

Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»

Кафедра інформаційних технологій

КУРСОВИЙ ПРОЕКТ

з дисципліни БАЗИ ДАНИХ

на тему: Проектування та розробка бази даних «Медична картотека, картотека хворого»

Студента 3 курсу, групи ІПЗ-3
напряму підготовки (спеціальності)
Інженерія програмного
забезпечення

Матіїв Т.Р.

Керівник кандидат технічних наук,
доцент кафедри інформаційних
технологій

Аннич А.Б.

Національна шкала: _____

Університетська шкала: _____

Оцінка ECTS: _____

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

(підпис) (прізвище та ініціали)

(підпис) (прізвище та ініціали)

м. Івано-Франківськ - 2018 рік

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

ЗАВДАННЯ НА КУРСОВИЙ ПРОЕКТ

(прізвище, ім'я, по батькові студента)

Кафедра _____ Дисципліна _____

Спеціальність _____ Курс _____ Група _____ Семестр _____

1. Тема проекту _____

2. Рекомендована література _____

3. Перелік питань, які підлягають розробці _____

4. Дата видачі завдання _____

Термін подачі до захисту _____

5. Студент _____ Керівник _____

КАЛЕНДАРНИЙ ПЛАН

[illegible]

РЕФЕРАТ

Пояснювальна записка: 29 сторінки (без додатків), 10 рисунків, 8 джерел, 2 додатки на 8 сторінках.

Ключові слова: БАЗА ДАНИХ, СИСТЕМА УПРАВЛІННЯ БАЗАМИ ДАНИХ, PostgreSQL, pgAdmin 4, ER-МОДЕЛЬ, ER-ДІАГРАМА, DUMP.

Об'єктом дослідження є база даних «Медична картотека».

Мета роботи – спроектувати та розробити необхідні компоненти бази даних «Медична картотека».

Стислий опис тексту пояснювальної записки:

У даному курсовому проекті описано основні етапи проектування та розробки бази даних засобами PostgreSQL, такі як: вибір моделі БД та СУБД, визначення переваг та недоліків середовищ розробки, проектування БД у вигляді ER-діаграми, реалізація БД на основі діаграми та приклади роботи з готовою базою даних.

ABSTRACT

Explanatory note: 29 pages (without appendixes), 6 photo, 8 sources, 2 attachments on 8 pages.

Keywords: DATA BASIS, DATA BASED MANAGEMENT SYSTEM, PostgreSQL, pgAdmin 4, ER-MODEL, ER-DIAGRAM, DUMP.

The object of research is the database "Medical card index".

The purpose of the work is to design and develop the necessary components of the "Medical Card" database.

Brief description of the text of the explanatory note:
x

This course project describes the main stages of designing and developing a database using PostgreSQL tools, such as: selecting a database model and DBMS, defining the advantages and disadvantages of development environments, designing a database in the form of ER diagrams, implementing a database based on a diagram and examples of work with the finished database. data.

ЗМІСТ

ВСТУП	7
Розділ 1. ІНСТРУМЕНТАРІЙ КУРСОВОЇ РОБОТИ	8
1.1 Загальний опис та можливості мови SQL	8
1.2 Правила побудови і нормалізації баз даних. Денормалізація	10
1.3 База даних PostgreSQL	14
1.4 Основні можливості PostgreSQL	15
Розділ 2. ОПИС І ПОБУДОВА СТРУКТУРИ БАЗИ ДАНИХ	17
2.1 Постановка задачі	17
2.2 Перелік проблем для вирішення	17
2.3 Конструювання структури	19
Розділ 3. РЕАЛІЗАЦІЯ ПРОЕКТУ	25
3.1 Створення таблиць і зв'язків між ними	25
3.2 Заповнення створених таблиць	27
ЗАГАЛЬНІ ВИСНОВКИ	28
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	29
ДОДАТКИ	30
Додаток А	30
Додаток В	39

					КП.ІПЗ-12.ПЗ							
Зм.	Арк.	№ докум.	Підпис	Дата	Проектування та розробка бази даних «Медична картотека, картотека хворого»				Літ.	Аркуш	Аркуші	
Розроб.		Матіїв Т.Р.										
Перев.		Козич О.В								6	40	
Н. контр.									ПНУ ІПЗ-3			
Затверд.		Аннич А.М.										

ВСТУП

В даній курсовій роботі представлено реалізацію бази даних електронного словника, перекладача на основі мови SQL.

Для розробки використовувалась консольна версія PostgreSQL. Усі таблиці і діаграма взаємодії таблиць розроблялися в стандартній консолі.

Основними завданнями медичної картотеки є:

1. Перелік пацієнтів
2. Історія кожного пацієнта
3. Перелік пацієнтів які були госпіталізовані, відвідували лікаря або ж були на інфекції
4. Хвороби на які хворіли пацієнти, дата перший симптомів захворювання, дата повного одужання, рецепт по якому було проведено лікування.

Під час розробки було створено 3 пацієнта. Для першого пацієнта історія містила інформацію про те що він прийняв ін'єкцію. Для другого пацієнта в історії записано, що він був на прийомі у лікаря. Третій пацієнта було госпіталізовано в лікарню із ознаками внутрішнього крововиливу.

					КП.ІПЗ-12.ПЗ	Арк.
						7
Зм.	Арк.	№ докум.	Підпис	Дата		

1 ІНСТРУМЕНТАРІЙ КУРСОВОЇ РОБОТИ

1.1 Загальний опис та можливості мови SQL

SQL (англ. *Structured query language* – мова структурованих запитів) — декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних. Сама по собі SQL не є ані системою керування базами даних, ані окремим програмним продуктом. На відміну від дійсних мов програмування (C або Pascal), SQL може формувати інтерактивні запити або, будучи вбудованою в прикладні програми, виступати як інструкції для керування даними. Окрім цього, стандарт SQL містить функції для визначення зміни, перевірки та захисту даних.

SQL – це діалогова мова програмування для здійснення запиту і внесення змін до бази даних, а також керування базами даних. Багато баз даних підтримує SQL з розширеннями до стандартної мови. Ядро SQL формує командна мова, яка дозволяє здійснювати пошук, вставку, оновлення і видалення даних за допомогою використання системи керування і адміністративних функцій. SQL також включає CLI (Call Level Interface) для доступу і керування базами даних дистанційно.

Критика SQL включає відсутність крос-платформенності, невідповідну обробку відсутніх даних. Часто це неоднозначна граматики і семантика мови.

