НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ
(повна назва інституту/факультету)

кафедра БІОМЕДИЧНОЇ КІБЕРНЕТИКИ
(повна назва кафедри)

КУРСОВА РОБОТА

з дисципліни (кредитного модуля) **Системи баз даних**

спеціальність ***122 Комп'ютерні науки***

спеціалізація ***Комп'ютерні технології в біології та медицині***

На тему **Розробка бази даних для управління відділом психотерапії та психіатрії в лікарні**

(тема індивідуального завдання)

Виконав студент 3-го курсу гр. БС-03
ГОНЧАРУК ВЛАДИСЛАВ СЕРГІЙОВИЧ

Засвідчую, що у роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент (-ка) _____

Перевірив ст.викл. Віталій СЕРДАКОВСЬКИЙ

Бали за роботу студента відповідно до РСО за

Оформлення роботи (із 18 балів)	
Обґрунтування (із 12 балів)	
Захист роботи (із 60 балів)	
ВСЬОГО	

_____

(підпис викладачів)

Київ – 2022р.

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут (факультет) БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ
(повна назва)

Кафедра БІОМЕДИЧНОЇ КІБЕРНЕТИКИ
(повна назва)

ЗАВДАННЯ
на курсову роботу студенту

ГОНЧАРУК ВЛАДИСЛАВ СЕРГІЙОВИЧ
(прізвище, ім'я, по батькові)

1. Тема роботи **Розробка бази даних для управління відділом психотерапії та психіатрії в лікарні**

2. Термін подання студентом роботи ***20 грудня 2022 року***

3. Вихідні дані до роботи **Індивідуальне завдання до лабораторних робіт.**

4. Зміст роботи **1. Теоретична частина (визначення необхідних знань для реалізації) 2. Аналітична частина (аналітичний аналіз курсової роботи, бізнес процесів, побудова діаграми «сутність-зв'язок»). 3. Конструктивна частина (реалізація розробки бази даних).**

5. Дата видачі завдання ***20 вересня 2022 р*.**

Календарний план

№ з/п	Назва етапів виконання курсової роботи	Термін виконання етапів роботи	Примітка
1	Отримати завдання на КР	20 вересня 2022р.	
	Оформлення розділу з		
	Оформлення розділу з		
	Оформлення розділу з		
	Оформлення КР		
	Подання в електронному вигляді КР та анотації до неї на сайт кафедри.	До 26 грудня 2022р	
	Подання пакету документів по КР до захисту	26 грудня 2022р.	
	Захист КР	31 грудня 2022р	

Студент _____ Владислав ГОНЧАРУК
(підпис) (ініціали, прізвище)

Керівник роботи _____ Віталій СЕРДАКОВСЬКИЙ
(підпис) (ініціали, прізвище)

Анотація

«Курсова робота (Курсовий проект)» з дисципліни «Системи баз даних» являється частиною циклу **Загальної підготовки** дисциплін першого (бакалаврського) рівня вищої освіти ступеня «**бакалавр**», зі спеціальності 122 «Комп'ютерні науки» за спеціалізацією «Комп'ютерні технології в біології та медицині»).

Загальна трудомісткість освоєння модуля становить 1 кредит (ЕКТС), 30 годин.

Курсову роботу виконав **Гончарук Владислав Сергійович,** студент 3 курсу, гр. БС-03 кафедри Біомедичної кібернетики факультету Біомедичної інженерії НТУУ «КПІ ім. Ігоря Сікорського».

Тема роботи: Розробка бази даних для управління відділом психотерапії та психіатрії в лікарні.

Питання, розглянуті в курсовій роботі: концептуальне, логічне та фізичне проектування бази даних.

Отримані результати курсової роботи: було отримано навички з пошуку теоретичної інформації по поставлені задачі, навички створення діаграм концептуального та логічного етапів проектування баз даних, фізичного проетування: створення таблиць та запитів до них в системі керування базами даних MySQL, та навички аналізу виконання роботи.

Структура і обсяг роботи: курсова робота складається із вступу, опису завдання, трьох розділів, висновків, списку використаної літератури із 17 джерел і 6 додатків. Загальний обсяг курсової роботи становить 91 сторінок, основного тексту (без додатків) – 70 сторінок, ілюстрацій – 9, таблиць – 2.

Abstract

"Course work (Course project)" in the discipline "Database Systems" is part of the cycle of General training of disciplines of the first (bachelor's) level of higher education degree "Bachelor", specialty 122 "Computer Science" in the specialty "Computer Technology in biology and medicine ").

The total complexity of mastering the module is 1 credit (ECTS), 30 hours.

Course work was performed by Vladyslav Honcharuk Serhiiovych, 3rd year student, gr. BS-03 of the Department of Biomedical Cybernetics, Faculty of Biomedical Engineering, NTUU "KPI. Igor Sikorsky ».

R&D: Development of a database for managing the department of psychotherapy and psychiatry in the hospital.

The issues considered in the course work: conceptual, logical and physical database design.

The results of coursework: obtained skills in finding theoretical information on the tasks, skills in creating diagrams of conceptual and logical stages of database design, physical design: creating tables and queries to them in the MySQL database management system, and skills in analyzing the resultsof the work.

The structure and scope of work, course work consists of an introduction, description of the tasks, three sections, conclusions, list of references on 17 sources and 6 applications. The total volume coursework is 91 pages, main text (no attachments) – 70 pages, illustrations – 9, tables – 2.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

БД – база даних;

СКБД – система керування базами даних;

RDBMS – Relational Database Management System;

ІС – інформаційна система;

КР – курсова робота;

НФ – нормальна форма;

Рис. – рисунок;

SQL – Structured Query Language;

NoSQL – Not Only Structured Query Language;

ВСТУП

Сьогодні психічні розлади є одними з найбільш розповсюдженіших хвороб, які можуть мати люди, і для людей дуже важливо на ранніх етапах розвитку таких хвороб знайти спосіб позбавитися їх, а в випадку вродженої патології зробити все можливе для того, щоб полегшити життя людини, а також зробити її соціалізацію більш безпечною для інших. Психічні розлади напряму впливають на якість життя та загальний стан здоров'я людини, якщо їх не лікувати - це може призвести до недієздатності або суїциду в гірших випадках, та погіршення продуктивності, неможливості влитися в соціум в кращих. Але локалізація таких розладів не є простим завданням і вимагає втручання психолога або лікаря психіатра або ж психотерапевта, це залежить від типу розладу та його ступіню тяжкості та типу розладу.

Автоматизація такої системи лікування є надзвичайно важливою в наш період діджиталізації, адже за допомогою автоматизованої системи запису пацієнтів до лікаря можна зекономити час та організувати злагоджену роботу медичного закладу. Така система допомагає вести електронну історію лікування пацієнта, а також може містити інформацію про наявні хвороби та методи їх лікування в випадку, якщо однозначно було визначено психічний розлад, також за допомогою такої системи можна виписувати пацієнтам ліки напряму, використавши технологію блокчейн для підтвердження достовірності рецепту, таким чином пацієнти зможуть отримувати ліки в аптеці набагато простіше.

Для імплементації такої автоматизації необхідна база даних, яка зможе зберігати інформації про пацієнтів, лікарів, хвороби та лікарню. Інформація в цій базі даних повинна бути швидкодоступною, база даних повинна бути стійкою до помилок, а також має бути можливість розширення бази даних через те, що вона може містити інформацію про пацієнтів та лікарів з усієї країни, а не бути локальною.

РОЗДІЛ 1
ТЕОРЕТИЧНА ЧАСТИНА

1.1. Бази даних

База даних - це сукупність організованих даних, що зберігаються і доступні в електронному вигляді. Бази даних використовуються для зберігання та управління великими обсягами даних в широкому спектрі додатків, включаючи фінансові системи, системи управління медичними даними та інтернет-магазини.

Існує кілька типів баз даних, включаючи реляційні бази даних (RDB), які зберігають дані в таблицях, що складаються з рядків і стовпців, і бази даних NoSQL, які зберігають дані в різних форматах, включаючи пари ключ-значення, документи і графіки. Управління базами даних зазвичай здійснюється за допомогою систем керування базами даних (СКБД), які забезпечують інтерфейс для створення та зміни даних у базі даних, а також виконання запитів до даних.

1.2. Реляційна модель бази даних

У реляційній моделі бази даних дані організовані в таблиці, також відомі як відношення. Кожна таблиця має унікальне ім'я і складається з рядків і стовпців. Стовпці в таблиці називаються атрибутами, і кожен атрибут має ім'я та тип даних. Рядки в таблиці називаються записами, і кожен запис представляє екземпляр сутності.

У реляційній базі даних зв'язки між різними таблицями визначаються за допомогою зовнішніх ключів. Зовнішній ключ — це поле в одній таблиці, яке посилається на первинний ключ іншої таблиці. Це дозволяє базі даних забезпечувати цілісність посилань, а це означає, що вона може запобігти невідповідностям і помилкам під час оновлення або видалення записів у

таблиці, пов'язаній з іншими таблицями. Однією з головних переваг моделі реляційної бази даних є те, що вона дозволяє визначати зв'язки між різними сутностями у ваших даних, що полегшує запит та маніпулювання даними. Також підтримується використання мови структурованих запитів (SQL), яка є стандартною мовою взаємодії з базами даних.

1.3. Системи керування базами даних

Система керування базами даних (СКБД) - це програмне забезпечення, яке дозволяє користувачам створювати, редагувати бази даних, а також виконувати запити до даних, що містяться в них. Воно забезпечує інтерфейс між програмним додатком та даними, що зберігаються в базі даних[1]. А також включає в себе ряд інструментів і функцій для управління даними і забезпечення їх цілісності. Деякі з ключових функцій включають:

- Зберігання даних: СКБД зберігає дані в структурованому форматі, наприклад, у вигляді таблиць, що складаються з рядків і стовпців.
- Визначення даних: СКБД дозволяє користувачам визначати структуру даних, включаючи типи даних та атрибути для кожного фрагмента даних.
- Маніпулювання даними: СКБД надає інструменти для вставки, оновлення та видалення даних у базі даних.
- Безпека даних: СКБД включає функції захисту даних у базі даних, такі як автентифікація користувачів та контроль доступу.
- Цілісність даних: СКБД забезпечує цілісність даних, застосовуючи правила та обмеження до даних, такі як унікальні ключі та зовнішні ключі.

Існує багато різних СКБД, починаючи від безкоштовних варіантів і варіантів з відкритим кодом і закінчуючи комерційними продуктами. Приклади сучасних СКБД: MySQL, Oracle і Microsoft SQL Server.

1.4. MySQL

MySQL, найпопулярніша система управління базами даних SQL з відкритим вихідним кодом, розробляється, розповсюджується та підтримується корпорацією Oracle[2]. Вона широко використовується в Інтернеті для зберігання та управління даними. Це реляційна СКБД, що означає, що вона зберігає дані в таблицях, що складаються з рядків і стовпців, і використовує мову структурованих запитів (SQL) для доступу до даних і маніпулювання ними. MySQL відома своєю швидкістю, надійністю та гнучкістю, її використовують багато великих і малих організацій по всьому світу. Особливо популярна ця СКБД в розробці баз даних для веб-додатків, оскільки вона проста у використанні і має велику кількість підтримуваних мов програмування і бібліотек. MySQL є ключовим компонентом стека LAMP (Linux, Apache, MySQL, PHP/Perl/Python), який є популярним набором інструментів для веб-розробки.

На додаток до основної бази даних MySQL, існує також ряд інструментів та утиліт, наданих MySQL, які можна використовувати для управління та адміністрування бази даних, а також для проектування та розробки додатків, керованих базами даних. MySQL доступна для різних платформ, включаючи Windows, Linux і MacOS.

1.5. Проектування бази даних

Проектування бази даних - це процес створення структури бази даних, що дозволяє ефективно зберігати, організовувати і отримувати доступ до даних, які вона містить. Воно передбачає ідентифікацію даних, які необхідно зберігати, визначення зв'язків між різними суб'єктами даних та вибір відповідної системи управління базами даних (СКБД) та моделі (реляційної чи одного з різновидів NoSQL). Ефективне проектування бази даних є важливим, оскільки воно допомагає гарантувати, що база даних є ефективною, точною та

простою у використанні. Правильно розроблена база даних надає доступ до актуальної, точної інформації[3]. Це також допомагає мінімізувати надмірність даних і їх залежність, що може ускладнити ведення бази даних і збільшити ризик невідповідності даних.

У процесі проектування бази даних задіяно кілька етапів, зокрема:

1. Визначення призначення бази даних і типів даних, які вона буде зберігати.
2. Визначення зв'язків між різними сутностями даних та визначення структури даних.
3. Вибір СКБД і прийняття рішення про модель бази даних.
4. Створення логічного дизайну бази даних за допомогою інструменту побудови діаграм або шляхом написання коду.
5. Нормалізація даних для мінімізації надмірності та залежності.
6. Створення фізичного дизайну бази даних на основі обраної СУБД і апаратного забезпечення.
7. Фізична реалізація бази даних шляхом створення необхідних таблиць, індексів та інших об'єктів бази даних.
8. Тестування бази даних, щоб переконатися, що вона відповідає вимогам і працює належним чином.
9. Ведення бази даних шляхом її резервного копіювання, моніторингу її продуктивності та внесення змін за потреби.

Основні етапи можна виділити як такі:

1. Концептуальне проектування
2. Логічне проектування
3. Фізичне проектування

1.5.1. Концептуальне проектування

Концептуальне проектування бази даних - це процес створення високорівневого дизайну бази даних, який окреслює структуру бази даних і зв'язки між різними сутностями даних. Метою концептуального проектування бази даних є створення чіткої та всеосяжної моделі даних, яку користувачі легко розуміють і використовують. Процес концептуального проектування баз даних зазвичай включає в себе наступні етапи:

1. Визначення даних, які необхідно зберігати в базі даних.
2. Визначення зв'язків між різними сутностями даних.
3. Визначення структури бази даних, включаючи типи даних та атрибути для кожної сутності.
4. Створення схеми або моделі, яка ілюструє структуру та зв'язки даних.
5. Перевірка дизайну, щоб переконатися, що він відповідає потребам користувачів та вимогам програми.

1.5.1.1. Предметна область та її опис

Предметна область - це сфера частини бізнесу або організації, для підтримки якої розробляється база даних (наприклад, бізнес, фінанси, медицина). На етапі концептуального проектування проекту бази даних визначається та аналізується предметна область з метою визначення конкретних вимог та потреб бази даних. Це може включати визначення ключових суб'єктів (наприклад, клієнтів, продуктів, співробітників) та відносин, які повинні бути представлені в базі даних, а також конкретних елементів даних (наприклад, ім'я, адреса, заробітна плата), які потрібно відстежувати для кожної організації. Опис предметної області в концептуальному проектуванні є важливим етапом процесу проектування бази даних, оскільки допомагає гарантувати, що база даних адаптована під конкретні потреби бізнесу або організації і здатна ефективно підтримувати її діяльність.

1.5.1.2. Прецеденти використання

Прецедент - це опис того, як користувач буде взаємодіяти з базою даних і як система відповість на цей запит. Він може включати конкретні дії, які користувач може виконати (наприклад, запит даних, оновлення записів, створення нових записів), а також очікувані результати цих дій (наприклад, список результатів пошуку, повідомлення про підтвердження). Прецеденти часто використовуються на етапі концептуального проектування проекту бази даних, щоб допомогти визначити вимоги та потреби користувачів, а також забезпечити, щоб база даних була розроблена таким чином, щоб користувачам було легко зрозуміти та використовувати додаток. Прецеденти також можуть бути використані для виявлення будь-яких проблем, які можуть виникнути під час використання бази даних, а також для розробки рішень для вирішення цих проблем.

1.5.1.3. Діаграма прецедентів

Діаграма прецедентів - це візуальне представлення зв'язків і залежностей між різними елементами в базі даних, яке відображує зв'язки між різними сутностями бази даних (наприклад, лікарями, пацієнтами, записами) та способи їх взаємодії один з одним. Діаграма прецедентів також може показувати зв'язок між різними діями користувача (наприклад, запитами, оновленнями, видаленнями) та очікуваними відповідями системи на ці дії. Наприклад, діаграма прецедентів може показати, що коли користувач виконує пошук певного продукту, система відповідає, повертаючи список відповідних товарів з бази даних. Діаграми прецедентів можуть бути корисними для візуалізації загальної структури та поведінки системи, а також для виявлення будь-яких потенційних проблем. Вони також можуть бути корисними для донесення дизайну системи до зацікавлених сторін та для документування системи для подальшого використання.

Елементи діаграми прецедентів:

Сутності: різні речі, які представлені в системі, такі як клієнти, продукти, замовлення або співробітники.

Дії: конкретні дії, які користувачі можуть виконувати в системі, наприклад запити, оновлення або видалення.

Відповіді: очікувані відповіді системи на різні дії користувача, такі як повернення списку результатів пошуку або відображення повідомлення про підтвердження.

Відносини: способи, якими різні суб'єкти пов'язані один з одним, наприклад, як клієнти пов'язані із замовленнями або як продукти пов'язані з постачальниками.

Залежності: Зв'язки між різними елементами в системі, що показують, як зміни одного елемента можуть вплинути на інші елементи.

Умови: Конкретні обставини або умови, за яких можуть відбуватися різні дії або відповіді.

Загалом, елементи діаграми прецедентів допомагають проілюструвати структуру та поведінку системи, а також виявити будь-які потенційні проблеми чи проблеми, які можуть виникнути під час використання системи.

1.5.1.4. Морфологічний аналіз

Морфологічний аналіз у контексті проектування баз даних це розкладання складної проблеми на окремі її сутності (наприклад, лікарі, пацієнти) та взаємозв'язки(пацієнт та лікар використовуються для створення запису пацієнта до лікаря). Морфологічний аналіз може бути корисним інструментом для виявлення ключових сутностей та атрибутів(іменників) і зв'язків(дієслів) в проблемній області і для проектування бази даних, яка ефективно представляє ці сутності і зв'язки.

