UNICHECK
by Turnitin

Ім'я користувача:
приховано налаштуваннями конфіденційності

ID перевірки:
1015626497

Дата перевірки:
16.06.2023 13:53:48 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
16.06.2023 13:57:51 EEST

ID користувача:
100011372

Назва документа: **ОК-42 Каленик Богдан**

Кількість сторінок: **51** Кількість слів: **8317** Кількість символів: **62979** Розмір файлу: **701.00 KB** ID файлу: **1015272610**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

2.52%
Схожість

Найбільша схожість: **0.42%** з джерелом з Бібліотеки (ID файлу: **1009724942**)

Пошук збігів з Інтернетом не проводився

2.52% Джерела з Бібліотеки | **137** ..... Сторінка 53

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи | **1**

Підозріле форматування | **23 сторінки**

1 ЗАГАЛЬНА ХАРАКТЕРИСТИКА БАЗИ ДАНИХ

1.1 Поняття бази даних

Широке застосування різноманітних інформаційних систем стало невід'ємною частиною сучасного суспільства. Сьогодні інформація перетворилася на один з найважливіших ресурсів, тому інформаційні системи стали необхідним інструментом практично у всіх галузях діяльності. Під назвою інформаційної системи потрібно розуміти складні організовані структури, які включають в себе різні технічні та програмні засоби, призначені для збору, обробки та представлення інформації. Бази даних є ключовим компонентом навколо якого будуються такі системи.

База даних – це упорядкований комплекс логічно взаємопов'язаних даних, які використовуються спільно та призначені для задоволення інформаційних потреб користувачів. Власне бази даних можна розглядати як електронні архіви, що дають змогу зберігати та управляти даними різноманітних типів, таких як: текст, числа, зображення та відео.

База даних характеризується деякими ознаками:

- По – перше, БД – це єдине сховище даних, яке зберігає всю інформацію, пов'язану з конкретною діяльністю, організацією чи проєктом.
- По – друге, БД може використовуватися як загальний корпоративний ресурс, доступний для всіх працівників компанії або для користувачів зовнішніх організацій.
- По – третє, БД зберігає не тільки самі дані, але і їх опис, який визначає структуру і зв'язки між ними.

Усі бази даних повинні відповідати вимогам якості, що залежать від ряду властивостей, до яких відносять: цілісність, доступність, надійність, ефективність та безпечність. Ці властивості мають наступні пояснення:

- Цілісність – це властивість баз даних, що забезпечує послідовність та точність даних. Дані мають зберігатися певним чином, так щоб вони не були

змінені або втрачені внаслідок неправильних дій чи помилок. Для досягнення цілісності база даних повинна зберігати механізми контролю даних: обмеження на введення, перевірку на коректність, заборону на видалення, модифікацію.

- Доступністю називається властивість баз даних, яка вказує на можливість користувачів отримувати швидкий доступ до даних. Означає, що за необхідності БД повинна бути доступна для використання в будь–який час. Для того щоб гарантувати доступність, потрібно використовувати резервні копії даних, а також застосовувати механізми реплікації та моніторингу.
- Надійністю є властивість баз даних, що полягає в стійкості збереження даних навіть при виникненні непередбачуваних ситуацій, пов'язаних з відмовою обладнання або програмного забезпечення, втратою електропостачання, збоями в мережі тощо. Надійність баз даних одержується за рахунок використання транзакцій та механізмів відновлення.
- Властивість ефективності бази даних визначається її здатністю оперативно опрацьовувати дані. Основні чинники, що впливають на ефективність: структура БД, використання індексів, оптимізація запитів і налаштування апаратного забезпечення.
- Безпека – це властивість БД, котра гарантує захист даних. Безпека досягається за допомогою автентифікації та авторизації користувачів, вбереження від злому та вірусів, а також захисту від недобросовісних дій персоналу, в наслідок таких дій дані залишаються конфіденційними та захищеними від несанкціонованого доступу, модифікації або видалення.

До інших властивостей можна додати масштабованість – забезпечує здатність бази даних розширюватися зі зростанням обсягу даних і збільшенням числа користувачів, а також простоту використання та адміністрування, яка дозволяє зменшити витрати на розробку та підтримку системи баз даних.

За останні роки спостерігається значний розвиток технологій обробки та аналізу даних, що відкриває нові можливості для баз даних. Наприклад, використання штучного інтелекту, машинного навчання та аналізу великих обсягів даних дозволяє отримувати цінні інсайти та прогнозувати тренди в різних галузях.

1.2 Призначення та класифікація систем управління базами даних

Система управління базами даних – це узагальнена програмна система для проведення різних маніпуляцій з базами даних. По суті, це комп'ютеризована система діловодства, яка зберігає інформацію і дозволяє користувачам додавати, видаляти, змінювати, отримувати та оновлювати її за потребою. Іншими словами, СУБД – це набір програм, які дають можливість користувачам створювати та підтримувати базу даних. Основна мета СУБД – надати зручний та ефективний спосіб зберігання та пошуку інформації.

Кожна СУБД має своє середовище, що візуально показано на рисунку 1.1.

Рисунок 1.1 – Компоненти середовища СУБД

Таким чином, середовище СУБД має наступні компоненти:

- Апаратне забезпечення;
- Програмне забезпечення;
- Дані;
- Процедури;
- Користувачі.

Апаратне забезпечення середовища системи управління баз даних включає в себе всі фізичні пристрої, які складають систему БД. Сюди входять пристрої зберігання даних, процесори, пристрої введення та виведення, принтери, мережеві пристрої тощо.

Програмне забезпечення – це набір програм, які використовуються для контролю та управління всією базою даних. Сюди входить саме програмне забезпечення СУБД, операційна система, мережеве забезпечення, яке

використовується для обміну даними між користувачами, і прикладна програма, яка надає доступ до даних в СУБД.

СУБД існує для того, щоб збирати, зберігати, обробляти та надавати доступ до даних, які є її найважливішим компонентом. Малюнок зображує дані у вигляді моста, що з'єднує комп'ютер та людину, ілюструючи важливу роль даних у взаємодії між ними.

Процедурний компонент – це ніщо інше, як функція, яка регулює використання бази даних. Процедури – це набір інструкції та правил, які допомагають при роботі СУБД.

Користувачі – це ті, хто контролює та управляє базами даних і виконує різні типи операцій над ними в СУБД. Це може бути група людей, які отримали доступ до бази даних лише для вирішення своїх запитів, тобто кінцеві користувачі, а можуть бути і люди, які беруть участь у проектуванні бази даних, тобто проектувальники.

Системи управління базами даних можна класифікувати за трьома основними критеріями: моделями даних, розміщенням і способом доступу, схема такої класифікації зображена на рисунку 1.2. Кожна з цих класифікацій визначається конфігурацією певних параметрів.

Рисунок 1.2 – Схема класифікацій СУБД

Модель даних визначає структуру даних, яка може бути збережена в БД, а також відносини між цими даними. Існують різні моделі, які використовуються в СУБД, але в загальному виділяють чотири найважливіші: ієрархічну, мережеву, реляційну, об’єктно–реляційну. Класифікація за моделями даних:

1 Ієрархічна модель використовує концепцію ієрархії для організації даних у вигляді деревоподібної структури з батьківськими та дочірніми вузлами. У цій моделі дані представлені у вигляді записів та вузлів, де записи містять атрибути, а вузли визначають ієрархічні зв'язки між ними. Одна з особливостей ієрархічної моделі полягає у сильній залежності між записами окремих рівнів, що обмежує доступ до даних через певний шлях від вершини моделі до коріння.

2 Мережева модель даних була розроблена з метою вирішення обмежень ієрархічної моделі, пов'язаних із сильною залежністю між записами та обмеженим доступом до даних. У цій моделі дані представлені у вигляді зв'язків між записами, що утворюють мережу з різними вузлами. Одна з особливостей мережевої моделі полягає у тому, що дані можуть мати зв’язки «багато до багатьох» , таким чином один запис може бути пов'язаний з декількома іншими записами, і навпаки. Така гнучкість у визначенні зв'язків дозволяє зручно моделювати складні структури даних, однак можуть виникати проблеми з дублюванням та їх непослідовністю.

3 Реляційні моделі даних відрізняються від розглянутих вище мережевих та ієрархічних простою структурою, зручним для користувача табличним представленням та пришвидшеним доступом. Ця модель є сукупністю простих двовимірних таблиць (об’єктів моделі). Зв’язки між двома логічно зв’язаними таблицями в реляційній моделі встановлюються по рівності значень однакових атрибутів і створюють таблицю–відношення. Таблиця–відношення є універсальним об’єктом, що уможливлює уніфікацію даних у різних СУБД, які підтримують реляційну модель. В даний час ця модель є найпопулярнішою.

4 Об'єктно–орієнтована модель даних є альтернативою реляційній моделі. Вона заснована на концепціях об'єктно–орієнтованого програмування, і була розроблена з метою кращого моделювання реальних об'єктів та взаємозв'язків між

ними. У цій моделі, дані представлені у вигляді об'єктів, які мають стан (атрибути) і поведінку (методи). Особливістю об'єктно–орієнтованої моделі є поліморфізм, тобто об'єкти можуть мати різні форми та поводитися по–різному, навіть якщо вони належать до одного класу. Це дає можливість розширювати функціональність та гнучкість системи. Об'єктно–орієнтована модель даних поєднує властивості асоціації, агрегації та композиції для відображення взаємозв'язків, які дозволяють моделювати складні структури та залежності між об'єктами.