Мова SQL поділяється на кілька видів елементів:

- *Пункти (диз'юнкти)* (англ. *Clauses*), що є складовими частинами інструкцій та запитів. (Іноді вони не обов'язкові.)
- *Вирази* (англ. *Expressions*), які можуть генерувати скалярні значення, або таблиці з стовпчиками і рядками даних
- *Предикати* (англ. *Predicates*), які описують умови, результатом яких є значення тризначної логіки SQL (true/false/unknown) або

					КП.ІІЗ-12.ІЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		8

- *Булеві* значення істинності і які використовуються для обмеження ефекту інструкцій та запитів, або для зміни потоку виконання програми.
- *Запити* (англ. *Queries*), які отримують дані на основі заданих критеріїв.
- *Інструкції* (англ. *Statements*), які чинять дію на схему даних чи самі дані, або контролюють транзакції, потік виконання програми, з'єднання, сесії, та виконують діагностику. Інструкції SQL також включають крапку з комою (";") для позначення кінця інструкції. Хоча вона не є обов'язковою на кожній платформі, вона описується як стандартна частина граматики SQL.
- *Незначимі пропуски* загалом ігноруються в інструкціях і запитах SQL, дозволяючи формувати код SQL з метою покращення читабельності.

SQL (Structured query language — мова структурованих запитів), складається з:

- **DDL** (Data Definition Language) — робота зі структурою бази,
- **DML** (Data Manipulation Language) — робота з рядочками,
- **DCL** (Data Control Language) — робота з правами,
- **TCL** (Transaction Control Language) — робота з транзакціями.

Data Definition Language

- **CREATE** — створення об'єкта (наприклад, таблиці);
- **ALTER** — зміна об'єкта (наприклад, додавання/зміна полів таблиці);
- **DROP** — видалення об'єкта.

Data Manipulation Language

- **INSERT** — вставлення даних;
- **SELECT** — вибірка даних;
- **UPDATE** — зміна даних;
- **DELETE** — видалення даних.

					КП.ІІЗ-12.ІІЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		9

Data Control Language

- GRANT — надання прав користувачу;
- DENY — явна заборона для користувача;
- REVOKE — відміна заборони/дозволу користувачу.

Transaction Control Language

- BEGIN TRANSACTION — почати транзакцію;
- COMMIT — прийняти зміни прийняті в транзакції;
- ROLLBACK — відкат.

1.2 Правила побудови і нормалізації баз даних. Денормалізація

Для правильного моделювання бази даних потрібно слідувати алгоритму нормалізації бази даних.

Нормалізація схеми бази даних — покроковий процес розбиття одного відношення (на практиці: таблиці) відповідно до алгоритму нормалізації на декілька відношень на базі функціональних залежностей.

Нормальна форма — властивість відношення в реляційній моделі даних, що характеризує його з точки зору надмірності, яка потенційно може призвести до логічно помилкових результатів вибірки або зміни даних. Нормальна форма визначається як сукупність вимог, яким має задовольняти відношення.

Таким чином, схема реляційної бази даних переходить у першу, другу, третю і так далі нормальні форми. Якщо відношення відповідає критеріям нормальної форми n та всіх попередніх нормальних форм, тоді вважається, що це відношення знаходиться у нормальній формі рівня n .

Перелік нормальних форм:

Перша нормальна форма

Перша нормальна форма (1НФ, 1NF) утворює ґрунт для структурованої схеми бази даних:

					КП.ІІЗ-12.ІЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		10

- Кожна таблиця повинна мати основний ключ: мінімальний набір колонок, які ідентифікують запис.
- Уникнення повторень груп (категорії даних, що можуть зустрічатись різну кількість разів в різних записах) правильно визначаючи неключові атрибути.
- Атомарність: кожен атрибут повинен мати лише одне значення, а не множину значень.

Друга нормальна форма

Друга нормальна форма (2НФ, 2NF) вимагає, аби дані, що зберігаються в таблицях із композитним ключем, не залежали лише від частини ключа:

- Схема бази даних повинна відповідати вимогам першої нормальної форми.
- Дані, що повторно з'являються в декількох рядках, виносяться в окремі таблиці.

Третя нормальна форма

Третя нормальна форма (3НФ, 3NF) вимагає, аби дані в таблиці залежали винятково від основного ключа:

- Схема бази даних повинна відповідати всім вимогам другої нормальної форми.
- Будь-яке поле, що залежить від основного ключа та від будь-якого іншого поля, має вноситись в окрему таблицю.

Нормальна форма Бойса-Кодда

Відношення знаходиться в НФБК тоді і лише тоді, коли детермінант кожної функціональної залежності є потенційним ключем. Якщо це правило не виконується, то, щоб привести вказане відношення до НФБК, його слід розділити на два відношення шляхом двох операцій проекції на кожну

					КП.ІІЗ-12.ІЗ	Арк.
						11
Зм.	Арк.	№ докум.	Підпис	Дата		

функціональну залежність, детермінант якої не є потенційним ключем:

1. Проекція без атрибутів залежної частини такої функціональної залежності;
2. Проекція на всі атрибути цієї функціональної залежності.

Визначення НФБК не потребує жодних умов попередніх нормальних форм. Якщо проводити нормалізацію послідовно, то в переважній більшості випадків при досягненні 3НФ автоматично будуть задовольнятися вимоги НФБК. 3НФ не збігається з НФБК лише тоді, коли одночасно виконуються такі 3 умови:

1. Відношення має 2 або більше потенційних ключів.
2. Ці потенційні ключі складені (містять більш ніж один атрибут)
3. Ці потенційні ключі перекриваються, тобто мають щонайменше один спільний атрибут.

Четверта нормальна форма

Четверта нормальна форма (4НФ, 4NF) потребує, аби в схемі баз даних не було нетривіальних багатозначних залежностей множин атрибутів від будь чого, окрім надмножини ключа-кандидата. Вважається, що таблиця знаходиться у 4НФ тоді і лише тоді, коли вона знаходиться в НФБК та багатозначні залежності є функціональними залежностями. Четверта нормальна форма усуває небажані структури даних — багатозначні залежності.

П'ята нормальна форма

П'ята нормальна форма (5НФ, 5NF, PJ/NF) вимагає, аби не було нетривіальних залежностей об'єднання, котрі б не витікали із обмежень ключів. Вважається, що таблиця в п'ятій нормальній формі тоді і лише тоді, коли вона знаходиться в 4НФ та кожна залежність об'єднання зумовлена її ключами-кандидатами.