1.5.1.5. Ітеративний процес виділення сутностей, зв’язків та атрибутів

Ітераційний процес визначення сутностей, атрибутів і зв'язків є ключовою частиною проектування баз даних. Він передбачає визначення ключових елементів даних, які будуть зберігатися в базі даних, і організацію їх в логічні структури, які можуть бути використані для представлення реальних відносин і полегшення зберігання і пошуку даних на основі загального опису предметної області та прецедентів використання. Щоб почати цей процес, потрібно виявити сутності, які будуть зберігатися в базі даних. Сутність - це тип особи, об'єкта або речі, який необхідно описати в базі даних[4]. Наприклад, у базі даних керування ліками суб'єктами можуть бути ліки та провізори. Далі необхідно виявити атрибути кожної сутності. Атрибут - це фрагмент даних, який описує сутність. Наприклад, атрибути провізора можуть включати його ім’я, графік роботи, стаж. Після того, як сутності та їх атрибути визначені, вам визначити зв'язки між ними. Взаємозв’язок - це зв'язок між двома або більше сутностями. Наприклад, Провізор може замовляти необхідні ліки в провайдера.

1.5.1.6. Основні нотації діаграми Сутність-Зв’язок

На діаграмі зв'язків сутностей (ERD) в загальному випадку основними позначеннями, що використовуються для представлення сутностей, атрибутів і зв'язків, є:

Прямокутники: представляють сутності, назва сутності пишеться всередині прямокутника.

Еліпси: представляють атрибути, назва атрибута пишеться всередині овалу.

Ромби: представляють відносини, назва відносин написано всередині ромба.

Лінії: пов'язують сутності зі зв'язками або атрибутами. Лінія, що з'єднує сутність зі зв'язком, означає, що сутність бере участь у зв'язку. Лінія, що з'єднує сутність з атрибутом, означає, що атрибут належить сутності.

Воронячі лапки: використовуються для представлення відносин «багато-до-багатьох» [5]. Лапка ворони проведена в кінці лінії, що з'єднує дві сутності, що

вказує на те, що кожна сутність може мати кілька екземплярів іншої сутності.
Подвійні лінії: використовуються для представлення відносин один до одного між двома сутностями. Між двома сутностями проводиться подвійна лінія, яка вказує на те, що кожна сутність може мати тільки один екземпляр іншої сутності.

В нотації Чена елементи прямокутник, еліпс, ромб, а також лінія та подвійна лінія – виконують ті ж функції що і в загальному. Єдина різниця це відображення відносин «багато-до-багатьох», а також наявність штрих-пунктирної лінії:
Пунктирна лінія: використовуються для представлення відносин один(1) до багатьох(М) між двома сутностями. Між сутністю 1 і сутністю М проводиться пунктирна лінія, яка вказує на те, що сутність 1 може мати кілька екземплярів сутності М, але кожен екземпляр сутності М може мати лише один екземпляр сутності 1.
Зв’язок «багато-до-багатьох» в нотації Чена представлено двома окремими зв’язками «один-до-багатьох», з'єднаними лінією. Це робиться для того, щоб показати, що кожен екземпляр однієї сутності може бути пов'язаний з декількома екземплярами іншої сутності, і навпаки. Наприклад, скажімо, необхідно розробити базу даних для лікарні, де кожну хворобу, яка занесена до бази даних можна лікувати різними ліками, і навпаки, різні ліки можуть використовуватися для лікування різних хвороб. Щоб представити цей зв'язок у нотації Чена, необхідно намалювати два прямокутники, щоб представити сутності "хвороба" і "ліки", і з'єднати їх двома пунктирними лініями, одна з яких буде йти від "хвороби" до "ліків", а інша - від "ліків" до "хвороби". Потім необхідно провести лінію, що з'єднує два зв'язки один до багатьох, щоб показати, що вони пов'язані між собою.

1.5.2. Логічне проектування

Логічне проектування бази даних – це процес створення детального проектування бази даних, яка визначає типи даних, атрибути та зв'язки між різними сутностями даних. Метою логічного проектування бази даних є створення детального, технічного завдання бази даних, яка може бути використана для реалізації бази даних в конкретній СКБД. Логічне проектування повинно враховувати конкретні вимоги і обмеження використовуваної СКБД, а також потреби користувачів, які будуть взаємодіяти з базою даних. Процес логічного проектування баз даних зазвичай включає в себе наступні етапи:

1. Перегляд та уточнення концептуального проектування бази даних.
2. Визначення типів даних та атрибутів для кожної сутності в базі даних.
3. Визначення зв'язків між різними сутностями даних.
4. Нормалізація бази даних для зменшення надмірності та покращення цілісності даних.
5. Створення детального технічного оформлення бази даних, включно з усіма таблицями, стовпцями, ключами та зв'язками.
6. Валідація бази даних, щоб переконатися, що вона відповідає вимогам СКБД і потребам користувачів.

1.5.2.1. Аномалії зміни

Аномалії зміни в базі даних відносяться до проблем, які можуть виникнути при оновленні або зміні даних. Ці проблеми можуть виникнути, коли база даних не розроблена або не підтримується таким чином, щоб враховувати потенційну можливість змін даних з часом, та коли в ній присутня надмірна збитковість. Існує кілька типів аномалій змін, які можуть виникнути в базі даних:

Аномалія вставки: відбувається, коли новий запис не може бути доданий до таблиці, без присутності іншого атрибуту [6].

Аномалія видалення: відбувається, коли видалення запису з бази даних також видаляє пов'язані дані, необхідні для інших записів.
Аномалія оновлення: відбувається, коли оновлення запису в базі даних призводить до того, що дані стають непослідовними або неправильними і видалення одних даних відбувається не явно внаслідок видалення інших даних.

Щоб запобігти цим аномаліям, слід розробити базу даних з відповідними структурами даних та обмеженнями. Це може включати використання нормалізації для усунення надмірності та забезпечення логічного, послідовного зберігання даних. Правильні процеси управління даними, такі як регулярний перегляд та очищення даних, також можуть допомогти мінімізувати ризик виникнення аномалій.

1.5.2.2. Нормалізація

У реляційній базі даних нормалізація - це процес організації стовпців (атрибутів) і таблиць (зв'язків) в базі даних таким чином, щоб мінімізувати надмірність і залежність. Мета нормалізації полягає в тому, щоб гарантувати, що кожен фрагмент даних зберігається лише в одному місці, а залежності даних представлені через зв'язки між таблицями. Існує кілька нормальних форм, в яких може бути база даних, починаючи від першої нормальної форми (1NF) до п'ятої нормальної форми (5NF). Кожна наступна нормальна форма будується на попередніх, з метою досягнення повністю нормованої бази даних. Ідея нормалізації полягає в тому, щоб допомогти переконатися розробнику, що інформаційні елементи були розділені по відповідним таблицям[7].

Перша нормальна форма (1NF): База даних знаходиться в 1NF, якщо вона відповідає наступним правилам:

- База даних організована в таблиці (зв'язки).
- Кожна таблиця має унікальний первинний ключ.

- Кожен стовпець у таблиці містить одне значення (тобто без повторюваних груп).

Друга нормальна форма (2NF): База даних знаходиться в 2NF, якщо вона вже знаходиться в 1NF і задовольняє наступним додатковим правилам:

- Усі неключові стовпці в таблиці мають залежати від усього первинного ключа.
- Не повинно бути часткових залежностей (тобто стовпця, який залежить лише від частини первинного ключа).

Третя нормальна форма (3NF): База даних знаходиться в 3NF, якщо вона вже знаходиться в 2NF і задовольняє додатковим правилом про те, що усі неключові стовпці в таблиці мають залежати лише від первинного ключа, а не від будь-яких інших стовпців, які не належать до ключа.

Нормальна форма Бойса-Кодда (BCNF): База даних знаходиться в BCNF, якщо вона вже знаходиться в 3NF і задовольняє додаткове правило про те, що усі визначники (стовпці, які визначають значення іншого стовпця) є частиною первинного ключа.

П'ята нормальна форма (5NF): база даних знаходиться в 5NF, якщо вона знаходиться в 4NF і задовольняє додаткове правило про відсутність багатозначних залежностей (тобто стовпець, який визначається більш ніж одним стовпцем).

За допомогою нормалізації бази даних можна зменшити надмірність, покращити цілісність даних і зробити базу даних гнучкішою та простішою в обслуговуванні.

Щоб розкласти відношення на ці нормальні форми, можна використовувати різноманітні алгоритми, такі як алгоритм декомпозиції, який розбиває відношення на більш дрібні відносини, що знаходяться в бажаному

нормальному вигляді. Інші алгоритми, такі як алгоритм синтезу, можуть бути використані для реконструкції відношення з його розкладеної форми.

1.5.2.3. Ключі

У реляційній базі даних, ключ — це поле, яке однозначно ідентифікує запис. Існує кілька типів ключів, які можна використовувати в базі даних, в тому числі:

1. **Первинний ключ**: стовпець або набір стовпців, який однозначно ідентифікує кожен рядок у таблиці. Первинний ключ не може містити Null-значення та має бути унікальним для всіх рядків таблиці. Наприклад: номер соціального страхування, який не може бути повторений і є унікальним для кожної людини, яка його має[8].
2. **Зовнішній ключ**: стовпець або набір стовпців, який посилається на первинний ключ іншої таблиці. Зовнішній ключ використовується для встановлення зв'язку між двома таблицями.
3. **Потенційний ключ**: набір стовпців, які потенційно можна використовувати як первинний ключ для таблиці. Таблиця може містити кілька потенційних ключів, але лише один первинний ключ.
4. **Складений ключ**: це первинний ключ, що складається з двох або більше стовпців. Складений ключ дає змогу використовувати кілька стовпців для унікальної ідентифікації рядка в таблиці.
5. **Сурогатний ключ**: це унікальний ідентифікатор, який використовується як первинний ключ для таблиці. Сурогатні ключі часто використовуються замість природних ключів (таких як номер соціального страхування або ідентифікатор клієнта) для підвищення продуктивності та гнучкості бази даних.

1.5.2.4. Функціональна та багатозначна залежність

Функціональні залежності та багатозначні залежності - це поняття в проектуванні бази даних, які використовуються для визначення нормальної форми бази даних.

Функціональні залежності - це зв'язки між атрибутами в базі даних, які відображають той факт, що значення одного атрибута (залежного атрибута) однозначно визначається значенням іншого атрибута (визначника).

Багатозначні залежності виникають, коли один атрибут залежить від іншого атрибута, але залежний атрибут може мати кілька значень для одного значення визначника.

Функціональні залежності та багатозначні залежності використовуються в нормалізації бази даних для виявлення областей бази даних, які можуть бути надлишковими або такими, що можуть не відповідати певній нормальній формі. Розуміючи та усуваючи функціональні та багатозначні залежності, можна створити базу даних, яка є більш ефективною, гнучкішою та більш стійкою до невідповідностей даних.

1.5.2.5. Замикання атрибутів

Під поняттям замикання атрибутів розуміється набір атрибутів, які можуть бути виведені із заданого набору атрибутів у відношенні. Закриття атрибута вважається «повним», якщо воно містить всі атрибути, які можна вивести з вихідного набору атрибутів. Наприклад, розглянемо відношення з атрибутами «A», «B» і «C». Закриття атрибутів для цього відношення включатиме ці атрибути, а також будь-які інші атрибути, які можуть бути отримані з них. Якщо з "D" можна вивести з комбінації "A" і "B", а з "E" можна вивести з "C", то замикання атрибутів для цього відношення включатиме атрибути "A", "B", "C", "D" і "E". Обчислення замикання атрибутів для відношення може бути корисним при проектуванні бази даних, оскільки допомагає визначити, які атрибути є важливими для відношення, а які можуть бути отримані з інших

атрибутів. Це може допомогти забезпечити належну нормалізацію зв'язку та ефективне зберігання даних.

1.5.2.6. Діаграми класів UML етапу логічного проектування та типи зв’язків

На етапі логічного проектування бази даних діаграми класів UML можуть бути використані для моделювання понять і зв'язків, які будуть представлені в базі даних. Клас на діаграмі класів UML представляє концепцію або сутність у змодельованому домені. Класи можуть бути пов'язані асоціаціями, які представляють відносини між класами. Наприклад, клас "провізор" може бути пов'язаний з класом "замовлення", що вказує на те, що провізор може замовляти кілька різновидів ліків.

Клас асоціацій - це клас, який представляє відносини між двома іншими класами. Він з'єднаний з іншими класами лінією асоціації і містить атрибути та операції, специфічні для зв'язку. Підкласи, також відомі як похідні класи, - це класи, які успадковують атрибути та операції від батьківського класу (також відомого як базовий клас). Підкласи можуть використовуватися для представлення спеціалізації або варіації в межах домену.

Композиція і агрегація - це типи відносин, які можуть бути використані для моделювання відносин між класом і його пов’язаними елементами. Композиція вказує на те, що пов’язані елементи мають істотне значення для класу і не можуть існувати самостійно, тоді як агрегація вказує на те, що елементи можуть існувати самостійно.

Ці поняття можуть бути реалізовані в реляційній моделі даних шляхом створення таблиць для кожного класу та використання зовнішніх ключів для представлення асоціацій між класами. Класи асоціації можуть бути реалізовані у вигляді окремих таблиць, при цьому зовнішні ключі пов'язують їх з іншими класами. Підкласи можуть бути реалізовані за допомогою успадкування таблиць, в залежності від конкретних потреб бази даних. Композиція і

агрегація можуть бути представлені за допомогою зовнішніх ключів і обмежень значень NULL.

1.5.2.7. Вирішення зв’язків типу M:M

Щоб вирішити зв'язок M:M у базі даних, загальним підходом є створення нової сутності (яка називається зв'язувальною таблицею), яка з'єднує дві сутності. Ця таблиця з'єднань містить зовнішні ключі, які посилаються на первинні ключі двох сутностей, а також може містити будь-які атрибути, характерні для зв'язку. Наприклад, в базі даних ліків можна створити таблицю з'єднань під назвою "medication_ pharmacist", яка має зовнішні ключі для "medication_id" і " pharmacist_id", а також атрибут для кількості ліків, які замовив провізор. Такий підхід дозволяє представляти відносини «багато-до-багатьох» в базі даних, при цьому дотримуючись принципів нормалізації. Без нормалізації зв'язок «багато-до-багатьох» може призвести до надмірності та невідповідності даних, тому важливо правильно представляти його в дизайні бази даних.

1.5.3. Фізичне проектування

Фізичне проектування баз даних - це процес реалізації логічного проектування бази даних в певній СКБД з використанням SQL. Це завершальний етап у загальному процесі проектування баз даних, після концептуального та логічного проектування баз даних. Метою фізичного проектування баз даних є створення повнофункціональної бази даних, яка може бути використана для ефективного зберігання та отримання даних. Це передбачає створення дизайну внутрішньої схеми бази даних, що зображує таблиці даних, стовпці даних цих таблиць і зв'язки між таблицями[9]. Завантаження даних у базу даних та створення будь-яких необхідних індексів та обмежень для забезпечення цілісності даних та підвищення продуктивності. Процес фізичного проектування баз даних зазвичай включає в себе наступні етапи:

1. Перегляд та уточнення логічного оформлення бази даних.
2. Створення необхідних структур баз даних в СКБД за допомогою SQL, включаючи таблиці, стовпці, ключі і зв'язки.
3. Додавання даних в базу даних.
4. Створення будь-яких необхідних індексів та обмежень для забезпечення цілісності даних та підвищення продуктивності.
5. Тестування бази даних, щоб переконатися, що вона функціонує правильно та відповідає потребам користувачів.

1.5.3.1. Типи наборів команд SQL

У SQL існує кілька типів наборів команд, які можна використовувати для визначення та маніпулювання структурою та даними бази даних. До них відносяться:

Мова визначення даних (Data Definition Language, DDL): цей набір команд використовується для визначення схеми бази даних, включаючи створення, зміну та видалення об'єктів бази даних, таких як таблиці, індекси та подання. Приклади команд DDL включають CREATE, ALTER, DROP і TRUNCATE.

Мова запитів даних (Data Query Language, DQL): цей набір команд використовується для отримання та маніпулювання даними в базі даних. Приклад команд DQL – SELECT.

Мова керування даними (Data Control Language, DCL): цей набір команд використовується для керування доступом до даних у базі даних. Приклади команд DCL включають GRANT і REVOKE.

Мова маніпулювання даними (Data Manipulation Language, DML): цей набір команд являє собою комбінацію команд , які можна використовувати для запису та редагування даних в уже створених таблицях. Приклади команд - INSERT, UPDATE і DELETE. **Мова контролю транзакцій (Transaction Control Language, TCL):** цей набір команд використовується для управління

транзакціями в базі даних, які є послідовністю операцій з базою даних, які розглядаються як єдине ціле.

Основними є DDL та DML.

Основні команди DDL:

CREATE: використовується для створення нового об'єкта бази даних, наприклад таблиці або подання.

ALTER: використовується для зміни структури існуючого об'єкта бази даних.

DROP: використовується для видалення об'єкта бази даних.

TRUNCATE: використовується для видалення всіх даних з таблиці, але залишку структури таблиці незмінною.

Приклади команд DML включають:

SELECT: використовується для отримання даних з однієї або декількох таблиць у базі даних.

INSERT: використовується для додавання нових даних до таблиці в базі даних.

UPDATE: використовується для зміни наявних даних у таблиці бази даних.

DELETE: використовується для видалення даних із таблиці в базі даних.

1.5.3.2. Словник бази даних

Словник даних бази даних (database data dictionary) — це набір метаданих, який описує дані, що зберігаються в базі даних. Словник даних дуже важливий, оскільки містить таку інформацію, як те, що знаходиться в базі даних, кому дозволено доступ до неї, де фізично зберігається база даних [10]. Він також містить інформацію про структуру бази даних, таку як імена та типи даних таблиць і стовпців, а також зв'язки між таблицями. словник даних можна використовувати для різних цілей, включаючи:

Документування бази даних: словник даних може надати вичерпний огляд структури та вмісту бази даних, полегшуючи іншим розуміння та використання бази даних.

Підвищення продуктивності бази даних: словник даних може допомогти оптимізувати продуктивність бази даних, надаючи інформацію про структури даних і зв'язки в базі даних.

Забезпечення цілісності даних: словник даних може використовуватися для забезпечення дотримання обмежень цілісності даних, таких як унікальність та зв'язки зовнішніх ключів, щоб забезпечити точність і узгодженість даних у базі даних.

