфейс повинен забезпечувати можливість перегляду списку всіх наявних пацієнтів, медичних карт, лікарів, відділень, палат, які містяться в базі даних;
2) необхідна реалізація пошуку пацієнтів, лікарів і відділень за певними характеристиками, зокрема за ПІБ та назвою відділення;
3) необхідна реалізація видалення пацієнтів, лікарів, відділення, палати, медичних карток;
4) оскільки інформацію про пацієнтів в базу даних будуть вносити лікарі, необхідно створити систему для авторизації лікарів.

2.2. Вхідна і вихідна інформація

Вхідна інформація складається з призначених для користувача даних, введених з клавіатури та занесених в базу даних. В основному вона являє собою

інформацію про пацієнтів, лікарів, відділення, обстеження, призначення та госпіталізацію.

Вхідна інформація для пацієнтів:

- прізвище, імя та по–батькові;
- стать;
- дата народження;
- місто, адреса;
- контактні дані(номер телефону);
- номер палати, номер ліжка;
- додаткова інформація(діагноз, вага, зріст, пульс, тиск, група крові, інвалідність);
- прізвище, імя та по–батькові лікаря.

Вхідна інформація для лікарів:

- прізвище, імя та по–батькові;
- місто, адреса;
- контактні дані(номер телефону);
- назва відділення;
- посада;
- спеціалізація;
- категорія;
- логін та пароль.

Вхідна інформація для відділень:

- назва відділення;
- номер телефону.

Вхідна інформація для бази ліків:

- назва;
- форма випуску;
- одиниця виміру;
- виробник.

Вихідна інформація являє собою сукупність структурованих даних, які були введені користувачем. В інтерфейсі користувача вся інформація виводиться у зручному для читання вигляді. Цю інформацію можна знайти за певними параметрами та видалити неактуальні дані за необхідністю.

2.3. Логічна модель даних

Логічна модель даних даної предметної області складається з таких сутностей:

- Пацієнти (Patients). Ця сутність включає основну інформацію про пацієнта – це ПІБ, стать, дата народження та діагноз. Утворює з сутностями «Додаткова інформація», «Госпіталізація», «Обстеження», «Призначення» зв’язки «один–до–багатьох».
- Додаткова інформація (Additional information). Ця сутність містить додаткову інформацію про пацієнтів – це вага, зріст, група крові, тиск, пульс, інвалідність.
- Палати (Chambers). Ця сутність містить всю інформацію про палати – це номер палати, номер ліжка, кількість ліжок. Утворює з сутністю «Пацієнти» зв’язок «один–до–одного».
- Госпіталізація (Hospitalization). Ця сутність містить всю інформацію про госпіталізацію пацієнтів – це дата госпіталізації, дата виписки, статус госпіталізації.
- Обстеження (Examination). Ця сутність містить всю інформацію про обстеження пацієнтів – це назва обстеження, дата обстеження, результат.
- Призначення (Appointment). Ця сутність містить всю інформацію про призначення для пацієнтів – це назва призначення, дата, кількість та тривалість прийому назначеного препарату.
- Контакти (Contacts). Ця сутність включає всю контактну інформацію пацієнтів та лікарів – це sybjectIs (пацієнт чи лікар), місто, адреса, номер телефону. Утворює з сутностями «Пацієнти» та «Лікарі» зв’язок «один–до–багатьох».

- Лікарі (Doctors). Ця сутність включає всю інформацію про лікарів – це ПІБ, логін та пароль. Утворює з сутністю «Пацієнти» зв’язок «один–до–багатьох».
- Відділення(Departments). Ця сутність включає в себе всю інформація про відділення – це назва відділення та номер телефону. Утворює з сутністю «Лікарі» зв’язок «один–до–багатьох».
- Посади(Posts). Ця сутність включає в себе всю інформація про посади – це назва посади, спеціалізація, категорія. Утворює з сутністю «Лікарі» зв’язок «один–до–багатьох».
- База ліків(Medicament). Ця сутність включає в себе всю інформацію про ліки – це назва, форма випуску, одиниця виміру, виробник. Утворює з сутністю «Лікарі» зв’язок «один–до–багатьох».

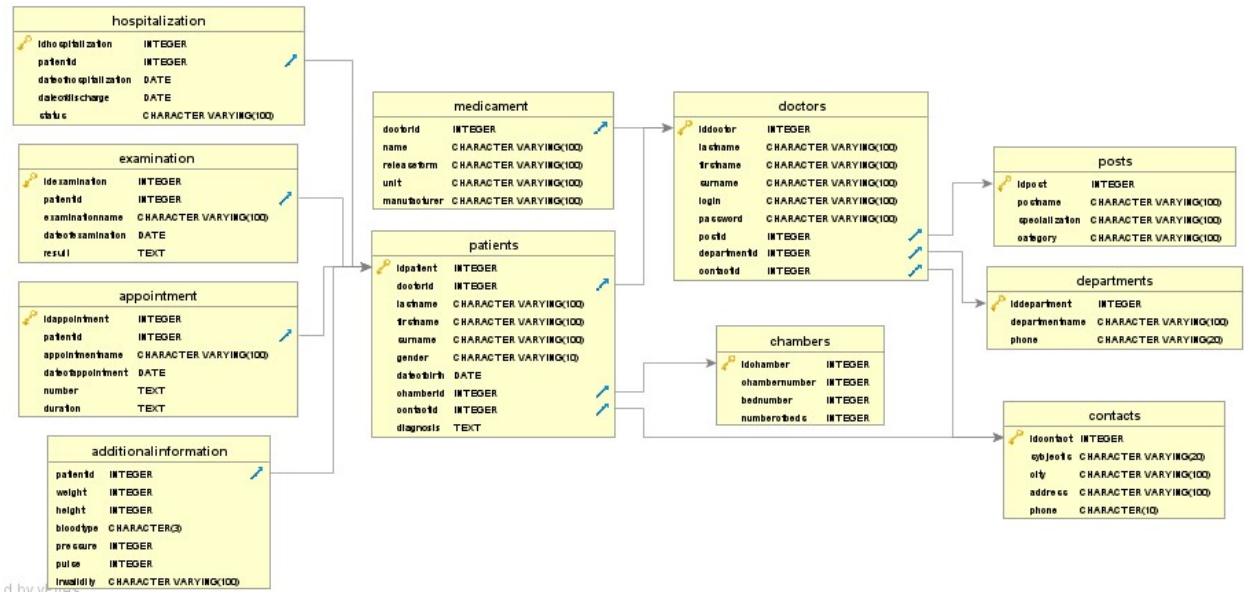

Рисунок 2.1 – Діаграма «Сутність–зв’язок»

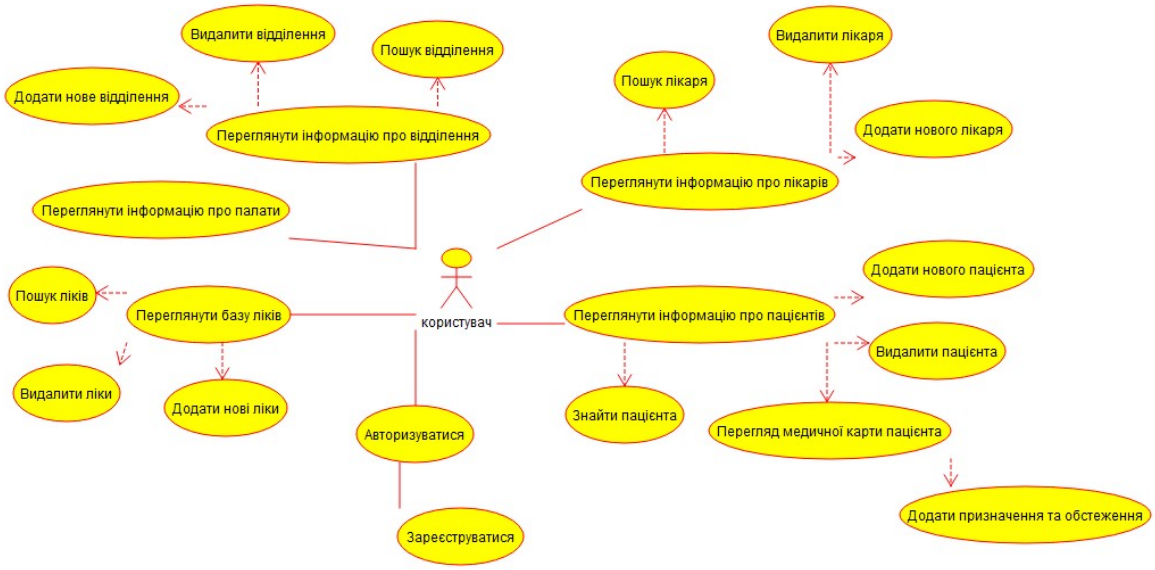

Рисунок 2.2 – Діаграма випадків застосування

Рисунок 2.3 – Діаграма класів

2.4. Реалізація таблиць

Зв’язки, розроблені на стадії формування логічної моделі даних, перетворюються в наступні таблиці:

- Patients;
- Additional information;
- Chambers;

- Hospitalization;
- Examination;
- Appointment;
- Contacts;
- Doctors;
- Departments;
- Posts;
- Medicament.

2.4.1. Таблиця «Patients»

Дана таблиця містить основну інформацію про пацієнта (рис. 2.4).

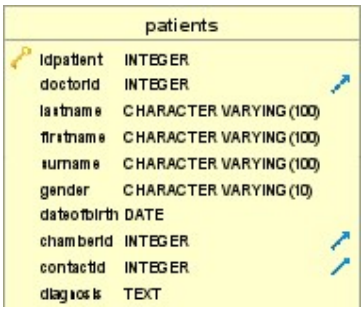

Рисунок 2.4 – Таблиця «Patients»

Таблиця містить наступні поля:

- idpatient – INT – первинний ключ;
- doctorid – INT – числове значення, вторинний ключ до таблиці «Doctors», що містить в собі всю інформацію про лікарів, та буде описана нижче;
- lastname – CHARACTER VARYING (100) – стрічкове значення, що містить в собі прізвище пацієнта;
- firstname – CHARACTER VARYING (100) – стрічкове значення, що містить в собі ім’я пацієнта;
- surname – CHARACTER VARYING (100) – стрічкове значення, що містить в собі по–батькові пацієнта;
- gender – CHARACTER VARYING (10) – стрічкове значення, що містить в собі стать пацієнта;

- dateofbirth – DATE – дата народження пацієнта;
- chamberid – INT – числове значення, вторинний ключ до таблиці «Chambers», що містить в собі всю інформацію про палати, та буде описана нижче;
- contactid – INT – числове значення, вторинний ключ до таблиці «Contacts», що містить в собі всю контактну інформацію пацієнтів та лікарів, та буде описана нижче;
- diagnosis – TEXT – текстове значення, що містить в собі діагноз пацієнта.

2.4.2. Таблиця «Additionalinformation»

Дана таблиця містить додаткову інформацію про пацієнта (рис. 2.5).

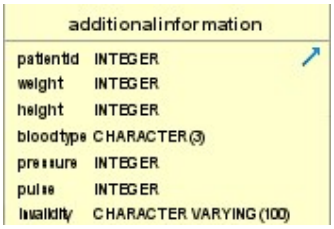

Рисунок 2.5 – Таблиця «Additionalinformation»

Таблиця містить наступні поля:

- patientid – INT – числове значення, вторинний ключ до таблиці «Patients», що описана вище;
- weight – INT – числове значення, що містить в собі значення ваги;
- height – INT – числове значення, що містить в собі значення зросту;
- bloodtype – CHARACTER VARYING (3) – стрічкове значення, що містить в собі групу крові;
- pressure – INT – числове значення, що містить в собі значення тиску;
- pulse – INT – числове значення, що містить в собі значення пульсу;
- invalidity – CHARACTER VARYING (100) – стрічкове значення, що містить в собі значення інвалідності.

2.4.3. Таблиця «Chambers»

Дана таблиця містить всю інформацію про палати (рис. 2.6).

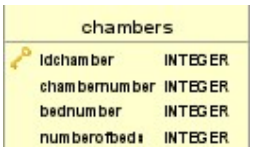

Рисунок 2.6 – Таблиця «Chambers»

Таблиця містить наступні поля:

- idchamber – INT – первинний ключ;
- chambernumber – INT – числове значення, що містить в собі номер палати;
- bednumber – INT – числове значення, що містить в собі номер ліжка;
- numberofbeds – INT – числове значення, що містить в собі кількість ліжок в палаті.

2.4.4. Таблиця «Hospitalization»

Дана таблиця містить всю інформацію про госпіталізацію пацієнта (рис. 2.7).

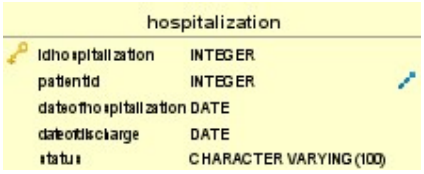

Рисунок 2.7 – Таблиця «Hospitalization»

Таблиця містить наступні поля:

- idhospitalization – INT – первинний ключ;
- patientid – INT – числове значення, вторинний ключ до таблиці «Patients», що описана вище;
- dateofhospitalization – DATE – дата госпіталізації;
- dateofdischarge – DATE – дата виписки;
- status – CHARACTER VARYING (100) – стрічкове значення, що містить в собі статус госпіталізації («в стаціонарі» чи «виписаний»).

