МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЦЕНТРАЛЬНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до виконання лабораторних робіт з навчальної дисципліни «Бази даних» (2 частина) для студентів денної та заочної форми навчання за спеціальностями F3 «Комп'ютерні науки», F7 «Комп'ютерна інженерія», F5 «Кібербезпека та захист інформації», G5 «Електроніка, електронні комунікації, приладобудування та радіотехніка»

ЗАТВЕРДЖЕНО
На засіданні кафедри кібербезпеки та програмного забезпечення,
протокол № 10 від 25.03.2025 року

КРОПИВНИЦЬКИЙ
2025

Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни «Бази даних» (2 частина) для студентів денної та заочної форми навчання за спеціальностями F3 «Комп'ютерні науки», F7 «Комп'ютерна інженерія», F5 «Кібербезпека та захист інформації», G5 «Електроніка, електронні комунікації, приладобудування та радіотехніка» / уклад. В.В. Босько, Л.В. Константинова, Л.І. Поліщук — Кропивницький: ЦНТУ, 2025. — 61 с.

Укладачі: Босько В. В., Константинова Л. В., Поліщук Л. І.

Рецензенти:

Смірнов О. А.	доктор технічних наук, професор, професор кафедри кібербезпеки та програмного забезпечення Центральноукраїнського національного технічного університету;
Мацуй А. М.	доктор технічних наук, професор, доцент кафедри автоматизації виробничих процесів Центральноукраїнського національного технічного університету, академік Академії технічних наук України, член-кореспондент Академії Прикладних Наук

Вступ

Методологія проектування баз даних – складається з комплексу певних методів, засобів та принципів, які застосовують при послідовній побудові структур баз даних. Уявлення про бази даних покращується при набуванні навичок створення баз даних та застосування інструментів для керування ними.

Для опанування основних навиків у роботі з базами даних необхідно ознайомитись з основними теоретичними поняттями, що входять до курсу «Бази даних», виконувати практичні завдання та вивчати деякі теми самостійно.

В цих методичних рекомендаціях пропонується виконання завдань з лабораторних робіт за темами, що представлено в таблиці 1. Вони є продовженням методичних рекомендацій до виконання лабораторних робіт з навчальної дисципліни «Бази даних» (1 частина).

Таблиця 1 – Теми лабораторних робіт

№	Теми лабораторних робіт
1	Проектування БД. Нормалізація БД.
2	Концептуальне проектування БД. ER-моделювання.
3	Моделювання даних. Модель «об’єкт-атрибут-зв’язок». Створення схем даних. Типи відношень.
4	Функції для маніпулювання даними.
5	Робота з представленнями.
6	Застосування індексів в БД.
7	Робота з семантичними помилками у запитах БД.

Для виконання завдань до лабораторних робіт за цими методичними рекомендаціями достатньо буде застосування MySQL, Microsoft Access (чи аналог) та Lucidchart (чи інший засіб) для побудови діаграм, а також текстовий редактор для підготовки звітів. Про MySQL вже згадувалось в методичних рекомендаціях до виконання лабораторних робіт з навчальної дисципліни «Бази даних» (1 частина) та можна ознайомитись в посібнику [1].

Продукт Microsoft Access зазвичай входить до складу пакету Microsoft Office, який є комерційним програмним забезпеченням. Проте існують кілька способів використання Microsoft Access безкоштовно або зі зниженими витратами:

1. Безкоштовний пробний період Microsoft 365. Microsoft пропонує безкоштовний пробний період для Microsoft 365, до складу якого входить Access. Це дозволяє користуватися програмою певний час без оплати.

2. Програми для студентів та навчальних закладів. Якщо ви студент або працівник навчального закладу, можливо, у вашому навчальному закладі є ліцензія Microsoft, яка надає доступ до Access безкоштовно.

3. Альтернативи Access. Якщо потрібно саме безкоштовна програма з функціями, схожими на Microsoft Access, можливо розглянути наступні:

- LibreOffice Base - це схожа на Access безкоштовна програма з відкритим кодом.
- Apache OpenOffice Base - безкоштовна альтернатива Access.
- Google Sheets або Airtable – це прості у використанні рішення для створення БД для онлайн застосування.

Access – це приклад СКБД для розв'язування широкого кола завдань користувачів без програмування. Одна з основних переваг Access полягає в тому, що в наявності прості та зручні засоби обробки кількох таблиць в одній базі даних. У Access є зручні та наочні засоби зв'язування таблиць, які зберігаються в одному файлі.

Методичні рекомендації містять теоретичні відомості до кожної з представлених лабораторних робіт, з якими пропонується ознайомитись перед виконанням робіт.

Вибравши й узгодивши з викладачем предметну область, над якою ви будете працювати, ви повинні виконати завдання до лабораторних робіт, а також відповісти на запитання вкінці кожної лабораторної роботи. Звіт повинен містити хід виконання завдань, а також графічні матеріали, що підтверджують правильність виконання цих завдань.

Форма підсумкового контролю: екзамен.

Максимальну кількість балів студент може одержати у випадку відвідування всіх лекцій, лабораторних занять, виконання і захисту виконаних завдань для самостійної роботи у встановлений термін проходження контролю.

При виконанні й захисті робіт після встановленого терміну, одержані бали перераховуються з коефіцієнтом: для самостійної роботи студента -0,3; лабораторної роботи -0,7.

Таблиця 2 - Шкала оцінювання: національна та ECTS

Сума балів за всі види навчальної діяльності	Оцінка ECTS	Оцінка за національною шкалою
		для екзамену, курсової роботи
90 – 100	**A**	відмінно
82-89	**B**	добре
74-81	**C**	
64-73	**D**	задовільно
60-63	**E**	
35-59	**FX**	незадовільно з можливістю повторного складання
0-34	**F**	незадовільно з обов'язковим повторним вивченням дисципліни

В кінці деяких робіт містяться посилання на додаткову відеоінформацію за темою, що полегшує сприйняття теоретичних відомостей.

Рекомендовані джерела інформації, що знаходяться в кінці методичних рекомендацій, містять список літератури, що застосовувалась при розробці рекомендацій та пропонується до перегляду й ознайомлення.

Лабораторна робота №1

Тема: Проектування БД. Нормалізація БД.

Мета: Вивчивши етапи проектування баз даних, навчитись створювати проекти та підвищувати якість проекту БД методом нормалізації даних.

Завдання:

1. За вибраним варіантом побудуйте таблицю (не нормалізовану) для своєї БД.
2. Згідно правил нормалізації приведіть таблицю до 1 НФ. Виведіть отриманий результат.
3. Згідно правил нормалізації приведіть таблицю до 2 НФ. Виведіть отриманий результат.
4. Видаліть транзитивні та функціональні залежності в таблиці та приведіть до 3НФ таблицю та винесіть відповідні дані до іншої таблиці.
5. Виведіть отриманий результат.
6. Виконайте денормалізацію (за вимогами і необхідністю).

Теоретичні відомості:

Процес **проектування БД** являє собою послідовність переходів від неформального мовного опису інформаційної структури предметної області до формалізованого опису об'єктів ПО в термінах деякої моделі. Проектування БД складається з таких етапів:

- системний аналіз ПО;
- концептуальне проектування;
- логічне проектування;
- фізичне проектування.

Важливим етапом проектування БД є процес, що називають нормалізацією [1]. Під **нормалізацією** розуміється спрощення конструкцій БД методом позбавлення збитковостей та повторів. Частіше за все збитковість ліквідують розміщуючи групи значень, що повторюються в окремі таблиці. А потім зв’язують іх за допомогою зовнішніх ключів. Це не тільки зменшує об’єм

дискового простору для БД, а також спрощує внесення змін в БД. За допомогою команди Update внесення змін відбувається одношагово в нормалізованій БД, оскільки немає дублікатів значень в різних таблицях.

Також процес нормалізації включає перевірку зв'язків в БД для виявлення небажаних перетинань та видалення некоректних залежностей. Це правильно, бо при створенні непрямих табличних зв'язків ускладнюється конструкція БД, що призводить до зменшення швидкості запитів до БД.

Є нормальні форми, що допомагають коректно нормалізувати БД.

Існує наступна послідовність нормальних форм [2]:

- перша нормальна форма (1НФ);
- друга нормальна форма (2НФ);
- третя нормальна форма (ЗНФ);
- посилена третя нормальна форма, або нормальна форма Бойса-Кодда (БКНФ);
- четверта нормальна форма(4НФ);
- п'ята нормальна форма (5НФ);
- доменно-ключева нормальна форма (ДКНФ);
- шоста нормальна форма (6НФ).

Перша нормальна форма (1НФ)

Змінна відношення знаходиться в 1НФ тільки тоді, якщо в будь-якому допустимому значенні відношення кожен його кортеж містить тільки одне значення для кожного з атрибутів.

За визначенням поняття відношення, у реляційній моделі, відношення завжди знаходиться в 1НФ. Що ж стосується різних таблиць, то вони можуть не бути правильними представленнями відношень і, відповідно, можуть не перебувати в першій нормальній формі.

Про повну й неповну функціональні залежності

Залежність неключового атрибуту від частини складного ключа (з кількох атрибутів) називають **неповною функціональною залежністю**.

Залежність неключового атрибуту від всіх одночасно атрибутів складного ключа називають **повною функціональною залежністю**.

Друга нормальна форма (2НФ)

Змінна відношення знаходиться в другій нормальній формі тільки тоді, якщо вона знаходиться в першій нормальній формі й кожен неключовий атрибут функціонально повно залежить від її потенційного ключа.

Третя нормальна форма (3НФ)

Змінна відношення знаходиться в третій нормальній формі тільки тоді, якщо вона знаходиться в другій нормальній формі, й відсутні транзитивні функціональні залежності неключових атрибутів від ключових.

Нормальна форма Бойса-Кодда (BCNF)

Змінна відношення знаходиться в нормальній формі Бойса-Кодда (тобто в посиленій третій нормальній формі) тільки тоді, якщо кожна її нетривіальна та не приводима зліва функціональна залежність має в якості свого детермінанта деякий потенційний ключ.

Четверта нормальна форма (4НФ)

Змінна відношення знаходиться в четвертій нормальній формі, якщо вона знаходиться в нормальній формі Бойса - Кодда та не містить нетривіальних багатозначних залежностей.

П’ята нормальна форма (5НФ)

Змінна відношення знаходиться в п'ятій нормальній формі (тобто, в проекційно-сполучній нормальній формі) тільки тоді, якщо кожна нетривіальна залежність з'єднання в ній визначається потенційним ключем (ключами) цього відношення.