Полегшення управління даними: словник може допомогти забезпечити належне управління та використання даних у базі даних відповідно до встановлених політик та процедур.

Словник даних бази даних може бути реалізований різними способами, наприклад, за допомогою окремої таблиці бази даних або системного каталогу, або за допомогою спеціалізованого програмного засобу для словника даних.

1.5.3.3. Предикативні твердження та обмеження цілісності

У базі даних таблиці - це структури, які зберігають дані в рядках і стовпцях. Кожен рядок представляє один запис, а кожен стовпець – певний фрагмент даних для цього запису. Таблиці використовуються для впорядкування даних у базі даних і керування ними та можуть бути пов'язані одна з одною за допомогою таких зв'язків, як зовнішні ключі.

Предикативні твердження - це твердження, які описують зв'язок між двома або більше змінними. У базах даних предикативні твердження можуть використовуватися для визначення обмежень цілісності, які є правилами, що забезпечують точність і узгодженість даних в базі даних. Наприклад, предикативна інструкція може вказати, що значення в одному стовпці має бути унікальним для всіх записів у таблиці або що значення в одному стовпці має дорівнювати значенню в іншому стовпці у пов'язаній таблиці. Існує кілька обмежень цілісності бази даних:

Обмеження даних у базі даних: обмеження цілісності можуть застосовуватися лише до даних, наявних у базі даних. Якщо дані введені неправильно або відсутні, це може призвести до невідповідностей даних або помилок.

Обмеження за допомогою типів даних: обмеження цілісності можна застосовувати лише до певних типів даних, таких як цілі числа або рядки. Вони можуть не застосовуватися до інших типів даних, таких як зображення або аудіофайли.

Обмеження за допомогою обмежень предметної області: Обмеження цілісності є специфічними для предметної області даних, що зберігаються в базі даних. Вони можуть бути непридатні до інших предметних областей. Наприклад: тип ліків може бути як твердий, наприклад у вигляді таблеток, рідкий, у вигляді сиропу, але не може бути наприклад зеленим.

1.5.3.4. Тригери

Тригер - це фрагмент коду, який автоматично виконується у відповідь на певні події, такі як вставка, оновлення або видалення даних. Це процедура, керована подіями, які управляються та зберігаються системою керування базами даних [11]. Тригери можна використовувати для забезпечення дотримання бізнес-правил, підтримки цілісності даних і виконання таких завдань, як оновлення інших таблиць або надсилання електронних листів. Тригери зазвичай пишуться тією ж мовою, що і механізм баз даних, наприклад SQL для реляційної бази даних. Вони пов'язані з певною таблицею або представленням і активуються, коли в цій таблиці або представленні відбувається вказана подія. Тригери можуть бути як на рівні рядків, так і на рівні операторів. Тригери на рівні рядка виконуються один раз для кожного рядка, на який впливає тригерна подія, тоді як тригери на рівні тверджень виконуються один раз для всього твердження. Тригери часто використовуються для забезпечення дотримання обмежень цілісності даних, таких як забезпечення того, що

значення знаходиться в певному діапазоні або що значення є унікальним. Вони також можуть використовуватися для автоматичного оновлення інших таблиць або виконання інших завдань при зміні даних. Однак важливо обережно використовувати тригери, оскільки вони можуть вплинути на продуктивність бази даних, якщо вони не реалізовані належним чином.

Існує кілька типів тригерів, які можуть використовуватися в СКБД, в тому числі:

Тригери вставки: ці тригери активуються, коли новий рядок вставляється в таблицю.

Тригери оновлення: ці тригери активуються, коли оновлюється наявний рядок у таблиці.

Тригери видалення: ці тригери активуються, коли видаляється наявний рядок у таблиці.

Тригери instead-of: ці тригери активуються замість операції вставлення, оновлення або видалення в представленні.

Тригери входу: ці тригери активуються, коли користувач входить до бази даних.

Тригери на рівні сервера: ці тригери активуються у відповідь на події, що відбуваються на рівні сервера, такі як запуск або зупинка обробника баз даних.

1.5.3.5. Представлення

У реляційній базі даних представлення – це віртуальна таблиця, яка зберігає набір результатів збереженого запиту [12]. Це не фізична таблиця, яка зберігається в базі даних, а віртуальна, основана на інструкції SELECT, яка використовується для вибору даних з однієї або декількох таблиць. Представлення можуть бути використані для ряду цілей, серед яких:

Спрощення складних запитів: створивши представлення, можна інкапсулювати складну інструкцію SELECT і використовувати її так, ніби це таблиця, що полегшує запит до бази даних.

Надання іншого погляду на дані: створюючи представлення, які вибирають різні підмножини даних із таблиці або таблиць, можна надавати різні точки зору на дані.

Безпека: представлення можна використовувати для обмеження доступу до певних даних, дозволяючи користувачам лише вибирати з представлення , а не дозволяючи їм безпосередньо отримувати доступ до базових таблиць.

Цілісність даних: представлення можна використовувати для забезпечення цілісності даних, дозволяючи змінювати дані лише через представлення , а не дозволяючи прямі оновлення базових таблиць.

Щоб модифікувати дані в представленні в системі керування базами даних (СКБД), можна використовувати тригер "instead-of". Тригер " instead-of " - це особливий тип тригера, який активується замість операції вставлення, оновлення або видалення в поданні, його не можна використовувати на оригінальних таблицях.

1.5.3.6. Матеріалізовані представлення

Матеріалізоване представлення — це попередньо складена та обрахована таблиця, яка базується на операторі SELECT. Матеріалізоване представлення може бути використане для підвищення продуктивності запитів, які отримують доступ до великих обсягів даних. Основною перевагою використання матеріалізованого представлення є те, що воно дозволяє значно скоротити час, необхідний для виконання запиту, так як дані в матеріалізованому представленні попередньо обраховуються і зберігаються. Коли запит виконується в матеріалізованому вигляді, СКБД може отримати дані з попередньо складеної таблиці, замість того, щоб обчислювати

результати на льоту за допомогою базових таблиць. Важливим є те, що матеріалізовані представлення не оновлюються автоматично при зміні базових даних, тому необхідно при зміні даних та необхідності створення такого представлення – обновлювати дані, які в ньому містяться.

1.5.3.7. Індекси

Індекс- це структура даних, яка дозволяє прискорити пошук і отримання даних. Індексування прискорює запити до стовпців, створюючи вказівники на місце, де зберігаються дані в базі даних[13]. Вона працює шляхом створення окремої структури даних, яка зберігає значення з певного стовпця в таблиці разом з посиланням на відповідний рядок в таблиці. Це дозволяє базі даних швидко здійснювати пошук по індексу, щоб знайти рядок, який містить потрібні дані, замість того, щоб шукати по всій таблиці.

Індекси можуть бути створені в одному або декількох стовпцях таблиці, а також для прискорення пошуку, а також для застосування обмежень унікальності для певних стовпців. Вони також можуть бути використані для підвищення продуктивності певних типів запитів, таких як ті, що використовують пункти WHERE, JOIN або ORDER BY.

Загалом, використання індексів може покращити продуктивність бази даних за рахунок зменшення обсягу роботи, яку доводиться виконувати движку баз даних для отримання потрібних даних. Однак важливо ретельно розглянути, які стовпці індексувати та як проектувати індекси, оскільки додавання занадто великої кількості індексів або їх погане проектування насправді може негативно вплинути на продуктивність.

Існує кілька типів індексів, які можуть використовуватися в СКБД, в тому числі:

Індекси B-дерева: Це найпоширеніший тип індексів, і вони використовують збалансовану деревоподібну структуру для зберігання даних індексу. Індекси

B-дерева можна використовувати для підтримки як рівності, так і пошуку діапазонів значень даних.

Хеш-індекси: використовують хеш-функцію для зіставлення даних індексу з певним місцем в індексі. Хеш-індекси ефективні для пошуку рівності, але вони не підтримують пошук по діапазонах.

Повнотекстові індекси: використовуються для підтримки повнотекстового пошуку, і вони зберігають слова та фрази, які з'являються у стовпці, а також посилання на рядок, який містить слово або фразу.

Просторові індекси: використовуються для підтримки запитів, які включають просторові дані, такі як точки, лінії та багатокутники.

Для створення індексу в СКБД можна використовувати такий синтаксис:

```
CREATE INDEX index_name
ON table_name (column_name);
```

Також можна вказати тип індексу за допомогою конструкції USE. Наприклад, для створення індексу B-дерева можна використовувати такий синтаксис:

```
CREATE INDEX index_name
ON table_name (column_name)
USING BTREE;
```

Що стосується планів виконання запитів, СКБД використовуватиме індекс, якщо визначить, що він покращить продуктивність запиту. Оптимізатор запитів розглядатиме такі фактори, як вибірковість індексу, розмір таблиці та тип виконуваного запиту, щоб визначити, чи слід використовувати індекс. Якщо СКБД вирішить використовувати індекс, він, як правило, виконує сканування індексу, яке передбачає зчитування даних індексу з метою знаходження рядків, які відповідають критеріям запиту. Якщо індекс не

використовується, то наприклад, MySQL починає пошук з першого рядка, і проводить послідовний пошук по всій таблиці, щоб знайти відповідні рядки[7].

1.5.3.8. General assertions

Загальні твердження (general assertions) - це твердження, які робляться про клас або сукупності об'єктів або подій. Ці твердження зазвичай формулюються природною мовою, і вони можуть бути використані для вираження широкого кола понять, таких як факти, думки, переконання та гіпотези. Наприклад, нижче наведені приклади загальних тверджень, сформульованих на природній мові:

«Всі примати - ссавці».

«Всі люди – прямоходячі».

«Сонце сходить на сході і сідає на заході»

Загальні твердження можуть бути реалізовані за допомогою різних сучасних засобів, таких як:

Формальна логіка: Загальні твердження можна представити за допомогою формальної логіки, яка являє собою систему символічних позначень і правил маніпулювання символами для представлення логічних зв'язків. Формальна логіка може бути використана для представлення структури загальних тверджень, і для того, щоб зробити з них логічні висновки.

Онтологія: це формальне представлення сукупності понять та їх зв'язків у межах предметної області. Онтологію можна використовувати для представлення загальних тверджень, визначаючи поняття та відносини, які мають відношення до предметної області, і використовуючи онтологію для представлення тверджень.

Графи знань: Граф знань - це засноване на графі представлення знань, яке використовується для представлення загальних тверджень та їх зв'язків. Графи знань можуть бути використані для представлення понять і відносин, які

мають відношення до предметної області, і для представлення тверджень, які робляться щодо цих понять і відносин.

Обробка природної мови (Neural Language Processing, NLP): NLP - це область штучного інтелекту, яка зосереджується на взаємодії між людьми та комп'ютерами за допомогою природної мови. NLP можна використовувати для вилучення загальних тверджень з тексту природною мовою та представлення їх у структурованому форматі, який може бути оброблений комп'ютером.

1.5.3.9. Запити з підзапитами, функції агрегації, HAVING, зовнішнє з'єднання

Підзапит — це запит, вкладений в інший запит. Підзапит можна використовувати для повернення даних, які використовуються в зовнішньому запиті.

Функція агрегації — це функція, яка виконує обчислення за набором значень і повертає одне скалярне значення. Деякі приклади функцій агрегації включають SUM, AVG, MIN і MAX.

HAVING використовується в інструкції SELECT для визначення умови фільтрації результатів функції агрегації. HAVING використовується замість функції WHERE із агрегатними функціями[14]. Тоді як WHERE використовується для фільтрування рядків таблиці.

Зовнішнє об'єднання – це тип об'єднання, що включає рядки з однієї або обох таблиць, які не мають відповідного рядка в іншій таблиці. Існує три типи зовнішніх з'єднань:

Ліве зовнішнє об'єднання (Left Outer Join): цей тип об'єднання включає всі рядки з лівої таблиці та лише відповідні рядки з правої таблиці. Якщо збігу немає, для правого стовпця повертаються значення NULL.

Праве зовнішнє об'єднання (Right Outer Join): цей тип об'єднання включає всі рядки з правої таблиці та лише відповідні рядки з лівої таблиці. Якщо збігу немає, для лівих стовпців повертаються значення NULL.

Повне зовнішнє об'єднання (Full Outer Join): цей тип об'єднання включає всі рядки з обох таблиць і повертає значення NULL для стовпців, які не збігаються в іншій таблиці. Ось приклад запиту з підзапитом, функціями агрегації та реченням HAVING:

```
SELECT t1.column1, t1.column2,
       (SELECT COUNT(*) FROM table2 WHERE t2.column1 = t1.column1) AS
count
FROM table1 t1
GROUP BY t1.column1, t1.column2
HAVING COUNT(*) > 10;
```

Цей запит обирає два стовпці з таблиці 1, а також включає підзапит, який рахує кількість рядків у таблиці 2, які мають відповідне значення у стовпці 1. Потім зовнішній запит групує результати за стовпцем 1 і стовпцем 2 та використовує HAVING для фільтрування будь-яких груп, кількість яких менша або дорівнює 10.

1.5.3.10. NULL значення

NULL-значення - це спеціальні значення, які представляють собою відсутність значення або NULL-заповнювача.У SQL значення NULL не збігається з порожнім рядком або нульовим значенням. Значення NULL використовуються для представлення відсутніх або невідомих даних, і вони розглядаються інакше, ніж інші значення в певних ситуаціях. SQL розміщує в полі NULL-значення за відсутності користувацького значення [15].

Ось деякі моменти, які слід враховувати при роботі зі значеннями NULL в SQL: Значення NULL не дорівнюють жодному іншому значенню, включно з іншими значеннями NULL.Це означає, що всі наведені нижче вирази повертатимуть значення FALSE: NULL = NULL, NULL = 0 і NULL = 'abc'. Оператори IS NULL і IS NOT NULL можуть бути використані для перевірки

для значень NULL. Оператор IS NULL порівнює значення з NULL-значенням і повертає true, якщо порівнюване значення дорівнює Null; в іншому випадку він повертає FALSE[16]. При роботі зі значеннями NULL і операторами DML важливо враховувати, як обробляються NULL-значення. Наприклад, якщо вставити значення NULL у стовпець, який має обмеження NOT NULL, вставити його не вдасться. Аналогічно, якщо оновити стовпець до NULL-значення, будь-які рядки, які мають значення, відмінне від NULL, буде оновлено до NULL.

Висновок до розділу 1

Було проведено ознайомлення з 17 джерелами для отримання нової інформації та закріплення вже відомої. Було розглянуто основну теорію, а саме бази даних та системи управління ними, проектування бази даних: концептуальне, логічне та фізичне проєктування, які були розглянуті в деталях. Отримані теоретичні навички знадобляться в наступних розділах для чіткого та ефективного проектування бази даних.

РОЗДІЛ 2
АНАЛІТИЧНА ЧАСТИНА

2.1. Опис предметної області

Відділення психотерапії та психіатрії в лікарні відповідає за надання послуг з психічного здоров'я пацієнтам. Ці послуги можуть включати психіатричну оцінку, управління ліками, індивідуальну терапію, групову терапію та інші форми психологічного лікування.

Пацієнти відділення психотерапії та психіатрії лікарні можуть мати широкий спектр психічних захворювань, таких як депресія, тривога, шизофренія, біполярний розлад та посттравматичний стресовий розлад (ПТСР).

Відділення може бути укомплектоване командою фахівців з психічного здоров'я, включаючи психіатрів, психологів, клінічних соціальних працівників, психіатричних медсестер та інших терапевтів психічного здоров'я. Ця команда може працювати разом, щоб оцінити пацієнтів та розробити індивідуальні плани лікування.

Психіатрична лікарня або психіатричне відділення в рамках загальної лікарні - це спеціалізована лікарня, яка спеціалізується на лікуванні розладів психічного здоров'я. Ці лікарні зазвичай надають широкий спектр послуг, включаючи стаціонарну та амбулаторну допомогу, часткову госпіталізацію та інтенсивні амбулаторні програми.

Відділення також може включати такі установи, як відділення кризового втручання, де можуть бути стабілізовані пацієнти в гострих кризах психічного здоров'я, і стаціонарне відділення для пацієнтів, які потребують більш інтенсивного лікування.

Важливо мати ефективну та точну систему ведення записів у таких відділеннях, щоб пацієнти отримували найкращу можливу допомогу, надаючи лікуючим фахівцям точну та актуальну інформацію про стан пацієнта, історію лікування та його прогрес.

База даних для управління відділенням психотерапії та психіатрії в лікарні, ймовірно, повинна містити інформацію про пацієнтів, плани лікування та персонал.

Інформація про пацієнта може включати особисті ідентифікаційні дані (ім’я, дата народження, адреса), контактну інформацію, історію хвороб, поточні діагнози та історію лікування. Він також може мати функцію для відстеження призначень пацієнтів, відвідування та сеансів терапії, які були завершені.

Плани лікування, можуть включати інформацію про конкретну терапію або схему прийому ліків, призначену для кожного пацієнта, а також будь-які нотатки чи спостереження, зроблені лікуючим терапевтом або психіатром.

Інформація про персонал може включати особисті ідентифікаційні дані, кваліфікацію, графік роботи та доступність, а також навантаження на пацієнтів.

База даних також може мати функцію управління платіжною та страховою інформацією (в даному проекті опущено можливість оформлення медичного страхування), а також відстеження результатів пацієнтів з часом, щоб допомогти персоналу оцінити ефективність різних підходів до лікування.

Деякі додаткові функції, які можна розглянути залежно від потреб лікарні, - це безпечний обмін повідомленнями між персоналом, планування записів пацієнтів до лікарів, нагадування про записи пацієнтам, створення різних звітів, таких як статистика по пацієнтам, статистика по лікуванню та статистика по персоналу.

СКБД, яка є безпечною та має зручний користувальницький інтерфейс, буде хорошим вибором, оскільки допоможе персоналу ефективно керувати записами, створювати звіти та швидко приймати рішення.

В подальшому можна буде використовувати здобутки машинного навчання для аналізу даних пацієнтів для розробки ефективних планів лікування.