2.4.5. Таблиця «Examination»

Дана таблиця містить всю інформацію про обстеження пацієнта (рис. 2.8).

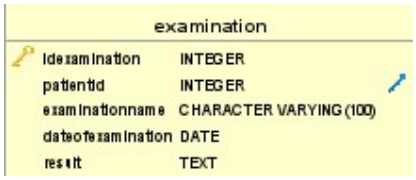

Рисунок 2.8 – Таблиця «Examination»

Таблиця містить наступні поля:

- idexamination – INT – первинний ключ;
- patientid – INT – числове значення, вторинний ключ до таблиці «Patients», що описана вище;
- examinationname – CHARACTER VARYING (100) – стрічкове значення, що містить в собі назву обстеження;
- dateofexamination – DATE – дата обстеження;
- result – TEXT – текстове значення, що містить в собі результат обстеження.

2.4.6. Таблиця «Appointment»

Дана таблиця містить всю інформацію про призначення (рис. 2.9).

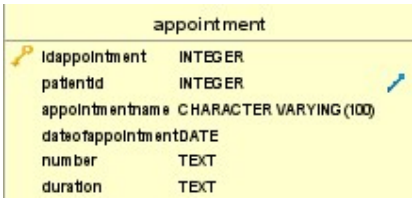

Рисунок 2.9 – Таблиця «Appointment»

Таблиця містить наступні поля:

- idappointment – INT – первинний ключ;
- patientid – INT – числове значення, вторинний ключ до таблиці «Patients», що описана вище;
- appointmentname – CHARACTER VARYING (100) – стрічкове значення, що містить в собі назву призначення;
- dateofappointment – DATE – дата призначення;
- number – TEXT – текстове значення, що містить в собі призначення лікаря;

- duration – TEXT – текстове значення, що містить в собі тривалість прийняття певного препарату.

2.4.7. Таблиця «Contacts»

Дана таблиця містить всю контактну інформацію пацієнтів та лікарів (рис. 2.10).

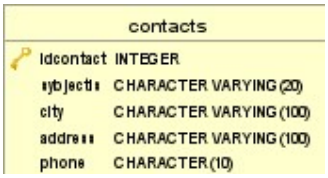

Рисунок 2.10 – Таблиця «Contacts»

Таблиця містить наступні поля:

- idcontact – INT – первинний ключ;
- sybjectis – CHARACTER VARYING (20) – стрічкове значення, що містить в собі визначення об’єкта («пацієнт» або «лікар»);
- city – CHARACTER VARYING (100) – стрічкове значення, що містить в собі назву міста;
- address – CHARACTER VARYING (100) – стрічкове значення, що містить в собі адресу;
- phone – CHARACTER VARYING (10) – стрічкове значення, що містить в собі номер телефону.

2.4.8. Таблиця «Doctors»

Дана таблиця містить всю інформацію про лікарів (рис. 2.11).

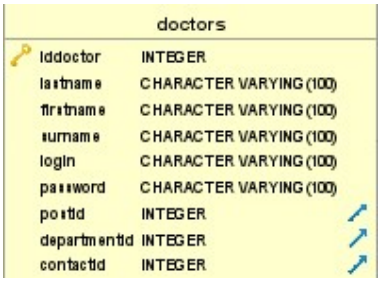

Рисунок 2.11 – Таблиця «Doctors»

Таблиця містить наступні поля:

- iddoctor – INT – первинний ключ;
- lastname – CHARACTER VARYING (100) – стрічкове значення, що містить в собі прізвище лікаря;
- firstname – CHARACTER VARYING (100) – стрічкове значення, що містить в собі ім’я лікаря;
- surname – CHARACTER VARYING (100) – стрічкове значення, що містить в собі по–батькові лікаря;
- login – CHARACTER VARYING (100) – стрічкове значення, що містить в собі логін;
- password – CHARACTER VARYING (100) – стрічкове значення, що містить в собі пароль;
- postid – INT – числове значення, вторинний ключ до таблиці «Posts», що містить в собі всю інформацію про посади, та буде описана нижче;
- departmentid – INT – числове значення, вторинний ключ до таблиці «Departments», що містить в собі всю інформацію про відділення, та буде описана нижче;
- contactid – INT – числове значення, вторинний ключ до таблиці «Contacts», що містить в собі всю контактну інформацію пацієнтів та лікарів, та описана вище.

2.4.9. Таблиця «Departments»

Дана таблиця містить всю інформацію про відділення (рис. 2.12).

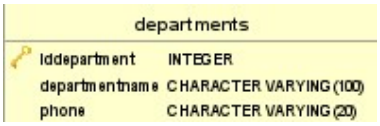

Рисунок 2.12 – Таблиця «Departments»

Таблиця містить наступні поля:

- iddepartment – INT – первинний ключ;
- departmentname – CHARACTER VARYING (100) – стрічкове значення, що містить в собі назву відділення;

- phone – CHARACTER VARYING (20) – стрічкове значення, що містить в собі номер телефону.

2.4.10.Таблиця «Posts»

Дана таблиця містить всю інформацію про посади (рис. 2.13).

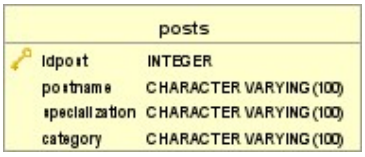

Рисунок 2.13 – Таблиця «Posts»

Таблиця містить наступні поля:

- idpost – INT – первинний ключ;
- postname – CHARACTER VARYING (100) – стрічкове значення, що містить в собі назву посади;
- specialization – CHARACTER VARYING (100) – стрічкове значення, що містить в собі спеціалізацію;
- category – CHARACTER VARYING (100) – стрічкове значення, що містить в собі категорію.

2.4.11.Таблиця «Medicament»

Дана таблиця містить всю інформацію про посади (рис. 2.14).

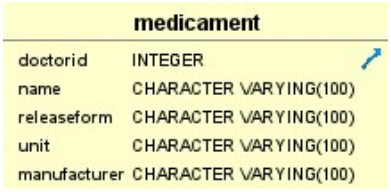

Рисунок 2.14 – Таблиця «Medicament»

Таблиця містить наступні поля:

- doctorid – INT – числове значення, вторинний ключ до таблиці «Doctors», що містить в собі всю інформацію про лікарів, та описана вище;
- name – CHARACTER VARYING (100) – стрічкове значення, що містить в собі назву ліків;

- releaseform– CHARACTER VARYING (100) – стрічкове значення, що містить в собі форму випуску ліків;
- unit – CHARACTER VARYING (100) – стрічкове значення, що містить в собі одиницю виміру;
- manufacturer – CHARACTER VARYING (100) – стрічкове значення, що містить в собі виробника ліків.

У базі необхідний контроль за введеними даними. Тому для деяких полів таблиць були введені певні обмеження. У таблиці Contacts для поля Phone виконується перевірка номера телефону за введеним шаблоном та в таблиці Patients виконується перевірка дати народження, а саме перевіряється чи коректно введена дата.

2.5. Нормалізація таблиць

Нормалізація – це спосіб організації даних у базі. Нормалізація розділяє великі таблиці на менші таблиці та пов'язує їх за допомогою зв’язків. Метою нормалізації є усунення зайвих (марних) даних та забезпечення логічного зберігання даних.

Винахідник реляційної моделі Едгар Кодд запропонував теорію нормалізації з введенням першої нормальної форми, і він продовжував розширювати теорію з другою та третьою нормальними формами. Пізніше він приєднався до Реймонда Ф. Бойса, щоб розробити теорію нормальної форми Бойса–Кодда.

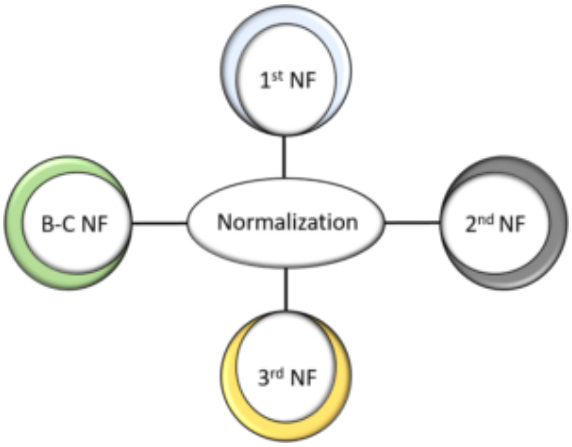

Рисунок 2.12 – Нормалізація бази даних (чотири нормальні форми)

2.5.1. Перша нормальна форма (1NF)

- Дані зберігаються в таблицях із рядками, однозначно ідентифікованими первинним ключем
- Дані всередині кожної таблиці зберігаються в окремих стовпцях у найбільш скороченому вигляді
- Немає повторюваних груп
- Таблиця повинна мати лише окремі (атомарні) цінні атрибути / стовпці
- Значення, що зберігаються у стовпці, повинні бути одного домену
- Усі стовпці таблиці повинні мати унікальні назви.
- І порядок збереження даних не має значення.

У першій нормальній формі ми вирішуємо проблему атомарності. Одна комірка не може містити декілька значень. Якщо таблиця містить складений або багатозначний атрибут, вона порушує першу звичайну форму.

2.5.2. Друга нормальна форма (2NF)

- Все від 1NF
- У кожній таблиці зберігаються лише дані, які стосуються первинного ключа таблиці
- Таблиця не повинна мати часткової залежності.

Перша умова 2–ої нормальної форми полягає в тому, що таблиця повинна бути в 1–ій нормальній формі. Таблиця також не повинна містити часткової залежності. Тут часткова залежність означає, що залежна підмножина ключа–кандидата визначає атрибут, який не є простим. Жоден непростий атрибут не залежить від залежної підмножини будь–якого ключового ключа таблиці.

2.5.3. Третя нормальна форма (3NF)

- Все від 2NF
- Між стовпцями в кожній таблиці немає залежностей між таблицями.

Таблиця повинна бути в 2–ій нормальній формі, перш ніж перейти до 3–ьої нормальної форми. Інша умова – не повинно бути перехідної залежності для

атрибутів, що не є простими. Це означає, що непрості атрибути (які не утворюють кандидатський ключ) не повинні залежати від інших непримітних атрибутів у даній таблиці. Отже, перехідна залежність – це функціональна залежність, при якій X → Z (X визначає Z) опосередковано в силу X → Y і Y → Z (де це не так, що Y → X).

2.5.4. Нормальна форма Бойса Кодда (BCNF)

Нормальна форма Бойса–Кодда, також відома як 3,5 NF. Його вища версія 3NF і була розроблена Реймоном Ф. Бойсом та Едгаром Ф. Коддом для вирішення певних типів аномалій, які не стосувалися 3NF. Перш ніж перейти до BCNF, таблиця повинна відповідати 3–й звичайній формі. У BCNF, якщо кожна функціональна залежність A → B, тоді A повинен бути супер ключем цієї конкретної таблиці.[7]

2.5.5. Переваги та недоліки нормалізації

Нормалізація надає численні переваги для бази даних. Основні переваги:

1) Більша загальна організація баз даних
2) Скорочення надмірних даних
3) Узгодженість даних у базі даних
4) Набагато гнучкіший дизайн бази даних
5) Краще керування безпекою баз даних
6) Менша база даних може підтримуватися, оскільки нормалізація усуває дублюючі дані. Загальний розмір бази даних зменшується в результаті.
7) Забезпечується краща ефективність роботи, яка може бути пов'язана з вищезгаданим моментом. Оскільки бази даних стають меншими за розміром, пропуск даних стає швидшим і коротшим, тим самим покращуючи час та швидкість реагування.
8) Більш вузькі таблиці можливі, оскільки нормалізовані таблиці будуть доопрацьовані та матимуть менші стовпці, що дозволяє робити більше записів даних на сторінку.

9) Менше індексів на таблицю забезпечує швидше завдання технічного обслуговування (відновлення індексу).Також реалізує можливість з'єднання лише необхідних таблиць.

Процес нормалізації полегшує роботу кожного, від користувача, який звертається до таблиць, до адміністратора бази даних, який відповідає за загальне управління кожним об'єктом в базі даних. Зменшується надмірність даних, що спрощує структури даних та економить простір на диску. Оскільки дублюючі дані зведені до мінімуму, можливість суперечливих даних значно зменшується. Оскільки база даних була нормалізована і розбита на менші таблиці, надається більша гнучкість щодо зміни існуючих структур.

Набагато простіше змінити невелику таблицю з невеликою кількістю даних, ніж змінити одну велику таблицю, яка містить всі важливі дані в базі даних. Нарешті, забезпечується також безпека в тому сенсі, що адміністратор може надати доступ до обмежених таблиць певним користувачам. Безпеку легше контролювати, коли відбулася нормалізація.

