НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

КУРСОВА РОБОТА

з дисципліни «Проєктування та використання баз даних»
(назва дисципліни)

на тему: «База даних для нотаріальної контори»

Студента групи ТР-25
зі спеціальності 122 «Комп'ютерні науки»
Кудрявцев Є. О.
(прізвище та ініціали)
Керівник доцент, Сегеда І. В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: _____ Оцінка: _____

Члени комісії

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)
_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)
_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ- 2024 рік

Національний технічний університет України
“ Київський політехнічний інститут ім. Ігоря Сікорського”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ
Рівень вищої освіти: бакалавр
За освітньою програмою «Цифрові технології в енергетиці»

Спеціальності 122 Комп’ютерні науки

ЗАВДАННЯ
НА КУРСОВУ РОБОТУ СТУДЕНТУ

Кудрявцев Єгор Олегович
(прізвище, ім’я, по батькові)

1.Тема роботи «База даних для нотаріальної контори»
керівник курсової роботи - Сегеда Ірина Василівна, доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
2. Строк подання студентом роботи 23 грудня 2024 року
3. Вихідні дані до проєкту (роботи): середовище розробки – MySQL Workbench
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) – Створення бази даних для нотаріальної контори

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1.	Отримання завдання до КР	1-3-й тижні	
2.	Вибір та дослідження скриптів і бази даних	4-6-й тижні	
3.	Реалізація основної структури	6-10-й тижні	
4.	Покращення та надання обмежень базі даних	11-и тиждень	
5.	Тестування успішно розробленої бази даних	11-й тиждень	
6.	Оформлення пояснювальної записки	12-й тиждень	
7.	Друк курсової роботи	13-й тиждень	

Студент _____ **Кудрявцев Єгор**
(підпис) (ім’я ПРИЗВИЩЕ)

Керівник курсової роботи _____ **Ірина Сегеда**
(підпис) (ім’я ПРИЗВИЩЕ)

АНОТАЦІЯ

Дана курсова робота представляє дослідження мети задачі, створення ефективної та надійної бази даних нотаріальної контори.

Результатом роботи є створена нова база даних із урахуванням усіх методів покращення ефективності, безпеки та надійності.

Виконана робота продемонструвала процес створення та роботи з базою даних, застосування принципів нормалізації для уникнення аномалій та забезпечення ефективного зберігання й обробки інформації, що є важливим для проектування масштабованих та надійних інформаційних систем.

Також здійснено виконання вибірок даних із застосуванням різноманітних SQL-конструкцій, таких як GROUP BY, HAVING, об'єднання таблиць та агрегація даних. Окрім цього, були розглянуті складні запити, що включають використання агрегатних функцій та умовних конструкцій CASE і UNION, що дозволяє ефективно обробляти дані в базі.

Для розробки бази даних було використано СКБД MySQL Workbench і Visual Paradigm для створення діаграм, відповідно до задачі.

Курсова робота складається зі вступу, 3х розділів, висновків, списку використаних джерел і Додатку А, Б, В. Загальна кількість сторінок 39 (без урахування додатків).

Ключові слова: MySQL Workbench, Visual Paradigm, нормалізація даних, база даних, СКБД.

ABSTRACT

This coursework presents a study of the purpose of the task, the creation of an effective and reliable database of a notary office.

The result of the work is a new database created taking into account all methods of improving efficiency, security and reliability.

The work performed demonstrated the process of creating and working with a database, applying the principles of normalization to avoid anomalies and ensuring effective storage and processing of information, which is important for the design of scalable and reliable information systems.

Data sampling was also performed using various SQL constructs, such as GROUP BY, HAVING, table joining and data aggregation. In addition, complex queries were considered, including the use of aggregate functions and conditional constructs CASE and UNION, which allows for effective processing of data in the database.

To develop the database, the MySQL Workbench and Visual Paradigm DBMS were used to create diagrams, in accordance with the task.

The coursework consists of an introduction, 3 sections, a conclusion, a list of sources used and Appendix A, B, C. Total number of pages 39 (excluding appendices).

Keywords: MySQL Workbench, Data Base, Visual Paradigm, data normalisation, data base, СКБД.

ЗМІСТ

ВСТУП

Нотаріальні контори є важливими юридичними установами, які щодня обробляють значні обсяги даних. Ця інформація включає дані про клієнтів, угоди, а також різноманітну документацію, яка потребує точності та надійності в обробці. У сучасному цифровому світі використання баз даних стає важливою умовою для оптимізації роботи нотаріальних контор. Застосування сучасних інформаційних технологій дозволяє спростити управління інформацією, забезпечити дистанційний доступ до даних, а також покращити загальну ефективність організації.

Метою даного проєкту є створення бази даних, яка відповідатиме сучасним вимогам нотаріальних контор. Такий підхід дозволяє спростити доступ до інформації для працівників контори, автоматизувати виконання частини рутинних завдань і забезпечити швидке та коректне оновлення даних.

Завдання дослідження цього проєкту включають розробку ефективної та безпечної бази даних. Для досягнення цих задач застосовуються інструменти, що сприяє зменшенню кількості помилок і підвищенню надійності роботи системи.

Об'єктом дослідження є нотаріальна контора, яка надає юридичні послуги та обслуговує різні категорії клієнтів. Предметом дослідження виступає розробка моделі бази даних для ефективного управління інформацією.

Практичне значення роботи полягає у підвищенні ефективності діяльності нотаріальної контори через оптимізацію обробки даних і автоматизацію рутинних процесів. Використання запропонованої системи дозволяє зменшити навантаження на працівників, підвищити якість обслуговування клієнтів, а також забезпечити більш швидке виконання операцій. Таким чином, результати даного проєкту можуть бути корисними як для конкретної нотаріальної контори, так і для інших організацій, які прагнуть впровадити сучасні технології для покращення своєї роботи.

Курсова робота складається зі вступу, трьох розділів, висновків і додатку.

РОЗДІЛ 1 АНАЛІЗ ВИМОГ

1.1 Постановка задачі

У сучасних умовах ефективне управління інформацією є однією з ключових складових успішної діяльності нотаріальних контор. Щоденна робота цих організацій включає обробку значних обсягів даних, таких як відомості про клієнтів, укладені угоди, надані послуги, а також інформацію про знижки й комісійні. Організація такої інформації потребує створення структурованої, масштабованої та захищеної бази даних, яка забезпечить зручний доступ до даних, автоматизацію рутинних процесів та можливість ефективного управління.

Розроблена база даних має включати декілька основних сутностей, пов’язаних між собою відповідно до логіки предметної області [1]. У системі можна виділити такі основні таблиці:

- Services – містить інформацію про всі доступні послуги нотаріальної контори, їх описи та вартість.
- Agreements – відображає укладені угоди, включаючи дату й час, загальну вартість, комісію, а також опис угоди. Усі відповідні зовнішні ключі можна встановлювати тут.
- Clients – включає дані про клієнтів, такі як імена, адреси, телефони та інформацію про зайнятість. Ця таблиця має бути розділена на декілька допоміжних таблиць для більш гнучкого управління.
- Service_discounts і Activity_discounts – зберігають інформацію про знижки, які застосовуються відповідно до категорій послуг або специфіки діяльності.

Головною метою проєкту є створення бази даних, яка відповідатиме сучасним вимогам інформаційної безпеки та дозволить автоматизувати обробку даних у нотаріальній конторі. База даних має бути масштабованою, легко модифікованою та забезпечувати високу продуктивність роботи при великій кількості записів [2].

Для досягнення мети було поставлено такі завдання

1. Розробити структуру бази даних із урахуванням нормалізації даних для уникнення дублювання інформації.
2. Забезпечити логічну цілісність бази даних за допомогою встановлення зв’язків між таблицями за допомогою зовнішніх ключів.
3. Проаналізувати створену дану базу даних і застосувати різні типи вибірок до неї.

Розроблена система стане основою для подальшого вдосконалення процесів управління даними в нотаріальній конторі та впровадження додаткових модулів, таких як інтеграція з зовнішніми інформаційними ресурсами чи застосунками [3].

1.2 Діаграма прецендентів

Діаграма прецедентів є ключовим елементом у процесі проєктування інформаційних систем, оскільки вона описує основні функції, які система повинна виконувати, а також відображає взаємодію користувачів із системою [4]. У контексті нашої бази даних для автоматизації процесів нотаріальної контори ця діаграма дозволяє структуровано представити основні дії, які виконують користувачі системи [5]. У відповідності з постановкою задачі було розроблено діаграму прецендентів (Рисунок 1.1).

На діаграмі система представлена у вигляді прямокутника із заголовком «Нотаріальна контора». У межах цього прямокутника знаходяться всі прецеденти, які описують функціональні можливості системи. Актори представляють користувачів або зовнішні компоненти, які взаємодіють із системою. Серед них: клієнт, який є фізичною або юридичною особою, що звертається до нотаріуса за наданням послуг; адміністратор, що відповідає за підтримку функціонування системи, управління базою даних та налаштування системи; система, яка є автоматизованою частиною, що забезпечує виконання певних функцій, таких як розрахунок цін та формування документів.