Типовими діями користувача-пацієнта такої системи будуть створення запису до лікаря, перегляд плану лікування, типовими діями користувача-лікаря будуть перегляд записів пацієнтів до них, перегляд історії лікування пацієнтів,

створення персонального плану лікування, створення записів для пацієнтів, що неспроможні самі їх створювати, або потребують регулярних відвідувань лікаря.

Відповідно до бази даних будуть додаватися такі потоки даних, як нові пацієнти, нові лікарі, нові історії лікування пацієнтів, нові записи пацієнтів до лікаря, нові плани лікування, нові ліки, і тп.

2.2. Приклади схожих існуючих ІС

Існує низка закордонних інформаційних медичних систем, які спеціально розроблені для управління відділенням психотерапії та психіатрії в лікарні, наразі немає саме наших вітчизняних розробок які б спеціалізувалися саме окремо на управлінні відділенням психотерапії та психіатрії в лікарні, але існують медичні інформаційні системи (МІС), які розроблені для будь-яких відділень та універсальні. Ось кілька прикладів закордонних розробок, а також 1 приклад відчизняної:

TheraNest: Ця система є веб-системою електронних медичних записів (Electronic Health Record, EHR), спеціально розробленою для лікарень спеціалізованих на психічному здоров'ї. Він включає інструменти для планування записів пацієнтів до лікарів, документування сеансів терапії, створення планів лікування та управління інформацією про рахунки та страхування пацієнтів.

MediRecords: Це платформа EHR, яка дозволяє керувати даними пацієнтів, планувати прийоми, керувати ліками та надає можливість звітності, а також безпечний обмін повідомленнями між лікарями та пацієнтами.

TherapyNotes: Ця система розроблена спеціально для практики психічного здоров'я та включає такі функції, як планування, документація, виставлення рахунків та безпечний клієнтський портал для спілкування.

EMCmed: українська медична інформаційна система (МІС), яка включає в свій функціонал системи електронних медичних записів. Вона розроблена,

щоб допомогти медичним працівникам керувати записами пацієнтів, планувати прийомами, виставляти рахунки страховим компаніям тощо. Не розповсюджена у світі, але успішно застосовується в Україні.

Це лише кілька прикладів, але на ринку закордонних ІС є багато інших медичних систем, які можна налаштувати відповідно до потреб конкретного відділення психотерапії та психіатрії в лікарні, але загалом всі вони приблизно мають один і той же функціонал, хоча деякі мають унікальний, що і дає їм перевагу в певному напрямку над іншими ІС.

2.3. Бізнес-процеси

Перелік назв прецедентів використання:

1. Реєстрація користувача в системі
2. Редагування персональних даних користувача
3. Додання персональних даних лікаря або пацієнта
4. Редагування персональних даних лікаря або пацієнта
5. Створення прийому пацієнта до лікаря
6. Перегляд прийомів до лікаря
7. Створення історії лікування пацієнта
8. Діагностування хвороби пацієнта
9. Створення персонального плану лікування
10. Підтримка бази даних ліків

Опис прецедентів використання в стислому форматі:

1. Прецедент - реєстрація користувача в системі

1.1. Користувач – людина.

Мета – створення облікового запису користувача в системі.

Завдання – надати існуючі дані про користувача.

Людина вводить обов’язкові дані в форму для реєстрації облікового запису, якщо вона хоче мати права лікаря то їй необхідно використати персональний

код, який вона повинна використати при реєстрації, що зроблено для того, щоб захистити систему від реєстрації користувачів з правами лікаря, які не є такими. Також при реєстрації користувач повинен ввести код з СМС для валідації мобільного номеру та перевірки на те, що даний користувач не робот.

2. Прецедент - Редагування персональних даних користувача

2.1. Користувач – користувач.

Мета – редагування персональних даних користувача

Завдання – редагувати дані користувача

Користувач редагує персональні дані, логін не може бути змінений, але він може бути бути створений автоматично, якщо відбувається аутинтефікація через сторонні API, так само як і всі дані будуть заповнені. Також не може бути змінений тип користувача, якщо лікар хоче бути пацієнтом та лікарем одночасно - йому необхідно створити 2 різні облікових записи з типами лікаря та пацієнта.

3. Прецедент - Додання персональних даних лікаря або пацієнта

3.1. Користувач – користувач.

Мета – додавання або редагування персональних даних пацієнта

Завдання – надати існуючі дані про пацієнта або лікаря,

Всі дані є обов’язковими для того, щоб можна було підібрати необхідного лікаря для пацієнта, деякі лікарі працюють лише з неповнолітніми пацієнтами, або навпаки.

4. Прецедент - Редагування персональних даних лікаря або пацієнта

4.1. Користувач – користувач.

Мета – Редагування персональних даних лікаря або пацієнта

Завдання – зміна персональних даних пацієнта або лікаря.

Кімната призначена для лікаря може бути змінена, також може бути змінена кваліфікація лікаря.

5. Прецедент - Створення прийому пацієнта до лікаря

5.1. Користувач – пацієнт.

Мета – Створення прийому пацієнта до лікаря пацієнтом

Завдання – створення прийому пацієнта до лікаря пацієнтом за допомогою вибору лікаря та вільної дати та часу в його графіку пацієнтом

5.2. Користувач – лікар.

Мета – Створення прийому пацієнта до лікаря лікарем

Завдання – створення прийому пацієнта до лікаря пацієнтом за допомогою вибору пацієнта та вільної дати та часу в графіці лікаря лікарем

Такий функціонал потрібен коли користувач завершив прийом у лікаря, і щоб пацієнт сам не записувався, лікар міг сам його записати. Також, якщо необхідно автомазувати проведення прийомів, тобто, наприклад зробити прийоми для певного пацієнта о 5 вечора кожного вівторка та п’ятниці, це буде можливо за допомогою функціоналу прав лікаря.

6. Прецедент - Перегляд прийомів до лікаря

6.1. Користувач – лікар.

Мета – Перегляд прийомів до лікаря

Завдання – лікар переглядає свій графік, коли до нього записані які пацієнти

Лікарю необхідна повна інформація про всіх записаних до нього пацієнтів на весь період вперед, а також історія записів пацієнтів за період минулого.

6.2. Користувач – пацієнт.

Мета – Перегляд прийомів до лікаря

Завдання – пацієнт переглядає графік потрібних лікарів, для того щоб знати на які дати та час в якого лікаря є вільний час

Пацієнт не повинен знати інформації про те, які пацієнти записані на який час та дату, лише зайняті часові проміжки

7. Прецедент - Створення історії хвороб пацієнта

7.1. Користувач – лікар.

Мета – створення історії хвороб пацієнта

Завдання – лікар записує свої спостереження про пацієнта, для подальшого аналізу та поставлення діагнозу, а також лікування пацієнта за допомогою терапії та/або медикаментів

Нерідко необхідний час для того, щоб встановити діагноз і десь необхідно зберігати замітки лікаря про пацієнта. В книгах по класифікації психічних розладів часто наводяться реальні випадки розладів з якими стикалися лікарі, і детально описані всі думки та дії лікаря з приводу конкретного випадку. Далі на основі цих даних можна робити певний статистичний аналіз.

8. Прецедент - Діагностування хвороби пацієнта

8.1. Користувач – лікар.

Мета – лікар діагностує хворобу пацієнта

Завдання – зробити можливим призначення пацієнту однієї хвороби, яку можна буде редактувати по мірі дослідження стану пацієнта

Кожний психічний розлад потребує свого лікування, тому важливо локалізувати хворобу, щоб знати як саме її лікувати, на перших порах діагноз може бути приблизним, адже зазвичай необхідно більше 10 сеансів терапій з лікарем, щоб точно визначити діагноз, але й в такому випадку точність визначення діагнозу не є стовідсотковою.

9. Прецедент - Створення персонального плану лікування

9.1. Користувач – лікар.

Мета – створити персональний план лікування для пацієнта

Завдання – зробити можливим специфікувати методи лікування певної хвороби

Важливо визначити способи лікування розладу, медикаменти та/або терапії за необхідності, а також необхідність госпіталізації до стаціонару при тяжких формах певних розладів для того щоб максимізувати ефективність лікування пацієнта.

10. Прецедент – Підтримка бази даних ліків

10.1. Користувач – адміністратор.

Мета – зробити базу даних ліків автооновлюваною

Завдання – зробити так, щоб ціни вже доданих ліків оновлювалися при оновленні їх в референсній базі даних, або додавалися нові медикаменти відповідно

Створити автоматичне оновлення бази даних ліків не складно, потрібно лише отримати доступ до бази даних ліків аптек певної країни чи регіону та проводити автоматично процес Extract-Transform-Load (ETL), для того щоб отримувати лише необхідні атрибути для ліків із загальної бази даних ліків.

Опис прецедентів використання в розгорнотому форматі:

7. Створення історії лікування пацієнта

Створення історії лікування є дуже важливим процесом в веденні електронного обліку стану здоров’я пацієнтів. Ця історія може включати, але не обмежується такою інформацією, як:

Ідентифікатор пацієнта: зовнішній ключ, що зв'язує запис історії лікування із записом пацієнта в основній таблиці.

Дата призначення лікування.

Тип лікування, який пацієнт отримував, наприклад, ліки, операція або терапія.

Постачальник лікування: ім'я постачальника, який проводив лікування, наприклад, лікаря, медсестри або терапевта.

Діагноз: Діагноз або стан, при якому було призначено лікування.

Примітки до лікування: Додаткова інформація про лікування, така як дозування, частота або будь-які побічні ефекти, з якими стикається пацієнт.

Результат: результат лікування, такий як поліпшення, відсутність змін або погіршення стану пацієнта.

Наступні кроки: Будь-які подальші дії, рекомендовані після лікування, такі як подальший прийом або додаткові тести.

5. Створення прийому пацієнта до лікаря

Створення прийомів пацієнта до лікаря є дуже важливим процесом в веденні електронного обліку стану здоров’я пацієнтів. Ці прийоми можуть включати, але не обмежується такою інформацією, як:

Унікальний ідентифікатор зустрічі.

Ідентифікатор пацієнта: зовнішній ключ, що пов'язує призначення з записом пацієнта в основній таблиці.

Ідентифікатор лікаря: зовнішній ключ, що зв'язує зустріч із записом лікаря в головній таблиці.

Дата і час призначення.

Тип призначення, наприклад, регулярний огляд, спостереження або консультація фахівця.

Причина візиту: причина призначення, наприклад, конкретна медична проблема або стан.

Статус призначення: поточний стан зустрічі, наприклад, запланований, підтверджений, скасований або завершений.

Нагадування: нагадування, яке встановлено для зустрічі, може включати електронну пошту, SMS або інші засоби нагадування.

Примітки: Будь-які додаткові примітки або інформація, пов'язана з призначенням.

Обидва списки можна продовжувати на декілька сторінок, але розробляти настільки досконалу базу даних, яка буде містити всі необхідні атрибути, та зможе позмагатися з сучасними аналогами в плані дизайну таблиць не є головним завданням курсового проекту.

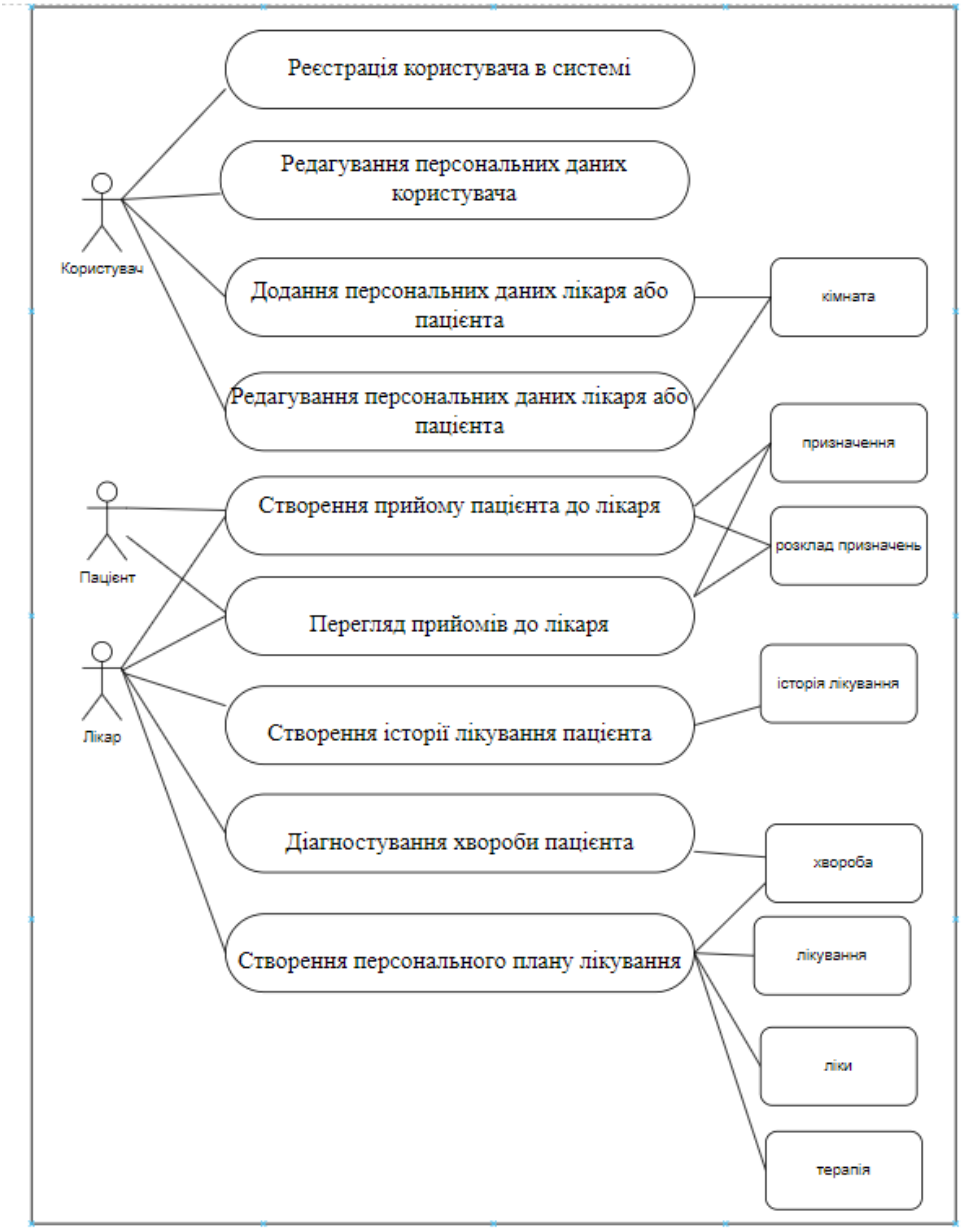

Рис.2.1 Діаграма прецедентів

2.4. Морфологічний аналіз

Іменники та їх аналіз:

Користувач – базова роль “користувача”, синонім до «людина»

Лікар – роль “користувача” з додатковими правами

Пацієнт – “користувач”, який не має права “лікар” та має свої права

Кімната - приміщення призначене для роботи в ньому “лікар”

Призначення - сутність прийому “лікар” “пацієнт”

Розклад призначень - розклад “призначення”

Ліки - медикаменти необхідні для лікування “пацієнт”

Хвороба - відхилення, або розлади, які має “пацієнт”

Історія лікування – інформацію, яку додав «лікар», що характеризує «пацієнт» на «призначення»

Лікування - методи позбавлення або полегшення “хвороба”, яку має “пацієнт”

Терапія - один з методів полегшення або позбавлення “пацієнт” від “хвороба”

Ім’я, прізвище, стать, дата народження, логін, пароль, номер телефону, email, ID пацієнта, ID лікаря - атрибути сутності “користувач”.

Посада, освіта, ID кімнати, ID лікаря, ID користувача - атрибути сутності “лікар”.

ID користувача, ID пацієнта, ID хвороби- атрибути сутності “пацієнт”.

ID кімнати, призначення, прилади - атрибути сутності “кімната”.

ID дати призначення, ID призначення, дата, час - атрибути сутності “розклад призначень”.

ID пацієнта, ID призначення, ID лікаря- атрибути сутності “призначення”.

ID хвороби, ID лікування, назва, симптоми, причини- атрибути сутності “хвороба”.

ID ліків, ID лікування, ID терапії, необхідність госпіталізації, додаткова інформація - атрибути сутності “лікування”.

ID ліків, назва, ціна, призначення - атрибути сутності “ліки”.

ID терапії, тип - атрибути сутності “терапія”.

Дієслова та їх аналіз:

1. Користувач реєструється в системі стаючи лікарем або пацієнтом, для лікаря він повинен представити персональний код, який я конфіденційним та видається для реєстрації кожному лікарю для того, щоб медична інформація була автоматизованою та потребувала мінімального втручання адміністратора для таких простих завдань, як додання нового лікаря

2. Користувач редагує пароль, номер телефону, або пошту, перш за все кожен зареєстрований лікар та пацієнт є користувачами МІС, тому вони змінюють свої дані користувача як користувачі, зміна логіну не передбачена, пароль повинен бути зашифрований, адже навіть адміністратор не повинен мати доступу до нього

3. Пацієнт додає/редагує персональні дані

4. Лікар додає/редагує персональні дані

5. Лікар створює прийом для пацієнта, якщо той знаходиться в нього на прийомі і пацієнту необхідний ще один прийом, або серія прийомів

6. Пацієнт створює прийом до потрібного лікаря, обираючи його по спеціалізації або імені

7. Лікар створює/редагує історію хвороб пацієнта, це необхідно, щоб констатувати певний психічний розлад, що передбачає певний період терапій(від 1 місяця до 0.5 року приблизно), а також запису спостережень лікаря для аналізу протікання хвороби

8. Лікар діагностує хворобу, призначає лікування, необхідні медикаменти та терапію.

2.5. Діаграма «сутність-зв’язок»

Діаграма «сутність-зв’язок» етапу концептуального проектування, а також фрагментована діаграма для кращої читаємості наведені в додатку А.

2.6. Нормалізація

4-а нормальна форма (4NF) - це рівень нормалізації бази даних, який будується поверх попередніх нормальних форм (1NF, 2NF і 3NF).Для досягнення 4NF база даних повинна відповідати наступним критеріям:

Вона повинна бути в 3-й нормальній формі (3НФ).

Вона не повинна мати будь-яких багатозначних та функціональних залежностей.