Основні недоліки нормалізації:

1) Таблиці будуть містити коди, а не реальні дані, оскільки повторні дані зберігатимуться як рядки кодів, а не справжні дані. Тому завжди є необхідність перейти до таблиці пошуку.
2) Моделі даних стає надзвичайно важко запитувати, оскільки оптимізована модель даних для додатків, а не для спеціальних запитів. (Спеціальний запит – це запит, який неможливо визначити до видачі запиту. Він складається з SQL, який будується динамічно і зазвичай будується за допомогою настільних інструментів запиту.). Отже, складно моделювати базу даних, не знаючи, чого бажає замовник.
3) У міру прогресування нормальної форми продуктивність стає повільнішою і повільнішою.
4) Для ефективного виконання процесу нормалізації необхідні належні знання про різні нормальні форми. Необережне використання може призвести до

жахливого дизайну, наповненого великими аномаліями та невідповідністю даних.

Хоча більшість успішних баз даних певною мірою нормалізуються, є один істотний недолік нормалізованої бази даних: зниження продуктивності бази даних. Прийняття зниженої продуктивності вимагає усвідомлення того, що коли запит або запит на транзакцію надсилається до бази даних, задіяні такі фактори, як використання процесора, використання пам'яті та введення / виведення. Щоб зробити короткий опис, для нормалізованої бази даних потрібно набагато більше процесора, пам'яті та вводу / виводу для обробки транзакцій та запитів баз даних, ніж денормалізована база даних.

Нормалізована база даних повинна знаходити запитувані таблиці, а потім з'єднувати дані з таблиць, щоб отримати або отримати запитувану інформацію, або обробити потрібні дані. [5]

2.6. Мова SQL

SQL (Structured Query Language) – це стандартизована мова програмування, яка використовується для управління реляційними базами даних та виконання різних операцій над даними в них. Спочатку створений у 1970–х роках, SQL регулярно використовується не тільки адміністраторами баз даних, але також розробниками, які пишуть сценарії інтеграції даних та аналітиками даних, які прагнуть налаштувати та запустити аналітичні запити.

Використання SQL включає зміни таблиць баз даних та індексних структур, додавання, оновлення та видалення рядків даних, отримання підмножини інформації з бази даних для обробки транзакцій та програм аналітики. Запити та інші операції SQL мають форму команд, записаних у вигляді запитів – загальновживані оператори SQL включають вибір, додавання, вставлення, оновлення, видалення, створення та зміни.

SQL став фактично стандартною мовою програмування для реляційних баз даних після їх появи в кінці 1970–х на початку 1980–х років. Реляційні системи, також відомі як бази даних SQL, містять набір таблиць, що містять дані в рядках і

стовпцях. Кожен стовпець у таблиці відповідає категорії даних при цьому кожен рядок містить значення даних для стовпця, що перетинається.

2.6.1. Історія SQL

У 1970–х роках, коли в лабораторіях IBM було створено нове програмне забезпечення баз даних – System R. І для управління даними, що зберігаються в System R, була створена мова SQL. Спочатку його називали SEQUEL – ім'я, яке досі використовується як альтернативна вимова для SQL, але пізніше було перейменовано на просто SQL.

У 1979 році компанія під назвою Relational Software, яка згодом стала Oracle, побачила комерційний потенціал SQL і випустила власну модифіковану версію, названу Oracle V2.

Тепер, на третьому десятилітті свого існування, SQL пропонує велику гнучкість для користувачів, підтримуючи розподілені бази даних, тобто бази даних, які можуть працювати в декількох комп'ютерних мережах одночасно. Сертифікований ANSI та ISO, SQL став стандартом мови запитів до бази даних, що лежить в основі різноманітних добре встановлених додатків баз даних в Інтернеті сьогодні. Він обслуговує як галузеві, так і академічні потреби та використовується як на окремих комп'ютерах, так і на корпоративних серверах. З розвитком технології баз даних програми на базі SQL стають все доступнішими для звичайного користувача. Це пов’язано з впровадженням різних баз даних SQL з відкритим кодом, таких як MySQL , PostgreSQL , SQLite, Firebird та багато іншого.

2.6.2. Стандарт SQL

Standard Стандарт SQL протягом багатьох років зазнав багато змін, які додали до стандарту багато нових функціональних можливостей, таких як підтримка XML, тригери, регулярне узгодження виразів, рекурсивні запити, стандартизовані послідовності та багато іншого. Завдяки великому обсягу SQL Standard, багато баз даних на базі даних, такі як MySQL або PostgreSQL, не реалізують весь стандарт.

У багатьох випадках поведінка бази даних для зберігання файлів або індексів недостатньо чітко визначена, і вирішити, як буде вести себе база даних, належить виробникам різних реалізацій SQL. Це причина, чому, хоча всі реалізації SQL мають однакову базу, вони рідко сумісні.

2.6.3. Елементи мови SQL

Мова SQL заснована на кількох елементах. Для зручності розробників SQL всі необхідні мовні команди у відповідних системах управління базами даних зазвичай виконуються через певний інтерфейс командного рядка SQL (CLI).

Статті – пункти є компонентами висловлювань та запитів.

Вирази – вирази можуть створювати скалярні значення або таблиці, які складаються з стовпців і рядків даних.

Предикати – вони задають умови, які використовуються для обмеження ефектів операторів та запитів або для зміни потоку програми.

Запити – запит отримає дані на основі заданих критеріїв.

Виписки – за допомогою операторів можна контролювати транзакції, потік програми, з'єднання, сеанси чи діагностику. У системах баз даних оператори SQL використовуються для надсилання запитів із клієнтської програми на сервер, де зберігаються бази даних. У відповідь сервер обробляє оператори SQL і повертає відповіді до клієнтської програми. Це дозволяє користувачам виконувати широкий спектр дивовижно швидких операцій по маніпулюванню даних від простих введення даних до складних запитів.

2.6.4. SQL запити

SQL–запити – це найпоширеніші та найважливіші операції SQL. За допомогою SQL–запиту можна шукати в базі даних необхідну інформацію. SQL–запити виконуються за допомогою оператора "SELECT". SQL–запит може бути більш конкретним за допомогою декількох пунктів:

- FROM – вказує таблицю, в якій буде здійснено пошук.

- WHERE – використовується для визначення рядків, в яких буде здійснюватися пошук. Усі рядки, для яких пункт WHERE не відповідає дійсності, будуть виключені.
- ORDER BY – це єдиний спосіб сортування результатів у SQL. В іншому випадку вони будуть повернуті у випадковому порядку [6].

Приклади реальних запитів наведені нижче на рисунках:

1) SELECT lastname, firstname, surname, gender, dateofbirth, diagnosis FROM patients WHERE gender = «чоловік» ORDER BY idpatient;

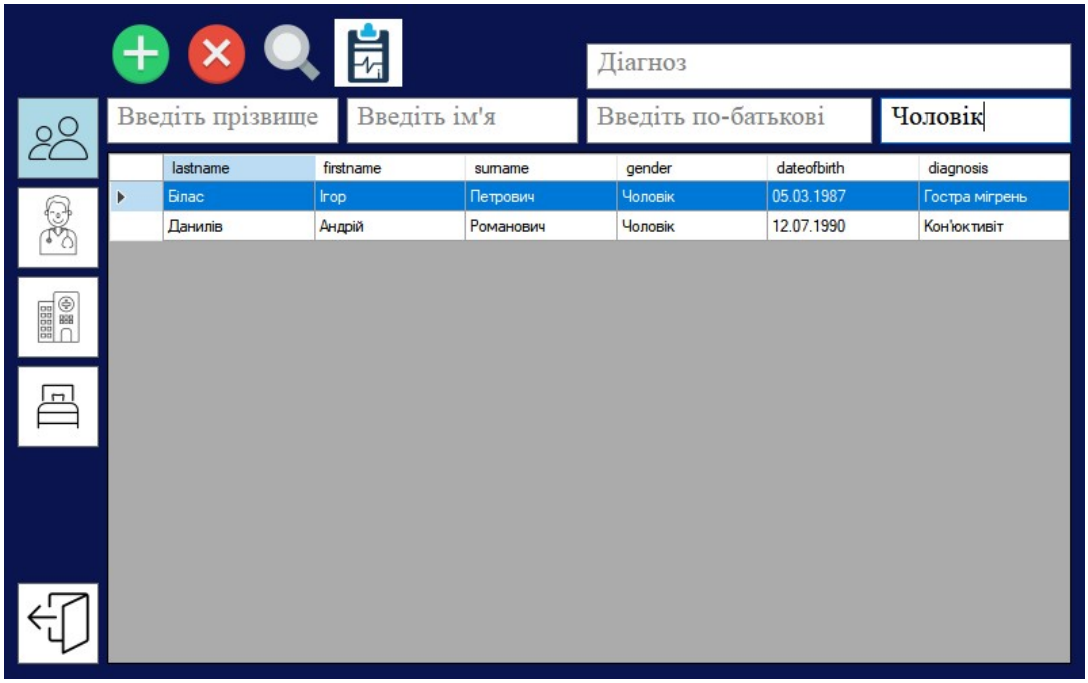

Рисунок 2.13 – Пошук пацієнтів за статтю «чоловік»

2) SELECT appointmentname, dateofappointment, number, duration FROM appointment JOIN patients ON appointment.patientid = patients.idpatient WHERE lastname = «Білас» and firstname = «Ігор» and surname = «Петрович»;

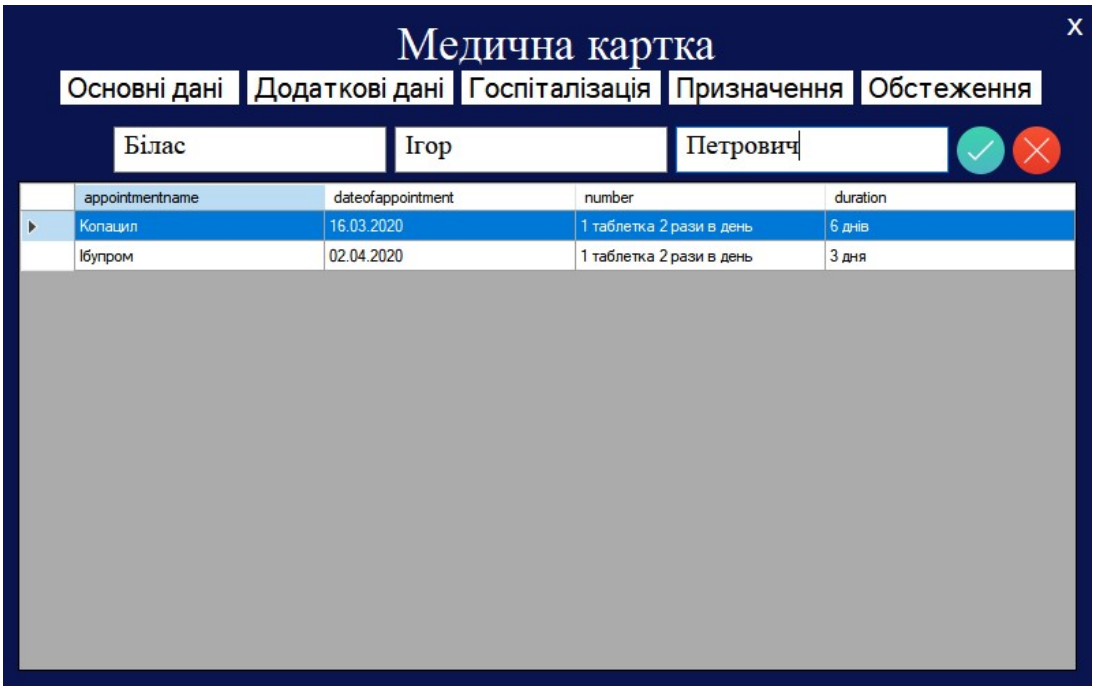

Рисунок 2.14 – Пошук призначень для певного пацієнта за прізвищем, ім’ям та по–батькові «Білас Ігор Петрович»

3) SELECT lastname, firstname, surname, postname, specialization, category FROM doctors INNER JOIN posts ON doctors.postid = posts.idpost WHERE category= «Вища» ORDER BY lastname;