					КП.ІПЗ-12.ІЗ	Арк.
						12
Зм.	Арк.	№ докум.	Підпис	Дата		

можливостей **Нормальна форма домен/ключ (DKNF)**

Ця нормальна форма вимагає, аби в схемі не було інших обмежень окрім ключів та доменів.

Шоста нормальна форма

Таблиця знаходиться у 6NF, якщо вона знаходиться у 5NF та задовольняє вимозі відсутності нетривіальних залежностей. Зазвичай 6NF ототожнюють з DKNF.

Проте для того, щоб база даних вважалась нормальною достатньо дотримуватись всього третьої нормальної форми. В практичному застосуванні використовується термін денормалізації бази даних.

Денормалізація (англ. *denormalization*) — навмисне приведення структури бази даних в стан, що не відповідає критеріям нормалізації, зазвичай проводиться з метою прискорення операцій читання з бази за рахунок додавання надлишкових даних.

Усунення аномалій даних відповідно до теорії реляційних баз даних вимагає, щоб будь-яка база даних була нормалізована, тобто відповідала вимогам нормальних форм. Відповідність вимогам нормалізації мінімізує надмірність даних в базі даних і забезпечує відсутність багатьох видів логічних помилок оновлення та вибірки даних.

Однак при запитах великої кількості даних операція з'єднання нормалізованих відносин виконується неприйнятно довго. Внаслідок цього в ситуаціях, коли продуктивність таких запитів неможливо підвищити іншими засобами, може проводитися денормалізація — композиція кількох відносин (таблиць) в одну, яка, як правило, знаходиться в другій, але не в третій нормальній формі. Нове ставлення фактично є збереженим результатом операції з'єднання вихідних відносин. За рахунок такого перепроєктування операція з'єднання при вибірці стає

					КП.ІІЗ-12.ІЗ	Арк.
						13
Зм.	Арк.	№ докум.	Підпис	Дата		

непотрібною і запити вибірки, які раніше вимагали з'єднання, працюють швидше.

1.3 База даних PostgreSQL. Консоль PostgreSQL

У ході роботи над курсовою використовувався движок бази даних PostgreSQL.

PostgreSQL — об'єктно-реляційна система керування базами даних(СКБД). Є альтернативою як комерційним СКБД (Oracle Database, MicrosoftSQL Server, IBM DB2та інші), так і СКБД з відкритим кодом (MySQL,Firebird,SQLite).

Порівняно з іншими проектами з відкритим кодом, такими як Apache, FreeBSD або MySQL, PostgreSQL не контролюється якоюсь однією компанією, її розробка можлива завдяки співпраці багатьох людей та компаній, які хочуть використовувати цю СКБД та впроваджувати у неї найновіші досягнення.

Сервер PostgreSQL написаний на мові C. Зазвичай розповсюджується у вигляді набору текстових файлів із серцевим кодом. Для інсталяції необхідно відкомпілювати файли на своєму комп'ютері і скопіювати в деякий каталог. Весь процес детально описаний в документації.

PostgreSQL — широко розповсюджена система керування базами даних з відкритим вихідним кодом. Прототип був розроблений в Каліфорнійському університеті Берклі в 1987 році під назвою POSTGRES, після чого активно розвивався і доповнювався. В червні 1990 року з'явилась друга версія із переробленою системою правил маніпулювання та роботи з таблицями, у 1991 році — третя версія, із доданою підтримкою одночасної роботи кількох менеджерів збереження, покращеним механізмом запитів і доповненою системою внутрішніх правил. В цей час POSTGRES використовувався для реалізації великих систем, таких як: система аналізу фінансових даних, пакет моніторингу функціональності потоків, база даних відстеження астероїдів, система медичної інформації, кілька географічних систем.

POSTGRES також використовувався як навчальний інструмент в кількох університетах. 1992 року POSTGRES став головною СКБД наукового комп'ютерного проекту Sequoia 2000. 1993 року кількість користувачів подвоїлась. Стало зрозуміло, що для підтримки й подальшого розвитку необхідні великі

					КП.ІІЗ-12.ІЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		14

Берклі було зупинено на версії 4.2. 1994 року *Andrew Yu* і *Jolly Chen* додали інтерпретатор мови SQL, вдосконалили сирцевий код і виклали в Інтернеті свою реалізацію під назвою Postgres95. 1996 року програмний продукт було перейменовано на PostgreSQL із початковою версією 6.0.

Подальшою підтримкою й розробкою займається група спеціалістів у галузі баз даних, які добровільно приєдналися до цього проекту.

1.4 Основні можливості PostgreSQL

Функції.

Функції дозволяють виконувати деякий код безпосередньо сервером бази даних. Ці функції можуть бути написані на SQL, який має деякі примітивні програмні оператори, такі як галуження та цикли. Але гнучкішою буде функція написана на одній із мов програмування, з якими PostgreSQL може працювати. До таких мов належать:

1. Вбудована мова, яка зветься PL/pgSQL , подібна до процедурної мови PL/SQL компанії Oracle.
2. Мови розробки сценаріїв: PL/Perl, PL/Perl, PL/Python/ PL/Pcl, PL/Ruby, PL/sh.
3. Класичні мови програмування C, C++, Java (за допомогою PL/Java).

Функції можуть виконуватись із привілеями користувача, який її викликав, або із привілеями користувача, який її написав.

Індекси

У PostgreSQL є підтримка індексів наступних типів: B-дерево, хем, R-дерево, GiST, GIN. При необхідності можна створити нові типи індексів.

Багатоверсійність (MVCC)

PostgreSQL підтримує одночасну модифікацію БД декількома користувачами за допомогою механізму Multiversion Concurrency Control (MVCC). Завдяки цьому виконуються вимоги ACID, і практично відпадає потреба в блокуванні зчитування.

					КП.ІПЗ-12.ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		15

Об'єкти користувача

PostgreSQL може бути розширено користувачем для власних потреб практично в будь-якому аспекті. Є можливість додавати власні:

1. Перетворення типів
2. Типи даних
3. Домени (для користувача типи з самого початку з накладеними обмеженнями)
4. Функції (включаючи агрегатні)
5. Індeksi
6. Оператори (включаючи перевизначення вже існуючих)
7. Процедурні мови

Успадкування

Таблиці можуть успадковувати характеристики та набори полів від інших таблиць (батьківських). При цьому дані, які додаються до породженої таблиці, автоматично будуть брати участь (якщо це не вказано окремо) в запитах до батьківської таблиці. Цей функціонал в поточний час не є повністю завершеним. Однак він достатній для практичного використання.