СУБД за розміщенням розділяють за двома основними типами: локальні та розподілені. Класифікація за розміщенням:

1 Локальні СУБД використовуються для управління базами даних, що знаходяться на одній фізичній машині чи локальній мережі. Головною перевагою локальних СУБД є швидкий доступу до даних, які зберігаються локально і обробляються безпосередньо на сервері. Вони зазвичай є простими в установці та налаштуванні, тому що не вимагають додаткових з'єднань з іншими машинами або вузлами мережі. Локальні СУБД часто використовуються в невеликих організаціях або на персональних комп'ютерах для одного користувача.

2 Розподілені СУБД використовуються для управління базами даних, які розташовані на різних фізичних машинах або в різних мережах. В таких системах дані розподілені між окремими вузлами, що дозволяє виконувати обробку паралельно на різних серверах . Перевагами розподілених СУБД є можливість опрацювання великих обсягів даних та підтримка розширюваності системи. Завдяки паралельному обробленню на різних серверах, розподілені СУБД можуть забезпечувати високу продуктивність та швидкодію.

Класифікація за методом доступу визначає поділ компонентів за категоріями на основі того, як користувачі та програми взаємодіють з даними та ресурсами. Існує три основні категорії СУБД: файл–серверні, клієнт– серверні, інтегровані Класифікація за способом доступу:

1 У файл–серверних СУБД дані зберігаються на центральному файловому сервері, до якого можуть підключатись різні клієнти. Клієнти взаємодіють з цим

сервером, використовуючи мережеві протоколи. Головною перевагою цього типу є централізоване зберігання даних і централізоване управління доступом до них.

2 У клієнт–серверних СУБД функції обробки даних розділені між клієнтськими та серверними компонентами. Серверна частина відповідає за зберігання та керування базою даних, а клієнтські компоненти взаємодіють з сервером для доступу до інформації та виконання операцій. Цей підхід дозволяє розподіляти обробку даних між клієнтами і серверами, забезпечуючи більшу ефективність та масштабованість системи.

3 Інтегровані СУБД поєднують у собі різні типи баз даних і гарантують їх єдиний доступ та управління. Вони дозволяють використовувати різні моделі даних в рамках однієї системи, що дає можливість ефективно працювати з різноманітними типами даних та виконувати складні операції, які потребують комбінації даних з різних джерел.

Системи управління базами даних виконують кілька важливих функцій, які вважаються їх перевагами. Більшість з цих функцій є прозорими для кінцевих користувачів. До них належать:

- Контроль за надмірністю;
- Несуперечливість;
- Спільне використання;
- Покращення цілісності;
- Підвищена безпека;
- Застосування стандартів;
- Можливість знаходження компромісу для суперечливих вимог;
- Підвищення доступності даних і їх готовність до роботи;
- Покращення показників продуктивності;
- Покращене керування паралельним доступом.

Проте, всупереч значним функціональні перевагам, СУБД також і мають суттєві недоліки:

- Збільшення витрат;
- Складність управління;

- Підтримка актуальності;
- Залежність від постачальника;
- Часті цикли оновлення/заміни.

Загалом ці фактори є ключовими при розгляді і виборі системи управління базами даних для конкретного проєкту або організації. Врахування переваг і недоліків СУБД допомагає забезпечити оптимальний вибір технології, яка відповідає вимогам проєкту.

1.3 Проектування бази даних

Проектування бази даних – це процес, який включає поетапний перехід від неформального опису інформаційної структури предметної області до формального опису об'єктів предметної області з використанням певної СУБД.

Проектування БД складається з таких етапів:

- системний аналіз предметної області;
- концептуальне проектування;
- логічне проектування;
- фізичне проектування;
- нормалізація.

Системний аналіз включає у себе мовний опис реальних об'єктів предметної області, визначення зв'язків між об'єктами, аналіз характеристик об'єктів і зв'язків.

Для визначення складу і структури предметної області використовуються або функціональний, або предметний підходи.

Функціональний підхід у системному аналізі баз даних використовується тоді, коли вже відомо, які функції мають виконувати майбутні користувачі системи і також відомі всі задачі, для яких потрібна ця база даних. За допомогою цього підходу можна чітко визначити мінімальний набір об'єктів у предметній області і з'ясувати, як вони взаємодіють між собою. Для цього використовуються виробничі документи, такі як описи процесів, інструкції, специфікації вимог до системи, а також проводяться опитування замовників та користувачів.

Предметний підхід у системному аналізі баз даних використовується, коли інформаційні потреби майбутніх користувачів не є чітко визначеними або заздалегідь відомими. У такому випадку неможливо точно визначити мінімальний набір об'єктів, який буде належати до предметної області. При застосуванні предметного підходу в опис предметної області включаються об'єкти і зв'язки, які є найбільш характерними і суттєвими для неї. База даних, побудована згідно з предметним підходом, називається предметною базою даних. Вона може бути використана для вирішення широкого спектру задач, які не були заздалегідь визначені.

У практичній діяльності використовують комплексний підхід до проектування, який дозволяє розв'язувати конкретні інформаційні та функціональні задачі, а також передбачає можливість додавання нових застосувань у майбутньому. Серед таких підходів слід виокремити: низхідне проектування, що полягає в переході від неформального опису інформаційної структури предметної області до формалізованого опису об'єктів в термінах конкретної СУБД, та висхідне проектування, що базується на створенні моделей об'єктів та їх зв'язків на основі аналізу потреб користувачів та вимог до функціональності. Схема цих підходів зображена на рисунку 1.3.

Рисунок 1.3 – Схема підходів у проектуванні БД

Концептуальне проектування базується на визначені лише основних сутностей, необхідних для зберігання інформації, та зв'язків між цими

сутностями, без уточнення деталей про кожен елемент. На цьому етапі створюється початкова модель, яка служить основою для подальшої розробки бази даних.

У концептуальному проектуванні можна використовувати різні способи візуалізації, наприклад, діаграму сутності–зв'язок (Entity–Relationship Diagram, ERD) або діаграму структури даних (Data Structure Diagram, DSD).

Логічне проектування відповідає за деталізацію концептуальної моделі даних і перетворення її в більш конкретну і структуровану форму. На цьому етапі враховуються вимоги до системи і визначаються точні характеристики сутностей, атрибутів, зв'язків та правил їх взаємодії. Іншими словами, логічне проектування враховує всі деталі для визначення інформаційної області. Однак створена під час цього проектування модель даних ще не залежить від потреби конкретної СУБД, і може бути використана для реалізації бази даних у будь–якій підтримуваній системі управління.

Більшість розробників програмного забезпечення (архітектори, розробники рішень і програмні аналітики) можуть пропустити етам концептуального проектування і почати безпосередньо з логічного.

Фізичний етап проектування є наступним після логічного етапу і спрямований на реалізацію бази даних з використанням певної реляційної системи управління. Однією з головних відмінностей між логічним та фізичним етапом проектування є те, що у фізичному потрібно використовувати специфічні імена таблиць і стовпців замість того, щоб вказувати їх загальні назви. Це дозволяє адаптуватися до обмежень і умовностей бажаної СУБД. .

Нормалізація бази даних (БД) — це процес проектування БД з використанням методу нормальних форм для досягнення оптимальної організації даних і забезпечення їх цілісності. Цей процес є ітераційним, оскільки полягає в послідовному перекладі стосунків з першої нормальної форми (1НФ) в нормальні форми вищого порядку згідно з певними правилами.

Виділяють таку послідовність нормальних форм, яка зображена на рисунку 1.4:

Рисунок 1.4 – Візуальна послідовність нормальних форм

Нормальні форми застосовуються в наступному порядку:

– перша нормальна форма (1НФ);
– друга нормальна форма (2НФ);
– третя нормальна форма (3НФ);
– посилена третя нормальна форма, або нормальна форма Бойса – Кодда (НФБК).

Метою першої нормальної форми є розділення БД на логічні одиниці – таблиці. Після створення таблиць для більшості з них будуть визначені ключові поля. Іншими словами, для переходу до першої нормальної форми БД повинна бути розбита на декілька логічних одиниць, в кожній з яких є визначений ключ і відсутні групи, що повторюються.

Друга нормальна форма розширює першу нормальну форму, вимагаючи, щоб неключові поля в таблиці були пов'язані з усіма атрибутами ключа і не залежали один від одного.

Для досягнення другої нормальної форми потрібно, щоб кожне поле в таблиці, яке не є ключовим, залежало від усього ключа, а не від його частини. Тобто, якщо у таблиці є складний ключ, складений з кількох атрибутів, то кожне неключове поле має залежати від цього комплексного ключа, а не від окремих атрибутів.

Основна ідея другої нормальної форми є в уникненні втрати інформації при модифікації даних. Якщо поля залежать від частин ключа, то при зміні цих частин може виникнути ситуація, коли дані стають некоректними.

Третя нормальна форма є наступним рівнем нормалізації бази даних і доповнює першу і другу нормальні форми. Головна мета третьої нормальної форми усунення транзитивних залежностей між неключовими полями.