Доменно-ключева нормальна форма (ДКНФ)

Змінна відношення знаходиться в ДКНФ тільки тоді, якщо кожне накладене на неї обмеження є логічним наслідком обмежень доменів та обмежень ключів, накладених на дану змінну відношення [3].

Шоста нормальна форма (6НФ)

Змінна відношення знаходиться в шостий нормальній формі тільки тоді, якщо вона задовільняє всім нетривіальним залежностям з'єднання. З визначення випливає, що змінна знаходиться в 6НФ тільки тоді, якщо вона не приводима,

тобто не може бути піддана подальшій декомпозиції без втрат. Кожна змінна відношення, яка знаходиться в 6НФ, також знаходиться й в 5НФ.

Можна означити основні властивості нормальних форм: кожна наступна нормальна форма за деяким критерієм краща ніж попередя нормальна форма; Під час переходу до наступної нормальної форми властивості попередніх нормальних форм зберігаються.

Найчастіше на практиці можна зустріти приведення таблиць до 1НФ, 2НФ та 3НФ.

Хоча ідеї нормалізації для проектування БД є корисними, проте вони не є універсальним або вичерпним засобом для підвищення якості проекту баз даних.

Денормалізація – це навмисний процес модифікації реляційної моделі, при якому ступінь нормалізації модифікованого відношення стає нижчим.

Денормалізацію використовують в таких ситуаціях, якщо вже нормалізована БД не задовільняє вимогам, які висуваються до продуктивності роботи системи.

Про концептуальне та логічне проектування: https://www.youtube.com/watch?v=CzQVjOovUnE

Приклад виконання можна розглянути за посиланням: Нормалізація бази даних через SQL https://www.youtube.com/watch?v=jorR4e6KZUw

Контрольні запитання до лабораторної роботи 1:

1. З яких етапів складається процес проектування БД?
2. Що розуміють під нормалізацією?
3. Які дії передбачає нормалізація БД?
4. До чого призводить нормалізація БД?
5. Яка форма називається 1 НФ?
6. Яка форма називається 2 НФ?
7. Яка форма називається 3 НФ?
8. Яка форма називається 4 НФ?
9. Яка форма називається 5 НФ?

10. Яка форма називається 6 НФ?

11. Яка форма називається нормальна форма Бойса-Кодда (BCNF)?

12. Яка форма називається ДКНФ?

13. Які нормальні форми найчастіше застосовують?

14. Чи є нормалізація вичерпним засобом для підвищення якості проекту БД?

15. Що називають денормалізацією?

Лабораторна робота № 2

Тема: Концептуальне проектування БД. ER-моделювання.

Мета: Навчитись будувати концептуальні схеми БД. Розглянути підходи до моделювання даних та набути навичок створення ER-діаграм.

Завдання:

1. Проаналізувати та дослідити ПО на визначення сутностей, атрибутів і первинних ключів та скласти таблицю (як приклад таблиця 2.2).
2. Подати сутності та атрибути у ER-діаграмах П. Чена й «Пташина лапка».
3. .На основі задач, що були поставлені, досліджень ПО та побудованої таблиці створити ER-діаграму як концептуальний проект.
4. Отриманий концептуальний проект треба перевірити на збитковість і на відповідність транзакціям користувачів.
5. Застосуйте Lucidchart (чи інший засіб) для побудови діаграм.

Теоретичні відомості:

Концептуальна схема БД починає створення з концептуального проектування. В основі лежить *концептуальна модель даних*. Концептуальна модель представляє загальний погляд на дані. Існують два головних підходи до моделювання даних при концептуальному проектуванні [1]:

- семантичні моделі;
- об'єктні моделі.

Семантичні моделі загалом відображають структури даних. Найбільш поширеною семантичною моделлю є модель "сутність – зв'язок" (Entity Relationship model, ER-модель). *ER-модель* складається із сутностей, зв'язків, атрибутів, доменів атрибутів, ключів. Моделювання даних відображає логічну структуру даних (схоже як блок-схеми алгоритмів відображають логічну структуру програми).

Об'єктні моделі, головним чином, описують поведінку об'єктів даних та засоби маніпуляції даними. Першорядне поняття таких моделей – це об'єкт, або сутність, що описується станом і поведінкою. Стан об'єкта визначають множиною його атрибутів, а поведінка об'єкта визначається набором операцій, що специфіковані саме для нього.

Ці моделі знаходять свою реалізацію в Extended Entity Relationship model (EER-модель), тобто в розширеному ER- моделюванні.

Модель "сутність-зв'язок"

ER-моделювання являє собою низхідний підхід до проектування БД, який починається з визначення найбільш важливих даних, які називаються *сутностями* (entities), і *зв'язків* (relationships) між даними, які повинні бути подані в моделі. Наступною дією в модель заноситься інформація про властивості сутностей та зв'язків, що називають *атрибутами*, а також обмеження, що відносяться до сутностей, зв'язків й атрибутів. Завдяки ER-моделі маємо графічне подання логічних об'єктів і їх відношень в структурі бази даних. Послідовність виконання ER-моделювання показана на рисунку 2.1.

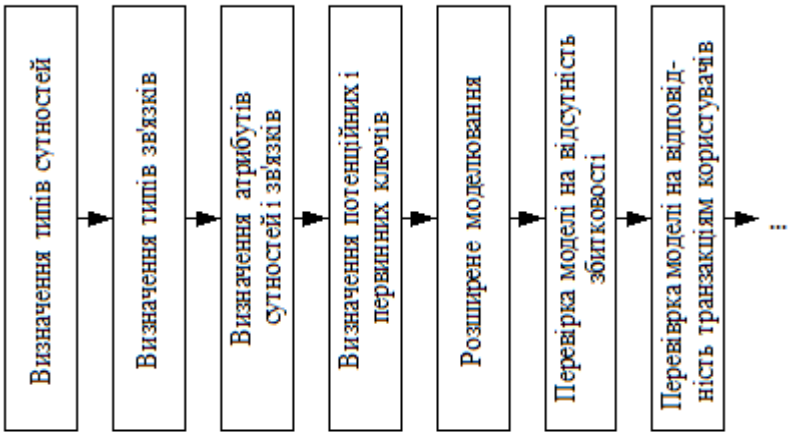


Рисунок 2.1 - Побудова моделі "сутність – зв'язок" поетапно [1]

Діаграми Чена набули розвитку у багатьох роботах з ER- моделювання. Розглянемо відомі наступні моделі:

- «Пташина лапка» (розробник К.М. Бахман);
- IDEF1X (розробник Т.Л. Ремей);
- на основі UML;

– Баркера модель

...

Сутності

Сутність дає можливість виконувати моделювання класу однотипних об'єктів. В межах системи, що моделюється, сутність має унікальне ім'я. Через те, що сутність ототожнюється з деяким класом однотипних об'єктів, то логічно, що в системі існує багато екземплярів цієї сутності. Об'єкт, якому відповідає сутність, має набір атрибутів, які характеризують його властивості, наприклад, сутність Викладач може мати такі атрибути: Індивідуальний або табельний номер, Прізвище, Ім'я, По батькові, Посада, Вчене звання.

Набір атрибутів, який однозначно визначає конкретний екземпляр сутності, називають **ключем**. У прикладі, що надано, для сутності *Викладач* ключем є *Табельний номер,* бо табельні номери всіх викладачів різні. Екземпляром сутності *Викладач* буде опис конкретного викладача. Загальноприйняте позначення сутності - прямокутник (рисунок 2.2).

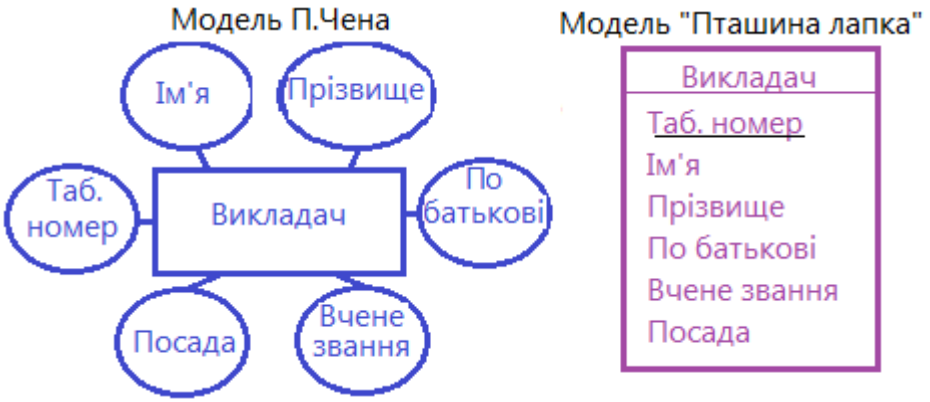


Рисунок 2.2 - Подання сутностей та атрибутів у ER-діаграмах П. Чена й «Пташина лапка»

Зв'язки

Зв’язки між сутностями створюються, та за ними можна визначити, яким чином сутності взаємодіють між собою чи співвідносяться. Зв’язки бувають таких видів:

– Бінарні, тобто між двома сутностями;

– Тернарні, тобто між трьома сутностями ;

- N-арний зв'язок , тобто між N сутностями;
- Рекурсивні, тобто між однією сутністю.

Більш розповсюдженими є бінарні зв'язки. За зв’язком визначають, яким чином екземпляри сутностей пов'язані між собою. Є **бінарні** зв'язки таких видів:

- 1:1 (один до одного);
- 1:M (один до багатьох);
- N:M (багато до багатьох).

На рисунках 2.3 та 2.4 подаються зв'язки у різних ER-моделях.

Зв'язок 1:1, наприклад, завідуючий кафедрою може керувати тільки однією кафедрою, а кожною кафедрою керує тільки один завідувач:

Рисунок 2.3 - Зв’язок 1:1 на діаграмі Чена

Зв'язок 1:M, наприклад, на кафедрі працює багато викладачів, а кожен викладач працює тільки на одній кафедрі:

Рисунок 2.4 - Зв’язок 1:M на діаграмі Чена

Зв'язок N:M, наприклад, студента навчають багато викладачів, а кожен викладач навчає багатьох студентів.

Рисунок 2.5 - Зв’язок N:M на діаграмі Чена

На рисунку 2.6 зображено представлення зв'язків (*а* - 1:1; *б* - 1:M; *в* - N:M) між відношеннями на діаграмі «Пташина лапка».