Тригери

Тригери визначаються як функції, що ініціюються DML-операціями. Наприклад, операція INSERT може запускати тригер, що перевіряє доданий запис на відповідність певним умовам. Тригери можна писати різними мовами програмування. Вони пов'язані з визначеною таблицею. Множинні тригери виконуються в алфавітному порядку.

Надійність

Згідно з результатами автоматизованого дослідження різного ПЗ на предмет помилок, у вихідному коді PostgreSQL було знайдено 20 проблемних місць на 775 000 рядків вихідного коду (в середньому, одна помилка на 39 000 рядків коду). Для порівняння:

1. MySQL — 97 проблем, одна помилка на 4 000 рядків коду;
2. FreeBSD (цілком) — 306 проблем, одна помилка на 4 000 рядків коду;
3. Linux (тільки ядро) — 950 проблем, одна помилка на 10 000 рядків коду.

					КП.ПЗ-12.ПЗ	Арк.
						16
Зм.	Арк.	№ докум.	Підпис	Дата		

2 ОПИС УМОВИ І ПОБУДОВА БАЗИ ДАНИХ

2.1 Постановка задачі

Задачею курсової роботи є розробка електронної медичної картотеки .

Медична картотека — інформація про пацієнта. Вся інформація зберігається у вигляді записів (книжка). У простонародді реєстратура.

Електронна медична картотека — комп'ютерна база даних, що містить ієрархію таблиць, зв'язків. Які в сукупності визначають суть медичної картотеки. Дає можливість дізнатися медичну історію кожного пацієнта зокрема: які ін'єкції приймав пацієнт, чи був госпіталізований, яких лікарів був на прийомі тощо.

Електронну медичну картотеку потрібно реалізувати у вигляді бази даних. Медична картотека використовується у медичних закладах, для збереження, аналізу, виводу інформації про пацієнта.

Суть електронної медичної картотети потрібно реалізувати на мові SQL.

2.2 Перелік проблем для вирішення

Медична картотека використовується в усіх видах медичних закладів, і має поширення в усіх медичних системах світу. Так як вона систематизує усі дані про пацієнта. Дає зрозуміти, лікарю на що пацієнт звертається найчастіше, і які проблеми притаманні тому чи іншому пацієнтів. Так як до кожного пацієнта потрібний індивідуальний підхід.

Робить більш зручним збір інформації про пацієнта. Та як історія пацієнта і його особисті дані — це дві окремі таблиці. Це дає змогу окремо записувати особисті дані і історію пацієнта.

Реалізувавши електронну медичну картотеку за допомогою баз даних, ми автоматизуємо процес медичної реєстратури. Якщо ж окрім бази реалізувати якийсь веб-аплікацію, це полегшить роботи працівників реєстратури медичного закладу

					КП.ІПЗ-12.ПЗ	Арк.
						17
Зм.	Арк.	№ докум.	Підпис	Дата		

Для Так як при старій системі реєстратури, черговому медичному працівнику доводилось у ручну із великої кількості медичних карт найти потрібного пацієнта, це марна трата часу, особливо якщо йде мова про місто, де є проблеми з “живою чергою”.

Проблеми які є основними:

1. Систематизувати дані пацієнта
2. Полегшити пошук пацієнтів
3. Ведення зручної історії
4. Полегшити пошук інформації про конкретного пацієнта
5. Можливість визначати які пацієнти не пройшли інфекцію
6. Можливість визначати які пацієнти не були госпіталізовані

Систематизувати дані пацієнта

Дані, які будуть зберігатися в базі будуть розподілено по таблицям відповідно по сутностям, що саме по собі добавить читабельність базі, для інших розробників, і краще розуміння для користувачів бази.

Полегшити пошук пацієнтів

У теперішній реаліях пошук пацієнтів в реєстратурі виглядає дуже затратною по часу і одночасно важкою роботою. Ця проблема є основною для цієї галузі. Тому такого типу автоматизації дасть велику перевагу у часі і затраті людського ресурсу.

Ведення зручної історії

Вище сказане досить непогано пояснює цю проблема. Так як будь-яка вирішення цих проблем призведе до зменшення затратного часу на процедуру пошуку та ведення історії про пацієнта.

					КП.ІІЗ-12.ІЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		18

Полегшити пошук інформації про конкретного пацієнт

Якщо потребується знайти про пацієнта все, що є для медичного працівника досить затратним процесом. Та автоматизація у вигляді баз даних спростить пошук всієї інформації про конкретного пацієнта.

Можливість визначати які пацієнти не пройшли інфекцію

Так як ін'єкції, які були проведені записується в історію для конкретного пацієнта, це спрощує процес пошуку проін'єкційованих пацієнтів. Або ж пацієнтів, які навпаки не пройшли ін'єкціювання, і в ній потребуються.

Можливість визначати які пацієнти не були госпіталізовані

Для лікаря інколи важливо знати чи був його пацієнт госпіталізований, допоможе визначити чи були у пацієнта раніше проблеми з тим чи іншим захворюванням. Це напряду вплине на процес одужання хворого пацієнта.

2.3 Конструювання структури

При конструюванні структури бази даних прийнято використовувати ER-модель.

ER-модель – модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків. ER-модель — це мета-модель даних, тобто засіб опису моделей даних. Існує ряд моделей для представлення знань, але одним з найзручніших інструментів уніфікованого представлення даних, незалежного від програмного забезпечення що його реалізує, є модель «сутність-зв'язок».

У разі реляційної бази даних, в якій зберігаються дані в таблицях, кожен рядок кожної таблиці являє собою один екземпляр сутності. Деякі поля даних в цих таблицях вказують на індекси в інших таблицях. Такі поля є показниками

					КП.ІПЗ-12.ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		19

фізичної реалізації зв'язків між сутностями.

У базі даних електронної медичної картотек мають бути сутності, які відповідають за пацієнта та історію, що зберігаються у базі, а також взаємодію між ними. Кожне пацієнт має мати унікальний ідентифікатор, назву і опис. Ідентифікатор потрібен для історії, що мати можливість звертатися до історії по її ідентифікатору.

Таким чином можна виділити одну таблицю, для зберігання пацієнтів. Взаємодія двох таблиць дасть нам можливість зберігати історію про окремого пацієнта. Таблиця паєнта окрім зв'язку з історією має особисту інформацію про пацієнта його ім'я, Прізвище, по-батькові, стать, вік, дата народження, а звичайно поле з індетифікатором користувача яке буде записуватися в історію.