Вимоги 3НФ:

- Кожне неключове поле повинно залежати тільки від ключа таблиці;
- Якщо в таблиці є транзитивна залежність між неключовими полями, необхідно розбити цю таблицю на дві або більше таблиць, де кожна таблиця буде мати свій ключ і буде містити тільки поля, що залежать від цього ключа.

Третя нормальна форма допомагає уникнути проблем з дублюванням даних, сприяє зменшенню розміру таблиць, полегшує зміну даних і запитів, а також покращує продувність БД.

НФБК виправляє аномалії, які виникають через перекривання ключів. У ній вимагається, щоб кожна функціональна залежність між неключовим полем і ключем була залежністю від всього ключа, а не лише від його підмножини.

Якщо відношення перебуває в третій нормальній формі (3НФ), і в ньому немає ключів, що перекриваються, то воно автоматично відповідає нормальній формі Бойса–Кодда (НФБК). Це означає, що НФБК є "досконалою" нормальною формою, оскільки вона враховує тільки функціональні залежності.

1.4 Мова програмування запитів SQL

SQL (Structured Query Language) – це стандартна мова для управління та маніпулювання реляційними базами даних. Вперше SQL було розроблено в компанії IBM в 1970–х роках. Початково вона називалась SEQUEL (Структурована англійська мова запитів). Пізніше назву змінили на SQL, і в 1981 році IBM випустила першу комерційну реалізацію мови в рамках СУБД System R.

SQL швидко стала стандартною мовою для роботи з реляційними базами даних. У 1986 і 1987 роках вона набула поширення серед виробників СУБД, таких як Oracle, Sybase і Microsoft.

З часом SQL все розвивалася і отримувала нові можливості. Наприклад, ANSI SQL–92 ввів підтримку тригерів, збережених процедур і представлень. SQL:1999 додала підтримку рекурсивних запитів і користувацьких типів даних.

Наразі існують кілька популярних реалізацій SQL з відкритим вихідним кодом, таких як MySQL і PostgreSQL, які широко використовуються. Хоча з'явилися й інші технології, на кшталт NoSQL, Hadoop і Spark, спрямовані на роботу з неструктурованими даними і розподіленими обчисленнями. Але не зважаючи на це, SQL досі є одним з найважливіших інструментів для створення баз даних.

Загальна структура SQL складається з таких елементів:

– DDL – мови визначення даних;
– DML – мови маніпулювання даними;
– DQL – мови запитів;
– DCL – мови управління даними;
– TCL – мови управління транзакціями.

DDL (Data Definition Language) – це підкатегорія мови SQL, яка використовується для визначення структури бази даних і об'єктів, які зберігаються в ній. Команди DDL дозволяють створювати нові об'єкти, змінювати їх структуру або видаляти існуючі.

Основні команди DDL:

- CREATE – команда створює нові об'єкти бази даних: таблиці, індекси, перегляди, процедури, функції тощо;
- ALTER – команда змінює структури об'єктів бази даних;
- DROP – команда видаляє об'єкти бази даних;
- RENAME – команда змінює назви об'єктів бази даних.

DML (Data Manipulation Language) – це підкатегорія мови SQL, яка реалізує управління даними. Тут використовуються команди для вставки, оновлення, видалення та вибору даних у таблицях.

Основні команди DML:

- INSERT – команда виконує вставку нових записів в таблицю БД;

- UPDATE – команда використовується для оновлення існуючих записів в таблиці. Використовується разом з умовою WHERE;
- DELETE – команда відповідає за видалення записів з таблиці бази даних. Якщо вказати умову WHERE, то таким способом будуть видалятися лише обрані записи.

DQL (Data Query Language) – це підкатегорія мови SQL, яка служить для виконання запитів до бази даних з метою отримання необхідної інформації. Команди DQL дають можливість вибирати, фільтрувати та об'єднувати дані з однієї або декількох таблиць.

Основні команди та оператори DQL:

- SELECT – команда дозволяє вибирати колонки та дані з таблиць;
- FROM – оператор вказує таблиці або перегляди, з яких потрібно вибрати дані;
- WHERE – оператор фільтрації записів на основі певних умов;
- GROUP BY – оператор групування результатів запиту;
- HAVING – оператор фільтрації групованих результатів на основі певних умов;
- ORDER BY – оператор сортування результатів запиту за певними колонками. Можна вказати, як потрібно сортувати дані – по зростанню або спаданню.

DCL (Data Control Language) – це підкатегорія мови SQL, що використовується для керування доступом до бази даних і об'єктів, що зберігаються в ній. Команди DCL призначені встановлювати привілеї доступу, надавати або забирати права користувачів та регулювати рівні безпеки.

Основні команди DCL:

- GRANT – команда надання привілеїв доступу до бази даних або конкретних об'єктів;
- REVOKE – команда відкликання наданих раніше привілеїв доступу;
- DENY – команда заборони доступу до бази даних або об'єктів.

TCL (Transaction Control Language) – це підкатегорія мови SQL, яка забезпечує керування транзакціями в базі даних. Команди TCL дозволяють розпочинати, підтверджувати або скасувати транзакції, забезпечують контроль над цілісністю даних та забезпечують можливість відновлення бази даних до попереднього стану в разі помилки.

Основні команди TCL:

- COMMIT – команда для підтвердження транзакції;
- ROLLBACK – команда скасування транзакції і відновлення бази даних до попереднього стану;
- SAVEPOINT – команда створення знаку (точки відновлення) в межах транзакції.;
- SET TRANSACTION – команда встановлення різних параметрів транзакції, таких як ізольованість, рівень ізоляції та інші.

Наявна у SQL проблема зі стандартизацією пов'язана з тим, що виробники СУБД використовують різні діалекти цієї мови, які часто несумісні між собою.

Щоб запобігти даній проблемі, компаніями ANSI і NIST були розроблені "Рівні відповідності", серед яких виділяють:

– Entry (базовий): Найнижчий рівень відповідності, який вимагає мінімального набору можливостей, необхідних для підтримки SQL стандарту;

– Transitional (перехідний): Перехідний рівень, перевірку на відповідність якого проводить лише інститут NIST;

– Intermediate (проміжний): Цей рівень включає додаткові функції та можливості, які розширюють базовий набір стандарту SQL.

– Full (повний): Найвищий рівень відповідності, який включає всі функції і можливості, передбачені SQL стандартом.

Ці рівні допомагають визначити, які функції та можливості підтримуються в конкретній реалізації SQL. Програмісти можуть використовувати рівні відповідності для забезпечення переносимості свого коду між різними СУБД, вибираючи рівень відповідності, який відповідає їх потребам.

1.5 Інструмент візуального проектування баз даних MySQL Workbench

Засобом розробки було обрано візуальний інструмент проектування MySQL Workbench, який дозволяє створювати бази даних на вибір: або шляхом використання графічного інтерфейсу, або використанням SQL–коду.

MySQL Workbench є високоефективним програмним середовищем, призначеним для полегшення роботи адміністраторів баз даних, архітекторів баз даних і розробників MySQL. Він пропонує можливість створення моделей даних, розроблення SQL–запитів та інструментів для управління і налаштування серверів БД.

MySQL Workbench – це кросплатформений інструмент з відкритим вихідним кодом. Цей інструмент об'єднує в собі функціональності проектування, розробки, створення, адміністрування та обслуговування баз даних.

Порівняно з деякими іншими системами управління базами даних: Oracle, DB2, MS Access і Microsoft SQL Server, MySQL Workbench пропонує кілька значущих переваг, на які слід звернути увагу при виборі..

На відмінну від перелічених СУБД MySQL здатен підтримувати декілька механізмів зберігання. До того ж, завдяки своїй простоті та дизайну він забезпечує збільшену продуктивність, що робить його привабливішим для користувачів.

MySQL Workbench може створювати моделі та керувати ними, перетворювати динамічну базу даних на модель, а також створювати і редагувати таблиці та вставляти дані.

Ще цей графічний інтерфейс дозволяє створювати, керувати та налаштовувати з'єднання і параметри підключення до серверів баз даних MySQL. Він також дає змогу виконувати SQL–запити до цих з'єднань за допомогою вбудованого редактора.

Редактор Visual SQL створює, керує і виконує запити. Він має автозаповнення і кольорові виділення, які допомагають легко працювати з операторами SQL.

Також MySQL Workbench значно спрощує управління користувачами. Таким чином, можна легко:

- Переглядати інформацію про облікові записи всіх користувачів;
- Додавати та видалити користувачів;
- Надавати та відкликати привілеї;
- Змінювати глобальні дозволи та дозволи бази даних;
- Змінювати паролі;
- Проводити аудит аби дізнатися, хто що коли робив.

Щоб встановити MySQL Workbench в рамках дипломного проєкту, необхідно завантажити MySQL Installer потрібної версії з офіційного веб–сайту Oracle. Під час встановлення MySQL Workbench через MySQL Installer можна вибрати інші компоненти та модулі, які також можуть бути корисними для роботи з MySQL. Наприклад, можна встановити сервер баз даних MySQL, клієнт MySQL Shell, адміністративні інструменти, драйвери для різних мов програмування тощо.

Після вибору необхідних компонентів та модулів MySQL Installer автоматично завантажує та встановлює їх на комп'ютері. Крім того, під час встановлення можна налаштувати параметри підключення до бази даних, наприклад, вказати користувача та пароль для доступу до MySQL Server. Запуск стартового вікна MySQL Workbench після встановлення зображено на рисунку 2.1.