Ось як виконати нормалізацію 4НФ:

1. Визначити будь-які багатозначні або функціональні залежності в базі даних,. Знайшовши будь-які багатозначні залежності, створити нову таблицю для атрибута неосновного ключа та включити атрибути первинного ключа вихідної таблиці як зовнішні ключі в нову таблицю.
2. Перевірити всі таблиці, щоб переконатися, що вони знаходяться в 3NF, якщо ні то довести їх до 3NF.

Повторювати ітераційно процес вище, доки з бази даних не буде видалено всі багатозначні та функціональні залежності.

Єдиною причиною чому база даних ще не знаходиться в 4 нормальній формі є те, що існує M:M зв’язок між сутностями лікування та ліки, адже для кожного певного лікування певної хвороби можуть використовуватися багато ліків, і кожні ліки можуть використовуватися для безлічі лікувань. Для того, щоб нормалізувати ці відношення необхідно створити таблицю об’єднання первинних ключів обох відношень, як було описано в першому розділі.

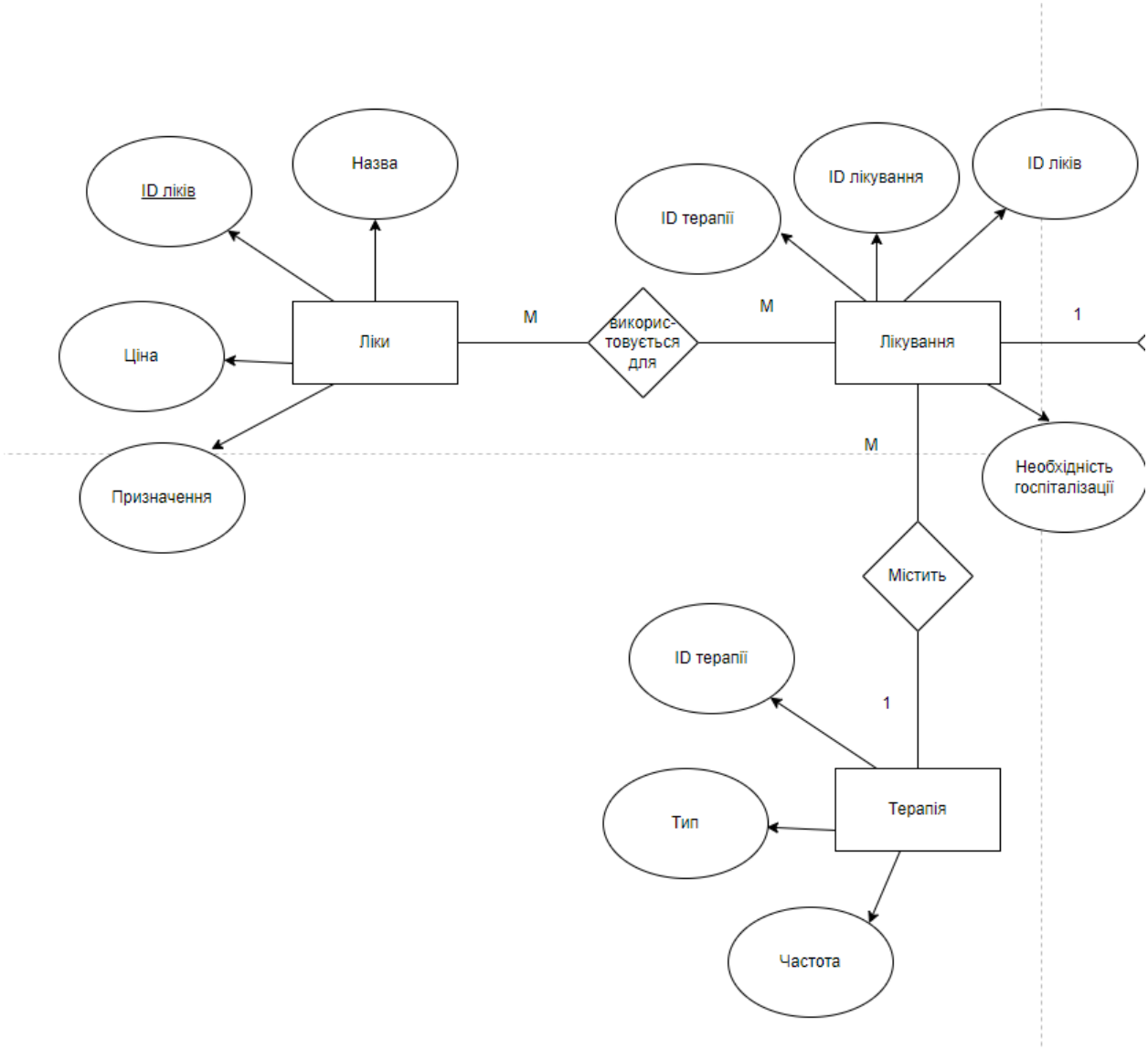

Рис.2.2 Фрагмент ER діаграми для виконаня декомпозиції

`MedicationID` - id ліків,

`Name` - назіва ліків

`Price` - ціна ліків

`TreatmentID`- id лікування,

`TherapyID`- id терапії,

`MedicationID` - id ліків,

`Hospitalization`- необхідність госпіталізації,

`TherapyID`- id терапії,

`Type` - тип ліків,

`Frequency`- частота терапії в тижень,

Опис (природньою мовою) та перелік (формули) ФЗ та БЗ

Користувач, який має певний логін, пароль, електронну пошту, мобільний номер, id пацієнта чи лікаря:

TreatmentID → TherapyID (1), MedicationID (2), Hospitalization (3),

Лікар, який має певне ім’я, прізвище, стать, дату народження, посаду, освіту та номер кімнати:

MedicationID → Name (4), Price (5)

Пацієнт, який має певне ім’я, прізвище, стать і дату народження

TherapyID → Type (6), Frequency (7)

Проведення декомпозиції

R(TreatmentID, TherapyID, MedicationID, Hospitalization, Name, Price, Type, Frequency).

Ключ TreatmentID = {MedicationID, TherapyID, Hospitalization; Name, Price; Type, Frequency}.

ФЗ 6-7 на R

R1(TherapyID, Type, Frequency)

R2(MedicationID, TherapyID, Hospitalization; Name, Price, TreatmentID)

Ключ на R1: TherapyID ФЗ 6-7

Ключ на R2: TreatmentID ФЗ 1-5

ФЗ 4-5 на R2

R3(MedicationID, Name, Price)

R4(MedicationID, TherapyID, Hospitalization, TreatmentID)

Ключ на R3: MedicationID ФЗ 4-5

Ключ на R4: TreatmentID ФЗ 1-3

ФЗ 1-3 на R4

R5(TherapyID, Hospitalization, TreatmentID)

R6(MedicationID, TreatmentID)

Ключ на R5: TreatmentID ФЗ 1-3

Ключ на R6: (MedicationID, TreatmentID) ФЗ 1-3

Результат декомпозиції:

Therapy((TherapyID, Type, Frequency)) – R1

Medication(MedicationID, Name, Price) – R3

Treatment(TherapyID, Hospitalization, TreatmentID) - R5

TreatmentMedication(TreatmentID, MedicationID) - R6

Отже, в результаті декомпозиції було отримано додаткове відношення з композитним ключем (MedicationID, TreatmentID), яке допомогло розв’язати M:M зв’язок. Таким чином створивши цю таблицю лікувань-ліків, ми видалили зв'язок M:M, розбивши його на два зв'язки 1:M, один між ліками та лікуваннями-ліками, а інший між лікуваннями та лікуваннями-ліками.

Ця база даних тепер знаходиться в 4NF, тому що:

- Кожна таблиця містить тільки атомарні значення,
- Кожна таблиця має зв'язок 1:M або 1:1*,
- Немає багатозначних і функціональних залежностей (MVD)

В ідеалі четверта нормальна форма не має мати зв’язків «один-до-одного», але видалення зв'язків 1:1 не є специфічною нормальною формою. Зв'язок 1:1 не викликає жодних аномалій оновлення або надмірності даних. Однак у деяких випадках, таких як підвищення продуктивності та зменшення складності, може існувати сенс видалити зв'язок 1:1 шляхом об'єднання таблиць. Так як, загалом, основною метою нормалізації є усунення надмірності даних і підвищення цілісності даних, оптимізація бази даних, за рахунок видалення зв'язків 1:1, є вторинним розглядом, який настає після забезпечення того, щоб дані були в нормальній формі, а зв'язки правильні. Ще одна причина наявності зв’язків 1:1 це те, що в даній роботі необхідно зпроектувати базу даних, яка повинна бути зрозумілою для замовника та легкою в проектуванні, знищення зв’язків 1:1 призведе до стрімкого зменшення кількості таблиць, а також зросте складність розуміння таблиць, адже за рахунок зменшення таблиць всі таблиці отримають атрибути знищених таблиць з якими в них існували зв’язки.

2.7. Діаграма класів UML етапу логічного проектування

Діаграма класів UML етапу логічного проектування, оновлена після етапу декомпозиції, наведена в додатку Б.

Висновок до розділу 2

Висновок до розділу. Отримані та закріплені теоретичні знання з проведення аналізу предметної області, проведено концептуальне та логічне проектування реляційної бази даних. Описано 10 прецедентів використання (2 в розгорнутому форматі). Спроектовано 12 відношень з 47 атрибутами.

РОЗДІЛ 3
ПРАКТИЧНА ЧАСТИНА

3.1. Обмеження предметної області

Обмеження природною мовою:

- ID користувача повинно мати числове значення та не може не мати значення, повинно бути головним ключем
- Ім’я користувача повинно бути рядком з довжиною до 20 символів, та не може не мати значення
- Прізвище користувача повинно бути рядком з довжиною до 20 символів, та не може не мати значення
- Стать користувача повинна бути одним символом, не може не мати значення, та повинно бути ‘F’ для жінок, ‘M’ для чоловіків, ‘O’ для осіб, стать яких інша
- День народження користувача повинно бути датою та не повинно не мати значення
- Логін користувача повинен бути рядком з довжиною до 10 символів, та не може не мати значення, повинен бути унікальним та складатися лише з латинських літер малого та великого регістру, цифр та нижніх підкреслень
- Пароль користувача повинен бути рядком з довжиною до 20 символів, та не може не мати значення, повинен складатися лише з латинських літер малого та великого регістру, цифр та нижніх підкреслень та символів: .!#$%&'*+-/=?^_`{|}~
- Email користувача повинен бути рядком з довжиною до 63 символів, та не може не мати значення, повинен бути унікальним та складатися з: перший символ - латинських літер малого та великого регістру, цифр, наступні - будь які символи, далі собачка, знову перший символ з латинських літер малого та великого регістру, цифр, далі те ж саме тільки з крапками, дефісами та нижніми підкреслюванями, далі можливо ще додати латинських літер малого та великого регістру, цифр, далі крапка і латинські літери малого та великого регістру
- Мобільний телефон користувача повинен бути рядком з довжиною до 14 символів, та не може не мати значення, повинен бути унікальним та складатися лише з +, за яким слідують цифри в кількості від 4 до 13
- Користувач повинен мати ID пацієнта та ID доктора, які мають значення 0 за замовчуванням, якщо користувач є доктором, то ID пацієнта повинно дорівнювати 0, і навпаки, якщо користувач - пацієнт

до ID лікаря = 0, інше ж ID, роль якого виконує користувач повинно мати значення більше 0, ці ID є унікальними та мають числовий тип даних, не можуть не мати значень, та виконують роль зовнішніх ключів

- ID терапії повинно мати числове значення та не може не мати значення, повинно бути головним ключем
- Тип терапії повинен бути рядком з довжиною до 30 символів та не повинен не мати значення
- Кількість прийомів на тиждень в терапії повинно бути цілим числовим значенням від 1 до 7 включно , не повинно не мати значення

- ID ліків повинно мати числове значення та не може не мати значення, повинно бути головним ключем
- Назва ліків повинна бути рядком з довжиною до 30 символів, повинна бути унікальною та не може не мати значення
- Ціна ліків повинна бути раціональним числом більшим за 0

- ID ліків повинно мати числове значення та не може не мати значення, повинно бути головним ключем
- Лікування може мати ID терапії, яке буде виступати в ролі зовнішнього ключа
- Лікування може мати ID ліків, яке буде виступати в ролі зовнішнього ключа
- Лікування повинно мати булеве значення госпіталізації, 0 якщо госпіталізація та лікування в стаціонарі не потрібне, 1 в противному випадку

- ID хвороби повинно мати числове значення та не може не мати значення, повинно бути головним ключем
- Назва хвороби повинна бути рядком з довжиною до 30 символів, повинна бути унікальною та не може не мати значення
- Симптоми хвороби повинні бути рядком з довжиною до 100 символів, не можуть не мати значення
- Причини хвороби повинні бути рядком з довжиною до 100 символів, не можуть не мати значення

- Лікування повинне мати унікальне ID лікування, яке буде виступати в ролі зовнішнього ключа

- Пацієнт може мати ID хвороби, яка була йому діагностована, це ID буде виступати в ролі зовнішнього ключа

- ID кімнати повинно мати числове значення та не може не мати значення, повинно бути головним ключем
- Спеціалізація кімнати повинна бути рядком з довжиною до 20 символів
- Інструмент кімнати повинен бути рядком з довжиною до 100 символів

- Посада лікаря повинна бути рядком з довжиною до 20 символів, та не може не мати значення
- Освіта лікаря повинна бути рядком з довжиною до 20 символів, та не може не мати значення
- Лікар може мати ID кімнати, в якій він працює, це ID буде виступати в ролі зовнішнього ключа

- ID призначення повинно мати числове значення та не може не мати значення, повинно бути головним ключем
- Дата призначення повинна бути рядком з довжиною до 20 символів, та не може не мати значення
- Призначення повинно мати ID пацієнта, який прийде на прийом, це ID буде виступати в ролі зовнішнього ключа та повинно мати числове значення

- ID часу та дати призначення повинно мати числове значення та не може не мати значення, повинно бути головним ключем
- Час та Дата призначення повинна дату типу дата, та не може не мати значення
- Час та Дата призначення повинна час типу час, та не може не мати значення
- Час та Дата призначення повинно мати унікальний ID призначення, це ID буде виступати в ролі зовнішнього ключа та повинно мати числове значення

- ID історії лікування повинно мати числове значення та не може не мати значення, повинно бути головним ключем
- Спостереження історії лікування повинно мати текстове значення з кількістю символів до 100 та не може не мати значення
- Історія лікування повинна мати унікальний ID пацієнта, це ID буде виступати в ролі зовнішнього ключа та повинно мати числове значення

Так, як в СКБД MySQL, в якій я реалізую фізичний дизайн бази даних не підтримує ASSERTIONS в режимі розробника і не повністю функціональна, обмеження предметної області були створені при розробці схем таблиць, а також за допомогою тригерів (будуть наведені далі).

Приклади реалізованих фізично обмежень природної області при створенні схем таблиць:

```
CREATE TABLE `user`(
     `UserID` INT NOT NULL AUTO_INCREMENT,
    `Login` VARCHAR(10) UNIQUE NOT NULL CHECK (`Login` REGEXP
"^[a-zA-Z0-9_]*"),
     `Password` VARCHAR(20) NOT NULL CHECK (`Password` REGEXP
"[a-zA-Z0-9.!#$%&'*+-/=?^_`{|}~]*"),
     `Email` VARCHAR(63) UNIQUE NOT NULL CHECK (`Email` REGEXP
"^[a-zA-Z0-9][a-zA-Z0-9.!#$%&'*+-/=?^_`{|}~]*?[a-zA-Z0-
9._-]?@[a-zA-Z0-9][a-zA-Z0-9._-]*?[a-zA-Z0-9]?\\.[a-zA-
Z]{2,63}$"),
     `MobileNumber` VARCHAR(14) UNIQUE NOT NULL CHECK
(`MobileNumber` REGEXP '^[+][0-9]{4,13}$'),
     `FirstName` VARCHAR(20) NOT NULL,
    `LastName` VARCHAR(20) NOT NULL,
    `Sex` CHAR(1) NOT NULL CHECK (Sex in ('M', 'F', 'O')),
    `BirthDate` DATE NOT NULL,
     PRIMARY KEY(UserID)
);


CREATE TABLE `therapy`(
     `TherapyID` INT NOT NULL AUTO_INCREMENT,
    `Type` VARCHAR(30) NOT NULL,
```

```
    `TimesPerWeek` INT NOT NULL CHECK (1 <= TimesPerWeek <= 7),
     PRIMARY KEY(`TherapyID`)
);
```

3.2. Приклади операторів створення таблиць

Найбільш показові приклади операторів створень таблиць (деякі вже були наведені в попередньому підрозділі):

```
CREATE TABLE `treatment`(
     `TreatmentID` INT NOT NULL AUTO_INCREMENT,
    `TherapyID` INT,
    `Hospitalization` BOOL NOT NULL CHECK(`Hospitalization` in (0,
1)) ,
     PRIMARY KEY(`TreatmentID`),
     FOREIGN KEY (TherapyID)
    REFERENCES therapy(TherapyID)
);

CREATE TABLE `disease`(
     `DiseaseID` INT NOT NULL AUTO_INCREMENT,
    `Name` VARCHAR(30) NOT NULL,
    `Symptoms` VARCHAR(100) NOT NULL,
    `Causes` VARCHAR(100) NOT NULL,
    `TreatmentID` INT NOT NULL,
     PRIMARY KEY(`DiseaseID`),
     FOREIGN KEY (TreatmentID)
    REFERENCES treatment(TreatmentID)
);

CREATE TABLE `therapy`(
     `TherapyID` INT NOT NULL AUTO_INCREMENT,
    `Type` VARCHAR(30) NOT NULL,
    `TimesPerWeek` INT NOT NULL CHECK (1 <= TimesPerWeek <= 7),
     PRIMARY KEY(`TherapyID`)
);
```

```
CREATE TABLE `medication`(
     `MedicationID` INT NOT NULL AUTO_INCREMENT,
    `Name` VARCHAR(30) UNIQUE NOT NULL,
    `Price` FLOAT CHECK (Price > 0),
     PRIMARY KEY(`MedicationID`)
);
```

3.3. Обмеження цілісності. Запит до словника даних

Далі наведено характеристику атрибутів таблиць та їх обмеження цілісності, обмеження природньою мовою наведено в першому підрозділі. Запит для отримання таблиці наведений в додатку Д.