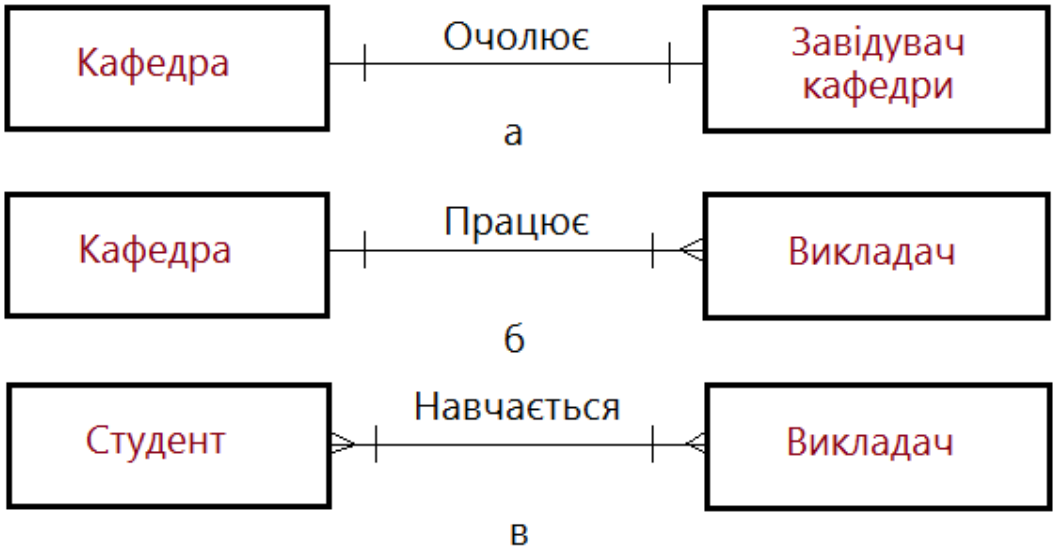


Рисунок 2.6 – Зв’язки 1:1, 1:M та N:M на діаграмі «Пташина лапка»

Атрибути є властивостями сутності. Значення кожного атрибута відбирають з відповідного набору значень, який включає всі потенційні значення, що можуть бути привласнені атрибуту. Ця множина значень називається *доменом*. Наприклад, атрибут Оцінка може приймати чотири значення: незадовільно, задовільно, добре, відмінно. Ці значення і складають домен цього атрибута.

Атрибути залежно від складності значень, які вони можуть приймати поділяються на категорії або типи (Таблиця 2.1). Це прості, складові, однозначні, багатозначні й похідні атрибути. Властивості й приклади видно в таблиці.

Таблиця 2.1 – Типи атрибутів [1]

Тип	**Властивість**
Простий	Атрибут, який не може бути поділений на інші атрибути. Приклад: Прізвище; посада
Складовий	Атрибут, який можна поділити на інші атрибути. Приклад: Адреса; Прізвище, Ім'я, По батькові
Однозначний	Атрибут, що може приймати тільки одне значення. Приклад: Табельний номер, ідентифікаційний номер

Багатозначний	Атрибут, що може приймати багато значень. Приклад: Телефон, Адреса (постійне місце проживання, тимчасове проживання, гуртожиток)
Похідний	Атрибут, що не зберігається в БД, а обчислюється за допомогою певного алгоритму Приклад. Вік (обчислюється по даті народження), стаж, кількість студентів в групі

Для приклада розглядається сутність **Студент** (рисунок 2.7).

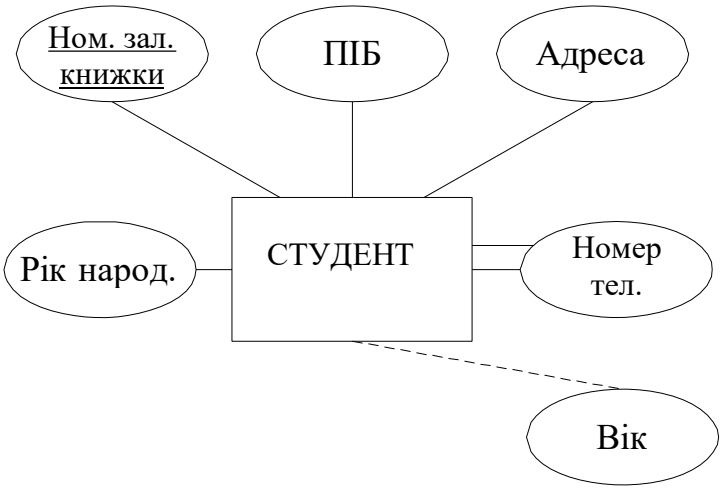


Рисунок 2.7 - ER-діаграма - сутність **Студент**

Атрибути **Номер залікової книжки, Рік народження** є простими.

Атрибути **ПІБ** і **Адреса** є складовими. **ПІБ** може бути поділений на атрибути: **прізвище, ім'я, по батькові**, а **Адреса** - на **індекс, місто, вулиця, будинок, квартира**.

Атрибут **Вік** є похідним: він обчислюється за значенням атрибута **Рік народження** (позначається пунктирною лінією).

Щодо атрибуту **Номер залікової книжки** – він є однозначним, тому, що він не має можливості приймати два значення для одного студента.

Атрибут **Номер телефону** є багатозначним: (позначається подвійною лінією).

Потенційний ключ - це атрибут або набір атрибутів сутності, які застосовуються для ідентифікації екземпляра сутності. Сутність може мати декілька потенційних ключів. В прикладі на рисунку потенційними ключами

можуть бути такі атрибути: **Номер залікової книжки, Прізвище Ім'я По батькові**.

Потенційний ключ, що вибрано для однозначної ідентифікації кожного екземпляра сутності певного типу, називається **первинним ключем**. Первинний ключ визначається з умовою, що є гарантії унікальності його значень, а також мінімальної довжини атрибутів, які входять в його склад. В прикладі - первинний ключ - **Номер залікової книжки**.

Потужність зв'язків

Потужність зв'язку (або кардинальність) показує певне число екземплярів сутностей, які пов'язані з одним екземпляром зв'язаної сутності. В моделі Чена потужність зв'язку відображається відповідними числами поруч з сутностями у форматі (*x, y*). Перше число визначає мінімальне значення потужності зв'язку, а друге - максимальне значення потужності зв’язку. Потужність вказує на число екземплярів у зв'язаній сутності.

Відомості про максимальне і мінімальне значення потужності зв'язку може застосовуватися у ПЗ або за допомогою тригерів, але на рівні таблиць СКБД не оперує з потужністю зв'язків.

Для приклада розглядається зв'язок **Група-Студент** (рис. 2.8).

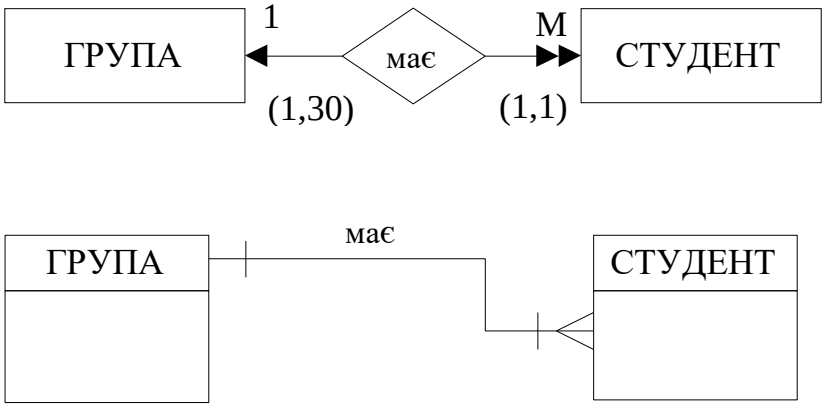


Рисунок 2.8 - ER-діаграми показують зв'язок і потужність [1]

Значення потужності (1,30) поруч з **Група** вказує, що в групі може бути від 1 до 30 студентів. А значення (1,1) - вказує, що студент може бути в списку тільки в одній групі (мінімум 1, максимум 1). У моделі «Пташина лапка» діапазон значень потужності не позначається в ER-діаграмах.

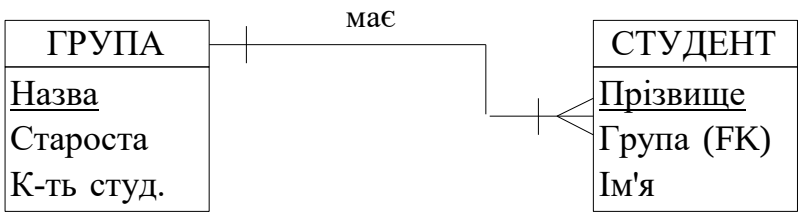
Сильні та слабкі зв'язки

Сутність є **незалежною** від існування, якщо може існувати незалежно від інших сутностей. І навпаки: якщо сутність залежить від існування інших сутностей, то вона є **залежною** від існування. Наприклад, сутності **Студент** і **Група** - незалежні, а сутність **Нагорода студента** є залежною від сутності **Студент**, бо існувати без неї не може.

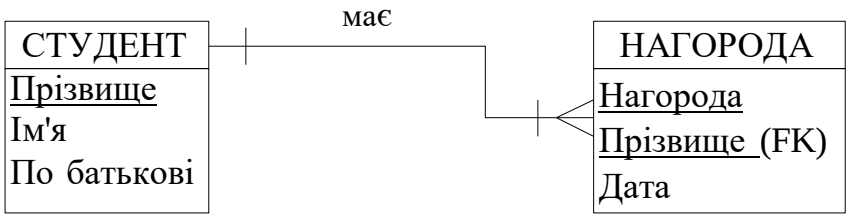
Якщо сутність незалежна від існування іншої, то зв'язок між ними називається **неідентифікаційним зв'язком** або **слабким зв'язком**. На ER-діаграмах "пташина лапка" слабкий зв'язок відображається штриховою лінією.

Ідентифікаційний зв'язок (або **сильний)** має місце у тому випадку, коли одна зв'язана сутність залежить від існування іншої. На ER-діаграмах "пташина лапка" сильний зв'язок відображається суцільною лінією.

Для прикладу розглянемо зв'язки **Група-Студент** і **Студент- Нагорода** (рисунок 2.9).



а



б

Рисунок 2.9 - На ER-діаграмах неідентифікаційний (*а*) та ідентифікаційний (*б*) зв'язки [1]

Якщо сутність **Студент** не має залежності від сутності **Група**, то лінія зв'язку між сутностями виглядає штриховою, а **Назва групи - атрибут** сутності **Студент** виконує роль зовнішнього ключа або Foreign Key (FK).

Якщо сутність **Нагорода** залежна від сутності **Студент**, то лінія зв'язку між сутностями позначається суцільною, а **Прізвище студента,** атрибут

сутності **Нагорода,** є одночасно частиною первинного ключа (Primary Key, PK) та Foreign Key (FK).

Атрибути зв'язків

Подібно сутностям, зв'язки можуть мати атрибути.