Рисунок 1.5 – Стартове вікно MySQL Workbench

Встановлення середовища розробки MySQL Workbench дозволяє безпосередньо перейти до етапу програмної реалізації бази даних.

2 ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Постановка задачі

Задача проектування бази даних для лікарні ветеринарної медицини полягає в створенні системи зберігання, отримання та оновлення інформації про клієнтів, тварин і їх медичних історій, ветеринарів, записів в лікарню, а також інформацію про надані послуги та сплати цих послуг.

Для розв'язання даної задачі потрібно враховувати наступні потреби:

- Зберігання даних про ветеринарів: особисті дані, професійну та контактну інформацію;
- Зберігання даних про клієнтів: особисті дані, контактну інформацію;
- Зберігання даних про тварин: особисті дані, фізіологічні особливості, медичну історію;
- Зберігання даних про записи, включно з датою, скаргами та призначеннями;
- Зберігання даних про послуги, а саме назви послуг і ціни;
- Зберігання даних про оплату послуг: сума сплати та час.

При розробці баз даних потрібно звернути увагу на такі питання як: які типи даних необхідні для зберігання, які зв'язки між таблицями потрібні, які запити мають бути забезпечені системою та які додаткові функції можна впровадити для покращення роботи ветеринарної лікарні.

2.2 Опис предметної області

Предметна область ветеринарної лікарні охоплює діяльність пов'язану з обліком тварин та з наданням ветеринарних послуг. Послуги застосовуються для лікування, попередження захворювань та проведення медичних діагностик. Ветеринарна лікарня може надавати послуги з профілактики і лікування хвороб, проводити різноманітні медичні дослідження та тестування.

У контексті даної предметної області, клієнтом називається власник тварини. Для того, щоб можна було внести клієнта в базу даних від нього буде вимагатися особиста інформація, така як: ім'я, прізвище, ім'я по батькові, домашня адреса і для встановлення зв'язку обов'язково потрібно зазначити контактну інформацію, а саме номер телефону і за наявності електронну пошту.

Під назвою «пацієнт» буде розумітися тварина, котра власне і потребує лікарської допомоги. Для проведення обліку тварини необхідно отримати інформацію про її вид та породу, деякі фізіологічні дані: вік та вагу. А також потрібно додати медичну історію, яка міститиме інформацію про вакцинації, хвороби і операції; з застосуванням такої історії ветеринару буде простіше відслідкувати стан здоров'я пацієнта.

Ветеринар – це працівник лікарні, який займається оглядом тварин та наданням послуг для них. Щоб можна було визначити особу ветеринара, потрібно за аналогією до клієнта вказати його особисті та контактні дані, також важливо вказати його стаж та посаду в ветеринарній лікарні, оскільки ця інформація допоможе організувати роботу з ветеринаром і визначити його кваліфікацію.

Вся комунікація між ветеринарами і клієнтами має проводитись під час запису. В записі буде зазначатись причина з якої пацієнту потрібна допомога, час цього запису та ветеринарні призначення.

Згідно з призначень будуть надаватися послуги, які в свою чергу зобов'язаний в кінці оплатити клієнт.

2.3 Встановлення сутностей та взаємозв'язків

На основі поставленої задачі та опису предметної області можна встановити наступні сутності:

- Ветеринари;
- Послуги;
- Клієнти;
- Пацієнти;

- Медичні картки;
- Медичні записи;
- Оплати.

Взаємозв'язки цих сутностей:

Сутність «Ветеринари» пов'язується з сутністю «Медичні записи» згідно типу «один до багатьох», тому що один ветеринар може приймати безліч записів.

Сутність «Клієнти» пов'язується з сутністю «Пацієнти» згідно типу «один до багатьох», тому що один клієнт може мати одразу кілька тварин.

Сутність «Пацієнти» пов'язується з сутністю «Медичні картки» згідно типу «один до одного», тому що за окремим пацієнтом можна закріпити лише одну медичну картку.

Сутність «Пацієнти» пов'язується з сутністю «Медичні записи» згідно типу «один до багатьох», тому що кожен пацієнт може бути записаний на прийом неодноразово.

Сутність «Медичні записи» пов'язується з сутністю «Послуги» згідно типу «багато до багатьох», тому що під час запису може надаватися кілька різних послуг, а одні й ті самі послуги можуть надаватися під час різних записів. В результаті такого зв'язку створюється зв'язна сутність.

Сутність «Оплати» пов'язується з сутністю «Медичні записи» згідно типу «один до одного», тому що кожен запис оплачується одноразово.

Таким чином, для кожної з сутностей потрібно додати необхідні атрибути та дати їм визначення :

Сутність «Ветеринари» відображає інформацію про ветеринарів, що працюють в ветеринарній лікарні. Характеристики:

- Прізвище – призначене для зберігання інформації про прізвище ветеринара;
- Ім'я – призначене для зберігання інформації про ім'я ветеринара;
- Ім'я по батькові – призначене для зберігання інформації про ім'я по батькові ветеринара;

- Посада – призначене для зберігання інформації про посаду ветеринара;
- Стаж – призначене для зберігання інформації про стаж ветеринара, який визначається в роках;
- Номер телефону – призначене для зберігання інформації про номер телефону ветеринара;
- Електронна пошта – призначене для зберігання інформації про електронну пошту ветеринара.

Сутність «Клієнти» відображає інформацію про власників тварин, які звертаються за допомогою до ветеринарної лікарні. Характеристики:

- Прізвище – призначене для зберігання інформації про прізвище клієнта;
- Ім’я – призначене для зберігання інформації про ім’я клієнта;
- Ім’я по батькові – призначене для зберігання інформації про ім’я по батькові клієнта;
- Адреса – призначене для зберігання інформації про адресу проживання клієнта;
- Номер телефону – призначене для зберігання інформації про номер телефону клієнта;
- Електронна пошта – призначене для зберігання інформації про електронну пошту клієнта.

Сутність «Послуги» відображає інформацію про послуги, які надають в ветеринарній лікарні. Характеристики:

- Назва послуги – впроваджується, щоб зберігати найменування послуги;
- Одиниця вимірювання – призначене для зберігання інформації про одиницю в якій вимірюється послуга;
- Ціна – призначене для зберігання інформації про ціну послуги.

Сутність «Пацієнти» відображає інформацію про тварин, які потребують ветеринарної допомоги. Характеристики:

- Кличка – призначене для зберігання інформації про кличку тварини;
- Вид – призначене для зберігання інформації про вид тварини;
- Порода – призначене для зберігання інформації про породу тварини;
- Вік – призначене для зберігання інформації про вік тварини, який вимірюється в роках.
- Вага – призначене для зберігання інформації про вагу тварини, яка вимірюється в кілограмах;
- Стать – призначене для зберігання інформації про приналежність тварини до чоловічої чи жіночої статі.

Сутність «Медичні записи» » відображає інформацію про конкретний запис пацієнта до ветеринарного лікаря. Характеристики:

- Дата запису – призначене для зберігання інформації про дату та час запису;
- Причина – призначене для зберігання інформації про скарги на здоров'я пацієнта, які стали причиною запису;
- Призначення – призначене для зберігання інформації про призначення які дає ветеринар під час запису.

Сутність «Оплати» відображає інформацію про оплату клієнтами наданих під час запису послуг. Характеристики:

- Сума оплати – призначене для зберігання інформації про суму оплати;
- Дата оплати – призначене для зберігання інформації про дату та час оплати;

Сутність «Медичні карти» » відображає інформацію про медичні картки, які кріпляться за пацієнтами в ветеринарній лікарні. Характеристики:

- Вакцинації – призначене для зберігання інформації про вакцинації пацієнта, включно з назвою вакцини і датою останнього надання;
- Хвороби – призначене для зберігання інформації про хвороби пацієнта, а саме: назву хвороби, дату виявлення та період лікування;
- Операції – призначене для зберігання інформації про операції пацієнта, де вказується назва операції та дата її проведення.

2.4 Побудова ER-діаграми

За допомогою MySQL Workbench із зібраної інформації було створено ER-діаграму даних, яка зображується на рисунку 2.1.

Рисунок 2.1 - ER-діаграма для заданої предметної області

2.5 Проектування структури даних

Під час проектування структури даних БД, або іншими словами побудови логічної моделі, відбувається впровадження структури таблиць із зазначенням первинних ключів, а також «зв'язків» між відносинами, що представляють собою зовнішні ключі. На цьому етапі вказуються наступні властивості даних: назви атрибутів в СУБД, тип, розмір та застосування обмежень NOT NULL.

Проектування структури даних таблиці «VETS» на основі опису сутності «Ветеринари» в табл. 3.1.

Таблиця 2.1 – Таблиця «VETS»

Назва атрибуту	Тип даних	Первинний ключ	Зовнішній ключ	NOT NULL
vetID	INT	*		
vetFname	VARCHAR(50)			*
vetSname	VARCHAR(50)			*
vetPname	VARCHAR(50)			
position	VARCHAR(100)			*
experienceYears	INT			*
vetPhone	VARCHAR(13)			*
vetEmail	VARCHAR(50)			*

Проектування структури даних таблиці «SERVICES» на основі опису сутності «Послуги» в табл. 2.2.