Таблиця 3.1

Назва таблиці	Чи є ключем?	Назва атрибуту	Тип даних	Чи може бути NULL
appointment	PK	AppointmentID	int(10)	NOT NULL
appointment	FK	DoctorID	int(10)	NOT NULL
appointment	FK	PatientID	int(10)	NOT NULL
disease	PK	DiseaseID	int(10)	NOT NULL
disease		Name	varchar(30)	NOT NULL
disease		Symptoms	varchar(100)	NOT NULL
disease		Causes	varchar(100)	NOT NULL
disease	FK	TreatmentID	int(10)	NOT NULL
doctor	PK	DoctorID	int(10)	NOT NULL
doctor	FK	UserID	int(10)	NOT NULL
doctor		Position	varchar(20)	NOT NULL
doctor		Education	varchar(20)	NOT NULL
doctor	FK	RoomID	int(10)	NULL
medication	PK	MedicationID	int(10)	NOT NULL
medication	UK	Name	varchar(30)	NOT NULL
medication		Price	float(12)	NULL
patient	PK	PatientID	int(10)	NOT NULL
patient	FK	UserID	int(10)	NOT NULL
patient	FK	DiseaseID	int(10)	NULL
room	PK	RoomID	int(10)	NOT NULL
room		Specialization	varchar(20)	NULL
room		Tools	varchar(100)	NULL
therapy	PK	TherapyID	int(10)	NOT NULL
therapy		Type	varchar(30)	NOT NULL

therapy		TimesPerWeek	int(10)	NOT NULL
timetable	PK	TimetableID	int(10)	NOT NULL
timetable		Date	date(3)	NOT NULL
timetable		Time	time(3)	NOT NULL
timetable	FK,UK	AppointmentID	int(10)	NOT NULL
treatment	PK	TreatmentID	int(10)	NULL
treatment	FK	TherapyID	int(10)	NULL
treatment		Hospitalization	tinyint(3)	NOT NULL
treatment_history	PK	RecordID	int(10)	NOT NULL
treatment_history		Observations	varchar(100)	NOT NULL
treatment_history	FK	PatientID	int(10)	NOT NULL
treatment_medication	FK,PK	TreatmentID	int(10)	NOT NULL
treatment_medication	FK,PK	MedicationID	int(10)	NOT NULL
user	PK	UserID	int(10)	NOT NULL
user	UK	Login	varchar(10)	NOT NULL
user		Password	varchar(20)	NOT NULL
user	UK	Email	varchar(63)	NOT NULL
user	UK	MobileNumber	varchar(14)	NOT NULL
user		FirstName	varchar(20)	NOT NULL
user		LastName	varchar(20)	NOT NULL
user		Sex	char(1)	NOT NULL
user		BirthDate	date(3)	NOT NULL

Запит до словника даних для отримання переліку обмежень головного та зовнішнього ключів, а також обмежень унікальності атрибута:

```
SELECT
  table_schema,
  table_name,
  constraint_name,
  constraint_type
FROM
  information_schema.table_constraints
WHERE
  constraint_type IN ('PRIMARY KEY', 'FOREIGN KEY', 'UNIQUE') AND
  table_schema = 'psychiatric-ward'
ORDER BY
```

```
table_schema,
table_name,
constraint_name;
```

Таблиця отримана в результаті:

Таблиця 3.2

Назва таблиці	Назва обмеження	Тип обмеження
appointment	appointment_ibfk_1	FOREIGN KEY
appointment	appointment_ibfk_2	FOREIGN KEY
appointment	PRIMARY	PRIMARY KEY
disease	disease_ibfk_1	FOREIGN KEY
disease	PRIMARY	PRIMARY KEY
doctor	doctor_ibfk_1	FOREIGN KEY
doctor	doctor_ibfk_2	FOREIGN KEY
doctor	PRIMARY	PRIMARY KEY
medication	Name	UNIQUE
medication	PRIMARY	PRIMARY KEY
patient	patient_ibfk_1	FOREIGN KEY
patient	patient_ibfk_2	FOREIGN KEY
patient	PRIMARY	PRIMARY KEY
room	PRIMARY	PRIMARY KEY
therapy	PRIMARY	PRIMARY KEY
timetable	AppointmentID	UNIQUE
timetable	PRIMARY	PRIMARY KEY
timetable	timetable_ibfk_1	FOREIGN KEY
treatment	PRIMARY	PRIMARY KEY
treatment	treatment_ibfk_1	FOREIGN KEY
treatment_history	PRIMARY	PRIMARY KEY
treatment_history	treatment_history_ibfk_1	FOREIGN KEY
treatment_medication	PRIMARY	PRIMARY KEY
treatment_medication	treatment_medication_ibfk_1	FOREIGN KEY
treatment_medication	treatment_medication_ibfk_2	FOREIGN KEY
user	Email	UNIQUE
user	Login	UNIQUE
user	MobileNumber	UNIQUE
user	PRIMARY	PRIMARY KEY

3.4. Коментування таблиць та стовпців

Коментування таблиці user та двох її атрибутів Email та Login:

```
ALTER TABLE user
COMMENT 'Table of users, both patients and doctors';

ALTER TABLE user
MODIFY Email varchar(63) COMMENT 'Email of the user, accepts any
mail service';

ALTER TABLE user
MODIFY Login varchar(10) COMMENT 'Login of the user, must be
unique';
```

3.5. Приклади представлень

Представлення, яке відображає повне ім’я пацієнта, тип терапії та кількість її разів на тиждень для пацієнта:

```
CREATE VIEW therapy_info AS
SELECT CONCAT(FirstName, ' ', LastName) AS 'Full Name', Type,
TimesPerWeek
FROM therapy
JOIN treatment ON therapy.TherapyID = treatment.TherapyID
JOIN disease ON treatment.TreatmentID = disease.TreatmentID
JOIN patient ON disease.DiseaseID = patient.DiseaseID
JOIN user ON user.UserID = patient.UserID;
```

Наступне представлення об'єднує таблицю користувачів, пацієнтів, хвороб, лікування та терапії. Воно показує ім'я пацієнта, ім'я захворювання пацієнта та тип лікування, об'єднуючи дані з усіх таблиць.

```
CREATE VIEW patient_disease_treatment AS
SELECT user.FirstName, user.LastName, disease.Name, therapy.Type
FROM user
JOIN patient ON patient.UserID = user.UserID
JOIN disease ON disease.DiseaseID = patient.DiseaseID
JOIN treatment ON treatment.TreatmentID = disease.TreatmentID
JOIN therapy ON therapy.TherapyID = treatment.TherapyID;
```

Наступне представлення об'єднує таблицю пацієнта з користувачем, хворобою, лікуванням, терапією разом і відображує повне ім'я пацієнта, дату народження, поточну назву захворювання, тип терапії, що використовується для пацієнта.У поле зору також включений статус госпіталізації.

```
CREATE VIEW patient_history AS
SELECT CONCAT(FirstName, ' ', LastName) AS 'Full Name', BirthDate,
Disease.Name AS 'Disease', Hospitalization, Type
FROM patient
JOIN user ON patient.UserID = user.UserID
JOIN disease ON patient.DiseaseID = disease.DiseaseID
JOIN treatment ON disease.TreatmentID = treatment.TreatmentID
JOIN therapy ON treatment.TherapyID = therapy.TherapyID;
```

Представлення з повним зовнішнім з’єднанням:

```
CREATE VIEW available_rooms_and_doctors AS
SELECT Room.Specialization, Room.Tools, Doctor.FirstName,
Doctor.LastName
FROM room
LEFT OUTER JOIN doctor ON room.RoomID = doctor.RoomID
UNION
SELECT Room.Specialization, Room.Tools, Doctor.FirstName,
Doctor.LastName
FROM room
RIGHT OUTER JOIN doctor ON room.RoomID = doctor.RoomID;
```

Це представлення спочатку вибере всі рядки з таблиці кімнати, а також будь-які відповідні рядки з таблиці лікаря. Потім вибере усі рядки з таблиці лікаря, а також усі відповідні рядки з таблиці кімнати. А потім за допомогою об'єднання результатів ліквідує дублікати. Представлення поверне всі кімнати з наявними і відсутніми лікарями, а також поверне всіх лікарів, навіть якщо вони не закріплені за жодним приміщенням.

```
CREATE VIEW therapy_count AS
```

```
SELECT Type, COUNT(Type) AS 'Number of Patients'
FROM therapy
JOIN treatment ON therapy.TherapyID = treatment.TherapyID
JOIN disease ON treatment.TreatmentID = disease.TreatmentID
JOIN patient ON disease.DiseaseID = patient.DiseaseID
GROUP BY Type
HAVING COUNT(Type) > 5;
```

Це представлення об'єднує таблицю терапії, лікування, хвороб та пацієнтів. Воно буде використовувати функцію COUNT для підрахунку кількості пацієнтів на один тип терапії. Результати групуються за типом терапії, і пункт HAVING поверне результати лише там, де кількість пацієнтів, що проходять певну терапії перевищує 5.

MySQL не підтримує автоматично змінювані представлення, тому немає способу створити представлення, яке автоматично оновлюється, коли базові дані змінюються, лише за допомогою створення таблиці ідентичної представленню та створенні тригеру, який буде виконуватися при намаганні додавати, змінювати чи видаляти дані з представлення, і буде виконувати всі ці операції над цією таблицею. Хоча в такий спосіб можна досягти мети виконання завдання, він не є оптимальним.

3.6. Приклади тригерів

Створимо наступні тригери:

Тригер, який буде виконуватися, якщо спеціалізація кімнати буде пустим, або нульовим значенням, а також, якщо інструменти кімнати будуть рядком довшим за 100 символів:

```
DELIMITER $$
CREATE TRIGGER check_insert_room
BEFORE INSERT ON room
FOR EACH ROW
BEGIN
```

```
    IF NEW.Specialization = '' OR NEW.Specialization IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Specializtion
must not be empty or NULL';
    END IF;
    IF LENGTH(NEW.Tools) > 100 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Tools
attribute must be less than 100 characters';
    END IF;
END$$
DELIMITER ;
```

Для наступних тригерів було вирішено створити додаткові log таблиці, які будуть виконувати роль сховищ, для старих значень. Тому було створено дві таблиці user_log та doctor_log з ідентичними до user та doctor схемами.

Другий тригер буде записувати в таблицю doctor_log старі значення певного лікаря, коли його атрибути будуть оновлюватися.

```
DELIMITER $$
CREATE TRIGGER update_doctor_log
AFTER UPDATE ON Doctor
FOR EACH ROW
BEGIN
    INSERT INTO doctor_log (DoctorID, UserID, Position, Education,
RoomID)
    VALUES (OLD.DoctorID, OLD.UserID, OLD.Position, OLD.Education,
OLD.RoomID);
END$$
DELIMITER ;
```

Третій тригер буде записувати в таблицю doctor_log старі значення певного лікаря, коли буде видалятися запис з таблиці лікар.

```
DELIMITER $$
CREATE TRIGGER delete_doctor_log
AFTER DELETE ON Doctor
FOR EACH ROW
```

```
BEGIN
    INSERT INTO doctor_log (DoctorID, UserID, Position, Education,
RoomID)
    VALUES (OLD.DoctorID, OLD.UserID, OLD.Position, OLD.Education,
OLD.RoomID);
END$$
DELIMITER ;
```

Четвертий тригер буде записувати в таблицю user_log старі значення певного користувача, коли буде оновлюватися якийсь атрибут певного користувача, наприклад його прізвище, або email, цю таблицю можна буде використовувати в подальшому, наприклад у випадку, коли користувач вводить старий пароль, змінивши його, можна буде нагадати користувачу що він змінив свій пароль, і він відрізняється від старого.

```
CREATE TRIGGER update_user_log
BEFORE UPDATE ON User
FOR EACH ROW
BEGIN
    INSERT INTO user_log (UserID, Login, Password, Email,
MobileNumber, FirstName, LastName, Sex, BirthDate)
    VALUES (OLD.UserID, OLD.Login, OLD.Password, OLD.Email,
OLD.MobileNumber, OLD.FirstName, OLD.LastName, OLD.Sex,
OLD.BirthDate);
END$$
DELIMITER ;
```

3.7. Приклади індексів

Додамо 26 записів до таблиці medication, для того щоб побачити різницю в вартості виконання запитів до таблиць з індексом на ім'я певних ліків та без індексу. Виконаємо наступний запит на пошук ціни ліків з ім’ям Лоразепам.

```
SELECT price FROM medication WHERE name='Lorazepam';
```

Створемо індекс на ім’я ліків:

```
CREATE INDEX med_name
ON medication (name);
```

Виконаємо запит 100 мільйонів разів, адже через невелику кількість даних в таблиці запити будуть виконуватися неймовірно швидко, це збільшить вартість пропорційно:

```
SELECT BENCHMARK(100000000, (SELECT price FROM medication WHERE name='Lorazepam'));
```

Отримали результати:

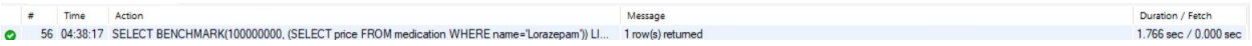

Рис. 3.1 Виконання запиту без індексу

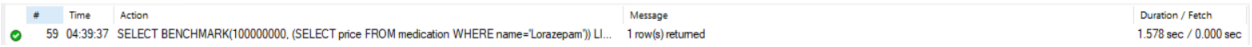

Рис. 3.2 Виконання запиту з індексом

В результаті маємо що вартість виконання запиту до індексованої таблиці приблизно на 13% менша.

Повний перелік створених індексів знаходиться в додатку Е.

Повний перелік доданих до таблиць даних знаходиться в додатку Ж.

Висновок до розділу 3

Отримано та закріплено знання з SQL. Набуто навички по роботі з СУБД MySQL. Створено 14 таблиць та 61 атрибутів.

РЕЗУЛЬТАТ РОБОТИ. ВИСНОВКИ

В ході роботи закріплені отримані теоретичні знання та практичні навички з проектування та використання БД, навички аналізу теоретичного матеріалу, навички концептуального та логічного проектування, створення діаграм, а також навички фізичного проектування, а саме розробки бази даних в СКБД MySQL. В аналітичній частині описано 10 прецедентів використання, 2 з яких — в розгорнутому форматі; надано формулювання для 48 обмежень предметної області та 20 бізнес-правил. На етапі концептуального проектування отримано 11 сутностей, 11 зв’язків та 45 атрибутів. На етапі логічного проектування в ході нормалізації до 4НФ отримано 12 відношень з 47 атрибутами. В практичній частині для забезпечення узгодженості та цілісності у відповідності із сформульованими обмеженнями та бізнес-правилами предметної області розроблено 6 обмежень цілісності, з них: 16 — унікальності (14 — первинного ключа), 11 — перевірки, 10 — зовнішнього ключа; 5 представлень та 4 тригерів. Всього в пояснювальній записці наведено 4 рисунки, 2 таблиць та 23 фрагментів (прикладів) коду

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Peterson R. What is DBMS (Database Management System)? Application, Types & Example [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/what-is-dbms.html
2. Oracle Corporation. MySQL 8.0 Reference Manual. MySQL [Електронний ресурс] – Режим доступу до ресурсу: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html
3. Microsoft. Database design basics [Електронний ресурс] – Режим доступу до ресурсу: https://support.microsoft.com/en-us/office/database-design-basics-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5
4. Embarcadero. Identifying Entities and Attributes. [Електронний ресурс] – Режим доступу до ресурсу: http://docwiki.embarcadero.com/InterBase/2020/en/Identifying_Entities_and_Attributes
5. Alalouf, Adam. Cardinality in Data Modeling. Temple University. [Електронний ресурс] – Режим доступу до ресурсу: https://community.mis.temple.edu/mis3506digitaldesignfall2018/files/2018/10/Adam-Alalouf_Cardinality.pdf
6. Prateek, T. L. What is Normalization in SQL and What are Its Types? Edureka! [Електронний ресурс] – Режим доступу до ресурсу: https://www.edureka.co/blog/normalization-in-sql/
7. Microsoft. Database Design Basics. Microsoft Support [Електронний ресурс] – Режим доступу до ресурсу: https://support.microsoft.com/en-us/office/database-design-basics-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5
8. dbBee. What is Unique Identifier and How to Use It? [Електронний ресурс] – Режим доступу до ресурсу: https://dataschool.com/sql-optimization/how-indexing-works/
9. Wambler, Scott. Data Modeling 101. Agile Data [Електронний ресурс] – Режим доступу до ресурсу: http://www.agiledata.org/essays/dataModeling101.html
10. Meador D. What is Data Dictionary [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/What-is-Data-Dictionary
11. tutorialandexample. Trigger in DBMS [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialandexample.com/trigger-in-dbms
12. includehelp. Views in Database Management System (DBMS) [Електронний ресурс] – Режим доступу до ресурсу: https://www.includehelp.com/dbms/views-in-dbms.aspx

13. Barnhill, Blake. Indexing. The Data School. [Електронний ресурс] – Режим доступу до ресурсу: https://dataschool.com/sql-optimization/how-indexing-works/
14. Harrison, Luke. SQL Cheat Sheet. Website Setup [Електронний ресурс] – Режим доступу до ресурсу: https://websitesetup.org/sql-cheat-sheet/
15. MySQL. 8.3.1 How MySQL Uses Indexes. MySQL [Електронний ресурс] – Режим доступу до ресурсу: https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html
16. GeekForGeeks. SQL NULL Values [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/sql-null-values/
17. SQL Tutorial. “SQL Logical Operators.” SQL Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.sqltutorial.org/sql-logical-operators/

Додаток А. Діаграми Сутність-Зв’язок етапу концептуального проектування.