Для прикладу атрибути **День** і **Аудиторія** належать до зв'язку між сутностями **Студент** і **Дисципліна** (рисунок 2.10).

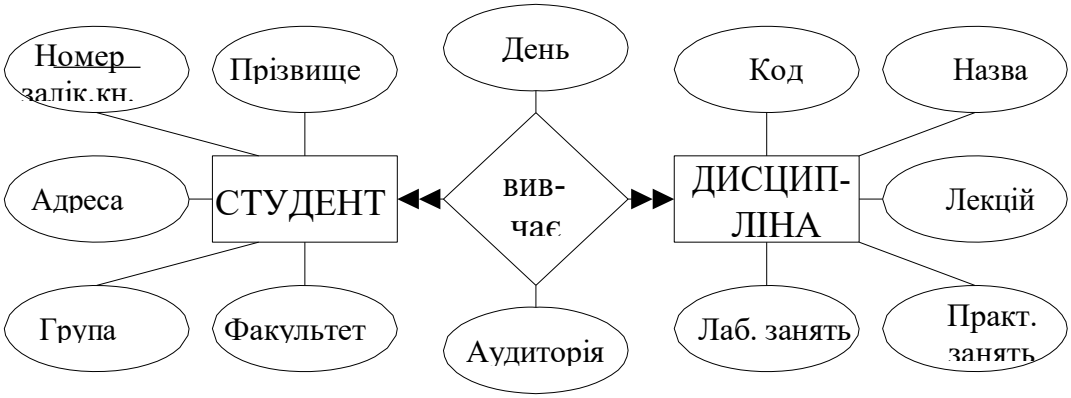


Рисунок 2.10 - Атрибути зв'язку **День** і **Аудиторія** на ER-діаграмі

Обов'язкові / необов'язкові зв'язки

Участь сутності у зв’язку називають необов’язковою, у випадку, якщо один з екземплярів сутності не має потреби в наявності відповідного екземпляра сутності в окремому зв'язку. І навпаки. У зв’язку сутностей, наприклад, **Студент-Нагорода,** не всі студенти мають нагороди. Це означає, що у таблиці **Студент** не кожен екземпляр мусить мати екземпляр сутності в таблиці **Нагорода**. В такому випадку сутність **Нагорода** визначається як необов'язкова відносно до сутності **Студент.** Необов'язкова сутність схематично позначається у вигляді невеликого кола з боку необов'язкової сутності (рисунок 2.11). Якщо є необов'язковість, то це означає, що мінімальне значення потужності зв'язку для необов'язкової сутності буде дорівнювати 0.

Якщо кожен екземпляр сутності має обов’язкову потребу в відповідному екземплярі сутності в окремому зв'язку, то ця участь сутності у зв'язку буде **обов'язковою**. Якщо обов'язковий зв'язок, то для обов'язкової сутності значення мінімальної потужності зв'язку буде дорівнювати 1.

Наприклад, зв'язок між сутностями **Студент** і **Нагорода** є необов'язковим (рисунок 2.11). Може бути така ситуація, що не у коженого студента є

нагорода, але якщо вона є, то нагорода обов'язково пов'язана з певним студентом.

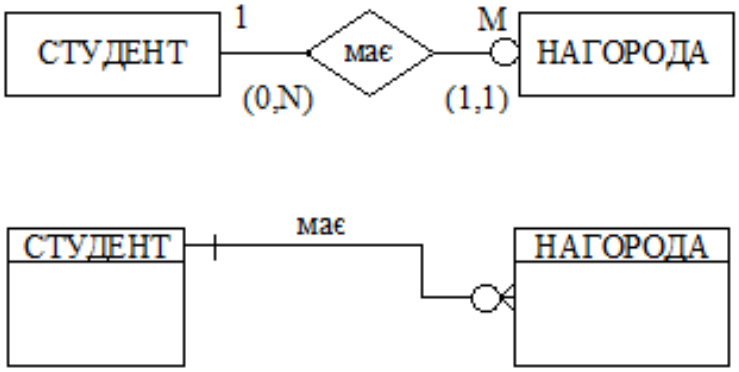


Рисунок 2.11 - Необов'язкова сутність в ER-діаграмах [1]

Слабкі сутності

Слабкою сутністю називається сутність, яка задовільняє таким умовам:

– залежності від існування сутності з якою вона зв'язана;

– первинний ключ цієї сутності частково або повністю отримано з іншої сутності.

Слабка сутність на діаграмі Чена позначається подвійним прямокутником, а на діаграмі "пташина лапка" невеликими сегментами в кожному з кутів прямокутника [1].

Для прикладу на рисунку 2.12 Сутність **Нагорода** є слабкою по відношенню до сутності **Студент**: вона залежить від існування цієї сутності і в її первинний ключ входить первинний ключ сутності **Студент**.

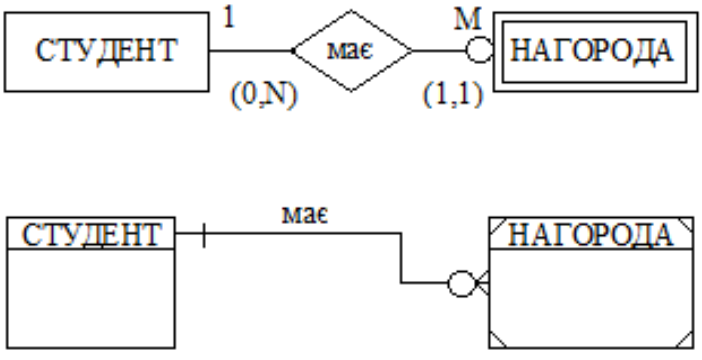


Рисунок 2.12 Позначення слабких сутностей на діаграмах П. Чена і "Пташина лапка"

Складні зв'язки

Використання зв'язків більш високого порядку дає можливість у багатьох випадках краще позначити семантику проблемної області.

Для прикладу розглянуто сутності **Викладач**, **Дисципліна** й **Екзамен**, що утворюють тернарний зв'язок (рисунок 2.13).

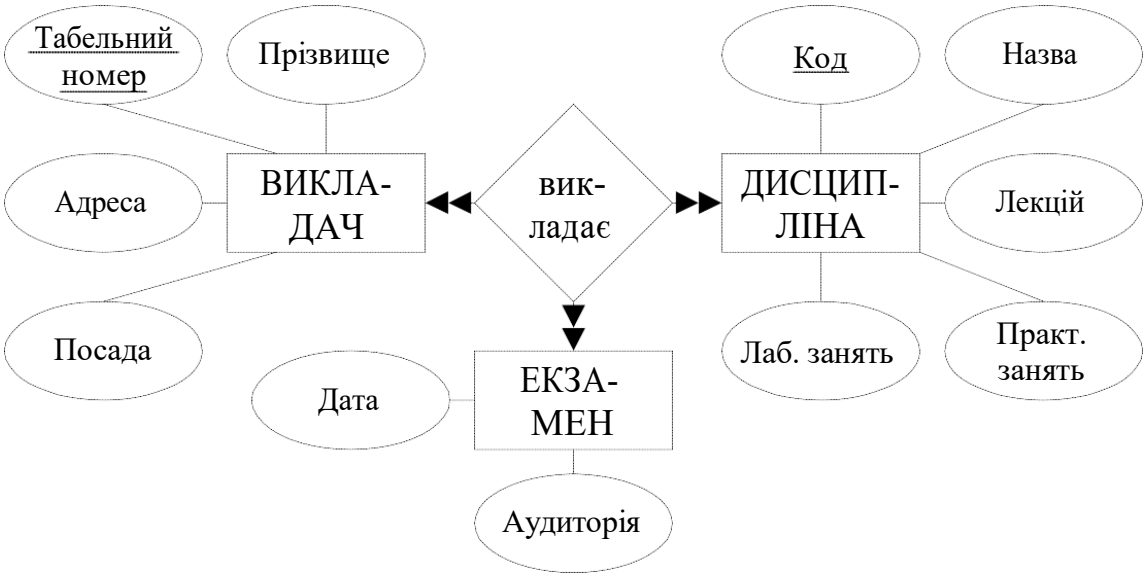


Рисунок 2.13 – Приклад тернарного зв'язку між трьома сутностями на діаграмі П. Чена

Приклад побудови таблиці з сутностей, атрибутів та первинних ключів в таблиці 2.2.

Таблиця 2.2 - Сутності, атрибути й первинні ключі ПО ЗВО [1]

Сутність	**Атрибути**	**Первинний ключ**
ФАКУЛЬТЕТ	Код факультету Назва Декан	Код факультету
СПЕЦІАЛЬНІСТЬ	Код спеціальності Назва Вимоги	Код спеціальності
ГРУПА	Код групи Назва Кількість студентів Староста	Код групи
СТУДЕНТ	Номер залікової книжки ПІБ Адреса	Номер залікової книжки
КАФЕДРА	Код кафедри Назва Завідуючий кафедрою Кількість викладачів	Код кафедри
ВИКЛАДАЧ	Табельний номер ПІБ Посада Науковий ступінь	Табельний номер викладача
ДИСЦИПЛІНА	Код дисципліни Назва Кількість годин Семестр	Код дисципліни

Приклад побудови концептуального проекту у вигляді ER-діаграми:

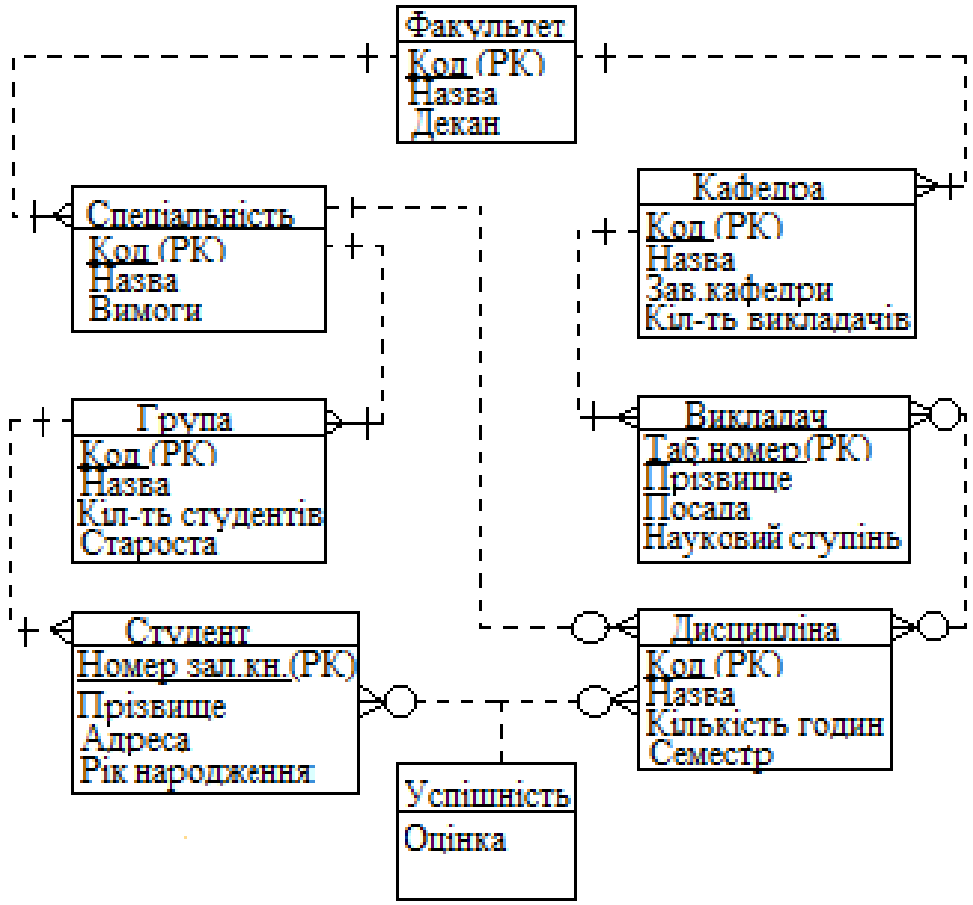


Рисунок 2.14 - ER-діаграма ПО ЗВО [1]

Використання ER-діаграм дозволяє забезпечити просте і наглядне уявлення про логіку головних об'єктів БД та про зв'язки між ними. Ще одна перевага ER-діаграм: вони добре інтегрують з реляційною моделлю.

Недолік ER-моделей: вони мають обмежені можливості для подання відношень і їх обмежень, при наявності багатьох об'єктів можуть бути складними, не мають засобів для опису операцій маніпулювання даними.

Про моделювання (Entity Relationship Diagram (ERD) Tutorial - Part 1: https://www.youtube.com/watch?v=xsg9BDiwiJE&list=PLUoebdZqEHTxpGCwKrb82cIvHNoNaBb4R , а також Part 2: https://www.youtube.com/watch?v=hktyW5Lp0Vo&list=PLUoebdZqEHTxpGCwKrb82cIvHNoNaBb4R&index=2 .

Як створити діаграми в Lucidchart (ERDs in Lucidchart): https://www.youtube.com/watch?v=xIXrnHmZmOA&list=PLUoebdZqEHTxpGCwKrb82cIvHNoNaBb4R&index=6

Як створити діаграму за допомогою ШІ в Lucidchart (Generate a diagram using AI): https://www.youtube.com/watch?v=e7dZNLpr34E&list=PLUoebdZqEHTxpGCwKrb82cIvHNoNaBb4R&index=8 .

Контрольні запитання до лабораторної роботи 2:

1. Що представляє собою сутніть.
2. Чим відрізняється сильна сутність від слабкої?
3. Що представляє собою атрибут?
4. Які бувають атрибути?
5. Що представляє собою поняття ступінь зв'язку?
6. Що таке кардинальне число?
7. Який зв’язок називають ідентифікаційний?
8. Який зв’язок називають неідентифікаційним?
9. Які символьні позначення застосовуються в діаграмах "сутність – зв'язок"?
10. Які є переваги і недоліки ER-моделі?

Лабораторна робота № 3

Тема: Моделювання даних. Модель «об’єкт-атрибут-зв’язок». Створення схем даних. Типи відношень.

Мета: Навчитись працювати з Схемою даних. Набути навичок створення різних типів відношень у БД.

Завдання:

1. Підготуйте необхідні за варіантом таблиці у БД (4 і більше).
2. Створіть відношення один-до-багатьох між таблицями своєї бази даних, що передбачають цей тип. Покажіть на схемі.
3. Продемонструвати зв’язок між таблицями з відношенням багато-до-багатьох. Показать на схемі даних.
4. Створіть зв’язок між таблицями з відношенням один-до-одного застосувати заздалегідь підготовлені таблиці. Показать на схемі.
5. Забезпечте цілісність для ваших даних у зв’язках.
6. Встановіть каскадне видалення. Встановіть каскадне відновлення.
7. Заповніть таблиці даними та перевірте правильність функціонування.

Теоретичні відомості:

Реляційна модель даних (РМД) деякої ПО є набором відношень, що змінюються з часом. Під час побудови інформаційної системи множина відношень дає можливість зберігати інформацію про об’єкти предметної області та створювати зв’язки між ними.

Встановлення взаємозв’язків виконується між таблицями, записи яких логічно пов’язані. Створивши таблиці та визначивши ключі для кожної з них задаються зв’язки й на їх основі можна брати дані з кількох таблиць та вміщувати їх в одну форму, запит або звіт.

Типи відношень бувають: «один-до-одного», «один-до-багатьох» та «багато-до-багатьох».

Між двома таблицями безпосередньо можна встановити відношень: “один-до-одного” та “один-до-багатьох”.

Найбільш розповсюдженим є відношення “один-до-багатьох” (one-to-many), коли одному запису одної таблиці відповідає декілька записів в іншій таблиці. У відношенні «один-до-багатьох» сторона «один» називається головною таблицею, а сторона «багато» – зв’язаною або підлеглою. Таблиця може бути зв’язаною в одному взаємозв’язку, та головною в іншому. Щоб встановити взаємозв’язок таблиць потрібно зв’язати ключ головної таблиці з співпадаючим полем у зв’язаній таблиці (зовнішнім ключем).

Рідше зустрічається відношення «один-до-одного» (one-to-one). В цьому випадку одному запису в головній таблиці відповідає один запис у зв’язаній таблиці.

Відношення «багато-до-багатьох» передбачає, що кожному запису в одній таблиці відповідає декілька записів в іншій. При цьому кожна сторона відношень виглядає як відношення один-до-багатьох. Але якщо розглядати взаємозв’язок таблиць з обох сторін, стає очевидним, що ні одна з таблиць не може бути головною й для їх зв’язування необхідна третя таблиця.

Існує два основних способи зв’язувати дані: це за допомогою полів підстановки (Lookup Wizard…) та шляхом визначення зв’язків у діалоговому вікні Схема даних (якщо розглядати це в Accass).

1). Метод підстановок. Встановлення зв'язків розглянемо на прикладі двох таблиць. Вміст таблиць **Список предметів** та **Консультації** на рисунку 3.1.

Необхідно, наприклад поле **Предмет** з таблиці **Список предметів** підставити у таблицю **Консультації**. Для цього необхідно відкрити структуру таблиці **Консультації** у режимі конструктора. У цій структурі можна створити ще одне поле з потрібним відповідним ім’ям. В цьому випадку це ім'я **Предмет**.

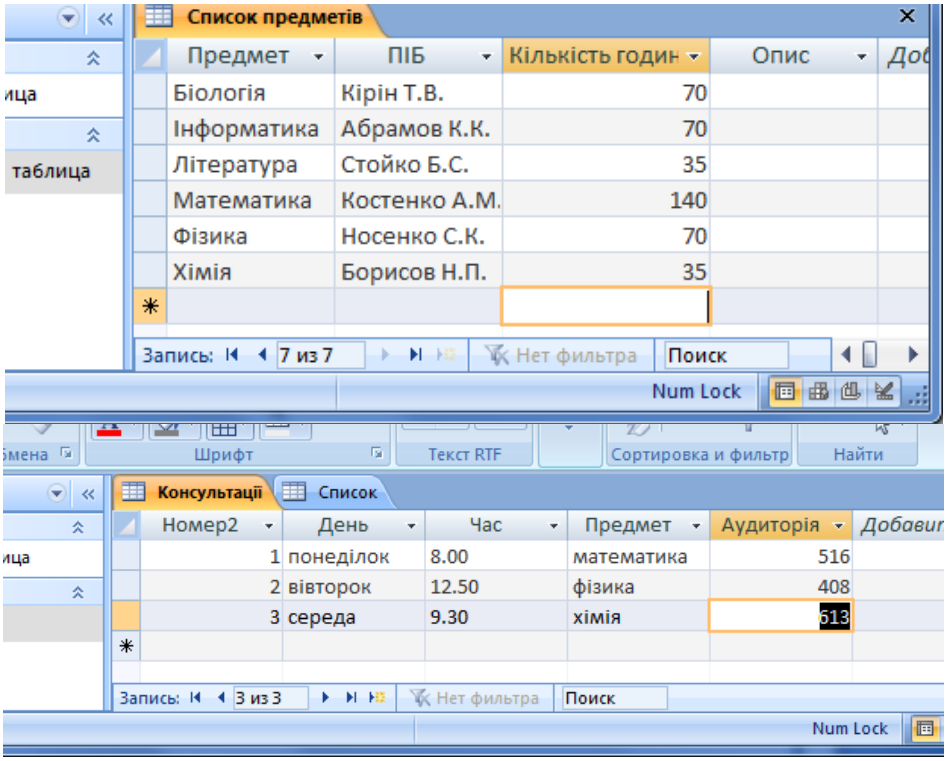


Рисунок 3.1 - Таблиці

Далі відкриємо значення **Тип даних**, де останнім у списку вибирається Майстер підстановок. Після активізації програми майстра на екрані відкривається вікно. У ньому вибираємо щоб о**б'єкт «стовбець підстановки» вибирав значення з таблиці** та притримуємось подальших інструкцій**.** Вибрали таблицю Список предметів.

У наступному необхідно виділити те поле, над яким здійснюється операція підстановки. У цьому випадку це поле – **Предмет** і стрілкою вибирається це поле. Після всіх дій треба закрити вікно структури таблиці **Консультації** і відкрити її для перегляду. Клацніть кнопкою миші в позиції першого рядка поля **Предмет** і розкрийте список його значень. Вікно набуде вигляду, що зображено на рисунку 3.2.