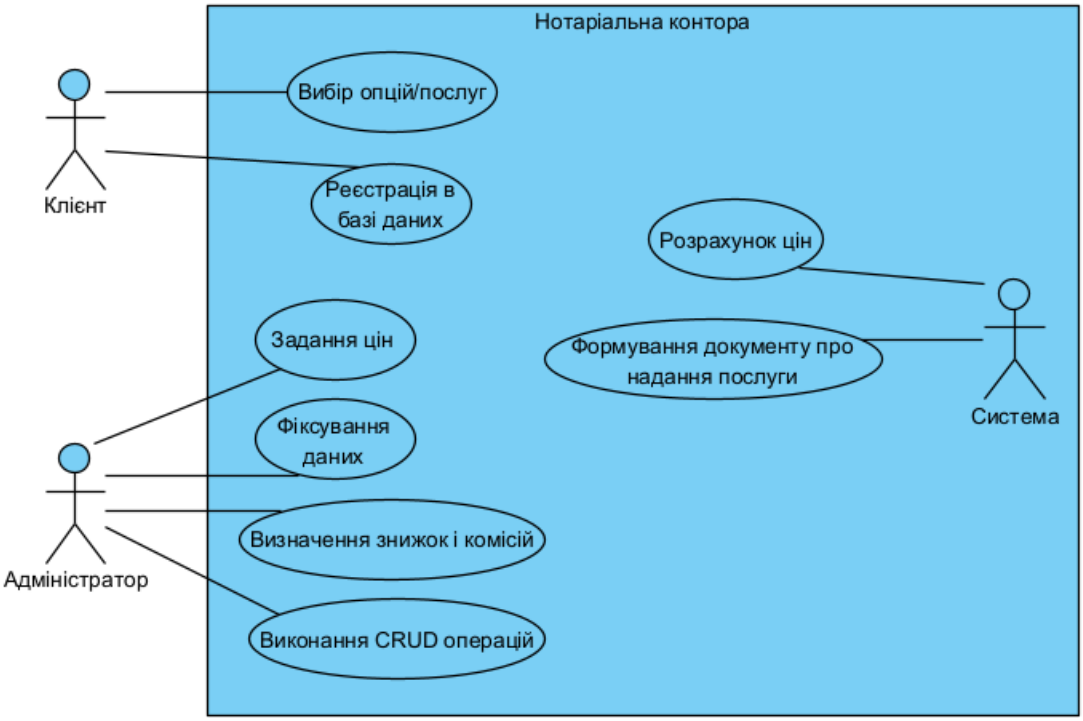


Рисунок 1.1 – Діаграма прецендентів

Прецеденти – це функції, які виконуються системою для забезпечення потреб користувачів. Діаграма містить такі ключові прецеденти: вибір опцій або послуг, реєстрація в базі даних, розрахунок ціни, формування документа про надання послуги, задання цін, фіксування даних, визначення знижок і комісій, виконання CRUD операцій [6]. Ці прецеденти охоплюють усі важливі аспекти функціонування системи, забезпечуючи її ефективність і зручність для користувачів.

Клієнт має можливість вибирати опції чи послуги із запропонованого списку, наприклад, оформлення договору, підтвердження довіреності чи легалізації документів. Вибрані послуги надалі використовуються для розрахунку вартості угоди. Після вибору послуг дані клієнта, такі як ім’я, контактна інформація та інші атрибути, фіксуються в базі даних, що дозволяє забезпечити зручний доступ до інформації для подальшого використання в процесі оформлення угод.

Адміністратор визначає базову вартість послуг, яка зберігається у таблиці послуг. Це забезпечує можливість встановлення коректних тарифів на кожен тип послуги. Адміністратор також вводить основну інформацію до системи, зокрема про

клієнтів, угоди та послуги. Правильне фіксування даних гарантує безперебійну роботу системи. Він також налаштовує правила надання знижок для окремих клієнтів або угод, а також визначає комісійні відрахування. Наприклад, знижки можуть надаватися на певні послуги при виконанні певних умов, таких як оформлення кількох угод одночасно. Важливою частиною роботи адміністратора є виконання стандартних операцій із базою даних, таких як створення нових записів, читання даних, оновлення існуючої інформації та видалення застарілих даних. Це забезпечує підтримку актуальності та цілісності даних у системі.

Система автоматично розраховує вартість обраних клієнтом послуг. У процесі розрахунку враховуються базова ціна послуги, можливі знижки, встановлені адміністратором, та комісії чи додаткові витрати. Після розрахунку ціни та вибору послуг система генерує офіційний документ, що містить деталі угоди, включаючи вартість, перелік послуг та інформацію про клієнта.

Діаграма прецедентів показує взаємозалежність між функціями системи. Прецеденти «Вибір опцій/послуг» та «Реєстрація в базі даних» є підґрунтям для виконання більш складних функцій, таких як розрахунок ціни та формування документа. Задання цін безпосередньо впливає на функціонування прецеденту «Розрахунок ціни», оскільки ціна послуг є базовим параметром у розрахунках. Прецеденти адміністратора, такі як «Визначення знижок і комісій», забезпечують налаштування правил, які використовуються системою під час розрахунку цін.

Діаграма прецедентів для бази даних нотаріальної контори дозволяє всебічно оцінити функціональні можливості системи, ролі її користувачів і їхню взаємодію з функціоналом. Вона є основою для подальшого етапу розробки, зокрема створення діаграм, що деталізують реалізацію описаних функцій [7].

1.3 Нормалізація даних

Важливим етапом у розробці бази даних є урахування нормалізації даних. Нормалізація даних – це процес організації та структуризації даних у базі даних таким

чином, щоб забезпечити мінімізацію надмірності та уникнути можливих аномалій при вставці, оновленні або видаленні даних [8]. Основною метою нормалізації є створення такого дизайну бази даних, який буде легким для підтримки, масштабування і, одночасно, забезпечить цілісність даних [9].

У процесі нормалізації застосовуються певні правила, які визначають так звані нормальні форми. Кожна нормальна форма надає певний рівень структурування і допомагає вирішити різні типи проблем, що виникають при роботі з даними. Існує кілька нормальних форм, від першої до п'ятої, кожна з яких має свої особливості та вимоги [10].

Перша нормальна форма вимагає, щоб усі стовпці таблиці містили атомарні значення, тобто кожне поле повинно зберігати тільки одне значення, а не множину значень або список. Таблиця також повинна мати унікальний ідентифікатор для кожного рядка, що зазвичай є первинним ключем. Це забезпечує однозначність записів та запобігає дублюванню даних.

Для досягнення 1НФ треба розділити такі множинні значення на окремі рядки або створити додаткові таблиці.

Друга нормальна форма передбачає, що таблиця повинна бути в 1НФ, а також всі неключові атрибути повинні повністю залежати від первинного ключа. Іншими словами, якщо таблиця має складний (композитний) ключ, то всі атрибути, які не є частиною цього ключа, повинні залежати від всього ключа, а не лише від частини його. Це називається відсутністю часткових залежностей [11].

Щоб привести таблицю до 2НФ, потрібно створити окрему таблицю для продуктів, де зберігаються тільки атрибути, пов'язані з продуктами.

Третя нормальна форма вимагає, щоб таблиця була в 2НФ, крім того, щоб не було транзитивних залежностей між атрибутами. Це означає, що жоден неключовий атрибут не повинен залежати від іншого неключового атрибута. Іншими словами, всі атрибути, що не є частиною первинного ключа, повинні залежати безпосередньо від ключа, а не від інших атрибутів.

Четверта нормальна форма вимагає, щоб таблиця була в 3НФ і не мала багатозначних залежностей. Багатозначна залежність виникає, коли один атрибут залежить від іншого, але ці залежності не пов'язані з основним ключем.

Загалом нормалізація має велике значення для побудови баз даних, оскільки вона допомагає уникати надмірності та забезпечує цілісність даних. Проте нормалізація також може призвести до деяких проблем, таких як складність запитів, необхідність виконання додаткових з'єднань (JOIN) між таблицями, що може вплинути на продуктивність. Тому в реальних проєктах часто застосовують де-нормалізацію, яка знижує рівень нормалізації, щоб покращити продуктивність[12].

Нормалізація є важливим інструментом у створенні ефективних і зручних баз даних, але її необхідно застосовувати з розумом, враховуючи вимоги конкретної системи.

РОЗДІЛ 2 РОЗРОБКА БАЗИ ДАНИХ

2.1 Створення таблиць спроєктованої бази даних

Створення таблиць спроєктованої бази даних є ключовим етапом реалізації бази даних, що забезпечує її функціональність та ефективність. Цей процес передбачає реалізацію фізичної структури бази даних на основі розробленої EER-діаграми, яка детально описує зв'язки між сутностями, їхні атрибути, а також типи відносин. З огляду на діаграму, раніше представлену для нотаріальної контори, ми можемо перейти до визначення основних таблиць, які забезпечать організацію даних для обробки запитів і виконання бізнес-логіки системи. Для відповідної бази даних було сконструйовано EER-діаграму (Рисуснок 2.1).