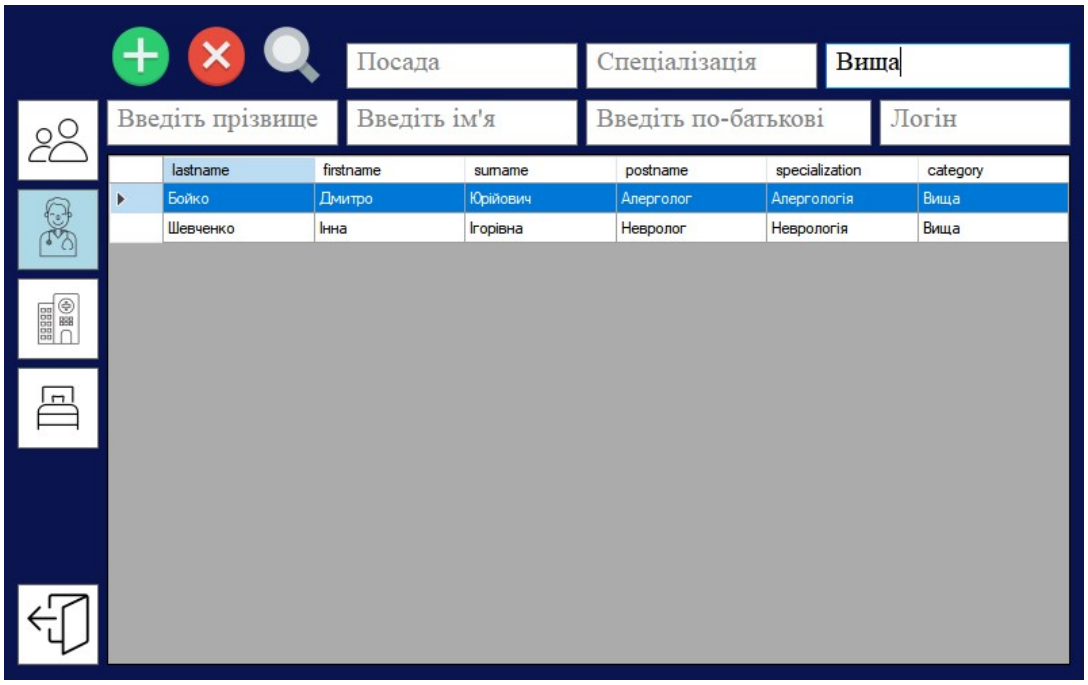

Рисунок 2.15 – Пошук лікарів за категорією «Вища»

4) SELECT departmentname, phone FROM departments WHERE departmentname= «Алергологія» ORDER BY departmentname;

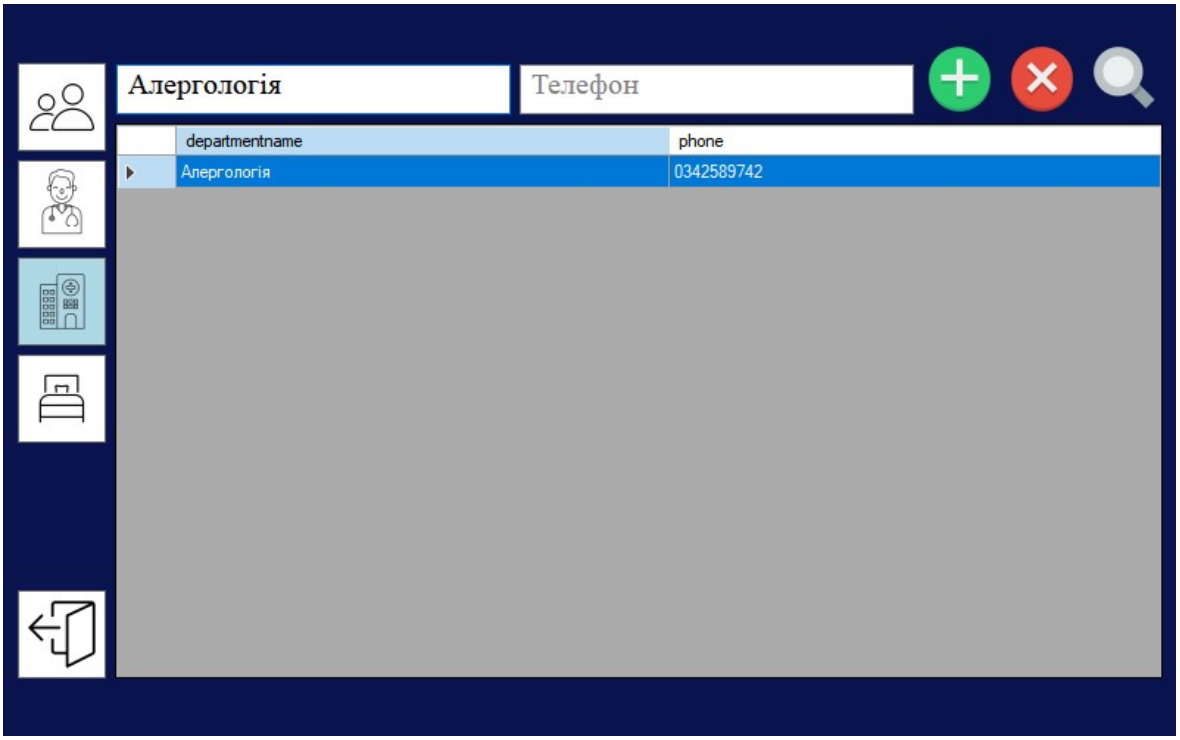

Рисунок 2.16 – Пошук відділень за назвою відділення «Алергологія»

3. РОЗРОБКА ІНТЕРФЕЙСУ КОРИСТУВАЧА

Метою розробки графічного інтерфейсу користувача є створення ефективного засобу взаємодії між людиною та системою. Інтерфейс повинен підтримувати увесь наявний функціонал серверної частини, надавати доступ до перегляду усієї інформації, що зберігається в базі даних, а також можливість редагування та доповнення даної інформації.

Для реалізації клієнтської частини було обрано технологію Windows Forms в Visual Studio. Було створено 12 форм:

- LoginForm;
- RegisterForm;
- MainForm;
- PatientForm;
- InsertForm;
- MedicalCardForm;
- AppointmentForm;
- DoctorForm;
- DepartmentForm;
- ChambersForm;
- InsertMedicamentsForm;
- MedicamentForm.

3.1. Форма «LoginForm»

LoginForm – це форма авторизації користувача, в нашому випадку лікаря (рис. 3.1). Для того, щоб увійти, потрібно ввести логін та пароль у відповідні поля, якщо користувач не зареєстрований, то при натисканні на відповідний надпис під кнопкою ввійти, можна перейти на форму реєстрації, яка буде описана нижче. При введені потрібних даних і натисканні кнопки «Ввійти», програма переходить на головну форму, яка буде описана нижче.

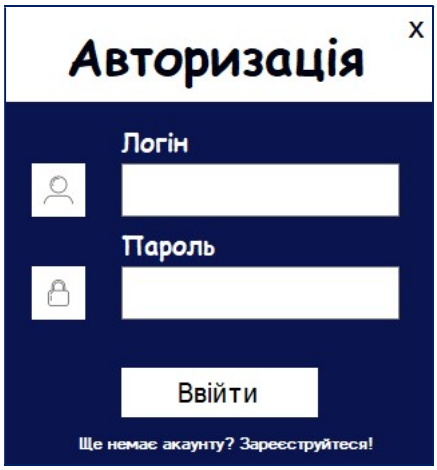

Рисунок 3.1 – Форма «LoginForm»

3.2. Форма «RegisterForm»

RegisterForm – це форма реєстрації користувача (лікаря) (рис. 3.2). Як вже було зазначено, щоб перейти на цю форму потрібно натиснути на відповідний надпис. Для того, щоб зареєструватися потрібно ввести необхідні дані і натиснути кнопку «Зареєструватися», щоб авторизуватися потрібно натиснути кнопку «Авторизуватися». Всі введені дані записуються в базу даних за допомогою процедури insert_data, описаної в додатках.

Рисунок 3.2 – Форма «RegisterForm»

3.3. Форма «MainForm»

MainForm – це головна форма, де можна вибрати куди потрібно перейти, натиснувши на відповідну кнопку (рис. 3.3). Відповідно можна перейти на 4 форми: «Пацієнти», «Лікарі», «Відділення» та «Палати», щоб вийти з програми потрібно натиснути відповідну кнопку внизу справа.

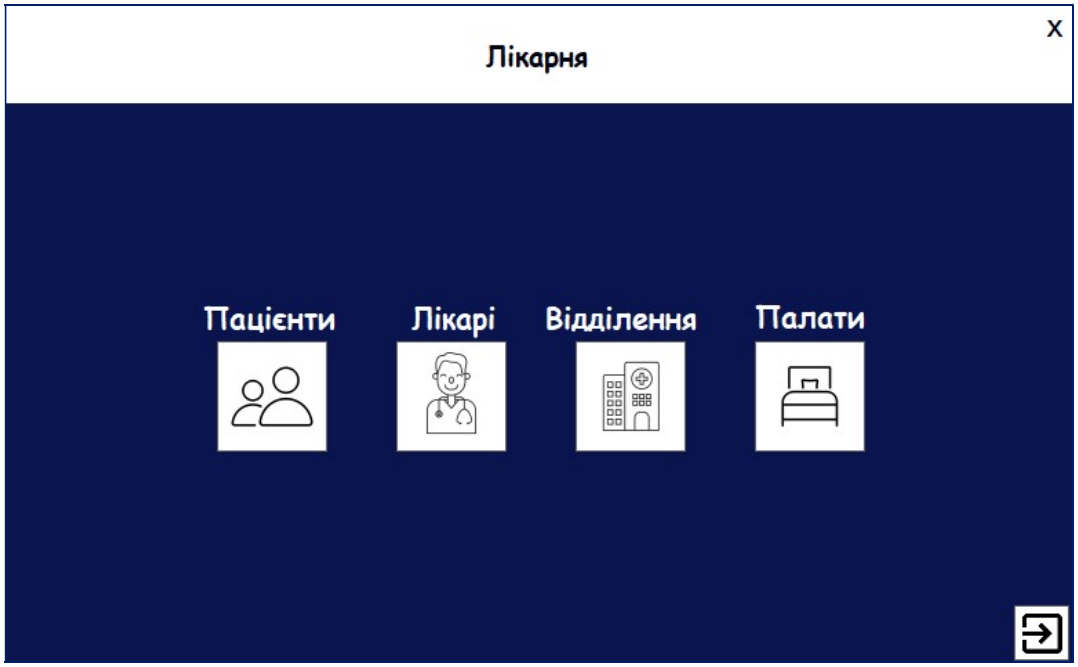

Рисунок 3.3 – Форма «MainForm»

3.4. Форма «PatientForm»

PatientForm – це форма, яка показує основну інформацію про пацієнтів (рис. 3.4). В цій формі є можливість перейти до інших форм, таких як «Лікарі», «Відділення» та «Палати», щоб вийти з програми потрібно натиснути відповідну кнопку внизу зліва.

Щоб додати нового пацієнта, потрібно натиснути на відповідну клавішу зверху форми. При натисканні цієї кнопки відкриється форма реєстрації нового пацієнта, яка буде описана нижче.

Щоб видалити всі записи про пацієнта, необхідно ввести дані в виділені поля – це ПІБ та діагноз. Пошук реалізовано за ПІБ та окремо за статтю, для цього потрібно ввести необхідні дані в виділені поля і натиснути відповідну кнопку зверху форми.

Щоб перейти до медичної карти пацієнта потрібно натиснути на потрібну кнопку зверху форми, програма перейде до форми медичної карти, яка буде описана нижче.

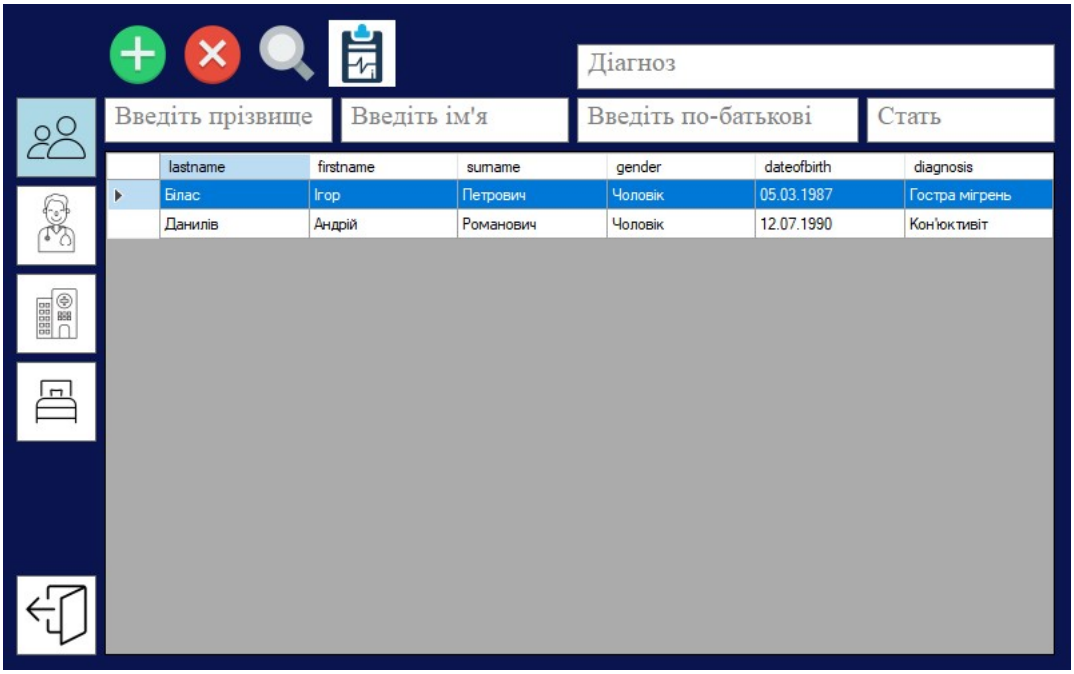

Рисунок 3.4 – Форма «PatientForm»

3.5. Форма «InsertForm»

InsertForm – це форма для реєстрації нових пацієнтів (рис. 3.5). Для того, щоб зареєструвати нового пацієнта потрібно ввести необхідні дані і натиснути кнопку «Готово».