Переконайтесь у тому, що у списку зберігаються всі значення поля **Предмет** зв'язаної таблиці **Список предметів.** Тепер кожне з цих значень може бути перенесено у поточну комірку поля **Предмет.**

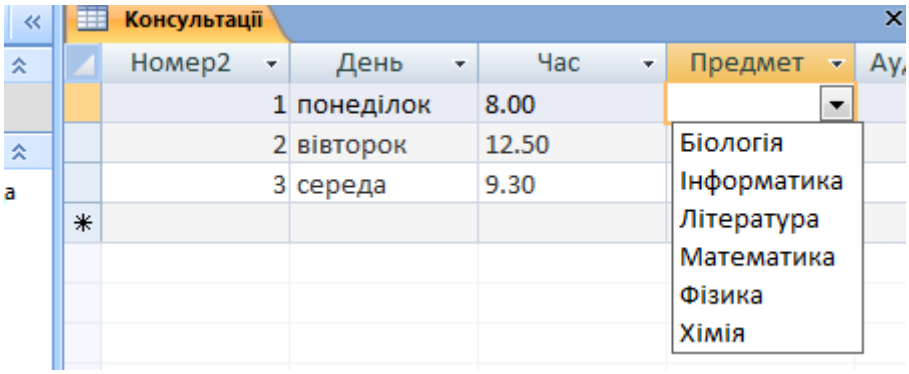


Рисунок 3.2 - Результат роботи підстановки

Таким чином, поле **Предмет** приєднано до таблиці **Консультації.** Система Access дозволяє приєднати до іншої таблиці не тільки одне поле, але й декілька. Методика виконання такої операції аналогічна розглянутій вище.

Щоб переконатися в наявності зв'язку між таблицями, треба відкрити **Схему даних** (Relationships) з вкладки робота з БД. Зв’язок між таблицею **Список предметів** і таблицею **Консультації** буде зображено лінією зв’язку, наприклад рисунок 3.3.

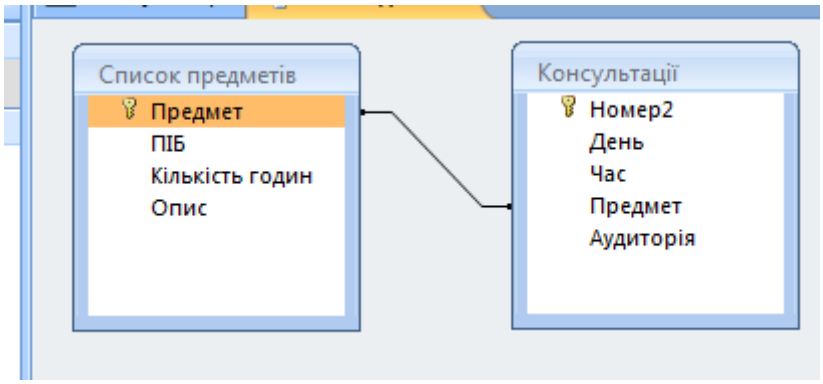


Рисунок 3.3 – Зв’язок зображено лінією зв’язку

В вікні схеми даних щоб прибрати зв'язок між таблицями, необхідно клацнути лінію, яка з'єднує таблиці і в контекстному меню, яке з’явиться необхідно виконати команду видалити**.** Якщо збереженого макету схеми даних немає, можливо вибрати з вкладки конструктор або з контекстного меню додати таблиці, що дозволяє додати таблиці або запити в вікні схеми даних. У вікні Relationships також можна створювати відношення та змінювати. Для зв'язування полів треба вибрати поле в одній таблиці й перетягнути його за допомогою миші на відповідне поле в іншій таблиці. Зв'язані поля не обов'язково повинні мати однакові імена, однак, вони повинні мати однакові

типи даних і мати зміст одного типу. Крім того, поля, які зв’язують, числового типу повинні мати однакові значення властивості розмір поля.

В більшості випадків зв'язують ключове поле (наведене в списку полів напівжирним шрифтом) однієї таблиці з відповідним йому полем (яке часто має те ж саме ім'я), що називають полем зовнішнього ключа в іншій таблиці.

За допомогою вікна схеми даних можна задавати як прості відношення, так і складні взаємозв’язки особливо в таких випадках:

- Якщо треба встановити цілісність даних;
- Якщо ключ вміщує більше ніж одне поле;
- Якщо немає прямого зв’язку між двома таблицями.

Для прикладу візьмемо таблиці: Доставка, яка містить дані про тарифи на перевезення товару та Транспортні фірми, яка містить дані про фірми, які займаються перевезенням. Вартість перевезення залежить від транспортної фірми та пункту перевезення. Оскільки кожна фірма виконує перевезення у різні регіони, то одній фірмі відповідають декілька тарифів, тобто між таблицями зв’язок “один-до-багатьох”.

Відкриємо Схему даних. Додаємо таблиці Транспортні фірми та Доставка (рисунок 3.4).

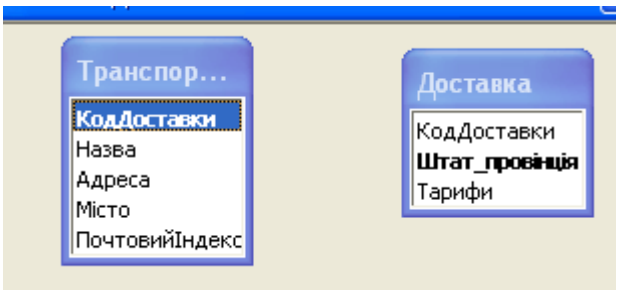


Рисунок 3.4 – Таблиці на схемі

Взаємозв’язок між таблицями Транспортні фірми та Доставка зв’яже інформацію про транспортні фірми з даними про тарифи на перевезення. Для цього необхідно перетягнути поле Код доставки з таблиці Транспортні фірми в таблицю Доставка. Напрямок перетягування ключа визначає, яка таблиця буде головною, а яка підлеглою. Таблиця, звідки переміщується ключове поле, є головною. Таблиця, куди воно переміщується - зв’язаною. З’являється діалогове вікно Зміна зв’язку із значенням “один-до-багатьох” в полі **Тип відношення**.

Якщо у діалоговому вікні Зміна зв'язку встановити прапорець в полі Забезпечення цілісності даних, то цілісність не допускає появи незв'язаних записів. Таблиця Транспорті фірми тепер буде пов'язана з таблицею Доставка, поля обох таблиць поєднані лінією, поміченою цифрою 1 і символом нескінченності, що вказує про відношення "один-до-багатьох" (Рисунок 3.5).

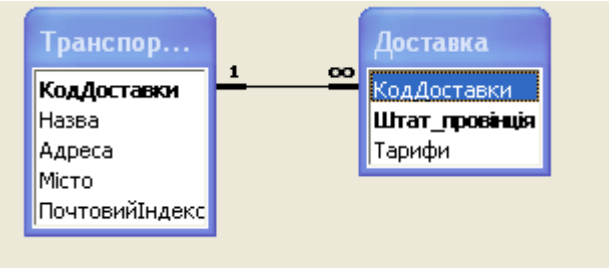


Рисунок 3.5 – Встановлено забезпечення цілісності

Встановлення забезпечення цілісності даних представляє собою в дії систему правил, які виключають вільну зміну зв'язаних записів. Таблиці пов'язані та зміни в одній з таблиць впливають на дані в іншій. Цілісність даних накладає наступні обмеження на введення та корективи інформації.

- Не можна вводити значення у зовнішній ключ зв'язаної таблиці, якщо не існує співпадаючого значення у ключовому полі головної таблиці. Наприклад, ви не зможете додати запис у таблицю **Замовлення**, вказавши **Код замовника**, якого немає в таблиці **Замовники** (не можна ввести замовлення для неіснуючого замовника).
- Забороняється видаляти записи в головній таблиці, якщо існують відповідні записи в зв'язаній таблиці. Не можна, наприклад, видаляти запис в таблиці **Замовники**, якщо їй відповідають записи в таблиці **Замовлення** (не можна видалити замовника, у якого є замовлення).
- Не можна змінити значення ключового поля в головній таблиці, якщо в зв'язаній таблиці є записи, які посилаються на це значення. Так, наприклад, вам не вдасться змінити **Код замовника** в таблиці **Замовники**, якщо в таблиці **Замовлення** існують записи для цього замовника.

Цілісність даних можна забезпечити з виконанням наступних умов.

- Співпадаюче поле в головній таблиці повинно бути ключовим.

- Всі значення зовнішнього ключа зв’язаної таблиці повинні бути присутніми в ключовому полі головної таблиці.
- Зв‘язані поля в обох таблицях повинні мати однаковий розмір і тип даних.
- Обидві таблиці повинні належати одній базі даних Access. Якщо зв’язані таблиці знаходяться в різних файлах, вони повинні мати формат Microsoft Access, а база даних, у якій зберігаються зв’язані таблиці, повинна бути відкритою.

Каскадне відновлення та видалення даних

В Access існує можливість обійти обмеження на зміну ключових полів і видалення даних у зв’язаних таблицях не порушуючи цілісність даних.

Каскадне відновлення забезпечує розповсюдження змін в головній таблиці на відповідні записи в зв’язаній таблиці. Так, наприклад, зі зміною коду замовника в таблиці **Замовники** відбудеться відновлення співпадаючого поля в таблиці **Замовлення**, і зв’язок між таблицями збережеться.

Якщо встановити каскадне видалення, з видаленням запису в головній таблиці видаляються і відповідні йому записи в зв’язаних таблицях. Наприклад, видалення замовника з таблиці **Замовники** включає видалення всіх його замовлень з таблиці **Замовлення**, а також видалення даних про склад замовлень в таблиці **Подробиці замовлень**. Тому що в цьому випадку видаляться і не виконані замовлення, каскадне видалення слід використовувати з обережністю.

Відношення багато-до-багатьох

У відношенні **багато-до-багатьох** є проміжна таблиця, яка служить мостом між двома таблицями у відношеннях багато-до-багатьох. Окрім ключових полів, зв’язуюча таблиця повинна мати хоча б одне поле, якого немає у зв’язуємих таблицях, але яке має значення для кожної з них. Таким чином, відношення багато-до-багатьох складається з відношень багато-до-одного та один-до-багатьох.

На прикладі розглядається, як за допомогою таблиці Подробиці реалізувати відношення багато-до-багатьох для таблиць Подарунки та Цукерки (рисунок 3.6).

Між таблицями Подарунки та Подробиці існує відношення один-до-багатьох через ключове поле Код подарунка. Крім цього таблиця Подробиці з таблицею Цукерки теж зв’язані відношенням багато-до-одого. Таблиця Подробиці виступає зв’язуючою між таблицями Подарунки та Цукерки (рисунок 3.6).