Таблиця 2.2 – Таблиця «SERVICES»

Назва атрибуту	Тип даних	Первинний ключ	Зовнішній ключ	NOT NULL
serviceID	INT	*		
serviceName	VARCHAR(100)			*
unitofMeasure	VARCHAR(20)			*
servicePrice	DECIMAL(10,2)			*

Проектування структури даних таблиці «CLIENTS» на основі опису сутності «Клієнти» в табл. 2.3.

Таблиця 2.3 – Таблиця «SERVICES»

Назва атрибуту	Тип даних	Первинний ключ	Зовнішній ключ	NOT NULL
clientID	INT	*		
clientFname	VARCHAR(50)			*
clientSname	VARCHAR(50)			*
clientPname	VARCHAR(50)			
clientAddress	VARCHAR(255)			*
clientPhone	VARCHAR(13)			*
clientEmail	VARCHAR(50)			

Проектування структури даних таблиці «PATIENTS» на основі опису сутності «Пацієнти» в табл. 2.4.

Таблиця 2.4 – Таблиця «PATIENTS»

Назва атрибуту	Тип даних	Первинний ключ	Зовнішній ключ	NOT NULL
patientID	INT	*		
patientName	VARCHAR(20)			*
species	VARCHAR(20)			*
breed	VARCHAR(50)			*
age	INT			*
weight	FLOAT			*
gender	ENUM('Чоловіча', 'Жіноча')			*
clientID	INT		*	*

Проектування структури данихтаблиці «MEDCARDS» на основі опису сутності «Медичні картки» в табл. 2.5.

Таблиця 2.5 – Таблиця «MEDCARDS»

Назва атрибуту	Тип даних	Первинний ключ	Зовнішній ключ	NOT NULL
cardID	INT	*	*	
vaccinationHistory	TEXT			
diseasesHistory	TEXT			
surgeryHistory	TEXT			

Проектування структури данихтаблиці «MEDRECORDS» на основі опису сутності «Медичні записи» в табл. 2.6.

Таблиця 2.6 – Таблиця «MEDRECORDS»

Назва атрибуту	Тип даних	Первинний ключ	Зовнішній ключ	NOT NULL
recordID	INT	*		
recordDatetime	DATETIME			*
reason	VARCHAR(255)			*
prescription	TEXT			*
patientID	INT	*		*
vetID	INT	*		*

Проектування структури данихтаблиці «PAYMENTS» на основі опису сутності «Оплати» в табл. 2.7.

Таблиця 2.7 – Таблиця «PAYMENTS»

Назва атрибуту	Тип даних	Первинний ключ	Зовнішній ключ	NOT NULL
paymentID	INT	*	*	
paymentAmount	DECIMAL(10,2)			*
paymentDatetime	DATETIME			*

Згідно опису взаємозв'язків з метою нормалізації бази даних, щоб уникнути зв'язку багато до багатьох між таблицями «MEDRECORDS» та «SERVICES» буде впроваджено проміжну таблицю «RECORD_SERVICE» представлену в табл.2.8.

Таблиця 2.8 – Таблиця «RECORD_SERVICE»

Назва атрибуту	Тип даних	Первинний ключ	Зовнішній ключ	NOT NULL
recordID	INT		*	*
serviceID	INT		*	*

3 ПРОГРАМНА РЕАЛІЗАЦІЯ БАЗИ ДАНИХ

3.1 Реалізація таблиць бази даних в програмному середовищі

Реалізація таблиць за допомогою обраних програмних засобів є важливою складовою процесу розробки баз даних. У даній роботі таким програмний засобом було обрано MySQL Workbench, який як вже вказувалось раніше дозволяє створювати таблиці як за допомогою графічного інтерфейсу, так і безпосередньо за допомогою команд мовою SQL. В подальшому буде використано саме другий спосіб, оскільки таким чином процес розробки стане більш гнучким та ефективним.

Таблиця «VETS». Код створення таблиці наведено далі:

```
CREATE TABLE VETS (
  vetID INT AUTO_INCREMENT PRIMARY KEY,
  vetFname VARCHAR(50) NOT NULL,
  vetSname VARCHAR(50) NOT NULL,
  vetPname VARCHAR(50),
  position VARCHAR(100) NOT NULL,
  experienceYears INT NOT NULL,
  vetPhone VARCHAR(13) NOT NULL,
  vetEmail VARCHAR(50) NOT NULL
);
```

Структура таблиці «VETS» має наступні пояснення:

- «vetID» – містить унікальний ідентифікатор ветеринара. Поле з цілим числовим типом, автоматичним приростом та первинним ключем;
- «vetFname» – містить прізвище ветеринара. Поле з символьним типом та обов'язковим заповненням довжиною до 50 символів;
- «vetSname» – містить ім’я ветеринара. Поле з символьним типом та обов'язковим заповненням довжиною до 50 символів;

- «vetPname» – містить ім’я по батькові ветеринара. Поле з символьним типом та обов'язковим заповненням довжиною до 50 символів;
- «position» – містить посаду ветеринара. Поле з символьним типом та обов'язковим заповненням довжиною до 50 символів;
- «experienceYears» – містить стаж роботи ветеринара. Поле з цілим числовим типом та обов'язковим заповненням;
- «vetPhone» – містить номер телефону ветеринара. Поле з символьним типом та обов'язковим заповненням довжиною до 13 символів;
- «vetEmail» – містить електронну пошту ветеринара. Поле з символьним типом та обов'язковим заповненням довжиною до 50 символів.

Таблиця «SERVICES». Код створення таблиці наведено далі:

```
CREATE TABLE SERVICES (
  serviceID INT AUTO_INCREMENT PRIMARY KEY,
  serviceName VARCHAR(100) NOT NULL,
  unitofMeasure VARCHAR(20) NOT NULL,
  servicePrice DECIMAL(10,2)  NOT NULL
);
```

Структура таблиці «SERVICES» має наступні пояснення:

- «serviceID» – містить унікальний ідентифікатор послуги. Поле з цілим числовим типом, автоматичним приростом та первинним ключем;
- «serviceName» – містить назву послуги. Поле з символьним типом та обов'язковим заповненням довжиною до 100 символів;
- «unitofMeasure» – містить одиницю виміру послуги. Поле з символьним типом та обов'язковим заповненням довжиною до 20 символів;
- «servicePrice» – містить вартість послуги. Поле з числовим типом десяткової точності з 10 символами до коми і 2 після та обов'язковим заповненням.

Таблиця «CLIENTS». Код створення таблиці наведено далі:

```
CREATE TABLE CLIENTS (
  clientID INT AUTO_INCREMENT PRIMARY KEY,
```

```
  clientFname VARCHAR(50) NOT NULL,
  clientSname VARCHAR(50) NOT NULL,
  clientPname VARCHAR(50),
  clientAddress  VARCHAR(255) NOT NULL,
  clientPhone  VARCHAR(13) NOT NULL,
  clientEmail  VARCHAR(50)
);
```

Структура таблиці «CLIENTS» має наступні пояснення:

- «clientID» – містить унікальний ідентифікатор клієнта. Поле з цілим числовим типом, автоматичним приростом та первинним ключем;
- «clientFname» – містить прізвище клієнта. Поле з символьним типом та обов'язковим заповненням довжиною до 50 символів;
- «clientSname» – містить ім’я клієнта. Поле з символьним типом та обов'язковим заповненням довжиною до 50 символів;
- «clientPname» – містить ім’я по батькові клієнта. Поле з символьним типом та опціональним заповненням довжиною до 50 символів;
- «clientAddresi» – містить адресу клієнта. Поле з символьним типом та обов'язковим заповненням довжиною до 255 символів;
- «clientPhone» – містить номер телефону клієнта. Поле з символьним типом та обов'язковим заповненням довжиною до 13 символів;
- «clientEmail» – містить електронну пошту клієнта. Поле з символьним типом та опціональним заповненням довжиною до 50 символів.

Таблиця «PATIENTS». Код створення таблиці наведено далі:

```
CREATE TABLE PATIENTS(
  patientID INT AUTO_INCREMENT PRIMARY KEY,
  patientName VARCHAR(20) NOT NULL,
  species VARCHAR(20) NOT NULL,
  breed VARCHAR(50) NOT NULL,
  age INT NOT NULL,
  weight FLOAT NOT NULL,
```

gender enum('Чоловіча', 'Жіноча') NOT NULL,
clientID INT NOT NULL,
FOREIGN KEY (clientID) REFERENCES CLIENTS(clientID)
);

Структура «PATIENTS» має наступні пояснення:

- «patientID» – містить унікальний ідентифікатор тварини. Поле з цілим числовим типом, автоматичним приростом та первинним ключем;
- «patientName» – містить кличку тварини. Поле з символьним типом та обов'язковим заповненням довжиною до 20 символів;
- «species» – містить вид тварини. Поле з символьним типом та обов'язковим заповненням довжиною до 20 символів;
- «breed» – містить породу тварини. Поле з символьним типом та обов'язковим заповненням довжиною до 50 символів;
- «age» – містить вік тварини. Поле з цілим числовим типом та обов'язковим заповненням;
- «weight» – містить вагу тварини. Поле з числовим типом з плаваючою крапкою та обов'язковим заповненням;
- «gender» – містить стать тварини. Поле з типом перерахунку зі значеннями на вибір «Чоловіча» або «Жіноча» стать та обов'язковим заповненням;
- «clientID» – ідентифікатор клієнта, якому належить тварина. Поле з цілим числовим типом та обов'язковим заповненням, яке є зовнішнім ключем, що посилається на «clientID» таблиці «CLIENTS».