Для того, щоб вийти з даної форми потрібно натиснути кнопку «Відмінити». Всі введені дані записуються в базу даних за допомогою процедури insert_p, описаної в додатках.

Рисунок 3.5 – Форма «InsertForm»

3.6. Форма «MedicalCardForm»

MedicalCardForm – це форма, в якій відображається вся інформація про пацієнта. Для того, щоб відобразити інформацію потрібно ввести ПІБ пацієнта і натиснути на відповідну кнопку. Після натискання кнопки відображається основна інформація про пацієнта – це ПІБ, стать, дата народження, діагноз, місто, адреса, номер телефону, номер палати та номер ліжка (рис. 3.6).

У формі можна також переглянути додаткові дані (рис. 3.7), інформацію про госпіталізацію пацієнта (рис. 3.8), призначення (рис. 3.9) та обстеження призначені лікарем (рис. 3.10). Для того, щоб відкрити необхідну інформацію потрібно натиснути відповідну кнопку.

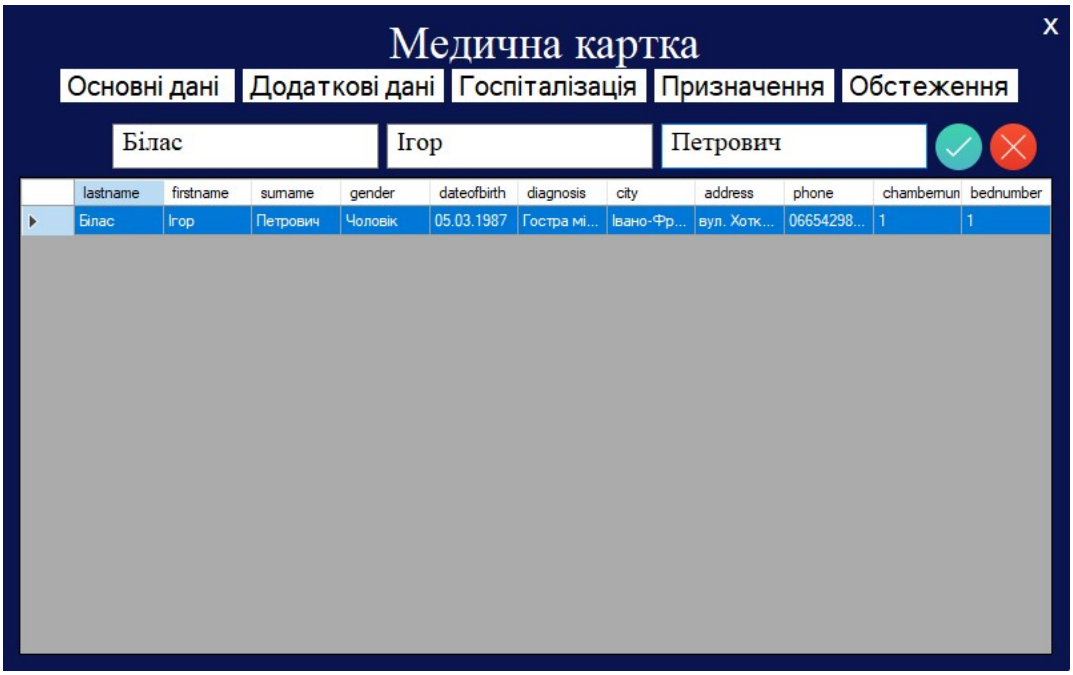

Рисунок 3.6 – Форма «MedicalCardForm» вкладка «Основні дані»

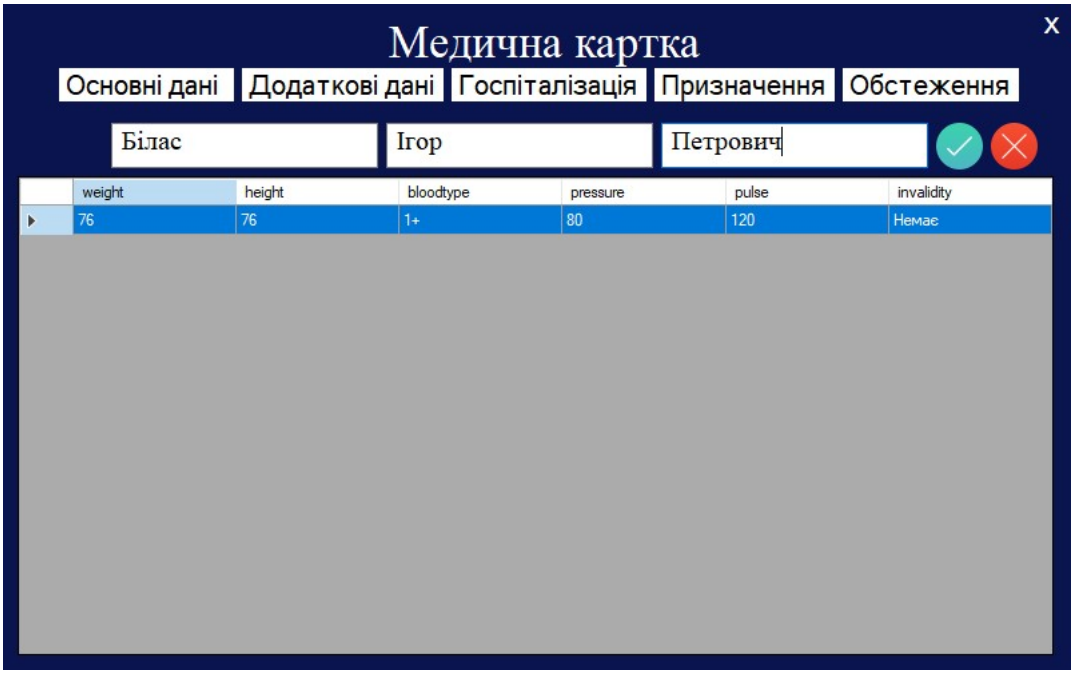

Рисунок 3.7 – Форма «MedicalCardForm» вкладка «Додаткові дані»

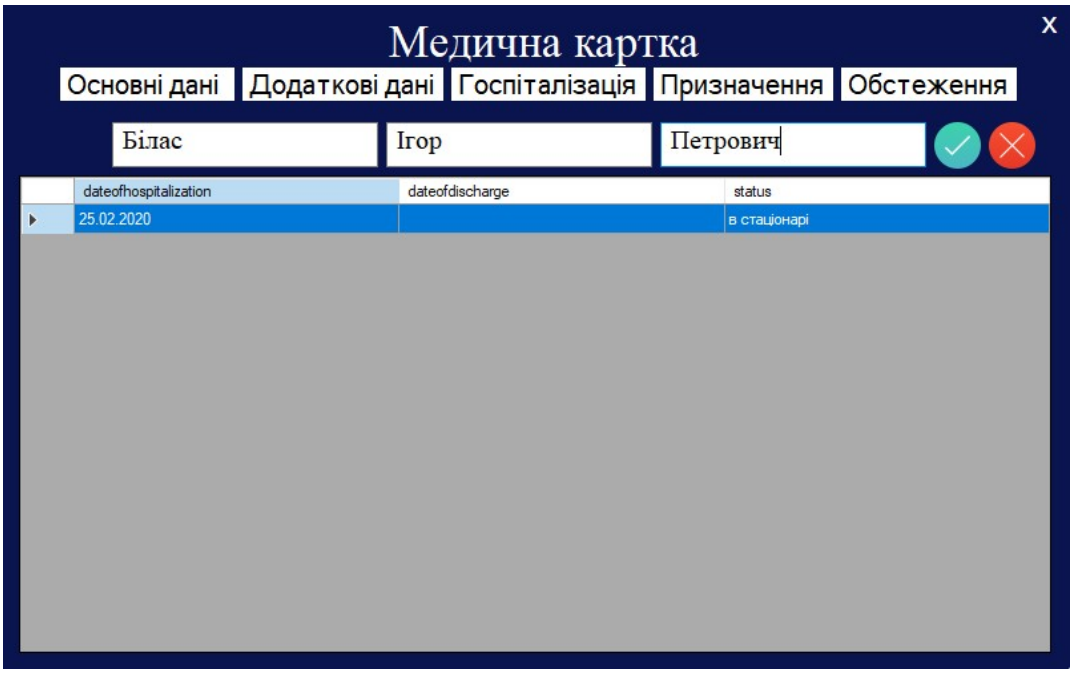

Рисунок 3.8 – Форма «MedicalCardForm» вкладка «Госпіталізація»

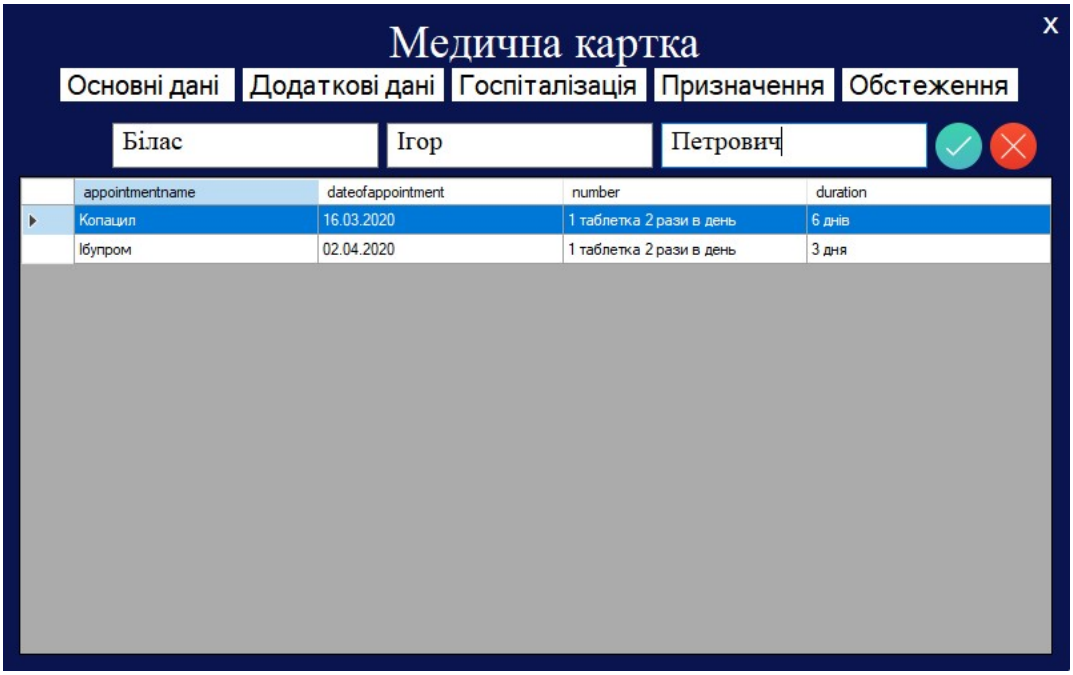

Рисунок 3.9 – Форма «MedicalCardForm» вкладка «Призначення»

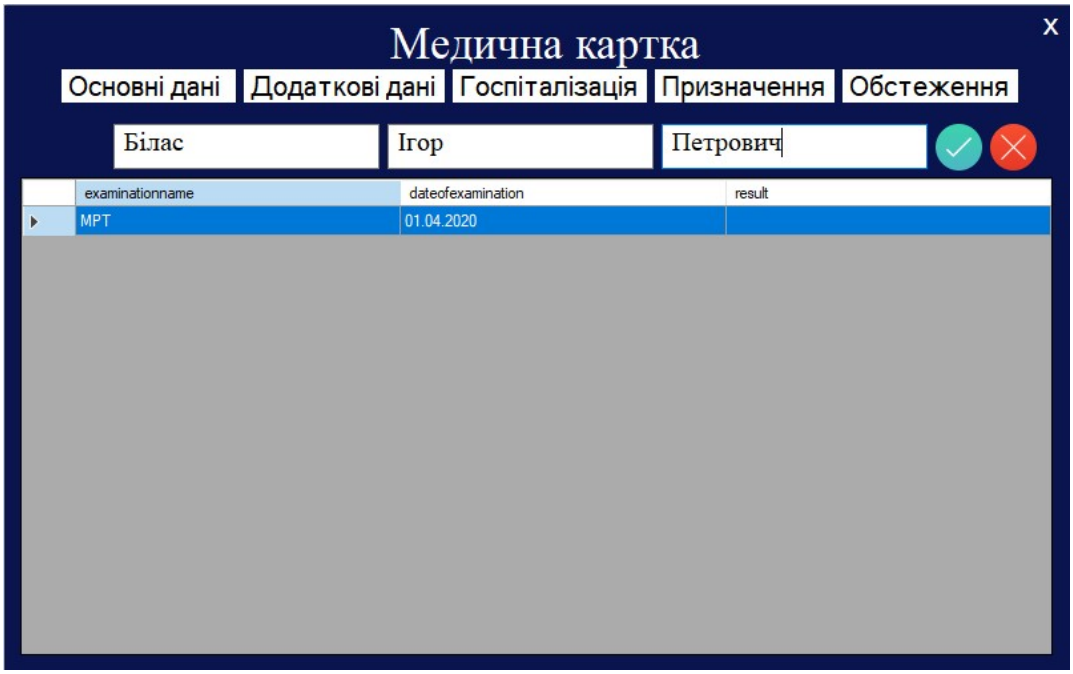

Рисунок 3.10 – Форма «MedicalCardForm» вкладка «Обстеження»

3.7. Форма «AppointmentForm»

AppointmentForm – це форма для додавання нового призначення або обстеження (рис. 3.11). Для того, щоб додати новий запис потрібно ввести необхідні дані і натиснути кнопку «Готово». Окремо можна додавати призначення, обстеження, дату виписки та результат обстеження. Щоб додати будь який новий запис обовязково потрібно заповнити поля ПІБ. Для того, щоб вийти з даної форми потрібно натиснути хрестик в правому верхньому куті форми.