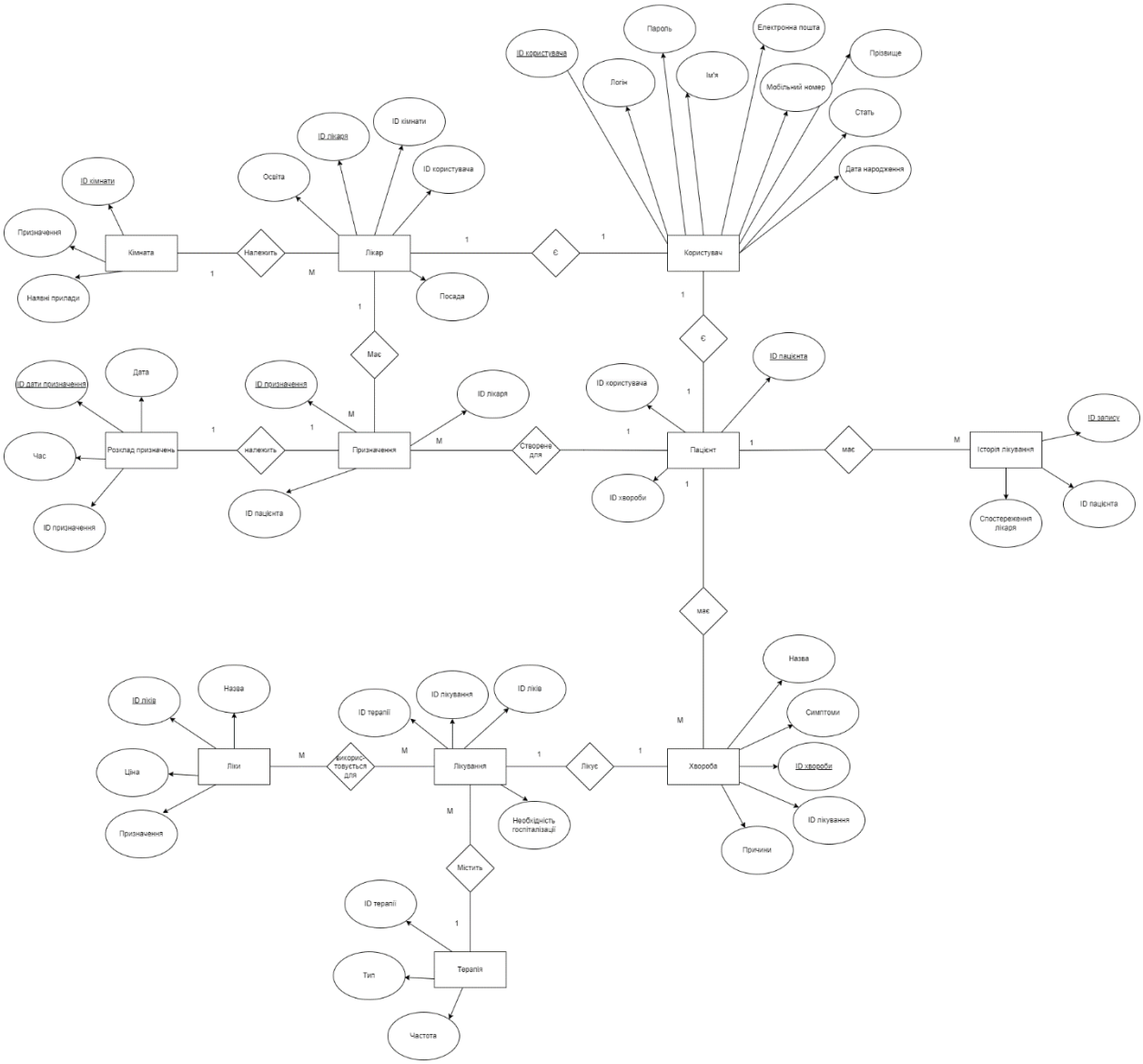

Рис А.1 Загальна діаграма сутність-зв’язок етапу концептуального проектування.

Для кращої читаємості далі наведено фрагменти вище наведеної діаграми:

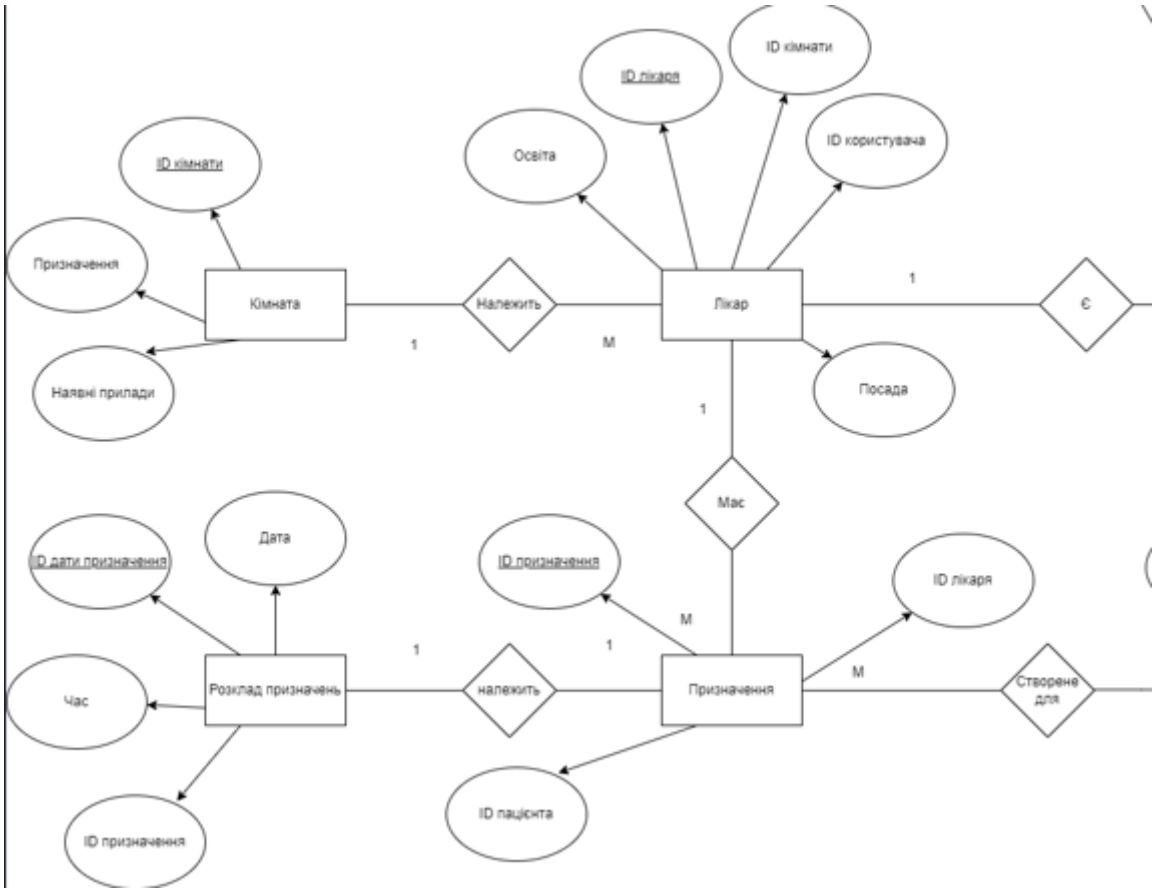

Рис А.2 Фрагмент діаграми сутність-зв’язок

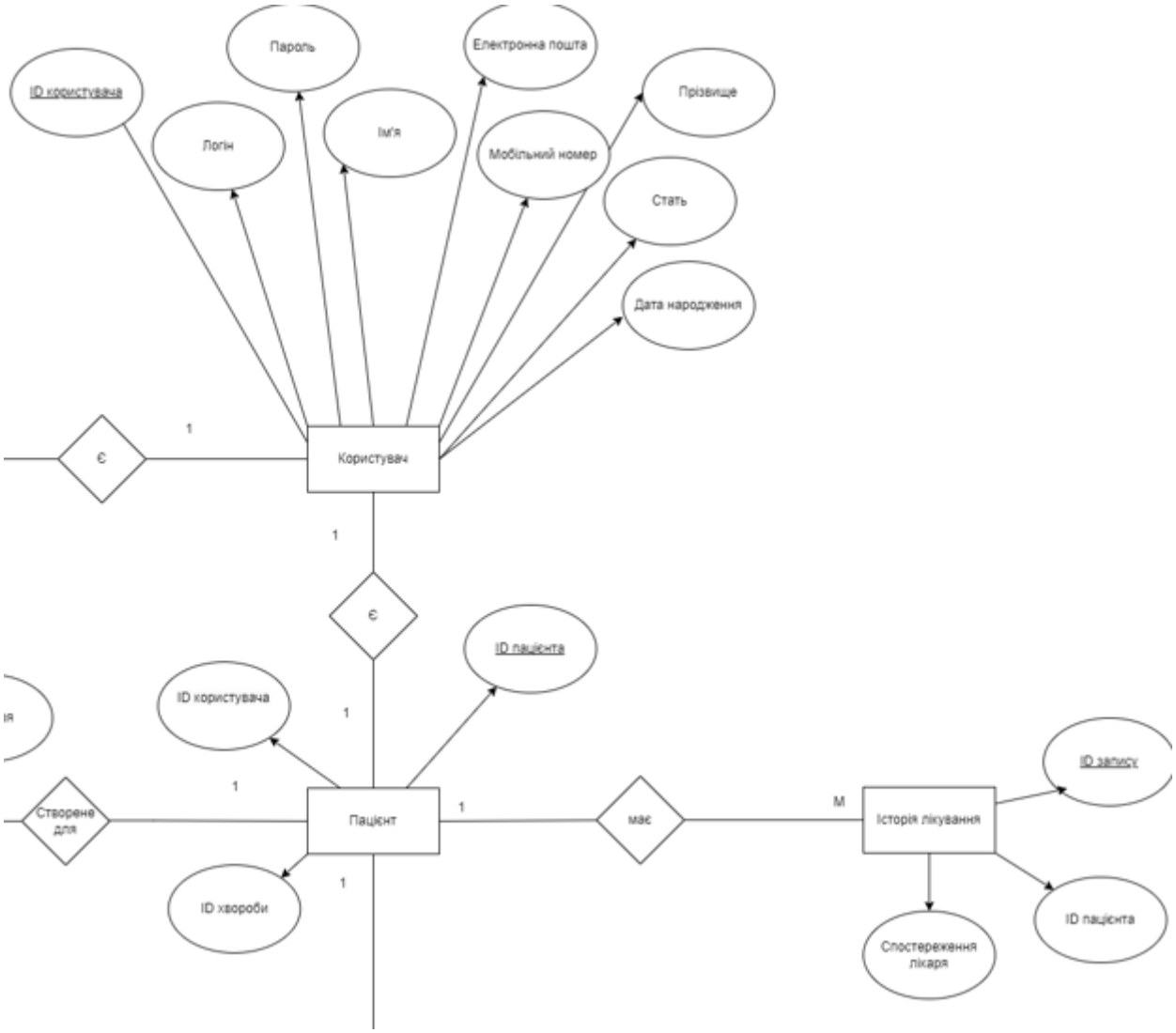

Рис А.3 Фрагмент діаграми сутність-зв’язок

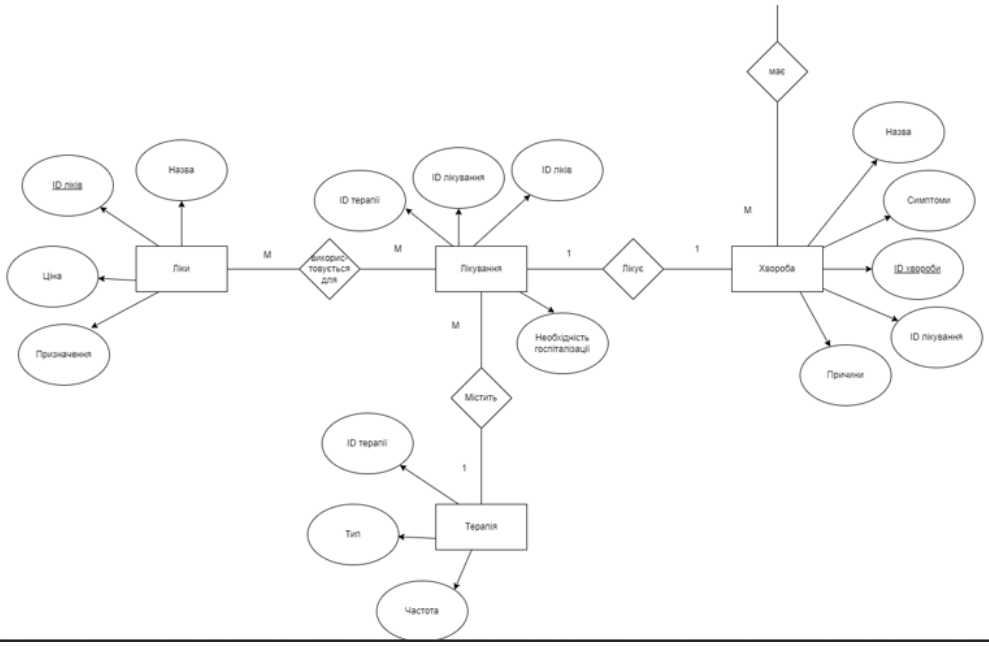

Рис А.4 Фрагмент діаграми сутність-зв’язок

Додаток Б . Діаграми класів UML етапу логічного проектування.