FOREIGN KEY (clientID) REFERENCES CLIENTS(clientID) – зв'язок з таблицею «CLIENTS», в полі «clientID».

Таблиця «MEDCARDS». Код створення таблиці наведено далі:

CREATE TABLE MEDCARDS (
cardID INT AUTO_INCREMENT PRIMARY KEY,
vaccinationHistory TEXT,
diseasesHistory TEXT,
surgeryHistory TEXT,

FOREIGN KEY (cardID) REFERENCES PATIENTS (patientID)

);

Структура «MEDCARDS» має наступні пояснення:

- «cardID» – містить унікальний ідентифікатор медичної картки. Поле з цілим числовим типом, автоматичним приростом та первинним ключем, яке є одночасно зовнішнім ключем, що посилається на “ patientID” таблиці “PATIENTS”;
- «vaccinationHistory» – містить історію вакцинацій пацієнта. Поле з текстовим типом;
- «diseasesHistory» – містить історію хвороб пацієнта. Поле з текстовим типом;
- «surgeryHistory» – містить історію операцій, проведених над пацієнтом. Поле з текстовим типом.

FOREIGN KEY (cardID) REFERENCES PATIENTS(patientID) – зв'язок з таблицею «PATIENTS», в якому поле «cardID» = «patientID».

Таблиця «MEDRECORDS». Код створення таблиці наведено далі:

```
CREATE TABLE MEDRECORDS  (
 recordID INT AUTO_INCREMENT PRIMARY KEY,
 recordDatetime DATETIME NOT NULL,
 reason VARCHAR(255) NOT NULL,
 prescription TEXT NOT NULL,
 patientID INT NOT NULL,
 vetID INT NOT NULL,
FOREIGN KEY (patientID) REFERENCES PATIENTS (patientID),
FOREIGN KEY (vetID) REFERENCES VETS (vetID)
);
```

Структура «MEDRECORDS» має наступні пояснення:

- «recordID» – містить унікальний ідентифікатор медичного запису. Поле з цілим числовим типом, автоматичним приростом та первинним ключем;

- «recordDatetime» – містить дату та час призначення запису. Поле з типом дати та часу та обов'язковим заповненням;
- «reason» – містить причину звернення до ветеринара. Поле з символьним типом та обов'язковим заповненням довжиною до 255 символів;
- «prescription» – містить призначення ветеринара для лікування. Поле з текстовим типом та обов'язковим заповненням;
- «patientID» – містить унікальний ідентифікатор тварини–пацієнта. Поле з цілим числовим типом та обов'язковим заповненням, яке є зовнішнім ключем, що посилається на «patientID» таблиці «PATIENTS»;
- «vetID» – містить унікальний ідентифікатор ветеринара. Поле з цілим числовим типом та обов'язковим заповненням, яке є зовнішнім ключем, що посилається на «vetID» таблиці «VETS».

FOREIGN KEY (patientID) REFERENCES PATIENTS(patientID) – зв'язок з таблицею «PATIENTS», в полі «patientID».

FOREIGN KEY (vetID) REFERENCES VETS(vetID) – зв'язок з таблицею «VETS», в полі «vetID».

Таблиця «RECORD_SERVICE. Код створення таблиці наведено далі:

```
CREATE TABLE RECORD_SERVICE (
  recordID INT NOT NULL,
  serviceID INT NOT NULL,
FOREIGN KEY (recordID) REFERENCES  MEDRECORDS (recordID),
FOREIGN KEY (serviceID) REFERENCES SERVICES (serviceID)
);
```

Структура «RECORD_SERVICES» має наступні пояснення:

- «recordID» – містить унікальний ідентифікатор медичного запису. Поле з цілим числовим типом та обов'язковим заповненням, яке є зовнішнім ключем, що посилається на «recordID» таблиці «MEDRECORDS»;
- «serviceID» – містить унікальний ідентифікатор послуги. Поле з цілим числовим типом та обов'язковим заповненням, яке є зовнішнім ключем, що посилається на «serviceID» таблиці «SERVICES».

FOREIGN KEY (recordID) REFERENCES MEDRECORDS(recordID) – зв'язок з таблицею «MEDRECORDS», в полі «recordID».

FOREIGN KEY (serviceID) REFERENCES SERVICES(serviceID) – зв'язок з таблицею «SERVICES», в полі «serviceID».

Таблиця «PAYMENTS». Код створення таблиці наведено далі:

```
CREATE TABLE PAYMENTS  (
 paymentID INT AUTO_INCREMENT PRIMARY KEY,
 paymentAmount DECIMAL(10,2) NOT NULL,
 paymentDatetime DATETIME NOT NULL,
FOREIGN KEY (paymentID) REFERENCES  MEDRECORDS (recordID)
);
```

Структура «PAYMENTS» має наступні пояснення:

- «paymentID» – містить унікальний ідентифікатор оплати. Поле з цілим числовим типом, автоматичним приростом та первинним ключем;
- «paymentAmount" – містить суму оплати. Поле з числовим типом десяткової точності з 10 символами до коми і 2 після та обов'язковим заповненням;
- «paymentDatetime» – містить дату та час оплати. Поле з типом дати та часу (DATETIME) та обов'язковим заповненням.

FOREIGN KEY (paymentID) REFERENCES MEDRECORDS (recordID) – зв'язок з таблицею "MEDRECORDS", в якому поле «paymentID» = «recordID».

Після того як таблиці були створені необхідно додати до них відповідні дані. Додавання виконується за допомогою команди INSERT. В загальному випадку синтаксис команди INSERT має наступний вигляд:

```
INSERT INTO назва_таблиці VALUES (значення1, значення2, …);
```

Результати додавання даних для кожної з таблиць продемонстровані в подальших рисунках. Детальний код цього процесу описаний у додатку А.

Вибірка таблиць відбувається з використанням запиту SELECT за таким синтаксисом:

SELECT * FROM назва_бази_даних.назва_таблиці;

Вигляд таблиці «VETS» після внесення даних зображено на рисунку 3.1.

Рисунок 3.1 – Вигляд таблиці «VETS»

Вигляд таблиці «SERVICES» після внесення даних зображено на рисунку 3.2.

Рисунок 3.2 – Вигляд таблиці «SERVICES»

Вигляд таблиці «CLIENTS» після внесення даних зображено на рисунку 3.3.

Рисунок 3.3 – Вигляд таблиці «CLIENTS»

Вигляд таблиці «PATIENTS» після внесення даних зображено на рисунку 3.4.

Рисунок 3.4 – Вигляд таблиці «PATIENTS»

Вигляд таблиці «MEDCARDS» після внесення даних зображено на рисунку 3.5.

Рисунок 3.5 – Вигляд таблиці «MEDCARDS»

Вигляд таблиці «MEDRECORDS» після внесення даних зображено на рисунку 3.6.

Рисунок 3.6 – Вигляд таблиці «MEDRECORDS»

Вигляд таблиці «RECORD_SERVICES» після внесення даних зображено на рисунку 3.7.

recordID	serviceID
1011	200771
1012	200771
1013	200775
1014	200791
1015	200793
1016	200793
1017	200777
1018	200777
1018	200778
1019	200792
1020	200792
1021	200792
1022	200793
1023	200793
1024	200783
1025	200793
1026	200794
1027	200794
1028	200773

Рисунок 3.7 – Вигляд таблиці «RECORD_SERVICES»

Вигляд таблиці «PAYMENTS» після внесення даних зображено на рисунку 3.8.

paymentID	paymentAmount	paymentDatetime
1011	76.71	2023-01-19 10:13:03
1012	76.71	2023-01-22 15:11:10
1013	289.44	2023-01-22 14:23:12
1014	52.19	2023-01-23 10:43:15
1015	211.93	2022-01-23 12:17:43
1016	211.93	2023-01-23 12:17:43
1017	39.15	2023-01-24 11:45:18
1018	57.69	2023-01-25 12:12:03
1019	22.15	2023-01-26 10:15:12
1020	22.15	2023-01-26 10:15:12
1021	22.15	2023-01-26 10:15:12
1022	59.82	2023-01-31 11:13:56
1023	59.82	2023-01-31 11:13:56
1024	132.56	2023-02-01 16:11:05
1025	211.93	2023-02-03 09:45:18
1026	119.31	2023-02-04 12:40:15
1027	119.31	2023-02-04 13:04:20
1028	124.54	2023-02-05 15:49:18

Рисунок 3.8 – Вигляд таблиці «PAYMENTS»

3.2 Формування програмованих запитів

Однією з головних особливостей баз даних є можливість швидкого та ефективного вилучення потрібної інформації. Забезпечення такого вилучення відбувається з застосуванням запитів, які дозволяють виконувати пошук і обробку даних у потрібному форматі та зручному для користувача вигляді.

1. Запит на вибірку всіх тварин певного власника за його прізвищем.