В той час в таблиці історія будуть записуватися ідентифікатори історії, пацієнта і те куд пацієнт звернувся (до лікаря, був госпіталізований або ж інфекційований).

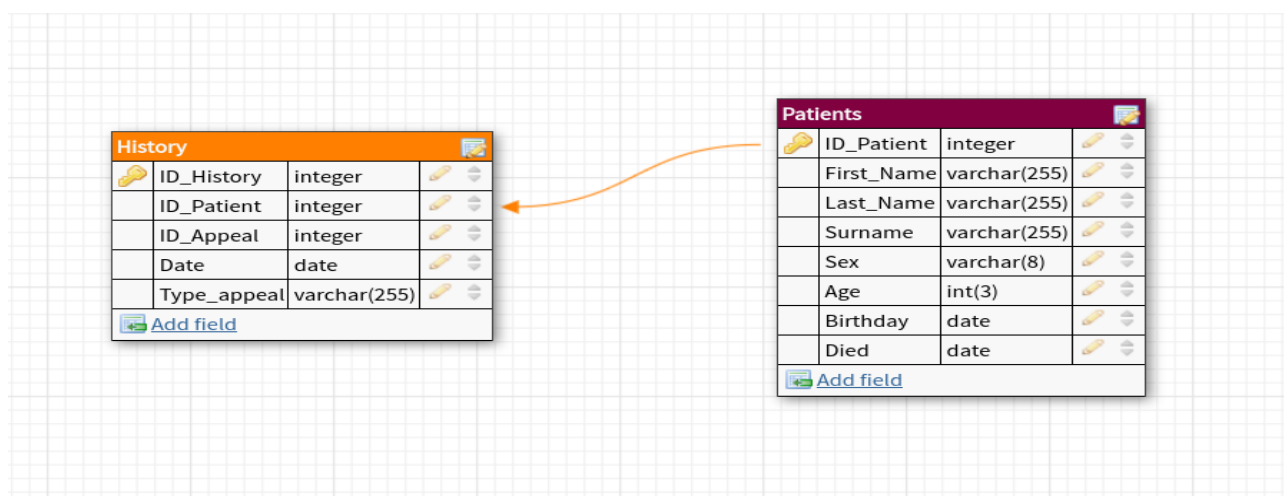


Рисунок 1. Зв'язок історії і пацієнта

Як можна помітити, зв'язок між таблицями тут 1 до 1 через унікальність ідентифікатора

Ідентифікатори в кожній з таблиць є типом INTEGER та з PRIMARY KEY для унікальності кожного ідентифікатора. Поле в таблиці ID_Patient не є PRIMARY KEY так як туди будуть записуватися ідентифікатори пацієнта. Якщо ж зробити його PRIMARY KEY, то поле не зможе приймати одній і тіж параметри. Тобто на одного пацієнта може буде декілька записів в історії. PRIMARY KEY не дозволить нам цього зробити, так як це додати унікальності полю.

Поле ID_Arreal створений для запису конкретного ідентифікатора лікаря, госпіталізації або ж ін'єкції. Поле Type_Arreal виконує функцію визначення тощо що відвідав пацієнт.

Дана структура погана тим, що є не оптимальною в плані витрати ресурсів пам'яті через дублювання великої кількості інформації і складності в обслуговуванні.

Наступною таблицею є Injections куди будуть записуватися ін'єкції пацієнтів, і медична назва ін'єкції.





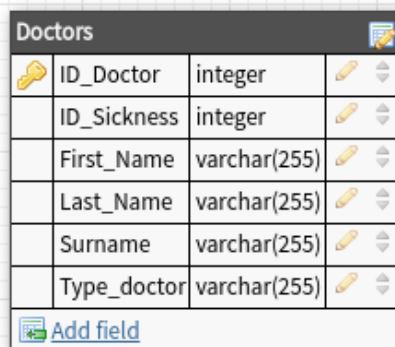
Injections			
	ID_Injection	integer	
	Nama_injection	varchar(255)	
 Add field			

Рисунок 2. Таблиця ін'єкцій

Якщо пацієнт відвідав лікаря для цього є окрема таблиця



ID_Doctor	integer
ID_Sickness	integer
First_Name	varchar(255)
Last_Name	varchar(255)
Surname	varchar(255)
Type_doctor	varchar(255)

Рисунок 3. Таблиця лікарів

Ця таблиця містить дані про конкретного лікаря. А саме: його ідентифікатор, ідентифікатор хвороби яку він визначив у пацієнта, його ім'я, прізвище, по-батькові і тип лікаря. В полі type_doctor буде записуватися тип лікаря, що буде означати до якого саме лікаря звернувся пацієнт.

Наприклад: пацієнт звернувся до сімейного лікаря, пацієнт скаржиться на біль у горлі.

Це означає, що в поле буде записано значення “family”.

Поле id_sickness як було написано вище буде звертатися до іншої таблиці яка в свою очиридь містить в собі хвороби які прописали лікарі своїм пацієнтам. Про цю таблицю більш детально описано нижче.

У випадку якщо пацієнту знадобилась термінова госпіталізація, для цього існує окрема таблиця “Hospitalization”








Hospitalization			
	ID_Hosp	integer	
	ID_Sickness	integer	
	Data	date	
	Address	varchar(255)	
	Reason	varchar(255)	
 Add field			

Рисунок 4. Таблиця пацієнтів, які госпіталізовані

В даному випадку ми маємо поле з ідентифікатором госпіталізації, поле з ідентифікатором хвороби з якою поступив пацієнт, дата госпіталізації, адреса лікарні, які прийняла пацієнта, та у зв’язку з чим пацієнт попав у лікарню, тобто які симптоми були притаманні пацієнту під час огляду.

Ідентифікатор таблиці буде записуватися в таблицю “history” в поле id_appeal. Також буде записуватися значення “hospitalization” в поле таблиці “history” type_appeal.

Так як дві таблиці ‘Doctors’ і ‘Hospitalization’ звязані з таблицею ‘sickness’ її вигляд буде наступним

					КП.ІПЗ-12.ПЗ	Арк.
						23
Зм.	Арк.	№ докум.	Підпис	Дата		








Sickness			
	ID_Sickness	integer	
	Name_sickness	varchar(255)	
	Begin_sickness	date	
	Recovery	date	
	recipe	text	
 Add field			

Рисунок 5. Таблиця хвороб

Таблиця містить ідентифікатор хвороби, її медична назва, початок хвороби, одужання і рецепт по якому було проведено лікування.