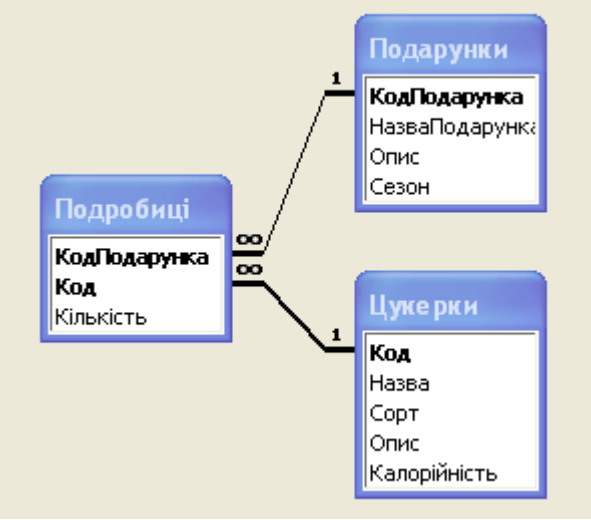


Рисунок 3.6 – Відношення «багато-до-багатьох»

Застосування відношення «один до одного» розглядається у випадках коли дані однієї таблиці поділяють на дві таблиці та більше з метою покращення продуктивності через частіше використання певної інформації або безпеки окремих даних.

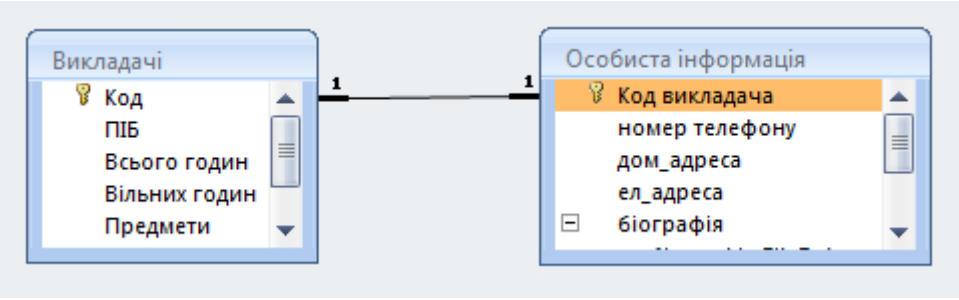


Рисунок 3.7 – Відношення «один-до-одного» на схемі

Приклад схеми даних із зв’язком «один до одного» зображено на рисунку 3.7. Як видно особиста інформація виділяється в окрему таблицю, що дає можливість мати доступ до неї обмеженій кількості персоналу.

Контрольні запитання до лабораторної роботи 3:

1. Які типи відношень (міжтабличних зв’язків) існують?
2. Які типи відношень можна встановити між двома таблицями безпосередньо?
3. Яким чином можна визначити зв’язки між таблицями?
4. Як виглядають зв’язані таблиці на Схемі даних?
5. Що означає визначення відношення один-до-багатьох?
6. Що означає визначення відношення один-до-одного?
7. Що означає визначення відношення багато-до-багатьох?
8. Як створити відношення багато-до-багатьох?
9. Якщо існує взаємозв’язок між таблицями, яку таблицю називають головною, а яку підлеглою?
10. Що таке цілісність даних?
11. Які обмеження на введення та корективи інформації накладає цілісність?
12. Що таке каскадне відновлення?
13. Що таке каскадне видалення?

Лабораторна робота № 4

Тема: Функції для маніпулювання даними.

Мета: Глибше ознаомитись з прийомами та синтаксисом SQL запитів, застосовуючи вбудовані функції та оператори CASE та LIKE.

Завдання:

1. Створити запит, що повертає значення тільки назви об’єктів(в залежності від теми БД), що були додані у базу більше як 5 років з дати внесення.
2. За допомогою запиту вивести таблицю, що складається з текстового поля ПІБ та відповідного окремо імені об’єкту.
3. Вивести запитом певні події чи назви об’єктів, що відповідають у полі дати високосному року.
4. Побудувати запит, що виводить значення поля назва об’єкта (персона, назва товару або ін.) таблиці та кількість років від дати створення чи народження об’єкта.
5. Створити запит, що виводить з таблиці, де є поле з адресою, тільки ті записи, де назва міста починається на «К» та має в назві кількість символів 4.

Теоретичні відомості:

Є ряд операцій, що виконуються над даними, як правило, для зручнішого опрацювання та модифікації інформації до бажаного вигляду завдяки функціям маніпулювання.

Функції - це фрагменти коду, які виконують деякі операції, а потім повертають результат. Функції дозволяють розширити можливості MySQL.

Реалізації SQL найчастіше розглядають наступні типи вбудованих функцій [5]:

• функції, що маніпулюють текстовими даними (виокремлення частин тексту, перетворення рядку в верхній чи нижній регістр та ін.);

• числові функції використовуються в математичних операціях, для обробки числових даних (знаходження модуля, виконання різних

арифметичних операцій);

• функції дати й часу, використовуються для обробки даних, що мають тип date, time, datetime, як правило, для виділення окремих елементів з цілого значення;

• системні функції видають специфічну інформацію, яка використовується в потрібній конкретній СКБД (наприклад, повертають ім'я поточного користувача СКБД, або інформацію про версію СКБД).

Функції для роботи з текстом

Розглянемо такий запит:

```
SELECT Name,   UPPER(Name) as Name_big  from students
ORDER BY Name_big
```

Результат виконання запиту розглянуто в таблиці 4.1.

Таблиця 4.1 – Результат запиту з Upper()

Name	Name_big
Шевченко Михайло Петрович	ШЕВЧЕНКО МИХАЙЛО ПЕТРОВИЧ
Іванців Лариса Львівна	ІВАНЦІВ ЛАРИСА ЛЬВІВНА
...	...

У прикладі видно, що функція Upper() конвертувала текст у верхній регістр, ім'я студента виведено двічі, перший раз - як поле збережене в таблиці Students, другий раз - перетворене функцією Upper, як поле Name_big.

Найчастіше використовують такі функції для роботи з текстом, що розглядаються в таблиці 4.2.

Таблиця 4.1 – Функції для роботи з текстом

Left()	Повертає вказану кількість символів стрічки, рахуючи зліва
Length()	Повертає довжину стрічки
Locate()	Шукає позицію однієї стрічки в іншій
Lower()	Перетворює символи стрічки в нижній регістр
LTrim()	Обрізає пробіли зліва
Right()	Повертає вказану кількість символів стрічки, рахуючи справа
RTrim()	Обрізає пробіли справа

SubString()	Повертає вирізану частину стрічки
Upper()	Перетворює символи стрічки у верхній регістр

Функції для роботи з датою та часом

Для зберігання дати й часу використовують специфічний внутрішній формат зберігання даних такого типу. Цей формат дозволяє компактніше розташовувати дані в таблиці та ефективніше їх сортувати й фільтрувати.

Найчастіше використовують функції роботи з датою й часом, що представлено в таблиці 4.2.

Таблиця 4.2 – Функції для роботи з датою й часом

AddDate()	Додавання до дати (дні, тижні тощо)
AddTime()	Додавання до часу (години, хвилини тощо)
CurDate()	Повертається поточна дата
CurTime()	Повертається поточний час
Date()	Повертається дата з даних типу datetime
DateDiff()	Обчислює різницю між двома датами
Date_Add()	Гнучка функція для додавання дат
Date_Format()	Повертає відформатовану стрічку дати чи часу
Day()	Повертає день з дати
DayOfWeek()	Повертає день тижня
Hour()	Повертає годину з часу
Minute()	Повертає хвилину з часу
Month()	Повертає місяць з дати

Now()	Видає поточну дату і час (в форматі datetime)
Second()	Повертає секунди з часу
Time()	Повертає час з даних типу datetime
Year()	Повертає рік з дати

При роботі з датою, особливо при фільтрації даних оператором WHERE, найчастіше використовують специфічні функції, розглянуті вище. При роботі з датою найперше, що слід знати, - це формат дати, що використовує **MySQL**. Кожного разу, коли необхідно вказувати конкретне значення дати при заповненні чи модифікації таблиць, дата повинна подаватися у форматі рррр-мм-дд. Так, *перше вересня 2008 року* виглядатиме *2008-09-01*. Можливо інтерпретатор розпізнав би й інші варіанти, але варто віддати перевагу запропонованому способу, тому що це виключає двозначність (наприклад, *01/09/08* можна сприйняти і як *9-те січня 2008 р*. і як *8-те вересня 2001 р*. тощо).

Роки, вказані за допомогою двох цифр, в **MySQL** теж підтримуються. Інтерпретатор сприймає *00- 69* як *2000-2069* і *70-99* як *1970-1999*. Але щоб уникнути непорозумінь і конфліктів з новішими версіями, бажано вказувати роки в чотирицифровому вигляді.

Числові функції

В таблиці 4.3 наводяться функції, що найчастіше використовуються для роботи з числами.

Таблиця 4.3 – Функції для роботи з числами

Abs()	Повертає модуль числа
Cos()	Повертає косинус вказаного кута
Exp()	Повертає для вказаного x значення e^x
Mod ()	Повертає остачу від ділення двох чисел
Pi()	Повертає число π

Rand()	Повертає випадкове число
Sin()	Повертає синус вказаного кута
SqrtO	Повертає корінь вказаного числа
Tan()	Повертає тангенс вказаного кута

Збережені функції

Збережені функції подібні до вбудованих, за винятком того, що визначати збережену функцію необхідно самостійно. Після створення збереженої функції її можна використовувати в операторах SQL, як і будь-яку іншу функцію.

Основний синтаксис для створення збереженої функції такий:

```
CREATE FUNCTION sf_name ([parameter(s)])
   RETURNS data type
   DETERMINISTIC
   STATEMENTS
```

- "CREATE FUNCTION sf_name ([parameter(s)]) " є обов'язковим і розповідає MySQL сервер для створення функції під назвою `sf_name' з необов'язковими параметрами, визначеними в дужках.
- «Тип даних RETURNS» є обов'язковим і вказує тип даних, який має повернути функція.
- **«**DETERMINISTIC**»** означає, що функція поверне однакові значення, якщо їй надано однакові аргументи.
- **«**STATEMENTS**»** це процедурний код, який виконує функція.

Визначені користувачем функції

MySQL також підтримує визначені користувачем функції, які розширюються MySQL. Визначені користувачем функції – це функції, які можна створити за допомогою мови програмування, наприклад C, C++ тощо, а потім додати їх до MySQL сервер. Після додавання їх можна використовувати як будь-яку іншу функцію.

Оператор Like

Оператор Like застосовують тільки до полів текстового типу, щоб знаходити підрядки, тобто відшукується у текстовому полі співпадіння з

умовою його вміст. Wildkards – спеціальні групові символи, які можуть відповідати, як умова для оператора.

Є символи двох типів:

- символ підкреслення (_), що заміняє один будь-який символ;

- знак відсотку (%), що замінює послідовність будь-якої кількості символів (і 0 також).