В прикладі потрібно звернутися до таблиці «PATIENTS» та вивести дані по всіх тваринах, власних яких має прізвище Цимбалюк. Відповідний код запиту:

```
SELECT patientID, patientName, gender, species, breed, age, weight, clientFname, clientSname
FROM PATIENTS
INNER JOIN CLIENTS ON PATIENTS.clientID = CLIENTS.clientID
WHERE clientFname = 'Цимбалюк';
```

Результат виконання запиту 1 можна побачити на рисунку 3.9.

Рисунок 3.9 – Результат виконання запиту 1

2. Запит на вибірку всіх тварин певної породи, які є молодшими за n років.

В прикладі потрібно звернутися до таблиці «PATIENTS», вивести дані по всіх тваринах, породу якої було обрано – безпородна, а вік менший ніж 5 років. Відповідний код запиту:

```
SELECT * PATIENTS
FROM
WHERE breed = 'Безпородна' AND age < 5;
```

Результат виконання запиту 2 можна побачити на рисунку 3.10.

Рисунок 3.10 – Результат виконання запиту 2

3. Запит на вибірку всіх ветеринарів з певним досвідом роботи.

В прикладі потрібно звернутись до таблиці «VETS» та вивести дані по всіх ветеринарах, досвід яких було обрано більшим ніж 20 років. Відповідний код запиту:

```
SELECT *
FROM VETS
WHERE experienceYears > 20;
```

Результат виконання запиту 3 можна побачити на рисунку 3.11.

Рисунок 3.11 – Результат виконання запиту 3

4. Запит на вибірку всіх записів медичних карток тварин, у яких згадується певна вакцинація.

В прикладі потрібно звернутись до таблиці «MEDCARDS», вивести дані про всіх тварин, які отримували щеплення проти сибірки. Відповідний код запиту:

```
SELECT *
FROM MEDCARDS
WHERE vaccinationHistory LIKE '%Вакцина проти сибірки%';
```

Результат виконання запиту 4 можна побачити на рисунку 3.12.

Рисунок 3.12 – Результат виконання запиту 4

5. Запит на вибірку всіх записів про прийоми, які здійснив певний ветеринар.

В прикладі потрібно звернутися до таблиці «MEDRECORDS» та вивести дані по всіх записах про прийом, які здійснив певний ветеринар, наприклад, з vetID = 10836. Відповідний код запиту:

```
SELECT recordID, appointmentDate, reason, prescription, patientID
FROM MEDRECORDS
WHERE vetID = '10836';
```

Результат виконання запиту 5 можна побачити на рисунку 3.13.

Рисунок 3.13 – Результат виконання запиту 5

6. Запит на вибірку всіх записів про платежі за прийоми за певний період.

В прикладі потрібно звернутися до таблиці «PAYMENTS», вивести дані по всіх записах про оплату, які здійснені за певний період, наприклад, з 2023–01–26 по 2023–02–04. Відповідний код запиту:

```
SELECT *
FROM PAYMENTS
WHERE paymentDatetime BETWEEN '2023–01–26' AND '2023–02–04';
```

Результат виконання запиту 6 можна побачити на рисунку 3.14.

Рисунок 3.14 – Результат виконання запиту 6

7. Запит на вибірку кількості тварин різних порід.

В прикладі потрібно звернутися до таблиці «PATIENTS», вивести дані про кількость тварин різних порід згруповних за породами. Відповідний код запиту:

```
SELECT breed, COUNT(*) as total
FROM PATIENTS
GROUP BY breed;
```

Результат виконання запиту 7 можна побачити на рисунку 3.15.

Рисунок 3.15 – Результат виконання запиту 7

8. Запит на вибірку кількості записів у медичних картках тварин певного власника.

В прикладі потрібно звернутися до таблиць «CLIENTS», «PATIENTS», «MEDCARDS» та вивести скільки існує записів карток тварин у власника з обраним прізвищем, наприклад Михайлюк. Відповідний код запиту:

```
SELECT CLIENTS.clientFname, COUNT(MEDCARDS.cardID) as total_cards
```

```
FROM CLIENTS
LEFT JOIN PATIENTS ON CLIENTS.clientID =PATIENTS.clientID
LEFT JOIN MEDCARDS ON PATIENTS.patientID = MEDCARDS.cardID
WHERE CLIENTS.clientFname = 'Михайлюк'
GROUP BY CLIENTS.clientFname;
```

Результат виконання запиту 8 можна побачити на рисунку 3.16

Рисунок 3.16 – Результат виконання запиту 8

9. Запит на вибірку ветеринарів, що надали найбільшу кількість послуг.

В прикладі нам потрібно проаналізувати кілька взаємопов'язаних таблиць та порахувати по кожному ветеринару загальну кількість наданих йому послуг, фільтрація в обрахунках встановлена по спаду. Відповідний код запиту:

```
SELECT VETS.vetID, VETS.vetFname, COUNT(RECORD_SERVICE.serviceID) as total_services
FROM VETS
INNER JOIN MEDRECORDS ON VETS.vetID = MEDRECORDS.vetID
INNER JOIN RECORD_SERVICE ON MEDRECORDS.recordID = RECORD_SERVICE.recordID
GROUP BY VETS.vetID, VETS.vetFname
ORDER BY total_services DESC;
```

Результат виконання запиту 9 можна побачити на рисунку 3.17.

Рисунок 3.17 – Результат виконання запиту 9

10. Запит на вибірку кількості тварин різної статі.

В прикладі потрібно звернутися до таблиці «PATIENTS», аби порахувати кількість тварин, які належать до кожної зі статей. Відповідний код запиту:

```
SELECT gender, COUNT(*) as total
FROM PATIENTS
GROUP BY gender;
```

Результат виконання запиту 10 можна побачити на рисунку 3.18.

Рисунок 3.18 – Результат виконання запиту 10

11. Запит на вибірку власників тварин та загальної кількості їх тварин.

В прикладі потрібно звернутися до таблиць «CLIENTS» і «PATIENTS», аби порахувати загальну кількість тварин, котрі кріпляться за кожним клієнтом. Відповідний код запиту:

```
SELECT CLIENTS.clientID, CLIENTS.clientFname, COUNT(*) as total_patients
FROM CLIENTS
LEFT JOIN PATIENTS ON CLIENTS.clientID =PATIENTS.clientID
GROUP BY CLIENTS.clientID, CLIENTS.clientFname;
```

Результат виконання запиту 11 можна побачити на рисунку 3.19.

Рисунок 3.19 – Результат виконання запиту 11

12. Запит на вибірку найбільшої кількості оплат, здійснених у певний день.

В прикладі потрібно звернутися до таблиці «PAYMENTS», аби дізнатися, якою була максимальна кількість оплат за день, додатковим параметром обрано зробити вибірку трьох кращих днів. Відповідний код запиту:

```
SELECT paymentDatetime, COUNT(*) as total_payments
FROM PAYMENTS
GROUP BY paymentDatetime
ORDER BY total_payments DESC
LIMIT 3;
```

Результат виконання запиту 12 можна побачити на рисунку 3.20.

Рисунок 3.20 – Результат виконання запиту 12

3.3 Впровадження тригерів

Тригери – це блоки коду, які виконуються автоматично при певних подіях, таких як вставка (INSERT), оновлення (UPDATE) або видалення (DELETE) даних у таблиці. Вони призначенні для проведення автоматичних операцій з даними та контролю дій користувачів БД.

Синтаксис тригерів в MySQL Workbench виглядає так:

```
DELIMITER &&
CREATE TRIGGER НАЗВА_ТРИГЕРА
{BEFORE | AFTER} {INSERT | UPDATE | DELETE} ON НАЗВА_ТАБЛИЦІ
FOR EACH ROW
BEGIN
ТІЛО_ТРИГЕРА
END &&
DELIMITER;
```

Пояснення синтаксису:

- CREATE TRIGGER – створення тригера;
- НАЗВА ТРИГЕРА – ім'я, яке вибирається для тригера. Воно повинно бути унікальним в межах бази даних;
- {BEFORE | AFTER} – вказує, коли тригер буде виконуватись – до або після операції;
- {INSERT | UPDATE | DELETE} – операції коли тригер буде спрацьовувати. Можливі значення: INSERT (вставка нового рядка в таблицю), UPDATE (оновлення існуючого рядка) або DELETE (видалення рядка з таблиці);
- ON назва тригера – вказує на якій таблиці тригер буде активний;
- FOR EACH ROW – тригер буде виконуватись для кожного рядка, який змінюється операцією бази даних;
- BEGIN і END – початок і кінець тіла тригера, тобто всіх інструкції, які будуть виконуватись.

DELIMITER – це ключове слово, яке використовується для зміни роздільника, наприклад на &&, що дозволяє застосовувати крапку з комою в середині тіла тригера, не розглядаючи його як кінець команди.

Отже, щоб вдосконалити базу даних ветеринарної лікарні потрібно запровадити деякі тригери.

1. Тригер для автоматичного створення медичної картки в таблиці «MEDCARDS» після додавання нового клієнта в таблиці «PATIENTS».