Ідентифікатор таблиці записується в дві таблиці 'Doctors' і 'Hospitalization'.

В сукупності таблиць та їх зв'язків її вигляд буде таким:

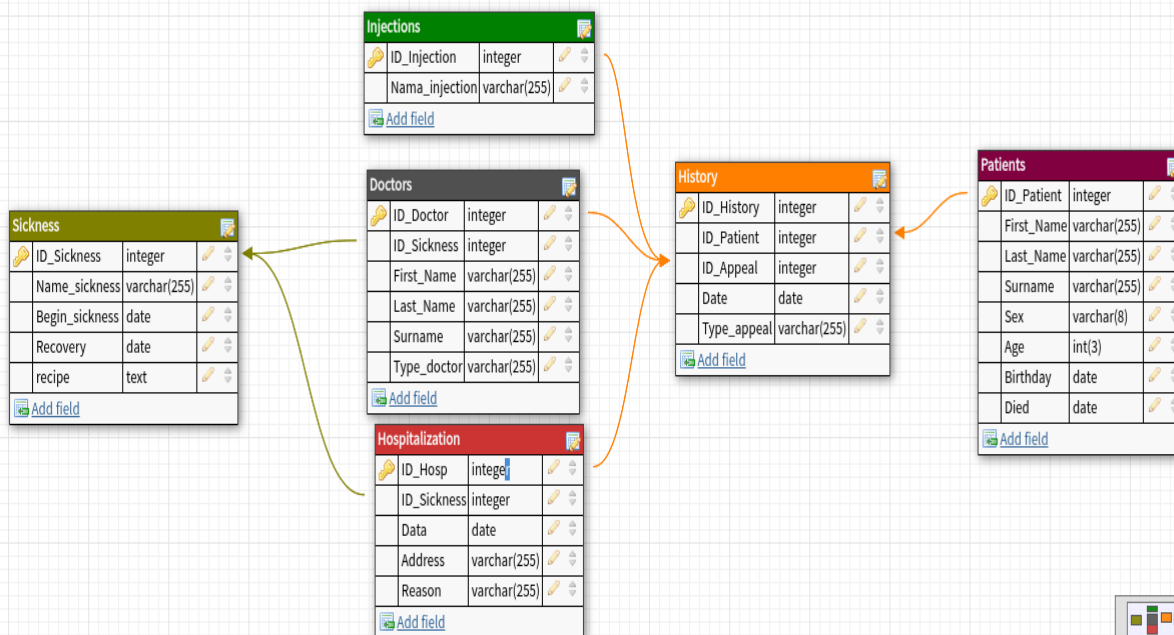


Рисунок 6. ER-Diagram бази даних

3 РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Створення таблиць і зв'язків між ними

Створюємо першу таблицю Patients

```
CREATE TABLE Patients (  
    ID_Patient INT UNIQUE,  
    First_Name varchar(255) NOT NULL,  
    Last_Name varchar(255) NOT NULL,  
    Surname varchar(255) NOT NULL,  
    Sex varchar(8) NOT NULL,  
    Age int NOT NULL,  
    Birthday DATE NOT NULL,  
    PRIMARY KEY (ID_Patient)  
);
```

Створюємо таблицю History

```
CREATE TABLE History (  
    ID_History SERIAL UNIQUE,  
    ID_Patient INT NOT NULL,  
    ID_Appeal INT NOT NULL,  
    Date DATE NOT NULL,  
    PRIMARY KEY (ID_History)  
);
```

Створюємо таблицю Injections

```
CREATE TABLE Injections (  
    ID_Injection SERIAL UNIQUE,  
    Name_injection varchar(255) NOT NULL,  
    PRIMARY KEY (ID_Injection)  
);
```

Створюємо таблицю Doctors

```
CREATE TABLE Doctors (  
    ID_Doctor SERIAL UNIQUE,  
    й
```

```
Для ID_Sickness SERIAL UNIQUE,  
    First_Name varchar(255) NOT NULL,  
    Last_Name varchar(255) NOT NULL,
```

Surname varchar(255) **NOT NULL**,
 Type_doctor varchar(255) **NOT NULL**,
PRIMARY KEY (ID_Doctor,ID_Sickness)

);

Створюємо таблицю Hospitalization

CREATE TABLE Hospitalization (
 ID_Hosp **SERIAL UNIQUE**,
 ID_Sickness **INT NOT NULL**,
 Data **DATE NOT NULL**,
 Address varchar(255) **NOT NULL**,
 Reason varchar(255) **NOT NULL**,
PRIMARY KEY (ID_Hosp)

);

Створюємо таблицю Sickness

CREATE TABLE Sickness (
 ID_Sickness **SERIAL UNIQUE**,
 Name_sickness varchar(255) **NOT NULL**,
 Begin_sickness **DATE NOT NULL**,
 Recovery **DATE NOT NULL**,
 recipe **TEXT NOT NULL**,
PRIMARY KEY (ID_Sickness)

);

Реалізовуємо зв'язки між таблицями за допомогою. **ALTER TABLE** table **ADD CONSTRAINT** name_constraint **FOREIGN KEY** (table_id) **REFERENCES** table2(table2_id)

ALTER TABLE History **ADD CONSTRAINT** Patients_history **FOREIGN KEY** (ID_Patient) **REFERENCES** Patients(ID_Patient);

ALTER TABLE Injections **ADD CONSTRAINT** Injections_history **FOREIGN KEY** (ID_Injection) **REFERENCES** History(ID_Appeal);

					КП.ІІЗ-12.ІІЗ	Арк.
						26
Зм.	Арк.	№ докум.	Підпис	Дата		

ALTER TABLE Hospitalization **ADD CONSTRAINT** Hospitalization_history
FOREIGN KEY (ID_Hosp) **REFERENCES** History(ID_Appeal);

ALTER TABLE Hospitalization **ADD CONSTRAINT** Hospitalization_sickness
FOREIGN KEY (ID_Sickness) **REFERENCES** Sickness(ID_Sickness);

ALTER TABLE Doctors **ADD CONSTRAINT** Doctors_history **FOREIGN KEY**
(ID_Doctor) **REFERENCES** History(ID_Appeal);

ALTER TABLE Doctors **ADD CONSTRAINT** Doctors_sickness **FOREIGN KEY**
(ID_Sickness) **REFERENCES** Sickness(ID_Sickness);

3.2 Заповнення створених таблиць

Для прикладу виведу INSERT-и для одного користувача

INSET INTO patients(first_name, last_name,surname,sex,age,birthday)

VALUES ('Taras', 'Matiiv', 'Romanovich', '', 18, '1999-03-04');

INSERT INTO history(id_patient, id_appeal, date, type_appeal)

VALUES (1, 1, '2018-11-01', 'Injection');

INSERT INTO injections(name_injections) **VALUES** ('chicken pox');

Усі SQL запити виконувалися в консолі PostgreSQL без використання
візуального СУБД (pgAdmin).