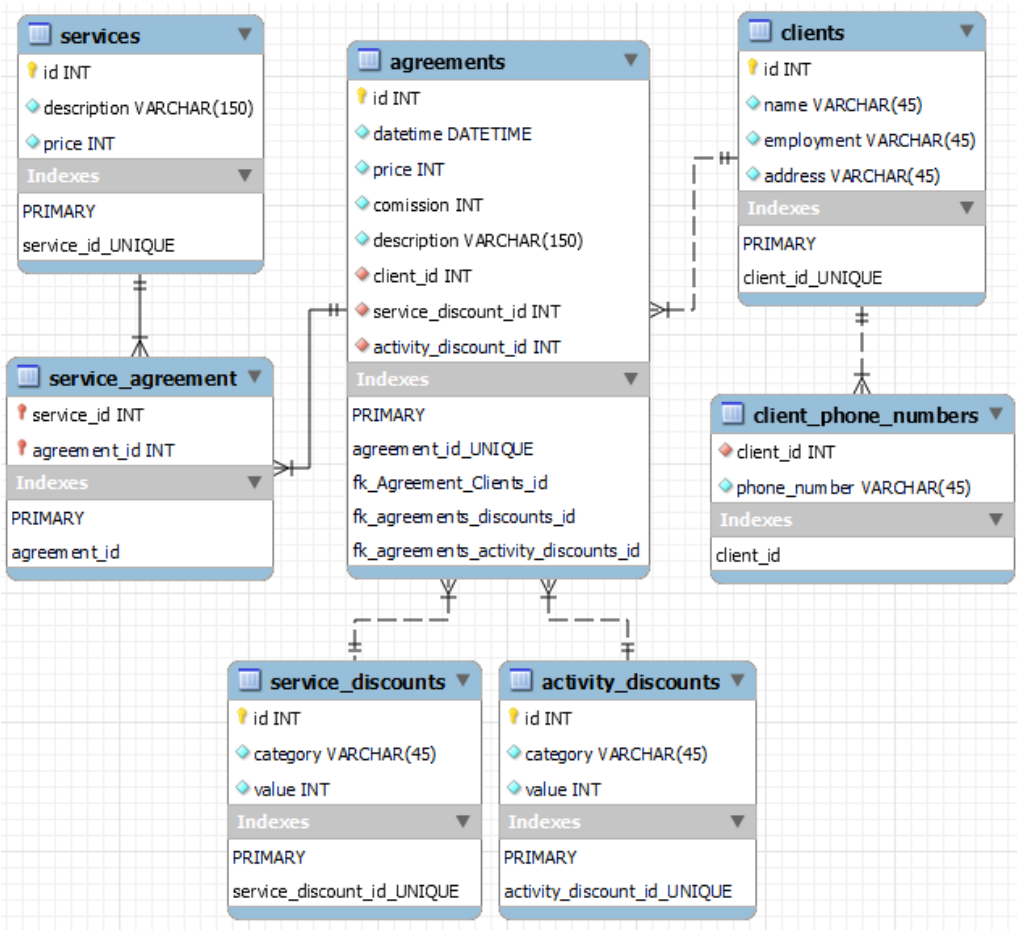


Рисунок 2.1 – EER-діаграма бази даних

EER-діаграма відображає кілька ключових сутностей, таких як Clients (Клієнти), Services (Послуги), Agreements (Угоди), Discounts (Знижки), а також їхні атрибути й зв'язки. Кожна сутність проєктується в таблицю, яка включає всі необхідні поля для зберігання інформації. Взаємозв'язки між сутностями представлені у вигляді зовнішніх ключів, що дозволяють підтримувати референційну цілісність бази даних.

Таблиця clients відповідає за зберігання основної інформації про клієнтів нотаріальної контори (Рисунок 2.2).

```
CREATE TABLE clients (
    id INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(45) NOT NULL,
    address VARCHAR(45) NOT NULL,
    employment VARCHAR(45) NOT NULL,
    PRIMARY KEY (id))
ENGINE = InnoDB;

-- Додавання додаткових атомарних значень для клієнтів телефони
CREATE TABLE client_phone_numbers (
    client_id INT NOT NULL,
    phone_number VARCHAR(45) NOT NULL,
    FOREIGN KEY (client_id) REFERENCES clients(id)
) ENGINE = InnoDB;
```

Рисунок 2.2 – Скрипт створення таблиці clients і client_phone_numbers

Вона включає унікальний ідентифікатор клієнта (client_id), а також інші поля: ім'я, прізвище, зайнятість, дату народження, адресу, контактний номер телефону та електронну пошту. Важливим рішенням було виділення контактної інформації клієнта (телефони) в окремі таблиці, щоб забезпечити можливість зберігання множинних значень. Це дозволяє системі зберігати більше даних для одного клієнта.

Для адрес клієнтів використовується окрема таблиця client_phone_numbers для зайнятостей окремого клієнта. Вона має зовнішній ключ client_id, що пов’язує її з таблицею clients. Це дозволяє клієнтам зберігати декілька телефонних номерів.

Таблиця services зберігає інформацію про всі послуги, які пропонує нотаріальна контора (Рисунок 3.3). Вона включає ідентифікатор послуги (service_id), назву послуги та її базову вартість.

```
CREATE TABLE services (
    id INT NOT NULL AUTO_INCREMENT,
    description VARCHAR(150) NOT NULL,
    price INT NOT NULL CHECK(price > 0),
    PRIMARY KEY (id),
    UNIQUE INDEX service_id_UNIQUE (id ASC) VISIBLE)
ENGINE = InnoDB;
```

Рисунок 2.3 – Скрипт створення таблиці services

Таблиця agreements відповідає за зберігання даних про всі угоди, що здійснюються в нотаріальній конторі. Вона включає ідентифікатор угоди, клієнта, працівника, послугу, дату угоди та остаточну вартість із урахуванням знижок (Рисунок 2.4).

```
CREATE TABLE agreements (
    id INT NOT NULL AUTO_INCREMENT,
    datetime DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    price INT NOT NULL CHECK(price > 0),
    comission INT NOT NULL,
    description VARCHAR(150) NOT NULL,

    -- Ідентифікатори про інші таблиці
    client_id INT NOT NULL,
    service_discount_id INT NOT NULL,
    activity_discount_id INT NOT NULL,

    PRIMARY KEY (id))
ENGINE = InnoDB;
```

Рисунок 2.4 – Скрипт створення таблиці agreements

Таблиця service_discounts й activity_discounts зберігає відповідні дані про знижки для сервісів та діяльностей (Рисунок 2.5).

```
CREATE TABLE service_discounts (
    id INT NOT NULL AUTO_INCREMENT,
    category_id INT NOT NULL,
    value INT NOT NULL CHECK(value >= 0),
    PRIMARY KEY (id),
    UNIQUE INDEX discount_id_UNIQUE (id ASC) VISIBLE)
ENGINE = InnoDB;


CREATE TABLE activity_discounts (
      id INT NOT NULL AUTO_INCREMENT,
      category VARCHAR(45) NOT NULL,
      value INT NOT NULL CHECK(value >=0),
      PRIMARY KEY (id),
      UNIQUE INDEX discount_id_UNIQUE (id ASC) VISIBLE)
ENGINE = InnoDB;
```

Рисунок 2.5 – Скрипт створення таблиці

Всі таблиці в проєктованій базі даних взаємопов’язані через зовнішні ключі, які відображають зв’язки, показані на EER-діаграмі. Наприклад, таблиця agreements має зв’язок із таблицями clients, services, activity_discounts та service_discounts, що дозволяє однозначно ідентифікувати клієнта, послугу та працівника, пов’язаних із кожною транзакцією.

При створенні бази даних також було враховано масштабованість і ефективність зберігання даних. Наприклад, використання окремих таблиць для адрес та телефонів дозволяє уникнути дублювання даних та забезпечити нормалізацію бази. Застосування зовнішніх ключів гарантує цілісність даних, а індекси забезпечують швидкий доступ до часто використовуваних записів.

Цей підхід до створення таблиць бази даних дозволяє забезпечити її цілісність і відповідність вимогам бізнес-логіки системи нотаріальної контори. Кожна таблиця

виконує чітко визначену функцію, що сприяє зручному адмініструванню й обробці даних.