Відповідний код тригера 1:

```
DROP TRIGGER IF EXISTS `animal_registry`.`create_medcard`;
DELIMITER $$
CREATE TRIGGER create_medcard
AFTER INSERT ON PATIENTS
FOR EACH ROW
BEGIN
INSERT INTO MEDCARDS (cardID) VALUES (NEW.patientID);
END$$
DELIMITER ;
```

Щоб перевірити роботу цього тригера додамо запис в таблицю «PATIENTS». Код додавання запису:

```
INSERT INTO `animal_registry`.`patients` (`patientID`, `patientName`, `species`, `breed`, `age`, `weight`, `gender`, `clientID`)
VALUES ('156126', 'Джек', 'Пес', 'Німецька вівчарка', '4', '21', 'Чоловіча', '210840');
```

Після цього в таблиці «MEDCARDS» теж з’являється новий запис, що продемонстровано на рисунку 3.21.

Рисунок 3.21 – Ефект спрацювання тригера 1

2. Тригер для автоматичного заповнення проміжної таблиці «RECORD_SERVICE» після додавання нового запису в таблицю «MEDRECORDS».

Відповідний код тригера 2:

```
DELIMITER $$
CREATE TRIGGER insert_record_service
AFTER INSERT ON MEDRECORDS
FOR EACH ROW
```

```
BEGIN
INSERT INTO RECORD_SERVICE (recordID, serviceID)
SELECT NEW.recordID, SERVICES.serviceID
FROM SERVICES
WHERE FIND_IN_SET(SERVICES.serviceName, NEW.prescription);
END $$
DELIMITER ;
```

Щоб перевірити роботу тригера додамо запис в таблицю «MEDRECORDS» Код додавання запису:

```
INSERT INTO `animal_registry`.`medrecords` (`recordID`, `recordDatetime`, `reason`, `prescription`, `patientID`, `vetID`)
VALUES ('1037', '2023–05–15 11:00:00', 'Позапланове щеплення та встановлення чипу', 'Профілактичні щеплення,Ідентифікація тварин(встановлення чипів)', '156126', '10830');
```

Таким чином, в таблиці ««RECORD_SERVICE»» з’являються нові записи, згідно з кількості наданих послуг під час візиту, що зображено на рисунку 3.22.

Рисунок 3.22 – Ефект спрацювання тригера 2

3. Тригер для автоматичного обрахунку суми наданих під час візиту послуг в таблиці «PAYMENTS».

Відповідний код тригера 3:

```
DELIMITER $$
CREATE TRIGGER calc_payment_amount
BEFORE INSERT ON PAYMENTS
FOR EACH ROW
BEGIN
DECLARE payment_id INT;
SET payment_id = NEW.paymentID;
```

```
SET NEW.paymentAmount = (SELECT SUM(SERVICES.servicePrice)
FROM SERVICES
JOIN RECORD_SERVICE ON SERVICES.serviceID = RECORD_SERVICE.serviceID
JOIN MEDRECORDS ON RECORD_SERVICE.recordID = MEDRECORDS.recordID WHERE MEDRECORDS.recordID = payment_id );
END$$
DELIMITER ;
```

Доречно буде перевірити працездатність даного тригера після додавання ще одного тригера до таблиці «PAYMENTS», який вказуватиме автоматичний час оплати.

4. Тригер для автоматичного встановлення часу оплати послуг в таблиці «PAYMENTS».

Відповідний код тригера 4:

```
DELIMITER $$
CREATE TRIGGER set_payment_time
BEFORE INSERT ON PAYMENTS
FOR EACH ROW
BEGIN
SET NEW.paymentDatetime = NOW();
END$$
DELIMITER ;
```

Для того щоб перевірити роботу тригерів 3 та 4 додамо новий запис в таблицю «PAYMENTS». Код додавання запису:

```
INSERT INTO `animal_registry`.`payments` (`paymentID`)
VALUES ('1037');
```

У результаті в таблиці «PAYMENTS» відбувається автоматичний обрахунок суми наданих послуг та автоматично встановлюється час їх надання, який відповідає часу додання нового запису в таблицю, ефект спрацювання зображено на рисунку 3.23.

Рисунок 3.23 – Ефект спрацювання тригерів 3 та 4

5. Тригер виводу повідомлення помилки при додаванні двох послуг з однією назвою в таблицю «SERVICES».

Відповідний код тригера 5:

```
DELIMITER $$
CREATE TRIGGER before_services_insert
BEFORE INSERT ON SERVICES
FOR EACH ROW
BEGIN
IF EXISTS (SELECT * FROM SERVICES WHERE serviceName = NEW.serviceName) THEN
 SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Послуга з цією назвою вже існує';
END IF;
END$$
DELIMITER ;
```

Щоб перевірити роботу тригера додамо в таблицю «SERVICES» нову послугу з вже існуючою там назвою. Код додавання запису:

```
INSERT INTO `animal_registry`.`services` (`serviceID`, `serviceName`, `unitofMeasure`, `servicePrice`)
VALUES ('200797', 'Ультразвукова діагностика вагітності', '1 дослідження', '111.23');
```

В результаті тригер не дозволяє нам здійснити вставку і з’являється повідомлення помилки, що зображено на рисунку 3.24.

Рисунок 3.24 – Ефект спрацювання тригера 5

6. Тригер виводу повідомлення помилки при внесені неповного номера телефону чи при внесені неправильного телефонного коду в таблиці «CLIENTS».

Відповідний код тригера 6:

```
DELIMITER $$
CREATE TRIGGER check_phone_numbers_clients
BEFORE INSERT ON CLIENTS
FOR EACH ROW
BEGIN
IF CHAR_LENGTH(NEW.clientPhone) < 13 THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Номер телефону містить менше 13 символів';
ELSEIF LEFT(NEW.clientPhone, 4) != '+380' THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Номер телефону починатися не з "+380"';
END IF;
END $$
DELIMITER ;
```

Перевіряємо роботу тригера внесенням неправильного запису. Код додавання запису:

```
INSERT INTO `animal_registry`.`clients` (`clientID`, `clientFname`, `clientSname`, `clientPname`, `clientAddress`, `clientPhone`, `clientEmail`)
VALUES ('210841', 'Лев', 'Єва', 'Янівна', 'Хмельницька область, Шепетівський район, смт Теофіполь, вул. Баюка,21', '+3068882132', 'lev.yeva@gmai.com');
```

Результат роботи тригера 6 зображується на рисунку 3.25.

Рисунок 3.25 – Ефект спрацювання тригера 6

Заповнимо номер до 13 символів, але при цьому ввівши неправильний код, наприклад «+368882132889», тоді виникає помилка, що зображена на рисунку 3.26.

Рисунок 3.26 – Ефект спрацювання тригера 6

Має сенс застосувати цей же тригер і до таблиці «VETS», оскільки вона також зберігає в собі поле з номерами телефонів.

Схожість

Джерела з Бібліотеки 137

1	Студентська робота	ID файлу: 1009724942	Навчальний заклад: National Technical University of Ukr	26 Джерело	0.42%
2	Студентська робота	ID файлу: 1003704716	Навчальний заклад: National University of Life and Envir	2 Джерело	0.36%
3	Студентська робота	ID файлу: 106406	Навчальний заклад: National University of Life and Environme	2 Джерело	0.32%
4	Студентська робота	ID файлу: 1014758121	Навчальний заклад: Lviv Polytechnic National University	29 Джерело	0.24%
5	Студентська робота	ID файлу: 1007956506	Навчальний заклад: National Technical University of Ukra	3 Джерело	0.2%
6	Студентська робота	ID файлу: 1008362411	Навчальний заклад: National Aviation University	11 Джерело	0.18%
7	Студентська робота	ID файлу: 106432	Навчальний заклад: National University of Life and Environme	2 Джерело	0.13%
8	Студентська робота	ID файлу: 1009496565	Навчальний заклад: National Technical University of Ukraine "Kyi...		0.12%
9	Студентська робота	ID файлу: 1011382784	Навчальний заклад: Interregional Academy of Personnel	30 Джерело	0.12%
10	Студентська робота	ID файлу: 1015062723	Навчальний заклад: Lviv Polytechnic National University		0.12%
11	Студентська робота	ID файлу: 1015270340	Навчальний заклад: Lviv Polytechnic National University		0.12%
12	Студентська робота	ID файлу: 1006954436	Навчальний заклад: National University of Life and Environmenta...		0.11%
13	Студентська робота	ID файлу: 1014802962	Навчальний заклад: Taras Shevchenko National University of Kyiv		0.11%
14	Студентська робота	ID файлу: 1015272890	Навчальний заклад: National Technical University of Ukra	2 Джерело	0.11%
15	Студентська робота	ID файлу: 1000068004	Навчальний заклад: National Technical University of Ukra	6 Джерело	0.1%
16	Студентська робота	ID файлу: 1015168753	Навчальний заклад: National Technical University of Ukraine "Kyi...		0.1%
17	Студентська робота	ID файлу: 1011477056	Навчальний заклад: National Technical University of Ukra	3 Джерело	0.1%
18	Студентська робота	ID файлу: 1011376384	Навчальний заклад: Lviv Polytechnic National University		0.1%
19	Студентська робота	ID файлу: 1000788359	Навчальний заклад: National Technical University of Ukr	11 Джерело	0.1%
20	Студентська робота	ID файлу: 1005686199	Навчальний заклад: National Technical University of Ukraine "Kyi...		0.1%

21	**Студентська робота**	ID **файлу**: 1015251438	**Навчальний заклад**: National Technical University of Ukraine “Kyi...	0.1%
22	**Студентська робота**	ID **файлу**: 1014720070	**Навчальний заклад**: Lviv Polytechnic National University	0.1%