ВИСНОВКИ

В даній курсовій роботі представлено реалізацію бази даних електронного словника, перекладача на основі мови SQL.

Для розробки використовувалась консольна версія PostgreSQL. Усі таблиці і діаграма взаємодії таблиць розроблялися в стандартній консолі.

Завдання, які були реалізовані під час реалізації курсової:

1. Перелік пацієнтів
2. Історія кожного пацієнта
3. Перелік пацієнтів які були госпіталізовані, відвідували лікаря або ж були на інфекції
4. Хвороби на, які хворіли пацієнти, дата перший симптомів захворювання, дата повного одужання, рецепт по якому було проведено лікування.

Дана курсова робота спрощує процес ведення історії про конкретного пацієнта.

					КП.ІПЗ-12.ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		28

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. <https://uk.wikipedia.org/wiki/SQL>
2. https://uk.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F_%D0%B1%D0%B0%D0%B7_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85
3. <https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BD%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F>
4. https://uk.wikipedia.org/wiki/Microsoft_SQL_Server
5. <https://uk.wikipedia.org/wiki/Transact-SQL>
6. https://ru.wikipedia.org/wiki/SQL_Server_Management_Studio
7. <https://uk.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B2%D0%BD%D0%B8%D0%BA>
8. https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%C2%AB%D1%81%D1%83%D1%82%D0%BD%D1%96%D1%81%D1%82%D1%8C_%E2%80%94%D0%B7%D0%B2%D1%8F%D0%B7%D0%BE%D0%BA%C2%BB

					КП.ІПЗ-12.ІЗ	Арк.
						29
Зм.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

Додаток А

```
--
-- PostgreSQL database dump
--
-- Dumped from database version 9.5.15
-- Dumped by pg_dump version 9.5.15

SET statement_timeout = 0;
SET lock_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false); SET
check_function_bodies = false;
SET client_min_messages = warning;
SET row_security = off;

CREATE EXTENSION IF NOT EXISTS plpgsql WITH SCHEMA pg_catalog;

COMMENT ON EXTENSION plpgsql IS 'PL/pgSQL procedural language';

SET default_tablespace = '';

SET default_with_oids = false;

CREATE TABLE public.doctors (
    id_doctor integer NOT NULL,
    id_sickness integer NOT NULL,
```

```
first_name character varying(255) NOT NULL,  
last_name character varying(255) NOT NULL,  
surname character varying(255) NOT NULL,  
type_doctor character varying(255) NOT NULL  
);
```

```
ALTER TABLE public.doctors OWNER TO taras;
```

```
CREATE SEQUENCE public.doctors_id_doctor_seq  
START WITH 1  
INCREMENT BY 1  
NO MINVALUE  
NO MAXVALUE  
CACHE 1;
```

```
ALTER TABLE public.doctors_id_doctor_seq OWNER TO taras;
```

```
ALTER SEQUENCE public.doctors_id_doctor_seq OWNED BY  
public.doctors.id_doctor;
```

```
CREATE SEQUENCE public.doctors_id_sickness_seq  
START WITH 1  
INCREMENT BY 1  
NO MINVALUE  
NO MAXVALUE  
CACHE 1;
```

```
ALTER TABLE public.doctors_id_sickness_seq OWNER TO taras;
```

```
ALTER SEQUENCE public.doctors_id_sickness_seq OWNED BY  
public.doctors.id_sickness;
```

```
CREATE TABLE public.history (  
    id_history integer NOT NULL,  
    id_patient integer NOT NULL,  
    id_appeal integer NOT NULL,  
    date date NOT NULL,  
    type_appeal character varying  
);
```

```
ALTER TABLE public.history OWNER TO taras;
```

```
CREATE SEQUENCE public.history_id_history_seq  
    START WITH 1  
    INCREMENT BY 1  
    NO MINVALUE  
    NO MAXVALUE  
    CACHE 1;
```

```
ALTER TABLE public.history_id_history_seq OWNER TO taras;
```

```
ALTER SEQUENCE public.history_id_history_seq OWNED BY  
public.history.id_history;
```

```
CREATE TABLE public.hospitalization (  
    id_hosp integer NOT NULL, id_sickness  
    integer NOT NULL,  
    data date NOT NULL,  
    address character varying(255) NOT NULL,
```



```
reason character varying(255) NOT NULL  
);
```

```
ALTER TABLE public.hospitalization OWNER TO taras;
```

```
CREATE SEQUENCE public.hospitalization_id_hosp_seq  
START WITH 1  
INCREMENT BY 1  
NO MINVALUE  
NO MAXVALUE  
CACHE 1;
```

```
ALTER TABLE public.hospitalization_id_hosp_seq OWNER TO taras;
```

```
ALTER SEQUENCE public.hospitalization_id_hosp_seq OWNED BY  
public.hospitalization.id_hosp;
```

```
CREATE TABLE public.injections (  
id_injection integer NOT NULL,  
name_injection character varying(255) NOT NULL  
);
```

```
ALTER TABLE public.injections OWNER TO taras;
```

```
CREATE SEQUENCE public.injections_id_injection_seq  
START WITH 1  
INCREMENT BY 1  
NO MINVALUE
```

NO MAXVALUE

CACHE 1;

ALTER TABLE public.injections_id_injection_seq OWNER TO taras;

**ALTER SEQUENCE public.injections_id_injection_seq OWNED BY
public.injections.id_injection;**

**CREATE TABLE public.patients (
 id_patient integer NOT NULL,
 first_name character varying(255) NOT NULL,
 last_name character varying(255) NOT NULL,
 surname character varying(255) NOT NULL, sex
 character varying(8) NOT NULL,
 age integer NOT NULL,
 birthday date NOT NULL
);**

ALTER TABLE public.patients OWNER TO taras;

**CREATE TABLE public.sickness (
 id_sickness integer NOT NULL,
 name_sickness character varying(255) NOT NULL,
 begin_sickness date NOT NULL,
 recovery date NOT NULL,
 recipe text NOT NULL
);**