2.2 Надання обмежень, індексів і ключів

Надавання ключів, зв’язків і обмежень у базі даних є критично важливими для забезпечення цілісності даних, уніфікації структури та забезпечення правильності функціонування системи. У попередньо розглянутих скриптах реалізовано кілька аспектів, які забезпечують коректне визначення ключів, встановлення зв’язків між таблицями та впровадження обмежень. Далі детально розглянемо всі аспекти, враховуючи структуру кожної таблиці, використання ключів, зв’язків та різних типів обмежень.

```
ALTER TABLE agreements ADD CONSTRAINT fk_Agreement_Clients
FOREIGN KEY (client_id) REFERENCES clients (id)
ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE agreements ADD CONSTRAINT fk_agreements_service_discounts
FOREIGN KEY (service_discount_id) REFERENCES service_discounts (id)
ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE agreements ADD CONSTRAINT fk_agreements_activity_discounts
FOREIGN KEY (activity_discount_id) REFERENCES activity_discounts (id)
ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE service_agreement ADD CONSTRAINT
FOREIGN KEY (service_id) REFERENCES services (id)
ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE service_agreement ADD CONSTRAINT
FOREIGN KEY (agreement_id) REFERENCES agreements (id)
ON DELETE NO ACTION ON UPDATE NO ACTION;
```

Рисунок 2.7 – Створення зовнішніх ключів

Для покращення продуктивності запитів додано індекси. Для полів, що зберігають числові значення, додано обмеження CHECK, яке забезпечує допустимі значення. Це дозволяє базі даних швидше знаходити необхідні записи. Первинні ключі є основою для ідентифікації унікальних записів у кожній таблиці. У кожній таблиці, де це необхідно, визначено PRIMARY KEY, що гарантує унікальність кожного рядка.

Зовнішні ключі встановлюють зв'язки між таблицями, дозволяючи створити ієрархічну структуру даних. У базі даних нотаріальної контори використовується кілька зовнішніх ключів, які гарантують цілісність даних та їх взаємозв'язок.

Також, зовнішні ключі мають важливі параметри, такі як ON DELETE і ON UPDATE, які визначають, що має статися при видаленні або оновленні запису, на який посилається зовнішній ключ.

Створення індексів у базах даних є важливим етапом оптимізації, оскільки індекси значно покращують швидкість доступу до даних. Індекс – це структура даних, яка дозволяє здійснювати швидкий пошук, сортування та фільтрацію записів у таблицях бази даних. У базі даних нотаріальної контори було надано всі необхідні індекси (Рисунок 2.8).

```
ALTER TABLE services ADD UNIQUE INDEX service_id_UNIQUE (id ASC) VISIBLE;
ALTER TABLE service_discounts ADD UNIQUE INDEX service_discount_id_UNIQUE (id ASC) VISIBLE;
ALTER TABLE clients ADD UNIQUE INDEX client_id_UNIQUE (id ASC) VISIBLE;
ALTER TABLE activity_discounts ADD UNIQUE INDEX activity_discount_id_UNIQUE (id ASC) VISIBLE;
ALTER TABLE agreements ADD UNIQUE INDEX agreement_id_UNIQUE (id ASC) VISIBLE;

ALTER TABLE agreements ADD INDEX fk_Agreement_Clients_id (client_id ASC) VISIBLE;
ALTER TABLE agreements ADD INDEX fk_agreements_discounts_id (service_discount_id ASC) VISIBLE;
ALTER TABLE agreements ADD INDEX fk_agreements_activity_discounts_id (activity_discount_id ASC) VISIBLE;
```

Рисунок 2.8 – Додавання індексів таблицям

Індекси дозволяють значно прискорити процес пошуку записів у великій таблиці. Без індексів система повинна буде здійснювати повний перебір усіх рядків таблиці, що називається повним скануванням таблиці. Якщо ж для стовпця створений

індекс, то база даних може використовувати його для значно швидшого доступу до потрібних рядків, оскільки індекс дозволяє швидко знаходити відповідні записи без необхідності переглядати всі дані.

Індекс може значно покращити швидкість сортування даних у запитах, які використовують оператор ORDER BY. Якщо індекс вже відсортований відповідно до певного стовпця, то база даних може просто повернути відсортовані дані, що значно пришвидшує виконання запиту.

Індекси забезпечують унікальність даних у таблиці. Індекс може бути створений для стовпця з унікальними значеннями, що забезпечить обмеження на дублювання значень (аналогічно до обмеження UNIQUE). Це дозволяє уникнути введення повторюваних даних у таблицю.

Структура бази даних спроєктована таким чином, щоб забезпечити її масштабованість, надійність і підтримку складних бізнес-логік. Використання зовнішніх ключів дозволяє підтримувати зв’язки між таблицями, а первинні ключі гарантують унікальність записів. Завдяки індексам оптимізується продуктивність запитів, особливо при роботі з великими обсягами даних. Ці заходи разом забезпечують правильну роботу бази даних, її гнучкість і відповідність вимогам системи.

2.3 Перевірка нормалізації

Нормалізація є важливою частиною проєктування бази даних, оскільки дозволяє організувати дані таким чином, щоб звести до мінімуму повторення та уникнути аномалій при оновленні, вставці чи видаленні записів. Один з основних принципів нормалізації – це поділ великих, складних таблиць на менші, чітко структуровані таблиці, що зменшує дублювання даних і сприяє кращій організації інформації. У цьому контексті, наявність правильно нормалізованої бази даних є запорукою її ефективності та коректності в довгостроковій перспективі.

У даному випадку маємо базу даних, що складається з кількох таблиць, кожна з яких зберігає певну категорію інформації, яка взаємодіє з іншими таблицями через зовнішні ключі. Усі ці таблиці взаємодіють одна з одною через зовнішні ключі, що забезпечує логічну структуру бази даних та дає можливість здійснювати зв'язки між записами різних таблиць.

Перша нормальна форма передбачає, що кожен стовпець таблиці містить лише атомарні значення, тобто кожне поле має містити тільки одне значення для кожного рядка. У нашій базі даних усі таблиці відповідають цій нормальній формі. Наприклад, в таблиці client_phone_numbers кожен номер телефону зберігається окремо для кожного клієнта, що гарантує відсутність повторів і забезпечує атомарність значень.

Отже, дана база даних відповідає 1 нормальній формі. Усі колонки містять лише атомарні значення.

Друга нормальна форма вимагає, щоб таблиці були у 1НФ, і додатково, щоб жоден ненульовий атрибут не залежав лише від частини складеного первинного ключа. Це означає, що всі атрибути, які не є частиною ключа, повинні залежати від всього первинного ключа, а не лише від його частини. У нашій базі даних таблиця service_agreement, наприклад, має складений ключ з двох стовпців: service_id та agreement_id. І кожен з них є невід'ємною частиною ключа, що запобігає частковим залежностям.

Таким чином дана база даних також відповідає 2НФ.

Третя нормальна форма вимагає, щоб таблиця була у 2НФ і при цьому всі атрибути, що не є частиною ключа, не залежали від інших атрибутів, які не є ключами (відсутність транзитивних залежностей). Іншими словами, жодна колонка, яка не є частиною первинного ключа, не повинна бути залежною від іншої колонки, яка також не є ключем.

У таблиці agreements є зовнішні ключі, які зв'язують її з іншими таблицями, такими як client_id, activity_discount_id, service_discount_id. Ці зв'язки є правильними, оскільки стовпці в таблиці agreements залежать лише від зовнішніх ключів і не мають транзитивних залежностей. Таким чином, таблиця agreements відповідає вимогам

3НФ. У таблицях clients, service_discounts, activity_discounts також немає транзитивних залежностей, і всі атрибути залежать тільки від первинних ключів, що робить їх відповідними 3НФ.

Отже, база даних добре організована і відповідає вимогам третьої нормальної форми. Всі таблиці правильно структуровані, і між ними встановлені коректні зв'язки за допомогою зовнішніх ключів. Завдяки цьому, дані в базі будуть зберігатися ефективно, і буде знижено ризик аномалій при вставці, оновленні чи видаленні записів. Таке проєктування дозволяє легко масштабувати базу даних у разі збільшення кількості даних, а також забезпечує її зручність та безпеку при роботі з великими обсягами інформації.

РОЗДІЛ 3 РОБОТА БАЗИ ДАНИХ

3.1 Вибірка стовбців

У результаті створення відповідної робочої бази даних і її наповнення необхідно перевірити її працездатність і переглянути результати декількох вибірок. Вибірка стовпців – це процес отримання певних даних із таблиці або кількох таблиць бази даних за допомогою запиту SELECT. SQL дозволяє вибирати як окремі стовпці, так і всі стовпці, а також фільтрувати, сортувати, групувати та застосовувати різні умови до даних.