Всі введені дані записуються в базу даних за допомогою процедур insert_ex (додати нове обстеження), insert_app (додати нове призначення), insert_discharge (додати дату виписки і автоматично змінити статус госпіталізації на «виписаний»), описаних в додатках.

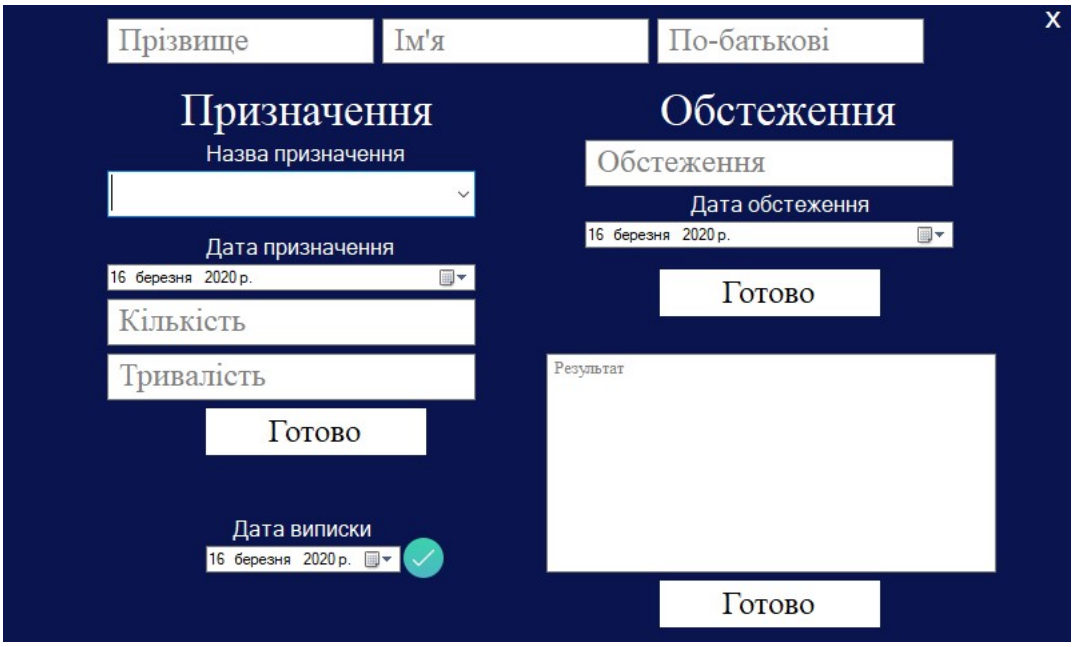

Рисунок 3.11 – Форма «AppointmentForm»

3.8. Форма «DoctorForm»

DoctorForm – це форма, яка показує всю інформацію про лікарів (рис. 3.12). В цій формі є можливість перейти до інших форм, таких як «Пацієнти», «Відділення» та «Палати», щоб вийти з програми потрібно натиснути відповідну кнопку внизу зліва.

Щоб додати нового лікаря, потрібно натиснути на відповідну клавішу зверху форми. При натисканні цієї кнопки відкриється форма реєстрації лікаря, яка була описана вище.

Щоб видалити всі записи про лікаря, необхідно ввести дані в виділені поля – це ПІБ та логін. Пошук реалізовано за ПІБ та окремо за посадою, спеціалізацією та категорією, для цього потрібно ввести необхідні дані в виділені поля і натиснути відповідну кнопку зверху форми.

Також в цій формі є можливість перейти до форми «База ліків», для цього потрібно натиснути необхідну кнопку зверху форми.

Рисунок 3.12 – Форма «DoctorForm»

3.9. Форма «DepartmentForm»

DepartmentForm – це форма, яка показує всю інформацію про відділення (рис. 3.13). В цій формі є можливість перейти до інших форм, таких як «Пацієнти», «Лікарі» та «Палати», щоб вийти з програми потрібно натиснути відповідну кнопку внизу зліва.

Щоб додати нове відділення, потрібно ввести необхідну інформацію в виділені поля – назва відділення та номер телефону і натиснути відповідну кнопку. Щоб видалити всі записи про відділення, необхідно ввести дані в виділені поля – це назва відділення та номер телефону і натиснути відповідну кнопку. Пошук реалізовано за назвою відділення, для цього потрібно ввести необхідні дані в виділені поля і натиснути відповідну кнопку зверху форми.

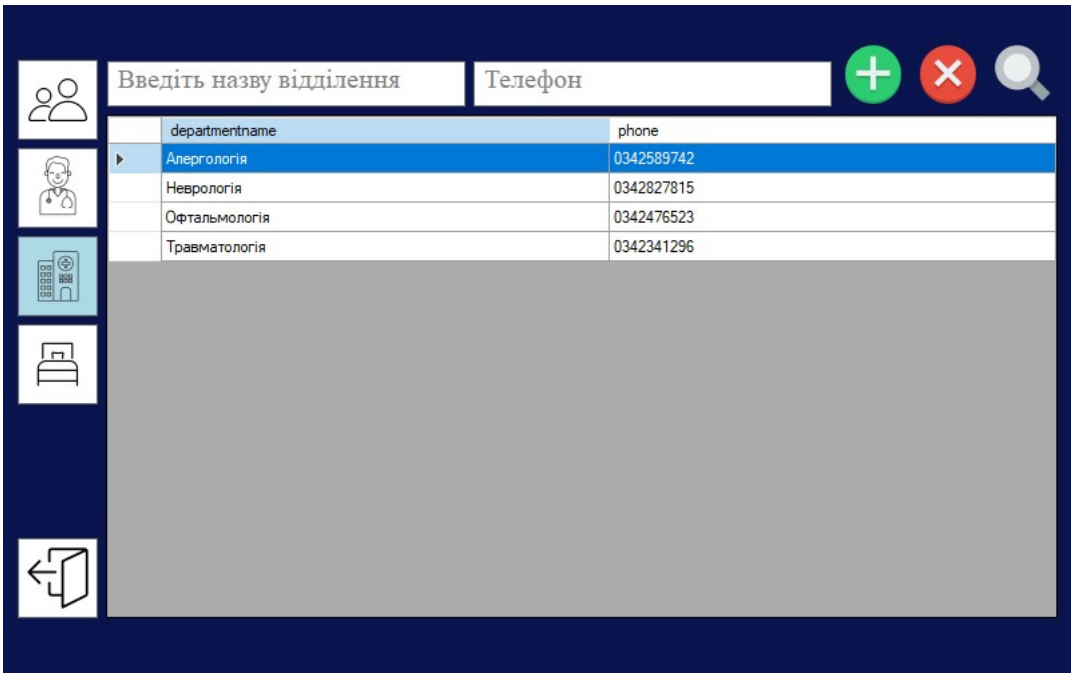

Рисунок 3.13 – Форма «DepartmentForm»

3.10. Форма «ChambersForm»

ChambersForm – це форма, яка показує всю інформацію про палати (рис. 3.14). В цій формі є можливість перейти до інших форм, таких як «Пацієнти», «Лікарі» та «Відділення», щоб вийти з програми потрібно натиснути відповідну кнопку.

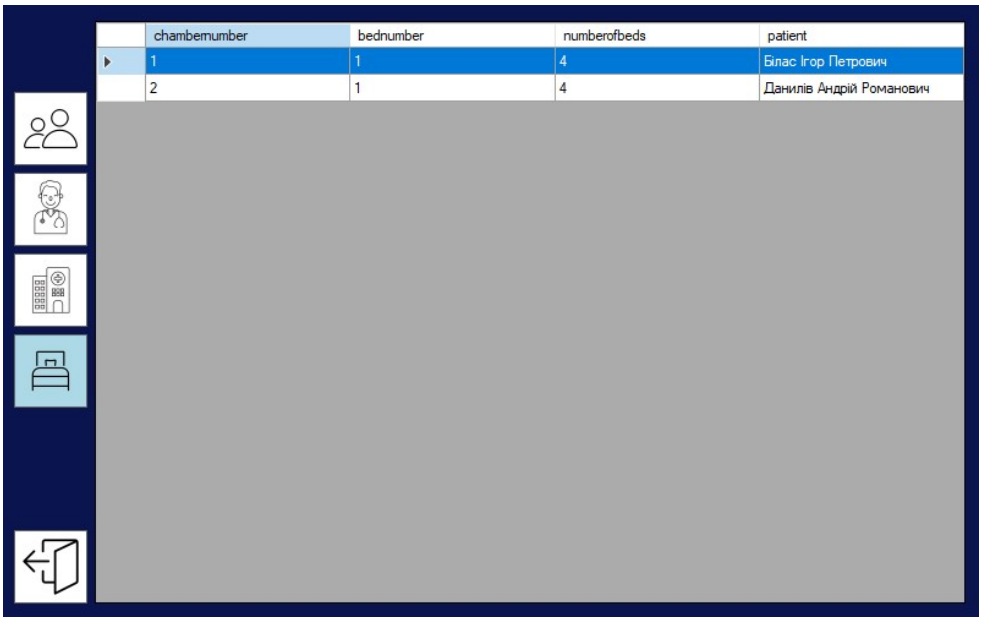

Рисунок 3.14 – Форма «ChambersForm»

3.11. Форма «MedicamentForm»

MedicamentForm – це форма, яка показує всю інформацію про базу ліків (рис. 3.15).

Щоб додати новий запис про лікарський засіб, потрібно натиснути на відповідну кнопку зверху форми. При натисканні цієї кнопки відкриється форма додавання нових ліків, яка буде описана нижче.

Щоб видалити всі записи про певний лікарський засіб, необхідно ввести дані в виділені поля – це назва ліків і натиснути відповідну кнопку. Пошук реалізовано за назвою ліків, формою випуску, одиницею виміру, виробником, для цього потрібно ввести необхідні дані в виділені поля і натиснути відповідну кнопку зверху форми. Для того, щоб вийти з даної форми потрібно натиснути хрестик в правому верхньому куті форми.

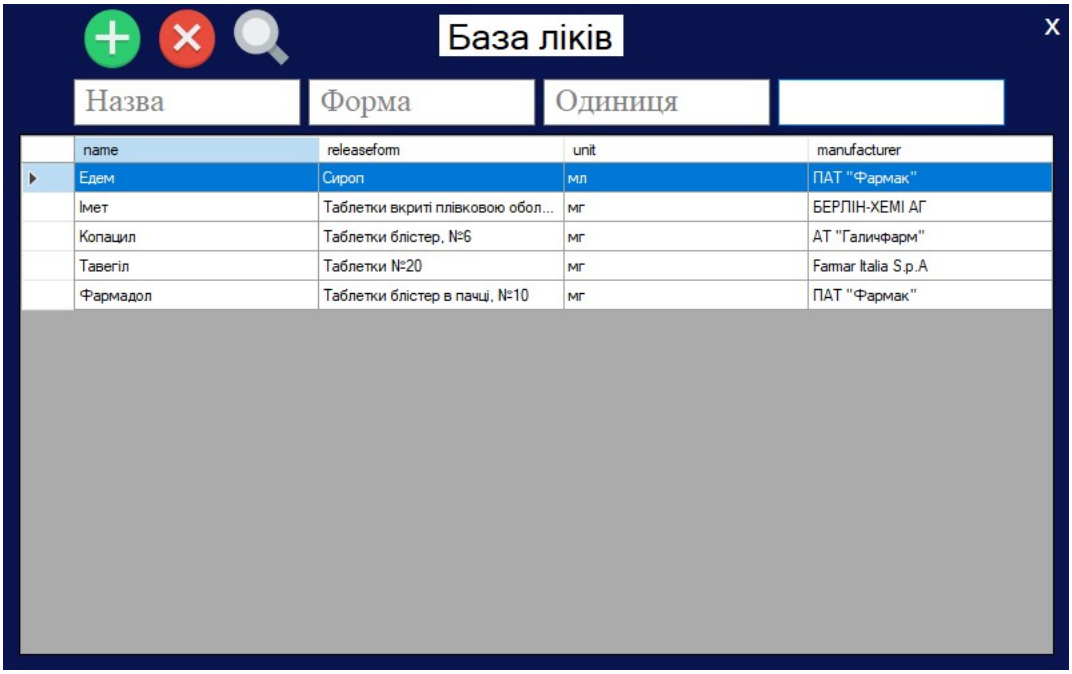

Рисунок 3.15 – Форма «MedicamentForm»

3.12. Форма «InsertMedicamentsForm»

InsertMedicamentsForm – це форма для додавання нового лікарського засобу (рис. 3.16). Для того, щоб додати новий запис потрібно ввести необхідні дані і натиснути кнопку «Готово». Для того, щоб вийти з даної форми потрібно натиснути кнопку «Відмінити».