ALTER TABLE public.sickness OWNER TO taras;

**CREATE SEQUENCE public.sickness_id_sickness_seq
START WITH 1
INCREMENT BY 1
NO MINVALUE
NO MAXVALUE
CACHE 1;**

ALTER TABLE public.sickness_id_sickness_seq OWNER TO taras;

**ALTER SEQUENCE public.sickness_id_sickness_seq OWNED BY
public.sickness.id_sickness;**

**ALTER TABLE ONLY public.doctors ALTER COLUMN id_doctor SET
DEFAULT nextval('public.doctors_id_doctor_seq'::regclass);**

**ALTER TABLE ONLY public.doctors ALTER COLUMN id_sickness SET
DEFAULT nextval('public.doctors_id_sickness_seq'::regclass);**

**ALTER TABLE ONLY public.history ALTER COLUMN id_history SET
DEFAULT nextval('public.history_id_history_seq'::regclass);**

**ALTER TABLE ONLY public.hospitalization ALTER COLUMN id_hosp SET
DEFAULT nextval('public.hospitalization_id_hosp_seq'::regclass);**

```
ALTER TABLE ONLY public.injections ALTER COLUMN id_injection SET  
DEFAULT nextval('public.injections_id_injection_seq'::regclass);
```

```
ALTER TABLE ONLY public.sickness ALTER COLUMN id_sickness SET  
DEFAULT nextval('public.sickness_id_sickness_seq'::regclass);
```

```
COPY public.doctors (id_doctor, id_sickness, first_name, last_name, surname,  
type_doctor) FROM stdin;
```

```
2      1      Alla   Petruchko Stepanivna family doctor
```

```
\.
```

```
SELECT pg_catalog.setval('public.doctors_id_doctor_seq', 3, true);
```

```
SELECT pg_catalog.setval('public.doctors_id_sickness_seq', 1, false);
```

```
COPY public.history (id_history, id_patient, id_appeal, date, type_appeal) FROM  
stdin;
```

```
2      1      1      2018-11-01 Injection
```

```
5      2      3      2017-06-21 doctors
```

```
10     3      2      2018-11-20 hospitalization
```

```
SELECT pg_catalog.setval('public.history_id_history_seq', 11, true);
```

```
COPY public.hospitalization (id_hosp, id_sickness, data, address, reason) FROM  
stdin;
```

```
2      2      2018-05-06 str. Mateika, 45      tablets, and sirops
```

```
SELECT pg_catalog.setval('public.hospitalization_id_hosp_seq', 1, true);
```

```
COPY public.injections (id_injection, name_injection) FROM stdin;
```

```
1      chicken pox
```

```
\.
```

```
SELECT pg_catalog.setval('public.injections_id_injection_seq', 1, true);
```

```
COPY public.patients (id_patient, first_name, last_name, surname, sex, age, birthday)
```

```
FROM stdin;
```

```
1      TarasMatiiv      Romanovich      male  18    1999-03-04
```

```
2      KolyaDanuliv     Maksumovich     male  18    2000-08-09
```

```
3      Maksim          Fenuk           Andriyovich     male  16    1981-04-26
```

```
COPY public.sickness (id_sickness, name_sickness, begin_sickness, recovery, recipe)
```

```
FROM stdin;
```

```
1      Virus HRV 2019-03-01 2018-03-07 antibiotics
```

```
2      internal hemorrhage      2017-01-03 2018-03-07
```

```
SELECT pg_catalog.setval('public.sickness_id_sickness_seq', 2, true);
```

```
ALTER TABLE ONLY public.doctors
```

```
ADD CONSTRAINT doctors_id_doctor_key UNIQUE (id_doctor);
```

```
ALTER TABLE ONLY public.doctors
```

```
ADD CONSTRAINT doctors_id_sickness_key UNIQUE (id_sickness);
```

ALTER TABLE ONLY public.sickness

ADD CONSTRAINT sickness_pkey PRIMARY KEY (id_sickness);

ALTER TABLE ONLY public.doctors

ADD CONSTRAINT doctors_sickness FOREIGN KEY (id_sickness)

REFERENCES public.sickness(id_sickness);

ALTER TABLE ONLY public.hospitalization

ADD CONSTRAINT hospitalization_history FOREIGN KEY (id_hosp)

REFERENCES public.history(id_appeal);

ALTER TABLE ONLY public.injections

ADD CONSTRAINT injections_history FOREIGN KEY (id_injection)

REFERENCES public.history(id_appeal);

ALTER TABLE ONLY public.history

ADD CONSTRAINT patients_history FOREIGN KEY (id_patient)

REFERENCES public.patients(id_patient);

REVOKE ALL ON SCHEMA public FROM PUBLIC;

REVOKE ALL ON SCHEMA public

FROM postgres; GRANT ALL

ON SCHEMA public TO postgres;

GRANT ALL ON SCHEMA

public TO PUBLIC;

Додаток В

Medicine Card

Вивід останнього пацієнта, який прийняв ін'єкцію:

```
```sql
```

```
SELECT patients.first_name, patients.last_name, history.date,
history.type_appeal, injections.name_injection FROM patients INNER JOIN
history ON patients.id_patient=history.id_patient INNER JOIN injections ON
history.id_appeal=injections.id_injection;
```

```
```
```

Вивід дат усіх звернень пацієнта:

```
```sql
```

```
SELECT history.date FROM patients
INNER JOIN history ON patients.id_patient=history.id_patient;
```

```
```
```

Вивід пацієнта, який відвідував лікаря:

```
```sql
```

```
SELECT patients.first_name, patients.last_name, history.date,
history.type_appeal, doctors.first_name, doctors.last_name,
doctors.type_doctor, sickness.name_sickness FROM patients INNER
JOIN history ON patients.id_patient=history.id_patient INNER JOIN
doctors ON history.id_appeal=doctors.id_doctor INNER JOIN sickness ON
doctors.id_sickness=sickness.id_sickness;
```

```
```
```

Вивід пацієнтів, які були госпіталізовані:

```
```sql
```

```
SELECT patients.first_name, patients.last_name, history.date,
history.type_appeal, hospitalization.data, hospitalization.address,
sickness.name_sickness FROM patients
INNER JOIN history ON patients.id_patient=history.id_patient
```

```
INNER JOIN hospitalization ON history.id_appeal=hospitalization.id_hosp INNER
JOIN sickness ONhospitalization.id_sickness=sickness.id_sickness;
```