Почнемо з найпростіших вибірки всіх і окремих полів(Рисунок 3.1).

```
SELECT name, employment FROM clients;
SELECT * FROM clients;
```

Рисунок 3.1 – Скрипти повної і часткової вибірки

У результаті отримуємо вивід відповідних даних таблиці (Рисунок 3.2).

name	employment
Hank Schrader	DEA Agent
Skyler White	Accountant
Marie Schrader	Radiologic Technologist
Saul Goodman	Lawyer
Mike Ehrmantraut	Private Investigator
Kim Wexler	Lawyer

id	name	address	employment
1	Hank Schrader	Albuquerque	DEA Agent
2	Skyler White	Albuquerque	Accountant
3	Marie Schrader	Albuquerque	Radiologic Technologist
4	Saul Goodman	Albuquerque	Lawyer
5	Mike Ehrmantraut	Albuquerque	Private Investigator
6	Kim Wexler	Albuquerque	Lawyer

Рисунок 3.2 – Результат використання скриптів вибірок

Таким чином вибірка конкретних стовпців прискорює виконання запитів, оскільки база даних не завантажує непотрібну інформацію. Вона використовується для оптимізації роботи з даними, коли потрібно відобразити лише окрему інформацію. У свою чергу вибірка всіх стовпців використовується, коли потрібно отримати всі дані з таблиці. Це зручно для початкового аналізу структури таблиці або для отримання повної картини даних. Для цього застосовується символ *, який означає вибір усіх стовпців.

3.2 Використання сортувань і вибірок даних із GROUP BY, HAVING

Одна з найкорисніших можливостей SQL-запитів є сортування. Можна сортувати дані за кількома стовпцями. Наприклад, спочатку за іменем клієнта в алфавітному порядку, а потім за зайнятістю в порядку спадання (Рисунок 3.3).

```
SELECT name, employment FROM clients
ORDER BY name ASC, employment DESC;
```

Рисунок 3.3 – Скрипт сортування за кількома полями

У результаті виконання даного скрипта можна отримати відповідну відсортовану таблицю (Рисунок 3.4).

name	employment
Bogdan Wolynetz	Car Wash Owner
Chuck McGill	Lawyer
Gale Boetticher	Chemist
Hank Schrader	DEA Agent
Howard Hamlin	Lawyer
Huell Babineaux	Bodyguard
Jane Margolis	Artist
Kim Wexler	Lawyer
Lydia Rodarte-Quayle	Executive

Рисунок 3.4 – Результат виконання скрипта сортування

Також гарним прикладом може бути сортування за місяцями й кількістю угод (Рисунок 3.5).

```
SELECT MONTH(datetime), COUNT(*) AS agreements_count FROM agreements
GROUP BY MONTH(datetime) ORDER BY agreements_count DESC, MONTH(datetime) ASC;
```

Рисунок 3.5 – Скрипт сортування за місяцями й кількістю угод

У результаті даний скрипт показує відповідну кількість угод у конкретні місяці та сортує за спаданням (Рисунок 3.6).

MONTH(datetime)	agreements_count
4	13
3	7

Рисунок 3.6 – Результат роботи скрипта з сортуванням

Також існує вибірка даних з використанням HAVING. Цей підхід використовується для групування даних і застосування умов до груп. Прикладом є скрипт для знаходження міст, кількість клієнтів із яких перевищує 2 (Рисунок 3.7).

```
SELECT address, COUNT(*) AS client_count FROM clients
GROUP BY address HAVING COUNT(*) > 2;
```

Рисунок 3.7 – Скрипт знаходження міст у яких клієнтів більше ніж 2

Відповідно отримуємо вивід даного скрипта з двома колонками: адреса та кількість клієнтів (Рисунок 3.8).

address	client_count
Albuquerque	16
Mexico	3

Рисунок 3.8 – Результат виконання скрипта

Загалом, сортування за кількома полями дозволяє впорядковувати результати запиту по двох або більше критеріях. Це використовується, коли потрібно організувати дані в певній логічній послідовності, наприклад, спочатку за пріоритетним полем, а потім за допоміжним.

Також за замовчуванням сортування виконується у зростаючому порядку (ASC). Для зворотного сортування використовується DESC. Кілька полів сортуються послідовно: спочатку за першим, потім за другим і так далі.

3.3 Створення об’єднаних таблиць

Об'єднання таблиць у SQL дозволяє працювати з даними з декількох таблиць одночасно, створюючи зв'язки між ними за певними умовами. Це одна з найважливіших функцій для роботи з реляційними базами даних, оскільки таблиці зазвичай нормалізовані і їхні дані зберігаються в різних частинах структури бази.

Основні типи об’єднань:

- INNER JOIN – повертає тільки ті записи, які мають відповідності в обох таблицях.
- LEFT JOIN (або LEFT OUTER JOIN) – повертає всі записи з лівої таблиці та відповідні записи з правої таблиці (або NULL, якщо відповідності немає).
- RIGHT JOIN (або RIGHT OUTER JOIN) – повертає всі записи з правої таблиці та відповідні записи з лівої таблиці (або NULL, якщо відповідності немає).

Розглянемо функцію INNER JOIN. Вона використовується для вибірки записів, які мають відповідності в обох таблицях. Якщо запис в одній таблиці не має відповідності в іншій, він не буде включений у результат (Рисунок 3.9).

```
SELECT agreements.id, agreements.client_id, clients.id FROM agreements
INNER JOIN clients ON agreements.client_id = clients.id;
```

Рисунок 3.9 – Скрипт функції INNER JOIN

У результаті отримуємо відповідні колонки з двох таблиць (Рисунок 3.10). Маємо 3 колонки: ідентифікатор угоди, дані про ідентифікатор клієнта в угоді та ідентифікатор клієнта відповідно.

id	client_id	id
18	1	1
21	1	1
22	1	1
19	2	2

Рисунок 3.10 – Результат виконання скрипта

Таблиця clients пов’язана з таблицею agreements через поле client_id. INNER JOIN поверне лише тих клієнтів, які мають угоди.

LEFT JOIN використовується для отримання всіх записів із лівої таблиці, навіть якщо у правій таблиці немає відповідностей. У випадку, якщо відповідності немає, дані з правої таблиці будуть заповнені NULL. Прикладом йому буде скрипт для таблиць service_discounts і activity_discounts (Рисунок 3.11).

```
SELECT * FROM activity_discounts
LEFT JOIN services ON activity_discounts.value = services.price;
```

Рисунок 3.11 – Скрипт LEFT JOIN

Відповідно отримуємо рядки лівої і правої таблиці, у разі відсутності відповідностей у правій – NULL (Рисунок 3.12).

id	category	value	id	description	price
1	Law Enforcement	300	NULL	NULL	NULL
2	Legal Practitioner	400	NULL	NULL	NULL
3	Corporate Executive	600	8	Contract Drafting	600
4	Small Business Owner	200	NULL	NULL	NULL
5	Healthcare Professional	250	NULL	NULL	NULL
6	Education Sector	150	NULL	NULL	NULL
7	Technology Specialist	350	NULL	NULL	NULL

Рисунок 3.12 – Результат виконання скрипта LEFT JOIN

RIGHT JOIN використовується для отримання всіх записів із правої таблиці, навіть якщо у лівій таблиці немає відповідностей. У разі відсутності відповідностей у лівій таблиці дані будуть заповнені NULL. Прикладом для цього буде скрипт для таблиць activity_discounts і service_discounts так само (Рисунок 3.13).

```
SELECT * FROM service_discounts
RIGHT JOIN activity_discounts ON service_discounts.value = activity_discounts.value;
```

Рисунок 3.13 – Скрипт RIGHT JOIN

Відповідно рядки правої і лівої таблиць, у разі відсутності відповідностей у лівій - NULL (Рисунок 3.14).

id	category	value	id	category	value
3	Seasonal Discount	300	1	Law Enforcement	300
NULL	NULL	NULL	2	Legal Practitioner	400
NULL	NULL	NULL	3	Corporate Executive	600
2	Loyalty Discount	200	4	Small Business Owner	200
5	Bulk Purchase Discount	250	5	Healthcare Professional	250
4	Referral Discount	150	6	Education Sector	150
6	Early Payment Discount	350	7	Technology Specialist	350
NULL	NULL	NULL	8	Government Official	500

Рисунок 3.14 – Скрипт RIGHT JOIN

Загалом об'єднання таблиць у SQL є одним із найбільш потужних інструментів для роботи з реляційними базами даних. Використання правильного типу JOIN залежить від поставленої задачі, структури таблиць і бажаного результату.

3.4 Основні типи умов у фільтруванні

Фільтрування даних у SQL – це один із найважливіших аспектів роботи з базами даних, який дозволяє витягувати тільки ті записи, що відповідають заданим критеріям. Умови фільтрування задаються в секції WHERE і можуть включати

різноманітні предикати та логічні вирази. Це здійснюється також за допомогою операторів, предикатів та ключових слів, таких як AND, OR, NOT, BETWEEN, IN, LIKE, NULL. Вибір правильного типу умови залежить від завдання, яке потрібно вирішити.