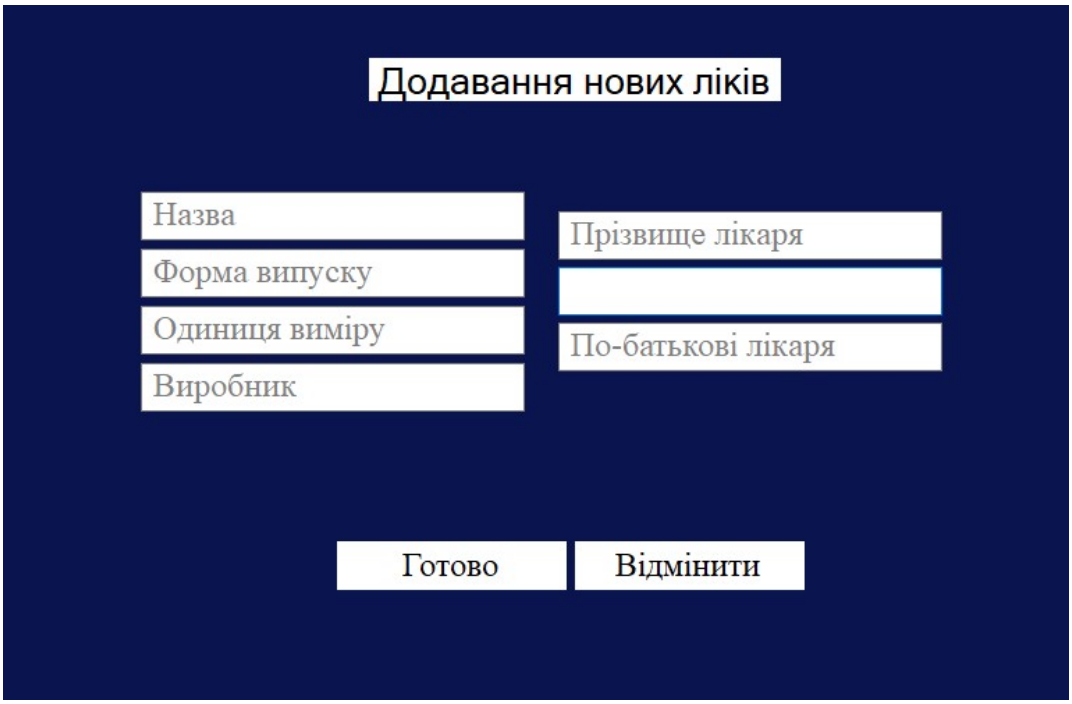

Рисунок 3.16 – Форма «InsertMedicamentsForm»

ВИСНОВКИ

У даній роботі було розроблено систему обробки даних діагностування і лікування пацієнтів, а саме було спроектовано базу даних та розроблено інтерфейс користувача.

На першому етапі було проведено аналіз обраних технологій – СУБД PostgreSQL та Windows Forms. При аналізі було досліджено архітектуру та основні можливості даних технологій. Обрана СУБД PostgreSQL задовольняє своїм функціоналом, допоміжним програмним забезпеченням, а також доступністю довідкових матеріалів потреби, що виникають при вирішенні даної задачі. Також дана СУБД є безкоштовною. За допомогою Windows Forms можна створювати графічно багаті програми, які легко розгортати та оновлювати. Вони можуть працювати, як при підключенні до Інтернету, так і без, можуть отримати доступ до ресурсів на локальному комп'ютері більш безпечним способом, ніж традиційні програми на базі Windows.

Другим етапом був опис вимог до бази даних та інтерфейсу користувача, які були виконані впродовж розробки програмного продукту, а також опис вхідної та вихідної інформації.

Третім етапом було проектування бази даних, а саме складання логічної моделі даних та реалізація таблиць. Було спроектовано базу даних, складено логічну модель, створено відповідні діаграми, реалізовано та описано 11 таблиць.

Наступним етапом був аналіз інформації про нормалізацію таблиць та нормалізація таблиць у даній базі даних. Було проаналізовано інформацію про нормалізацію таблиць, а саме про чотири основні нормальні форми, розглянуто переваги та недоліки нормалізації та було нормалізовано таблиці у базі даних.

Ще одним етапом був аналіз мови SQL, за допомогою якої були написані запити до бази даних. Було проаналізовано інформацію про мову SQL та показано реальні запити до даної бази даних.

Останнім етапом розробки системи була побудова інтерфейсу користувача. Створений інтерфейс відповідає клієнтській частині клієнт-серверної архітектури, яка взаємодіє з серверною частиною. Було використано технологію Windows

Forms, яка дозволила створити простий та зрозумілий інтерфейс користувача, який дозволяє проводити маніпуляції з даними, наявними у базі даних.

Отже, у даній роботі було розроблено компоненти системи обробки даних діагностування і лікування пацієнтів, яка являє собою ґрунтовну основу для подальшого розширення функціональності та оптимізації здобутих результатів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/about/
2. PostgreSQL 11.7 Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/11/index.html
3. PostgreSQL [Електронний ресурс] // Wikipedia. – 2020. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/PostgreSQL
4. PostgreSQL. Основы языка SQL. Учебное пособие 336с. – 2016р.
5. Techopedia Windows Forms [Електронний ресурс] // – Режим доступу до ресурсу: https://www.techopedia.com/definition/24300/windows–forms–net
6. Стаття «The Database Normalization Process» by Ronald Plew and Ryan Stephens Jan 24, 2003.
7. Джеймс Р. Грофф, Пол Н. Вайнберг, Ендрю Дж. Оппель – SQL. Полное руководство 960 с.
8. An Introduction to Database Systems (8th Edition) 8th Edition by C.J. Date
9. Windows Forms Programming in C# by Chris Sells
10. Windows Forms 2.0 Programming (Microsoft .NET Development Series) 2nd Edition by Chris Sells

ДОДАТКИ

Запит	Опис
CREATE TABLE patients (idpatient INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, doctorid INT NOT NULL, lastname VARCHAR(100) NOT NULL, firstname VARCHAR(100) NOT NULL, surname VARCHAR(100) NOT NULL, gender VARCHAR(10) NOT NULL, dateofbirth DATE NOT NULL, chamberid INT NOT NULL, contactid INT NOT NULL, FOREIGN KEY ("doctorid") REFERENCES doctors("iddoctor") ON DELETE CASCADE, FOREIGN KEY ("chamberid") REFERENCES chambers("idchamber") ON DELETE CASCADE, FOREIGN KEY ("contactid") REFERENCES contacts("idcontact") ON DELETE CASCADE, CONSTRAINT checkdate CHECK(dateofbirth < CURRENT_DATE));	Створення таблиці Patients, містить основну інформацію про пацієнтів. Зовнішні ключі: • doctorid (посилається на таблицю Doctors); • chamberid (посилається на таблицю Chambers); • contactid (посилається на таблицю Contacts).
CREATE TABLE additionalinformation (patientid INT NOT NULL, weight INT NOT NULL, height INT NOT NULL, bloodtype CHAR(3) NOT NULL, pressure INT NOT NULL, pulse INT NOT NULL, invalidity VARCHAR(100), FOREIGN KEY ("patientid")	Створення таблиці Additional information, містить додаткову інформацію про пацієнтів. Зовнішні ключі: • patientid (посилається на таблицю Patients).

REFERENCES patients("idpatient") ON DELETE CASCADE);	
CREATE TABLE chambers (idchamber INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, chambernumber INT NOT NULL, bednumber INT NOT NULL, numberofbeds INT NOT NULL);	Створення таблиці Chambers, містить всю інформацію про палати. Таблиця не має зовнішніх ключів.
CREATE TABLE hospitalization (idhospitalization INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, patientid INT NOT NULL, dateofhospitalization DATE NOT NULL, dateofdischarge DATE, status VARCHAR(100) NOT NULL, FOREIGN KEY ("patientid") REFERENCES patients("idpatient") ON DELETE CASCADE);	Створення таблиці Hospitalization, містить всю інформацію про госпіталізацію пацієнта. Зовнішні ключі: • patientid (посилається на таблицю Patients).
CREATE TABLE examination (idexamination INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, patientid INT NOT NULL, examinationname VARCHAR(100) NOT NULL, dateofexamination DATE NOT NULL, result TEXT, FOREIGN KEY ("patientid") REFERENCES patients("idpatient") ON DELETE CASCADE);	Створення таблиці Examination, містить всю інформацію про обстеження пацієнта. Зовнішні ключі: • patientid (посилається на таблицю Patients).

CREATE TABLE appointment (idappointment INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, patientid INT NOT NULL, appointmentname VARCHAR(100) NOT NULL, dateofappointment DATE NOT NULL, number TEXT NOT NULL, duration TEXT NOT NULL, FOREIGN KEY ("patientid") REFERENCES patients("idpatient") ON DELETE CASCADE);	Створення таблиці Appointment, містить всю інформацію про призначення. Зовнішні ключі: • patientid (посилається на таблицю Patients).
CREATE TABLE contacts (idcontact INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, sybjectis VARCHAR(20) NOT NULL, city VARCHAR(100) NOT NULL, address VARCHAR(100) NOT NULL, phone CHAR(10) NOT NULL, CONSTRAINT checkphone CHECK (phone LIKE '0%'));	Створення таблиці Contacts, містить контактну інформацію про пацієнтів та лікарів. Таблиця не має зовнішніх ключів.
CREATE TABLE doctors (iddoctor INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, lastname VARCHAR(100) NOT NULL, firstname VARCHAR(100) NOT NULL, surname VARCHAR(100) NOT NULL, login VARCHAR(100) NOT NULL, password VARCHAR(100) NOT NULL, postid INT NOT NULL, departmentid INT NOT NULL, contactid INT NOT NULL,	Створення таблиці Doctors, містить всю інформацію про лікарів. Зовнішні ключі: • postid (посилається на таблицю Posts); • departmentid (посилається на таблицю Departments); • contactid (посилається на таблицю Contacts).

diagnosis TEXT NOT NULL, FOREIGN KEY ("postid") REFERENCES posts("idpost") ON DELETE CASCADE, FOREIGN KEY ("departmentid") REFERENCES departments("iddepartment") ON DELETE CASCADE, FOREIGN KEY ("contactid") REFERENCES contacts("idcontact") ON DELETE CASCADE);	
CREATE TABLE departments (iddepartment INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, departmentname VARCHAR(100) NOT NULL, phone VARCHAR(20) NOT NULL, CONSTRAINT checkphone CHECK (phone LIKE '0342%'));	Створення таблиці Departments, містить інформацію про відділення. Таблиця не має зовнішніх ключів.
CREATE TABLE posts (idpost INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, postname VARCHAR(100) NOT NULL, specialization VARCHAR(100) NOT NULL, category VARCHAR(100) NOT NULL);	Створення таблиці Posts, містить інформацію про посади. Таблиця не має зовнішніх ключів.
CREATE TABLE medicament (doctorid INT NOT NULL, name VARCHAR(100) NOT NULL, releaseform VARCHAR(100) NOT NULL, unit VARCHAR(100) NOT NULL,	Створення таблиці Medicament, містить всю інформацію про базу ліків. Зовнішні ключі: • doctorid (посилається на таблицю Doctors).

manufacturer VARCHAR(100) NOT NULL, FOREIGN KEY ("doctorid") REFERENCES doctors("iddoctor") ON DELETE CASCADE);	
INSERT INTO departments (departmentname, phone) values (@departmentname, @phone);	SQL-запит для додавання нового запису в таблицю Departments.
BEGIN; CREATE PROCEDURE insert_p (_lastname VARCHAR, _firstname VARCHAR, _surname VARCHAR, _gender VARCHAR, _dateofbirth DATE, _selchamber INT, _bednumber INT, _sybjectis VARCHAR, _city VARCHAR, _address VARCHAR, _phone VARCHAR, _diagnosis TEXT, _seldoctorlname VARCHAR, _seldoctorfname VARCHAR, _seldoctorsname VARCHAR, _dateofhospitalization DATE, _weight INT, _height INT, _bloodtype CHAR, _pressure INT, _pulse INT, _invalidity VARCHAR)	Процедура, для додавання нового запису про певного пацієнта, дозволяє додавати записи відразу в чотири таблиці Patients, Contacts, Additionalinformation і Hospitalization.

```
LANGUAGE plpgsql
AS $$
DECLARE
	_patientid INT;
	_contactid INT;
	sel_doctor CURSOR for select iddoctor
from doctors where lastname =
_seldoctorlname;
	_seldoctorid INT;
	sel_chamber CURSOR for select
idchamber from chambers where
chambernumber = _selchamber and
bednumber = _bednumber;
	_selchamberid INT;
begin
OPEN sel_doctor;
FETCH FROM sel_doctor INTO _seldoctorid;
CLOSE sel_doctor;

OPEN sel_chamber;
FETCH FROM sel_chamber INTO
_selchamberid;
CLOSE sel_chamber;

insert into contacts ("sybjectis", "city",
"address", "phone")
values(
	"_sybjectis",
	"_city",
	"_address",
	"_phone")
returning idcontact into _contactid;

insert into patients("doctorid", "lastname",
```

```
"firstname", "surname", "gender",
"dateofbirth", "chamberid", "contactid",
"diagnosis")
values
(
        "_seldoctorid",
        "_lastname",
        "_firstname",
        "_surname",
        "_gender",
        "_dateofbirth",
        "_selchamberid",
        "_contactid",
        "_diagnosis")
returning idpatient into _patientid;

insert into additionalinformation ("patientid",
"weight", "height", "bloodtype", "pressure",
"pulse", "invalidity")
values(
        "_patientid",
        "_weight",
        "_height",
        "_bloodtype",
        "_pressure",
        "_pulse",
        "_invalidity"
);

insert into hospitalization("patientid",
"dateofhospitalization", "status")
values (
        "_patientid",
        "_dateofhospitalization",
```