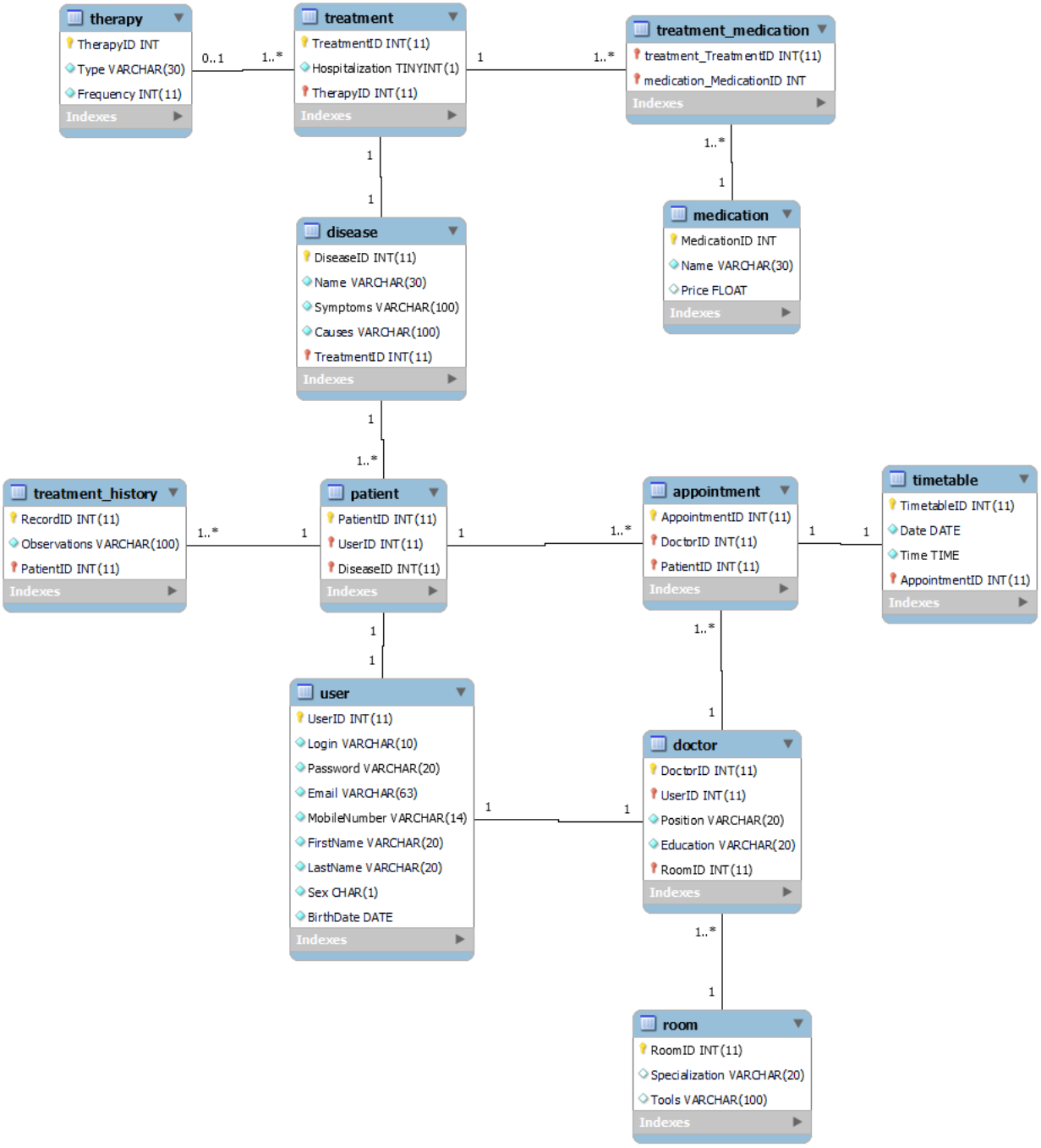

Рис Б.1 діаграма класів UML логічного проектування

Додаток В. Оператори DDL.

```
CREATE DATABASE `psychiatric-ward`;
USE `psychiatric-ward`;

CREATE TABLE `user`(
     `UserID` INT NOT NULL AUTO_INCREMENT,
    `Login` VARCHAR(10) UNIQUE NOT NULL CHECK (`Login` REGEXP
"^[a-zA-Z0-9_]*"),
     `Password` VARCHAR(20) NOT NULL CHECK (`Password` REGEXP "[a-
zA-Z0-9.!#$%&'*+-/=?^_`{|}~]*"),
     `Email` VARCHAR(63) UNIQUE NOT NULL CHECK (`Email` REGEXP
"^[a-zA-Z0-9][a-zA-Z0-9.!#$%&'*+-/=?^_`{|}~]*?[a-zA-Z0-
9._-]?@[a-zA-Z0-9][a-zA-Z0-9._-]*?[a-zA-Z0-9]?\\.[a-zA-
Z]{2,63}$"),
     `MobileNumber` VARCHAR(14) UNIQUE NOT NULL CHECK
(`MobileNumber` REGEXP '^[+][0-9]{4,13}$'),
     `FirstName` VARCHAR(20) NOT NULL,
    `LastName` VARCHAR(20) NOT NULL,
    `Sex` CHAR(1) NOT NULL CHECK (Sex in ('M', 'F', 'O')),
    `BirthDate` DATE NOT NULL,
     PRIMARY KEY(UserID)
) ;

CREATE TABLE `therapy`(
     `TherapyID` INT NOT NULL AUTO_INCREMENT,
    `Type` VARCHAR(30) NOT NULL,
    `TimesPerWeek` INT NOT NULL CHECK (1 <= TimesPerWeek <= 7),
     PRIMARY KEY(`TherapyID`)
);

CREATE TABLE `medication`(
     `MedicationID` INT NOT NULL AUTO_INCREMENT,
    `Name` VARCHAR(30) UNIQUE NOT NULL,
    `Price` FLOAT CHECK (Price > 0),
```

```
     PRIMARY KEY(`MedicationID`)
);

CREATE TABLE `treatment`(
     `TreatmentID` INT NOT NULL AUTO_INCREMENT,
    `TherapyID` INT,
    `Hospitalization` BOOL NOT NULL CHECK(`Hospitalization` in (0,
1)) ,
     PRIMARY KEY(`TreatmentID`),
     FOREIGN KEY (TherapyID)
    REFERENCES therapy(TherapyID)
);

CREATE TABLE `disease`(
     `DiseaseID` INT NOT NULL AUTO_INCREMENT,
    `Name` VARCHAR(30) NOT NULL,
    `Symptoms` VARCHAR(100) NOT NULL,
    `Causes` VARCHAR(100) NOT NULL,
    `TreatmentID` INT NOT NULL,
     PRIMARY KEY(`DiseaseID`),
     FOREIGN KEY (TreatmentID)
    REFERENCES treatment(TreatmentID)
);

CREATE TABLE `patient`(
     `PatientID` INT NOT NULL AUTO_INCREMENT,
     `UserID` INT NOT NULL,
    `DiseaseID` INT,
    PRIMARY KEY (`PatientID`),
    FOREIGN KEY (DiseaseID)
    REFERENCES disease(DiseaseID),
     FOREIGN KEY (UserID)
    REFERENCES user(UserID)
) COMMENT="Patient has personal information";
```

```
CREATE TABLE `room`(
     `RoomID` INT NOT NULL AUTO_INCREMENT,
    `Specialization` VARCHAR(20),
    `Tools` VARCHAR(100),
     PRIMARY KEY(`RoomID`)
);

CREATE TABLE `doctor`(
     `DoctorID` INT NOT NULL AUTO_INCREMENT,
     `UserID` INT NOT NULL,
    `Position` VARCHAR(20) NOT NULL,
    `Education` VARCHAR(20) NOT NULL,
    `RoomID` INT,
    PRIMARY KEY(`DoctorID`),
     FOREIGN KEY (RoomID)
    REFERENCES room(RoomID),
     FOREIGN KEY (UserID)
    REFERENCES user(UserID)
);

CREATE TABLE `appointment`(
     `AppointmentID` INT NOT NULL AUTO_INCREMENT,
    `DoctorID` INT NOT NULL,
    `PatientID` INT NOT NULL ,
     PRIMARY KEY(`AppointmentID`),
     FOREIGN KEY (DoctorID)
    REFERENCES doctor(DoctorID),
     FOREIGN KEY (PatientID)
    REFERENCES patient(PatientID)
);

CREATE TABLE `timetable`(
     `TimetableID` INT NOT NULL AUTO_INCREMENT,
```

```
    `Date` DATE NOT NULL,
    `Time` TIME NOT NULL,
    `AppointmentID` INT UNIQUE NOT NULL,
     PRIMARY KEY(`TimetableID`),
     FOREIGN KEY (AppointmentID)
    REFERENCES appointment(AppointmentID)
);

CREATE TABLE `treatment_history`(
     `RecordID` INT NOT NULL AUTO_INCREMENT,
    `Observations` VARCHAR(100) NOT NULL,
     `PatientID` INT NOT NULL,
     PRIMARY KEY(`RecordID`),
     FOREIGN KEY (PatientID)
    REFERENCES patient(PatientID)
);

CREATE TABLE `treatment_medication`(
     `TreatmentID` INT NOT NULL,
    `MedicationID` INT NOT NULL,
     PRIMARY KEY(`TreatmentID`, `MedicationID`),
     FOREIGN KEY (TreatmentID)
    REFERENCES treatment(TreatmentID),
     FOREIGN KEY (MedicationID)
    REFERENCES medication(MedicationID)
);

CREATE TABLE `user_log`(
     `UserID` INT NOT NULL AUTO_INCREMENT,
    `Login` VARCHAR(10) UNIQUE NOT NULL CHECK (`Login` REGEXP
"^[a-zA-Z0-9_]*"),
     `Password` VARCHAR(20) NOT NULL CHECK (`Password` REGEXP "[a-
zA-Z0-9.!#$%&'*+-/=?^_`{|}~]*"),
```

```
    `Email` VARCHAR(63) UNIQUE NOT NULL CHECK (`Email` REGEXP
"^[a-zA-Z0-9][a-zA-Z0-9.!#$%&'*+-/=?^_`{|}~]*?[a-zA-Z0-
9._-]?@[a-zA-Z0-9][a-zA-Z0-9._-]*?[a-zA-Z0-9]?\\.[a-zA-
Z]{2,63}$"),
    `MobileNumber` VARCHAR(14) UNIQUE NOT NULL CHECK
(`MobileNumber` REGEXP '^[+][0-9]{4,13}$'),
    `FirstName` VARCHAR(20) NOT NULL,
    `LastName` VARCHAR(20) NOT NULL,
    `Sex` CHAR(1) NOT NULL CHECK (Sex in ('M', 'F', 'O')),
    `BirthDate` DATE NOT NULL,
     PRIMARY KEY(UserID)
) ;

CREATE TABLE `doctor_log`(
     `DoctorID` INT NOT NULL AUTO_INCREMENT,
     `UserID` INT NOT NULL,
    `Position` VARCHAR(20) NOT NULL,
    `Education` VARCHAR(20) NOT NULL,
    `RoomID` INT,
    PRIMARY KEY(`DoctorID`),
     FOREIGN KEY (RoomID)
    REFERENCES room(RoomID),
     FOREIGN KEY (UserID)
    REFERENCES user(UserID)
);
```

Додаток Д. Атрибути таблиць.

```
USE `psychiatric-ward`;

WITH vars
AS (
  SELECT
    DATABASE()  AS v_SchemaName
  , 'NO'        AS v_TablesOnly
)

, baseTbl
AS (
  SELECT table_schema AS SchemaName
  , table_catalog
  , table_type, table_name, table_schema
  FROM INFORMATION_SCHEMA.TABLES
  WHERE TABLE_SCHEMA = (SELECT v_SchemaName FROM vars)
    AND (    (TABLE_TYPE = 'BASE TABLE')
          OR  ((SELECT v_TablesOnly FROM vars) = 'NO')
        )
)

, metadata
AS (
     SELECT
       bt.SchemaName     AS schema_nm
     , bt.table_name     AS table_nm
     , CASE WHEN bt.TABLE_TYPE = 'BASE TABLE' THEN 'TBL'
            WHEN bt.TABLE_TYPE = 'VIEW'  THEN 'VW'
            ELSE 'UK'
       END AS obj_typ
     , tut.ordinal_position   AS ord_pos
     , tut.column_name        AS column_nm
    , CONCAT(COALESCE(tut.data_type, 'unknown'),
```

```
      CASE WHEN tut.data_type IN('varchar','char')           THEN
CONCAT('(', tut.CHARACTER_MAXIMUM_LENGTH, ')')
            WHEN tut.data_type IN('date','time')             THEN
CONCAT('(3)')
            WHEN tut.data_type = 'datetime'                  THEN
CONCAT('(8)')
            WHEN tut.data_type = 'timestamp'                 THEN
CONCAT('(4)')
            WHEN                                    tut.data_type
in('tinyint','smallint','mediumint','int','bigint')          THEN
CONCAT('(', tut.NUMERIC_PRECISION, ')')
            WHEN tut.data_type = 'decimal'                   THEN
CONCAT('(', tut.NUMERIC_PRECISION, ',', tut.NUMERIC_SCALE, ')')
            WHEN tut.CHARACTER_MAXIMUM_LENGTH IS NOT NULL   THEN
CONCAT('(', tut.CHARACTER_MAXIMUM_LENGTH, ')')
             WHEN tut.DATETIME_PRECISION IS NOT NULL         THEN
CONCAT('(', tut.DATETIME_PRECISION, ')')
            WHEN tut.NUMERIC_PRECISION IS NOT NULL
              AND tut.NUMERIC_SCALE     IS NULL              THEN
CONCAT('(', tut.NUMERIC_PRECISION, ')')
            WHEN tut.NUMERIC_PRECISION IS NOT NULL
             AND tut.NUMERIC_SCALE      IS NOT NULL          THEN
CONCAT('(', tut.NUMERIC_PRECISION, ',', tut.NUMERIC_SCALE, ')')
             ELSE ''
    END ) AS data_typ
  , CASE WHEN tut.IS_NULLABLE = 'YES' THEN 'NULL' ELSE 'NOT NULL'
END AS nullable
  FROM       INFORMATION_SCHEMA.COLUMNS tut
  INNER JOIN baseTbl                    bt  ON bt.table_catalog =
tut.TABLE_CATALOG AND bt.table_name = tut.table_name
)

, meta_for_keys
AS (
```

```
  SELECT schema_nm, table_nm, column_nm
  , GROUP_CONCAT(is_key ORDER BY is_key SEPARATOR ',') AS is_key
  FROM (
    SELECT cons.TABLE_SCHEMA AS schema_nm
    , cons.TABLE_NAME        AS table_nm
    , kcu.COLUMN_NAME        AS column_nm
    , CASE WHEN cons.constraint_type = 'PRIMARY KEY' THEN 'PK'
           WHEN cons.constraint_type = 'UNIQUE'      THEN 'UK'
           WHEN cons.constraint_type = 'FOREIGN KEY' THEN 'FK'
           ELSE 'X'
      END AS is_key
    FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS      cons
    INNER JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE kcu
       ON cons.TABLE_SCHEMA = kcu.TABLE_SCHEMA
      AND cons.TABLE_NAME = kcu.TABLE_NAME
       AND cons.CONSTRAINT_NAME = kcu.CONSTRAINT_NAME
    WHERE cons.table_schema = (SELECT v_SchemaName FROM vars)
      AND  cons.table_name  IN(SELECT  DISTINCT  table_name  FROM
baseTbl)
      AND    cons.constraint_type    IN('PRIMARY    KEY','FOREIGN
KEY','UNIQUE')
    GROUP BY cons.TABLE_SCHEMA, cons.TABLE_NAME, kcu.COLUMN_NAME,
cons.constraint_type
  ) t
  GROUP BY schema_nm, table_nm, column_nm
)

, col_comm
AS (
     SELECT TABLE_SCHEMA AS SCHEMA_NM
     , TABLE_NAME        AS TABLE_NM
     , COLUMN_NAME       AS COLUMN_NM
     , COLUMN_COMMENT    AS column_descr
     FROM INFORMATION_SCHEMA.COLUMNS
```

```
     WHERE table_schema = (SELECT v_SchemaName FROM vars)
       AND table_name IN(SELECT DISTINCT table_name FROM baseTbl)
)

SELECT md.SCHEMA_NM, md.TABLE_NM, md.OBJ_TYP
, md.ORD_POS AS ord
, COALESCE(pk.is_key, ' ') AS is_key
, md.COLUMN_NM, md.DATA_TYP, md.NULLABLE, c.column_descr
FROM      metadata      md
LEFT JOIN meta_for_keys pk ON pk.SCHEMA_NM = md.SCHEMA_NM AND
pk.TABLE_NM = md.TABLE_NM AND pk.COLUMN_NM = md.COLUMN_NM
LEFT JOIN col_comm        c  ON c.SCHEMA_NM  = md.SCHEMA_NM AND
c.TABLE_NM  = md.TABLE_NM AND c.COLUMN_NM  = md.COLUMN_NM
ORDER BY md.SCHEMA_NM, md.TABLE_NM, md.ORD_POS
```

Додаток Е. Перелік індексів.

```
CREATE INDEX user_login_index ON user(Login);
CREATE INDEX user_email_index ON user(Email);
CREATE INDEX user_mobile_index ON user(MobileNumber);
CREATE INDEX therapy_type_index ON therapy(Type);
CREATE INDEX medication_name_index ON medication(Name);
CREATE INDEX disease_name_index ON disease(Name);
CREATE INDEX room_specialization_index ON room(Specialization);
```

Додаток Ж. Оператори DML, заповнення таблиць даними

```
INSERT INTO user (Login, Password, Email, MobileNumber, FirstName,
LastName, Sex, BirthDate) VALUES
("user1", "password1", "user1@example.com", "+1234567890", "John",
"Doe", "M", "1990-01-01"),
("user2", "password2", "user2@example.com", "+1234567891", "Jane",
"Doe", "F", "1992-02-02"),
("user3", "password3", "user3@example.com", "+1234567892", "Bob",
"Smith", "M", "1995-03-03"),
("user4",   "password4",   "user4@example.com",   "+1234567893",
"Samantha", "Johnson", "F", "1997-04-04"),
("user5", "password5", "user5@example.com", "+1234567894", "Mike",
"Williams", "M", "1999-05-05"),
("user6",   "password6",   "user6@example.com",   "+1234567895",
"Emily", "Jones", "F", "2001-06-06"),
("user7",   "password7",   "user7@example.com",   "+1234567896",
"Jacob", "Brown", "M", "2003-07-07"),
("user8",   "password8",   "user8@example.com",   "+1234567897",
"Ashley", "Miller", "F", "2005-08-08"),
("user9",   "password9",   "user9@example.com",   "+1234567898",
"Daniel", "Moore", "M", "2007-09-09"),
("user10",  "password10",  "user10@example.com",  "+1234567899",
"Jessica", "Taylor", "F", "2009-10-10");

INSERT INTO therapy (Type, TimesPerWeek) VALUES
("Cognitive Behavioral Therapy", 3),
("Art therapy", 2),
("Music therapy", 1),
("Psychoanalytic therapy", 4),
("Individual therapy", 5),
("Group therapy", 2),
("Behavioral therapy", 3),
("Family therapy", 1),
```

```
("Hypnotherapy", 2),
("Neurofeedback therapy", 3);

INSERT INTO medication (name, price) VALUES
  ('Aspirin', 5.99),
  ('Ibuprofen', 6.99),
  ('Acetaminophen', 4.99),
  ('Amoxicillin', 12.99),
  ('Metformin', 9.99),
  ('Lisinopril', 13.99),
  ('Atorvastatin', 18.99),
  ('Losartan', 14.99),
  ('Levothyroxine', 17.99),
  ('Simvastatin', 15.99),
  ('Metoprolol', 16.99),
  ('Sertraline', 19.99),
  ('Omeprazole', 11.99),
  ('Esomeprazole', 8.99),
  ('Gabapentin', 10.99),
  ('Trazodone', 7.99),
  ('Citalopram', 6.99),
  ('Alprazolam', 4.99),
  ('Clonazepam', 12.99),
  ('Diazepam', 9.99),
  ('Lorazepam', 13.99),
  ('Nitrofurantoin', 18.99),
  ('Furosemide', 14.99),
  ('Hydrochlorothiazide', 17.99),
  ('Metronidazole', 15.99),
  ('Azithromycin', 16);

INSERT INTO treatment (TherapyID, Hospitalization) VALUES
(1, 1),
(2, 0),
```

```
(3, 1),
(4, 0),
(5, 1),
(6, 0),
(7, 1),
(8, 0),
(9, 1),
(10, 0);

INSERT INTO disease (Name, Symptoms, Causes, TreatmentID) VALUES
("Depression", "Loss of interest, hopelessness, change in appetite or weight, insomnia, lack of energy", "Genetics, biology, environment", 1),
("Anxiety", "Restlessness, fatigue, trouble concentrating, muscle tension, sleep disturbance", "Trauma, life events, genetics", 2),
("Schizophrenia", "Hallucinations, delusions, disordered thinking, disordered speech, negative symptoms", "Genetics, brain chemistry", 3),
("Bipolar disorder", "Manic episodes, depressive episodes, mixed episodes", "Genetics, brain chemistry", 4),
("Post-traumatic stress disorder", "Re-experiencing the trauma, avoidance, negative changes in mood, changes in arousal and reactivity", "Trauma", 5),
("Eating disorder", "Abnormal eating habits, changes in weight, preoccupation with weight and food", "Genetics, environment, brain chemistry", 6),
("Obsessive-compulsive disorder", "Obsessions, compulsions", "Genetics, brain chemistry, environment", 7),
("Panic disorder", "Sudden and unexpected panic attacks", "Genetics, environment, brain chemistry", 8),
("Drug addiction", "Compulsive drug seeking, difficulty controlling use, continued use despite harm", "Genetics, environment, brain chemistry", 9),
```

```
("Gambling addiction", "Compulsive gambling, difficulty controlling use, continued use despite harm", "Genetics, environment, brain chemistry", 10);

INSERT INTO room (Specialization, Tools) VALUES
("Rehabilitation", "Therapy couch, resistance bands, balance board"),
("Therapy", "Whiteboard, markers, beanbags"),
("Group therapy", "Couch, armchair, tables"),
("Observation", "Bed, monitor, call button"),
("Research", "Computer, scanner, lab equipment"),
("Emergency", "Bed, defibrillator, oxygen tanks"),
("Intensive care", "Bed, ventilator, monitor"),
("Surgery", "Operating table, anesthesia machine, surgical instruments"),
("Imaging", "CT scanner, MRI machine, X-ray machine"),
("Pharmacy", "Dispensing cabinet, refrigerator, scales");

INSERT INTO patient (UserID, DiseaseID) VALUES
(1, 1),
(2, 2),
(3, 3),
(4, 4),
(5, 5),
(6, 6),
(7, 7),
(8, 8),
(9, 9),
(10, 10);

INSERT INTO doctor (UserID, Position, Education, RoomID) VALUES
(1, "Psychiatrist", "MD", 1),
(2, "Therapist", "MSW", 2),
(3, "Nurse", "BSN", 3),
```

```
(4, "Social worker", "MSW", 4),
(5, "Researcher", "PhD", 5);
```