Одним із найпростіших способів фільтрування є порівняння значень. Це дозволяє перевіряти, чи відповідає значення одного стовпця заданому критерію або значенню іншого стовпця. Для цього використовуються оператори порівняння, такі як =, <>, >, <, >=, <=. Наприклад, можна знайти всі записи, де значення поля більше або менше певного числа. Умови порівняння можна комбінувати за допомогою логічних операторів AND, OR і NOT, які визначають, чи мають виконуватися всі умови одночасно, чи достатньо виконання хоча б однієї. Прикладом цьому буде відповідний запит SQL (Рисунок 3.15).

```
SELECT id, description, price FROM services
WHERE price > 1200;
```

Рисунок 3.15 – Запит із порівнянням

У результаті даний скрипт повертає всі значення, вищі за 1200 (Рисунок 3.16).

id	description	price
2	Criminal Defense	2500
3	Corporate Law	1500
7	Intellectual Property	1300
12	Immigration Law	1400
14	Environmental Law	1600
19	Litigation Support	1250

Рисунок 3.16 – Результат виконання скрипта

Перевірка діапазонів використовується для визначення, чи належить значення до певного інтервалу. Оператор BETWEEN дозволяє зручно працювати з діапазонами чисел, дат або інших типів даних, включаючи обидва кінці діапазону. Наприклад,

можна відфільтрувати дані про клієнтів, вік яких знаходиться в межах певного інтервалу. Використання NOT BETWEEN дозволяє відфільтрувати дані, що не входять у цей діапазон. Прикладом буде скрипт для цін у таблиці послуг (Рисунок 3.17).

```
SELECT id, description, price FROM services
WHERE price BETWEEN 1200 AND 1300;
```

Рисунок 3.17 – Запит із порівнянням BETWEEN

У результаті отримуємо всі ціни від 1200 до 1300 (Рисунок 3.18).

id	description	price
7	Intellectual Property	1300
11	Civil Rights	1200
19	Litigation Support	1250

Рисунок 3.18 – Результат виконання скрипта

Іноді потрібно перевірити, чи належить значення до заздалегідь визначеного списку можливих варіантів. Для цього використовуються оператори IN і NOT IN. Вони дозволяють працювати зі списком значень без необхідності задавати довгі ланцюжки умов із використанням OR. Це спрощує запис умов і робить запити більш читабельними. Наприклад, можна визначити, чи проживає клієнт у конкретному списку міст або ж виключити клієнтів із цього списку (Рисунок 3.19).

```
SELECT id, comission FROM agreements
WHERE comission IN (30, 40, 60);
```

Рисунок 3.19 – Скрипт з умовою IN

У результаті отримуємо всі ідентифікатори угод, комісія яких дорівнює числам умови (Рисунок 3.20).

id	comission
1	30
2	60
9	40
11	30
14	60
18	60

Рисунок 3.20 – Результат виконання скрипта

Для роботи зі строковими даними використовується оператор LIKE, який дозволяє шукати записи, що відповідають певному шаблону. Шаблони можуть включати символи підстановки: % для будь-якої кількості символів і _ для одного символу. Наприклад, можна знайти всі імена, що починаються на певну літеру, або записи, які закінчуються на певну послідовність символів. Оператор NOT LIKE використовується для виключення даних, що відповідають заданому шаблону. Прикладом буде скрипт, що вибирає всі імена, які починаються на букву S (Рисунок 3.21)

```
SELECT id, name FROM clients
WHERE name LIKE 'S%';
```

Рисунок 3.21 – Скрипт вибору імен по букві

У результаті отримуємо всі імена, які починаються на букву S (Рисунок 3.22).

id	name
2	SKyler Grey
4	Saul Goodman

Рисунок 3.22 –Результат виконання скрипта

Оператор EXISTS перевіряє, чи існує хоча б один рядок, що відповідає підзапиту (Рисунок 3.23). Це дозволяє створювати запити, які враховують зв’язки між

таблицями. Наприклад, можна перевірити, чи має клієнт хоча б один пов’язаний запис у таблиці замовлень. Оператор NOT EXISTS використовується для пошуку записів, для яких не існує відповідних рядків у пов’язаній таблиці. Цей підхід є потужним інструментом для роботи з пов’язаними даними.

```
SELECT id, name FROM clients
WHERE NOT EXISTS (SELECT 1 FROM agreements WHERE agreements.client_id = clients.id);
```

Рисунок 3.23 – Скрипт для NOT EXISTS

У результаті виконання скрипта маємо 0 клієнтів, що не уклали угоди (Рисунок 3.24).

Рисунок 3.24 – Результат виконання скрипта

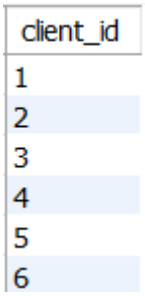
У SQL значення NULL означає відсутність даних. Оператори IS NULL і IS NOT NULL використовуються для перевірки, чи має стовпець значення або ж воно є невизначеним. Це особливо важливо для полів, у яких можуть бути пропущені дані, наприклад, телефонний номер або електронна адреса. Перевірка на NULL дозволяє правильно обробляти такі записи, уникаючи помилок у запитах. Прикладом буде скрипт, що покаже клієнтів, які вказали свої номери (Рисунок 3.25).

```
SELECT client_id FROM client_phone_numbers
WHERE phone_number IS NOT NULL;
```

Рисунок 3.25 – Скрипт перевірки наявності номерів клієнтів

У результаті отримуємо ідентифікатори всіх клієнтів, що вказали свої номери (Рисунок 3.26). У даному випадку всі клієнти вказали свої номери, тому отримано

послідовність усіх ідентифікаторів клієнтів. Обернена до цього дія буде якщо прибрати у скрипті NOT.



client_id
1
2
3
4
5
6

Рисунок 3.26 – Результат виконання скрипта

SQL дозволяє комбінувати різні типи умов у складних запитах, що робить можливим більш точне й деталізоване фільтрування даних. Наприклад, можна одночасно перевіряти належність до певного списку, порівнювати значення й виключати дані, що відповідають певному шаблону. Використання комбінацій логічних операторів AND, OR і NOT дозволяє гнучко налаштовувати запити.

Ефективне фільтрування допомагає значно зменшити обсяг даних, які обробляє запит, що позитивно впливає на продуктивність. Застосування індексів до стовпців, які часто фільтруються, може ще більше прискорити виконання запитів. Також важливо уникати обчислень у самих умовах фільтрування, які можуть знизити ефективність, наприклад, замість перевірки функції на стовпці краще використовувати діапазон.

Загалом фільтрування є основою роботи з даними, і правильне розуміння типів умов дозволяє будувати точні й оптимізовані запити, що відповідають потребам користувача.

3.5 Агрегатні функції

Агрегатні функції є невід'ємною частиною роботи з базами даних, оскільки дозволяють виконувати обчислення над множиною рядків і повертати одне

узагальнене значення. Ці функції застосовуються для підсумовування даних, підрахунку кількості записів, пошуку мінімальних або максимальних значень, обчислення середніх і виконання багатьох інших задач. Вони особливо корисні при аналізі великих обсягів інформації, таких як підготовка звітів або створення аналітичних моделей.

Агрегатні функції зазвичай використовуються в поєднанні з конструкціями GROUP BY і HAVING. Вони дозволяють групувати дані за певними категоріями та застосовувати агрегатні функції до кожної групи окремо. Наприклад, можна знайти середній дохід для кожного регіону або підрахувати кількість клієнтів у кожній віковій групі. Використання HAVING дозволяє фільтрувати результати на основі агрегованих значень, що робить аналіз ще більш гнучким. Прикладом цьому є універсальний скрипт для перегляду цін для угод (Рисунок 3.27).

```
SELECT COUNT(*) AS total_deals,
MIN(price) AS min_amount, MAX(price) AS max_amount,
AVG(price) AS avg_amount FROM agreements;
```

Рисунок 3.27. Скрипт для пошуку MIN, MAX, AVG цін

Відповідно отримуємо 4 колонки в результаті: кількість угод, мінімум, максимум, середнє (Рисунок 3.28).

total_deals	min_amount	max_amount	avg_amount
20	500	2000	1045.0000

Рисунок 3.28 – Результат виконання скрипта для пошуку MIN, MAX, AVG цін

Важливо пам’ятати, що агрегатні функції працюють лише з групами даних і не можуть безпосередньо використовуватися з іншими значеннями у вибірці, якщо ці значення не входять до конструкції GROUP BY. Крім того, агрегатні функції не враховують значення NULL, тому для отримання точних результатів необхідно обробляти такі випадки, наприклад, за допомогою функції COALESCE.

Застосування агрегатних функцій відкриває великі можливості для бізнесу, дозволяючи швидко аналізувати дані й отримувати ключову інформацію. Вони допомагають у створенні звітів про продажі, моніторингу продуктивності працівників, аналізі ринку та інших задачах, які вимагають узагальнення великих обсягів даних.

3.6 Використання CASE, UNION у вибірках даних

CASE – це потужна конструкція, яка дозволяє реалізувати умовну логіку безпосередньо у запитах SQL. Її можна використовувати для формування значень у результатах запитів залежно від певних умов, для виконання умовного групування або сортування. UNION же використовується для об'єднання результатів кількох SQL-запитів у єдиний набір даних, який відображається як один результат.