"в стаціонарі"); end $$; COMMIT;	

BEGIN;
CREATE PROCEDURE insert_ex
(
_lname VARCHAR,
_fname VARCHAR,
_sname VARCHAR,
_examinationname VARCHAR,
_dateofexamination DATE
)
LANGUAGE plpgsql
AS $$
DECLARE
_examinationid INT;
sel_patient CURSOR for select idpatient from patients where lastname = _lname and firstname = _fname and surname = _sname;
_patientid INT;
begin
OPEN sel_patient;
FETCH FROM sel_patient INTO _patientid;
CLOSE sel_patient;
 	
insert into examination ("patientid", "examinationname", "dateofexamination")
values(
"_patientid",
"_examinationname",
"_dateofexamination")
returning idexamination into _examinationid;
	Процедура, для додавання нового запису про обстеження пацієнта в таблицю Examination за його ПІБ.

end $$; COMMIT;	

BEGIN;
CREATE PROCEDURE insert_app
(
_lname VARCHAR,
_fname VARCHAR,
_sname VARCHAR,
_appointmentname VARCHAR,
_dateofappointment DATE,
_number VARCHAR,
_duration VARCHAR
)
LANGUAGE plpgsql
AS $$
DECLARE
_appointmentid INT;
sel_patient CURSOR for select idpatient from patients where lastname = _lname and firstname = _fname and surname = _sname;
_patientid INT;
begin
OPEN sel_patient;
FETCH FROM sel_patient INTO _patientid;
CLOSE sel_patient;
 	
insert into appointment ("patientid", "appointmentname", "dateofappointment", "number", "duration")
values(
"_patientid",
"_appointmentname",
	Процедура для додавання нового запису про призначення для пацієнта в таблицю Appointment за його ПІБ.

"_dateofappointment", "_number", "_duration") returning idappointment into _appointmentid; end $$; COMMIT;	
BEGIN; CREATE PROCEDURE insert_data (_lastname VARCHAR, _firstname VARCHAR, _surname VARCHAR, _selecteddepartment VARCHAR, _login VARCHAR, _password VARCHAR, _sybjectis VARCHAR, _city VARCHAR, _address VARCHAR, _phone VARCHAR, _postname VARCHAR, _specialization VARCHAR, _category VARCHAR) LANGUAGE plpgsql AS $$ DECLARE sel_department CURSOR for select iddepartment from departments where departmentname = _selecteddepartment; _contactid INT; _postid INT; _selectedid INT;	Процедура, для додавання нового запису про лікаря, дозволяє додавати записи відразу в три таблиці Doctors, Contacts, Posts.

```
begin
OPEN sel_department;
FETCH FROM sel_department INTO
_selectedid;
CLOSE sel_department;

insert into contacts ("sybjectis", "city",
"address", "phone")
values(
        "_sybjectis",
        "_city",
        "_address",
        "_phone")
returning idcontact into _contactid;

insert into posts ("postname", "specialization",
"category")
values(
        "_postname",
        "_specialization",
        "_category"
)
returning idpost into _postid;

insert into doctors("lastname", "firstname",
"surname", "postid",
"departmentid","contactid", "login",
"password")
values
(
        "_lastname",
        "_firstname",
        "_surname",
        "_postid",
```

"_selectedid", "_contactid", "_login", "_password"); end $$; COMMIT;	

BEGIN;
CREATE PROCEDURE insert_discharge
(
_lname VARCHAR,
_fname VARCHAR,
_sname VARCHAR,
_dateofdischarge DATE
)
LANGUAGE plpgsql
AS $$
DECLARE
sel_patient CURSOR for select idpatient from patients where lastname = _lname and firstname = _fname and surname = _sname;
_patientid INT;
begin
OPEN sel_patient;
FETCH FROM sel_patient INTO _patientid;
CLOSE sel_patient;
 	
update hospitalization set dateofdischarge = _dateofdischarge where patientid = _patientid;
update hospitalization set status = 'виписаний' where patientid = _patientid;
 	
end
	Процедура для оновлення запису про госпіталізацію (в цьому випадку про виписку) пацієнта за ПІБ в таблиці Hospitalization.

$$; COMMIT;	

BEGIN;
CREATE PROCEDURE exresult1
(
_examinationname VARCHAR,
_dateofexamination DATE,
_result TEXT
)
LANGUAGE plpgsql
AS $$
DECLARE
begin
 	
update examination set result = _result where examinationname = _examinationname and dateofexamination = _dateofexamination ;
 	
end
$$;
COMMIT;
	Процедура для оновлення запису про обстеження (в цьому випадку додається результат обстеження) пацієнта за назвою обстеження і датою в таблиці Examination.

BEGIN; CREATE PROCEDURE insert_medicament (_lname VARCHAR, _fname VARCHAR, _sname VARCHAR, _name VARCHAR, _releaseform VARCHAR, _unit VARCHAR, _manufacturer VARCHAR) LANGUAGE plpgsql AS $$	Процедура, для додавання нового запису про лікарський засіб в таблицю Medicament.

DECLARE sel_doctor CURSOR for select iddoctor from doctors where lastname = _lname and firstname = _fname and surname = _sname; _doctorid INT; begin OPEN sel_doctor; FETCH FROM sel_doctor INTO _doctorid; CLOSE sel_doctor; insert into medicament ("doctorid", "name", "releaseform", "unit", "manufacturer") values ("_doctorid", "_name", "_releaseform", "_unit", "_manufacturer"); end $$; COMMIT;	
SELECT departmentname, phone FROM departments ORDER BY departmentname;	SQL-запит для виведення необхідної інформації з таблиці Departments.
SELECT lastname, firstname, surname, postname, specialization, category, city, address, phone FROM doctors JOIN posts ON doctors.postid = posts.idpost JOIN contacts ON doctors.contactid = contacts.idcontact ORDER BY lastname;	SQL-запит для виведення необхідної інформації з таблиць Doctors і Contacts.
SELECT lastname, firstname, surname, gender, dateofbirth, diagnosis FROM patients ORDER BY idpatient;	SQL-запит для виведення необхідної інформації з таблиці Patients.

SELECT departmentname FROM departments ORDER BY iddepartment;	SQL-запит для виведення назв відділень для випадного списку з таблиці Departments.
SELECT lastname, firstname, surname, gender, dateofbirth, diagnosis, city, address, phone, chambernumber, bednumber FROM patients JOIN contacts ON patients.contactid = contacts.idcontact JOIN chambers ON patients.chamberid = chambers.idchamber WHERE lastname = @lname and firstname = @fname and surname = @sname ORDER BY lastname;	SQL-запит для виведення необхідної інформації з таблиць Patients і Contacts для певного пацієнта за ПІБ.
SELECT weight, height, bloodtype, pressure, pulse, invalidity FROM additionalinformation JOIN patients ON additionalinformation.patientid = patients.idpatient WHERE lastname = @lname and firstname = @fname and surname = @sname;	SQL-запит для виведення необхідної інформації з таблиць Patients і Additionalinformation для певного пацієнта за ПІБ.
SELECT dateofhospitalization, dateofdischarge, status FROM hospitalization JOIN patients ON hospitalization.patientid = patients.idpatient WHERE lastname = @lname and firstname = @fname and surname = @sname;	SQL-запит для виведення необхідної інформації з таблиць Patients і Hospitalization для певного пацієнта за ПІБ.
SELECT appointmentname, dateofappointment, number, duration FROM appointment JOIN patients ON appointment.patientid = patients.idpatient WHERE lastname = @lname and firstname = @fname and surname = @sname;	SQL-запит для виведення необхідної інформації з таблиць Patients і Appointment для певного пацієнта за ПІБ.
SELECT examinationname, dateofexamination, result FROM examination JOIN patients ON examination.patientid =	SQL-запит для виведення необхідної інформації з таблиць Patients і Examination для певного пацієнта за ПІБ.

patients.idpatient WHERE lastname = @lname and firstname = @fname and surname = @sname;	
SELECT departmentname, phone FROM departments WHERE departmentname=@departmentname ORDER BY departmentname;	SQL-запит для пошуку необхідної інформації за назвою відділення.
SELECT lastname, firstname, surname, postname, specialization, category FROM doctors INNER JOIN posts ON doctors.postid = posts.idpost WHERE lastname=@lname and firstname=@fname and surname=@sname or postname=@postname or specialization=@specialization or category=@category ORDER BY lastname;	SQL-запит для пошуку необхідної інформації за ПІБ або посадою лікаря.
SELECT lastname, firstname, surname, gender, dateofbirth, diagnosis FROM patients WHERE lastname=@lname and firstname=@fname and surname=@sname or gender=@gender ORDER BY idpatient;	SQL-запит для пошуку необхідної інформації за ПІБ пацієнта або за статтю.
DELETE FROM departments WHERE departmentname = @departmentname;	SQL-запит для видалення всіх даних з таблиці Departments за назвою відділення.
BEGIN; CREATE PROCEDURE delete_patients (_lastname VARCHAR, _firstname VARCHAR, _surname VARCHAR, _diagnosis VARCHAR) LANGUAGE plpgsql AS $$ DECLARE id_patient CURSOR for select idpatient from	Процедура для видалення всіх даних про пацієнта одразу з семи таблиць: Appointment, Examination, Hospitalization, Patients Additionalinformation, Chambers, Contacts за ПІБ та діагнозом.

```
patients where lastname = _lastname and
firstname = _firstname and surname =
_surname and diagnosis = _diagnosis;
_patientid INT;
id_contact CURSOR for select contactid from
patients where lastname = _lastname and
firstname = _firstname and surname =
_surname and diagnosis = _diagnosis;
_contactid INT;
id_chamber CURSOR for select chamberid
from patients where lastname = _lastname and
firstname = _firstname and surname =
_surname and diagnosis = _diagnosis;
_chamberid INT;

begin
OPEN id_patient;
FETCH FROM id_patient INTO _patientid;
CLOSE id_patient;

OPEN id_contact;
FETCH FROM id_contact INTO _contactid;
CLOSE id_contact;

OPEN id_chamber;
FETCH FROM id_chamber INTO
_chamberid;
CLOSE id_chamber;

delete from contacts where idcontact =
_contactid;
delete from chambers where idchamber =
_chamberid;
delete from patients where "lastname" =
```

_lastname and "firstname" = _firstname and "surname" = _surname and "diagnosis" = _diagnosis; delete from additionalinformation where patientid = _patientid; delete from hospitalization where patientid = _patientid; delete from examination where patientid = _patientid; delete from appointment where patientid = _patientid; end $$; COMMIT;	
BEGIN; CREATE PROCEDURE delete_doctors (_lastname VARCHAR, _firstname VARCHAR, _surname VARCHAR, _login VARCHAR) LANGUAGE plpgsql AS $$ DECLARE id_contact CURSOR for select contactid from doctors where lastname = _lastname and firstname = _firstname and surname = _surname and login = _login; _contactid INT; id_posts CURSOR for select postid from doctors where lastname = _lastname and firstname = _firstname and surname = _surname and login = _login;	Процедура для видалення всіх даних про лікаря одразу з трьох таблиць: Contacts, Posts, Doctors за ПІБ та логіном.

_postsid INT; begin OPEN id_contact; FETCH FROM id_contact INTO _contactid; CLOSE id_contact; OPEN id_posts; FETCH FROM id_posts INTO _postsid; CLOSE id_posts; delete from contacts where idcontact = _contactid; delete from posts where idposts = _postsid; delete from doctors where "lastname" = _lastname and "firstname" = _firstname and "surname" = _surname and "login" = _login; end $$; COMMIT;	

BEGIN;
CREATE PROCEDURE delete_medicament
(
_name VARCHAR
)
LANGUAGE plpgsql
AS $$
DECLARE
begin
delete from medicament where name = _name;
end
$$;
COMMIT;
	Процедура для видалення всіх даних про лікарський засіб з таблиці Medicament за назвою ліків.