CASE дозволяє створювати умовні вирази в запитах. Наприклад, можна класифікувати ціни послуг, як «Дорогі», «Дешеві» (Рисунок 3.29).

```
SELECT id, price,
CASE
    WHEN price > 1000 THEN 'Expensive'
    ELSE 'Affordable'
END AS price_label
FROM services;
```

Рисунок 3.29 – Скрипт функції CASE

У результаті отримуємо таблицю з даними та надписом відповідно до умови функції (Рисунок 3.30)

id	price	price_label
1	800	Affordable
2	2000	Expensive
3	1500	Expensive
4	900	Affordable

Рисунок 3.30 – Результат виконання функції CASE

UNION дозволяє об'єднувати результати двох або більше запитів в один набір даних. Наприклад можна поєднати колонки ідентифікаторів, назв і цін знижок на послугу та на діяльність (Рисунок 3.31).

```
SELECT * FROM service_discounts
UNION
SELECT * FROM activity_discounts;
```

Рисунок 3.31 – Скрипт вибірки з UNION

У результаті маємо три колонки, у яких розміщено дані з двох таблиць (Рисунок 3.32).

18	VIP Customer Discount	1550
19	Limited Time Discount	1650
20	Birthday Discount	1750
1	Law Enforcement	300
2	Legal Practitioner	400
3	Corporate Executive	600

Рисунок 3.32 – Результат використання UNION

Використання CASE та UNION значно розширює можливості SQL для аналізу даних і створення складних вибірок. Вони дозволяють підвищити гнучкість запитів, інтегрувати дані з різних джерел та застосовувати умовну логіку безпосередньо у базі даних.

Отже використання UNION дає можливість комбінувати результати кількох вибірок, дозволяючи структурувати й узагальнювати дані з різних джерел. Конструкція CASE є надзвичайно зручною для реалізації умовної логіки в межах SQL-запитів.

ВИСНОВКИ

У результаті виконання даної курсової роботи, було поставлено мету задачі та успішно розроблено базу даних відповідно до завдання. Загалом було розглянуто ключові аспекти проєктування та розробки бази даних, починаючи від аналізу вимог і закінчуючи її практичною реалізацією. Робота була спрямована на створення ефективної та нормалізованої бази даних, здатної забезпечити зберігання та обробку даних без аномалій та з максимальними гарантіями цілісності.

По-перше було розглянуто основні вимоги до бази даних, які включають правильну організацію таблиць, визначення атрибутів і зв'язків між таблицями. Визначено задачі, які має вирішити система, а також побудовано діаграму прецедентів, що дозволяє чітко побачити взаємодію користувачів та системи. Важливою частиною даного етапу стала нормалізація даних, яка дозволила організувати дані таким чином, щоб уникнути дублювання та зберегти їх логічну структуру. Цей процес став важливим кроком у забезпеченні коректності даних та їх ефективного зберігання.

Далі було детально описано створення таблиць спроєктованої бази даних, що включає визначення атрибутів, типів даних та первинних ключів для кожної таблиці. Окремо було зупинено увагу на наданні обмежень, індексів і ключів, що гарантують цілісність та ефективність виконання запитів. Перевірка нормалізації дозволила впевнитися в тому, що створена база даних відповідає вимогам нормальних форм, зокрема третій нормальній формі, що забезпечує відсутність транзитивних залежностей та дублювання даних.

Потім були виконані різноманітні операції з базою даних, такі як вибірка стовпців, сортування даних, групування за допомогою GROUP BY, використання умов фільтрації з HAVING, створення об’єднаних таблиць, агрегація даних за допомогою агрегатних функцій. Цей етап продемонстрував можливості бази даних у контексті складних запитів, що дозволяють ефективно отримувати та аналізувати

необхідну інформацію. Також було продемонстровано використання конструкцій CASE та UNION для додаткової обробки вибірок, що робить систему гнучкою і здатною обробляти складні запити користувачів.

Виконана курсова робота дозволила на практиці застосувати теоретичні знання, здобуті під час вивчення предмету, і отримати досвід у проєктуванні та розробці баз даних. В результаті були створені оптимізовані таблиці, визначені правильні зв'язки між ними, а також забезпечено ефективне використання SQL-запитів для роботи з даними. Курсова робота є чудовим прикладом того, як можна застосувати принципи нормалізації та нормальних форм для розробки бази даних, що задовольняє вимоги цілісності, ефективності та масштабованості. Також дану базу даних можна легко використовувати для початку розробки власної унікальної системи для нотаріальної контори й легко можна розширювати функціонал.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Alan Beaulieu, No Starch Press. O'Reilly Media, Learning SQL: Generate, Manipulate, and Retrieve Data, 2020, 377c.
2. Anthony DeBarros, Practical SQL: A Beginner's Guide to Storytelling with Data, 2022, 464c.
3. Anthony Molinaro, Robert de Graaf. O'Reilly Media. SQL Cookbook: Query Solutions and Techniques for All SQL Users, 2020, 567c.
4. Alistair Cockburn, Addison-Wesley Professional. Writing Effective Use Cases, 2000, 304c.
5. Daryl Kulak, Eamon Guiney, Addison-Wesley Professional. Use Cases: Requirements in Context, 2003, 272c.
6. Don Rosenberg, Matt Stephens, Springer. Use Case Driven Object Modeling with UMLTheory and Practice, 2007, 438c.
7. Kurt Bittner, Ian Spence, AddisonWesley Professional. Use Case Modeling, 2002, 368c.
8. Walter Shields, ClydeBank Media LLC. SQL QuickStart Guide, 2019, 242c.
9. Alice Zhao, O'Reilly Media. SQL Pocket Guide: A Guide to SQL Usage, 2021, 354c.
10. Ben Forta, Sams Publishing. SQL in 10 Minutes a Day, 2019, 256c.
11. Cathy Tanimura, O'Reilly Media. SQL for Data Analysis: Advanced Techniques for Transforming Data into Insights, 2021, 357c.
12. John Viescas, Addison-Wesley Professional. SQL Queries for Mere Mortals: A Hands-On Guide to Data Manipulation in SQL, 2018, 960c.

ДОДАТОК А

```
CREATE DATABASE IF NOT EXISTS notary_office;
USE notary_office;

-- Крок 1. Створення таблиць. Після цього переходимо до завпонення у файл Notary_office_insertData
CREATE TABLE clients (
	id INT NOT NULL AUTO_INCREMENT,
	name VARCHAR(45) NOT NULL,
	employment VARCHAR(45) NOT NULL,
	address VARCHAR(45) NOT NULL,
	PRIMARY KEY (id))
ENGINE = InnoDB;

CREATE TABLE client_phone_numbers (
   client_id INT NOT NULL,
   phone_number VARCHAR(45) NOT NULL,
   FOREIGN KEY (client_id) REFERENCES clients(id)
) ENGINE = InnoDB;

CREATE TABLE services (
	id INT NOT NULL AUTO_INCREMENT,
	description VARCHAR(150) NOT NULL,
	price INT NOT NULL CHECK(price > 0),
	PRIMARY KEY (id))
ENGINE = InnoDB;


CREATE TABLE service_discounts (
	id INT NOT NULL AUTO_INCREMENT,
	category VARCHAR(45) NOT NULL,
	value INT NOT NULL CHECK(value >= 0),
	PRIMARY KEY (id))
ENGINE = InnoDB;
```

```
CREATE TABLE activity_discounts (
            id INT NOT NULL AUTO_INCREMENT,
            category VARCHAR(45) NOT NULL,
            value INT NOT NULL CHECK(value >=0),
            PRIMARY KEY (id))
ENGINE = InnoDB;

CREATE TABLE agreements (
          id INT NOT NULL AUTO_INCREMENT,
   datetime DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
          price INT NOT NULL CHECK(price > 0),
          comission INT NOT NULL,
          description VARCHAR(150) NOT NULL,

   -- Ідентифікатори про інші таблиці
          client_id INT NOT NULL,
          service_discount_id INT NOT NULL,
          activity_discount_id INT NOT NULL,

          PRIMARY KEY (id))
ENGINE = InnoDB;

CREATE TABLE service_agreement (
   service_id INT NOT NULL,
   agreement_id INT NOT NULL,
   PRIMARY KEY (service_id, agreement_id))
ENGINE = InnoDB;
```

ДОДАТОК Б

```
USE notary_office_new;

-- Крок 2. Наповнення таблиць даними
-- Додавання можливих послуг та цін до таблиці сервісів. Далі переходимо до файлу Notary_office_addConstraints
INSERT INTO services (description, price) VALUES
('Legal Consultation', 800),
('Criminal Defense', 2000),
('Corporate Law', 1500),
('Tax Evasion Advice', 900),
('Personal Injury Claims', 700),
('Real Estate Transactions', 1100),
('Intellectual Property', 1300),
('Contract Drafting', 600),
('Employment Law', 1000),
('Family Law', 500),
('Civil Rights', 1200),
('Immigration Law', 1400),
('Bankruptcy Filing', 950),
('Environmental Law', 1600),
('Insurance Claims', 750),
('Estate Planning', 1050),
('Mediation Services', 850),
('Arbitration', 950),
('Litigation Support', 1250),
('Debt Collection', 650);

-- Додавання даних про клієнтів в однойменній таблиці
INSERT INTO clients (name, employment, address) VALUES
('Hank Schrader', 'DEA Agent', 'Albuquerque'),
('Skyler White', 'Accountant', 'Albuquerque'),
('Marie Schrader', 'Radiologic Technologist', 'Albuquerque'),
('Saul Goodman', 'Lawyer', 'Albuquerque'),
('Mike Ehrmantraut', 'Private Investigator', 'Albuquerque'),
```

```
('Kim Wexler', 'Lawyer', 'Albuquerque'),
('Chuck McGill', 'Lawyer', 'Albuquerque'),
('Howard Hamlin', 'Lawyer', 'Albuquerque'),
('Nacho Varga', 'Cartel Member', 'Mexico'),
('Lydia Rodarte-Quayle', 'Executive', 'Houston'),
('Todd Alquist', 'Lab Assistant', 'Albuquerque'),
('Gale Boetticher', 'Chemist', 'Albuquerque'),
('Tuco Salamanca', 'Cartel Member', 'Mexico'),
('Huell Babineaux', 'Bodyguard', 'Albuquerque'),
('Patrick Kuby', 'Private Investigator', 'Albuquerque'),
('Victor', 'Cartel Member', 'Mexico'),
('Tyrus Kitt', 'Bodyguard', 'Albuquerque'),
('Jane Margolis', 'Artist', 'Albuquerque'),
('Ted Beneke', 'Business Owner', 'Albuquerque'),
('Bogdan Wolynetz', 'Car Wash Owner', 'Albuquerque');

INSERT INTO client_phone_numbers (client_id, phone_number) VALUES
(1, '1234567890'), (2, '5678901234'), (3, '1234567890'), (4, '7778889999'), (5, '2223334444'),
(6, '8880002222'), (7, '1234567890'), (8, '5678901234'), (9, '6668880000'), (10, '5556667777'),
(11, '4446668888'), (12, '5678901234'), (13, '2223334444'), (14, '8880002222'), (15, '1234567890'),
(16, '3335557777'), (17, '1234567890'), (18, '6668880000'), (19, '4446668888'), (20, '5556667777');

-- Додавання знижок за діяльність в таблицю знижок
INSERT INTO activity_discounts (category, value) VALUES
('Law Enforcement', 300),
('Legal Practitioner', 400),
('Corporate Executive', 600),
('Small Business Owner', 200),
('Healthcare Professional', 250),
('Education Sector', 150),
('Technology Specialist', 350),
('Government Official', 500),
('Military Personnel', 450),
('Non-Profit Worker', 100),
('Lawyer', 450),
('Entrepreneur', 350),
('Financial Analyst', 400),
```

('Artist', 200),
('Writer', 250),
('Engineer', 300),
('Consultant', 500),
('Scientist', 550),
('Chef', 150),
('Athlete', 200);

-- Додавання знижок за послугу в таблицю знижок
INSERT INTO service_discounts (category, value) VALUES
('New Customer Discount', 100),
('Loyalty Discount', 200),
('Seasonal Discount', 300),
('Referral Discount', 150),
('Bulk Purchase Discount', 250),
('Early Payment Discount', 350),
('Promotional Discount', 450),
('Membership Discount', 550),
('Clearance Discount', 650),
('Holiday Discount', 750),
('Volume Discount', 850),
('Special Offer Discount', 950),
('Employee Discount', 1050),
('First-Time Buyer Discount', 1150),
('Festival Discount', 1250),
('Trade-In Discount', 1350),
('Anniversary Discount', 1450),
('VIP Customer Discount', 1550),
('Limited Time Discount', 1650),
('Birthday Discount', 1750);

-- Додавання створенних договорів в однойменну таблицю
INSERT INTO agreements (datetime, price, comission, description, client_id, activity_discount_id, service_discount_id) VALUES
('2024-03-22', 500, 30, 'Legal consultation for business setup', 4, 4, 4),
('2024-04-02', 800, 60, 'Defense in criminal case', 5, 5, 5),
('2024-03-26', 1500, 100, 'Corporate law consultation', 6, 6, 6),

('2024-03-27', 2000, 150, 'Tax evasion advice', 7, 7, 7),
('2024-04-03', 900, 70, 'Personal injury claims processing', 8, 8, 8),
('2024-03-28', 700, 50, 'Real estate transaction', 9, 9, 9),
('2024-03-29', 1100, 80, 'Intellectual property case', 10, 10, 10),
('2024-04-04', 1300, 90, 'Contract drafting for clients', 11, 11, 11),
('2024-03-30', 600, 40, 'Employment law case', 12, 12, 12),
('2024-04-05', 1000, 70, 'Family law consultation', 13, 13, 13),
('2024-03-31', 500, 30, 'Civil rights case', 14, 14, 14),
('2024-04-06', 1200, 90, 'Immigration law consultation', 15, 15, 15),
('2024-04-01', 1400, 100, 'Bankruptcy filing service', 16, 16, 16),
('2024-04-07', 950, 60, 'Environmental law case', 17, 17, 17),
('2024-04-02', 1600, 110, 'Insurance claims processing', 18, 18, 18),
('2024-04-08', 750, 50, 'Estate planning service', 19, 19, 19),
('2024-04-03', 1050, 70, 'Mediation services', 20, 20, 20),
('2024-04-09', 850, 60, 'Arbitration services', 1, 1, 4),
('2024-04-04', 950, 65, 'Litigation support', 2, 2, 5),
('2024-04-10', 1250, 85, 'Debt collection services', 3, 3, 6);

-- Додавання даних в проміжну таблицю послуг
INSERT INTO service_agreement (service_id, agreement_id) VALUES
(1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8), (9, 9), (10, 10),
(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9), (9, 10), (10, 1);

ДОДАТОК В

```
USE notary_office_new;

-- Крок 3. Додавання обмежень
-- Індекси
-- Додавання індексів
ALTER TABLE services ADD UNIQUE INDEX service_id_UNIQUE (id ASC) VISIBLE;
ALTER TABLE service_discounts ADD UNIQUE INDEX service_discount_id_UNIQUE (id ASC) VISIBLE;
ALTER TABLE clients ADD UNIQUE INDEX client_id_UNIQUE (id ASC) VISIBLE;
ALTER TABLE activity_discounts ADD UNIQUE INDEX activity_discount_id_UNIQUE (id ASC) VISIBLE;
ALTER TABLE agreements ADD UNIQUE INDEX agreement_id_UNIQUE (id ASC) VISIBLE;

ALTER TABLE agreements ADD INDEX fk_Agreement_Clients_id (client_id ASC) VISIBLE;
ALTER TABLE agreements ADD INDEX fk_agreements_discounts_id (service_discount_id ASC) VISIBLE;
ALTER TABLE agreements ADD INDEX fk_agreements_activity_discounts_id (activity_discount_id ASC) VISIBLE;

-- Видалення усіх індексів
-- ALTER TABLE services DROP INDEX service_id_UNIQUE;
-- ALTER TABLE service_discounts DROP INDEX service_discount_id_UNIQUE;
-- ALTER TABLE clients DROP INDEX client_id_UNIQUE;
-- ALTER TABLE activity_discounts DROP INDEX activity_discount_id_UNIQUE;
-- ALTER TABLE agreements DROP INDEX agreement_id_UNIQUE;

-- ALTER TABLE agreements DROP INDEX fk_Agreement_Clients_id;
-- ALTER TABLE agreements DROP INDEX fk_Agreement_Services_id;
-- ALTER TABLE agreements DROP INDEX fk_agreements_discounts_id;
-- ALTER TABLE agreements DROP INDEX fk_agreements_activity_discounts_id;


-- Додавання зовнішніх ключів
ALTER TABLE agreements ADD CONSTRAINT fk_Agreement_Clients
FOREIGN KEY (client_id) REFERENCES clients (id)
ON DELETE NO ACTION ON UPDATE NO ACTION;
```

```
ALTER TABLE agreements ADD CONSTRAINT fk_agreements_service_discounts
FOREIGN KEY (service_discount_id) REFERENCES service_discounts (id)
ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE agreements ADD CONSTRAINT fk_agreements_activity_discounts
FOREIGN KEY (activity_discount_id) REFERENCES activity_discounts (id)
ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE service_agreement ADD CONSTRAINT
FOREIGN KEY (service_id) REFERENCES services (id)
ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE service_agreement ADD CONSTRAINT
FOREIGN KEY (agreement_id) REFERENCES agreements (id)
ON DELETE NO ACTION ON UPDATE NO ACTION;

-- Видалення усіх зовнішніх ключів
-- ALTER TABLE agreements DROP CONSTRAINT fk_Agreement_Clients;
-- ALTER TABLE agreements DROP CONSTRAINT fk_Agreement_Services;
-- ALTER TABLE agreements DROP CONSTRAINT fk_agreements_service_discounts;
-- ALTER TABLE agreements DROP CONSTRAINT fk_agreements_activity_discounts;
```