Наприклад, є необхідність вивести всіх абонентів, прізвище яких починаються на «Смір»:

```
…SELECT * from Phone Where Abonent Like ‘Смір%’…
```

Оператор CASE

Часто виникає необхідність при побудові вибірки аналізувати значення полів і виводити в деякому стовпці одне значення при певній умові та інше, коли ця умова не виконується. Ось, наприклад, потрібно зробити вибірку з таблиці студентів Students, яка має містити два стовпці: "ім'я студента" (Name) та "форма навчання" (Form), де має бути слово "платна" коли відповідне поле paid в таблиці Students дорівнює 1 або "державна" - коли 0. Потрібну вибірку зручно робити, використавши функцію CASE. Конструкція CASE обчислює та перевіряє вираз, а також порівнює результат з одним зі значень, вказаним у конструкції WHEN. Якщо знайдено збіг, то повертається відповідне значення конструкції THEN. У протилежному випадку повертається значення за замовчуванням.

```
SELECT Name,
(CASE (paid) WHEN 1 THEN 'платна' WHEN 0 THEN 'державна'
ELSE 'невідома' END) as form FROM students
```

Синтаксис CASE є таким:

```
CASE вираз WHEN значення виразу THEN результат, що повертається
[WHEN [значення виразу] THEN результат, що повертається ... ]
[ELSE результат що, повертається за замовчуванням при невиконанні жодної з
умов]
END
```

Або альтернативний варіант:

```
CASE WHEN [умова1] THEN результат, що повертається1
[WHEN [умова2] THEN результат, що повертається2 ...]
[ELSE результат, що повертається при невиконанні жодної з умов]
END
```

Контрольні запитання до лабораторної роботи 4:

1. Що являють собою запити і для чого їх використовують?

2. До якого типу полів застосовують оператор Like?
3. Що представляє собою функція у MySQL?
4. Які дії можливі з функціями для роботи з датою і часом?
5. Які дії можливі з функціями для роботи з числами?
6. Які дії можливі з функціями для роботи з текстом?
7. У яких випадках використовують конструкцію CASE в запитах?
8. Що повертає функція Length()?
9. Що повертає функція RTrim()?
10. Що повертає функція Day()?
11. Як отримати випадкове число у запиті?
12. Які функції називають збереженими?
13. Які функції в MySQL називають визначеними користувачем?

Лабораторна робота №5

Тема: Робота з представленнями.

Мета: Навчитися використовувати механізм представлень при роботі з БД.

Завдання:

1. Створіть представлення для будь-якоі таблиці БД, візьміть два поля, одне текстове, а інше числового типу поле, значення якого більше чи дорівнюють 500.
2. Створіть представлення для таблиці об’єктів вашої БД (клієнти, працівники, постачальники тощо), де замість поля ПІБ буде два поля окремо (прізвище та ім’я), а поля адреса і телефон об’єднані в одне.
3. Створіть представлення marriedstat для БД, яке показує кількість одружених об’єктів (студентів, клієнтів тощо). У представленні повинно бути три поля. Код, Назва підприємства і Кількість одружених (назви полів у кожного свої). Застосовувати агрегатні функції та сортування.
4. Змінити будь-яке представлення своєї БД.
5. Видалити будь-яке представлення своєї БД.

Теоретичні відомості:

Механізм представлень (view) є потужним інструментом СКБД, що дозволяє приховати реальну структуру БД від небажаних користувачів за рахунок визначення представлень.

Представлення або ще називають подання (views) – це віртуальні таблиці. На відміну від звичайних таблиць, що містять дані, представлення є деяким запитом, що зберігається в БД, а для користувача не відрізняється за виглядом від базового відношення.

Подання також допомагає спростити запити, оскільки він пропонує менш складну версію схеми. Наприклад, розробнику додатків не потрібно створювати деталізовані багатотабличні з'єднання, натомість він може звернутися до подання в базовому операторі SELECT. Крім того, можливість представляти

абстрактну схему дозволяє модифікувати визначення таблиць, що лежать в основі подання, без порушення роботи додатків.

Є низка обмежень при роботі з представленнями. Наприклад, в MySQL не дозволяється створювати індекс на поданні, визначати на поданні тригер або посилатися на системну або визначену користувачем змінну в запиті подання.

Докладніше про обмеження в документації: https://dev.mysql.com/doc/refman/8.0/en/create-view.html.

Для кращого розуміння доцільності використання представлень розглядається приклад запиту в базі даних про студентів ЗВО. Необхідно визначити кількість студентів, що навчаються на платній формі за спеціальністю «Кібербезпека», наприклад [5]:

```
Select sp.Name as Name, count(*) as cnt From students as st
Left join speciality as sp on (sp.ID=st.speciality_ID)
Where st.paid=1 and sp.Name Like‘%Кібербезпека%’
Group by sp.Name
```

Для зміни умови спеціальності необхідно розбиратись у операторах і знати структуру БД. А якщо можна було б виділити загальний запит, без умови спеціальності, окремо у таблицю spec_paid, то завдання виглядає простішим:

```
Select * from spec_paid
Where Name Like‘%Кібербезпека%’
```

Саме в таких випадках і доречно використовувати представлення. У прикладі spec_paid – є представленням, і тому не має реальних даних чи стовпців. Натомість має запит, який виконується при звертанні до spec_paid.

Є правила, яких слід дотримуватись при застосуванні представлень:

- Щоб створити представлення необхідно мати відповідні права;

- Імена представлень, як і таблиць, повинні бути унікальні в БД;

- Представлення можуть бути вкладені, тобто запит, що в представленні, може брати дані з іншого представлення;

- Представлення можуть використовуватись разом з таблицями.

Конструкції для роботи з представленнями:

CREATE VIEW – створити представлення;

DROP VIEW – для знищення представлення;

CREATE OR REPLACE VIEW – застосовують щоб змінити представлення;

ALTER VIEW - для оновлення визначення представлення.

Представлення буде створено, якщо його ще немає, або заміниться, якщо представлення з даним іменем вже існує.

Синтаксис конструкції **CREATE VIEW**:

```
CREATE [OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

Тут view_name - ім'я створюваного уявлення; select_statement - оператор SELECT, який вибирає дані з таблиць та/або інших представлень, які будуть утримуватися в поданні.

Оператор CREATE VIEW містить 4 необов'язкові конструкції:

- OR REPLACE - при використанні даної конструкції у разі існування представлення з таким ім'ям старе буде видалено, а нове створено.

- ALGORITM - визначає алгоритм, що використовується при зверненні до подання.

- column_list - задає імена полів уявлення.

- WITH CHECK OPTION - при використанні всі рядки, що додаються або змінюються, перевірятимуться на відповідність визначенню представлення. У разі невідповідності ця зміна не буде виконана. Зверніть увагу, що при вказівці даної конструкції для поновлення подання виникне помилка і подання не буде створено.

Простий приклад створення представлення usersp1:

```
CREATE OR REPLACE VIEW usersp1 AS Select * from users
```

Контрольні запитання до лабораторної роботи 5:

1. Що називають представленням в БД?
2. Чим відрізняється представлення від таблиці БД?
3. Що означає конструкція CREATE VIEW?
4. Що означає конструкція DROP VIEW?
5. Що означає конструкція ALTERVIEW?

6. Яким чином можна створити представлення в БД?
7. Яким чином можна змінити представлення в БД?
8. Яким чином можна знищити представлення в БД?
9. Що відбувається при використанні OR REPLACE в конструкції Create VIEW?
10. Які є обмеження при роботі з представленнями в БД?
11. Які правила використання представлень в БД?

Лабораторна робота № 6.

Тема: Застосування індексів в БД.

Мета: Навчитися створювати індекси, розуміти вплив індексів на швидкодію сортування даних під час виконання запитів.

Завдання:

1. Створити таблицю з даними про клієнтів (працівників, постачальників), додати 10 записів.
2. Виконати два запити пошуку даних з предікатом для числового поля і для текстового, відмітити час виконання запитів.

```
SELECT * FROM tbl_name WHERE col1=95467 or col1=66467;
SELECT * FROM tbl_name WHERE txt_col LIKE “Cmir%”;
```

3. Створити індекси на відповідні поля в таблиці.
4. Виконати схожі два запити пошуку даних (завдання 2) з предікатом для числового поля і для текстового, відмітити час виконання запитів.
5. Сформуйте порівняльну таблицю зі значеннями швидкостей виконання запитів з таблиці без індексів і з таблиці з індексами. Майте на увазі, щоб при багаторазовому виконанні однакового запиту здійснювався реальний пошук з таблиці, необхідно перед виконанням очищати кеш командою RESET QUERY CACHE.
6. Створіть унікальний індекс в таблиці на три поля.
7. Видаліть непотрібний індекс.

Теоретичні відомості:

Індекси – це корисний інструмент, ним прийнято називати впорядкований список полів чи груп полів в таблиці [1]. Часто реальні таблиці мають дуже велику кількість записів, при тому ж не мають визначеного порядку, тому пошук певних даних може зайняти досить багато часу.

Якщо створюється індекс в полі, то БД запам’ятовує порядок всіх значень в області пам’яті, системі легше знайти потрібне значення в впорядкованому індексі, та дать інформацію, як знайти потрібний рядок таблиці.

Є і недоліки індексів. Застосування індексів сповільнює операції модифікації , особливо INSERT, DELETE. Крім того сам індекс займає місце теж. Треба приймати виважене рішення щодо створення індексів.

У яких випадках створюють індекси [5]:

- щоб швидко знайти значення, що відповідають виразу WHERE;

- щоб знайти рядки з інших таблиць при об'єднанні JOIN;

- для знаходження MAX(), MIN() для заданого індексованого стовпця;

- щоб здійснювати сортування або групування в таблиці, якщо ці операції виконуються на крайньому ліворуч префіксі ключа.

Існують такі типи індексів:

- Неунікальний (Non-unique index) - індекс, у якому значення можуть повторюватися.

- Унікальний (Unique index) - всі значення зустрічаються лише один раз. Якщо у стовпець з унікальним індексом намагатись додати неунікальне значення, видається помилка.

- Простий (Simple index) – індекс, що складається з одного поля.

- Складний (Composite Index) - індекс, що будується з кількох стовпців таблиці. У цьому типі індексу розташування полів є важливим.

- Деревоподібний (B-tree index) – індекс, який представлений у вигляді кореневої вершини та вузлів.

- Частковий (Partial Indexes) - індекс, що складається з підмножин рядків таблиці за певним виразом.

- Кластеризований індекс – це індекс, який сортує рядки з даними у таблиці. Кластеризований індекс зберігає дані у листі індексу.

- Некластеризований індекс – це індекс, який використовується для застосування індексів до неключових стовпців. Головна відмінність від кластеризованого індексу у тому, що некластеризований індекс не впорядковує дані фізично. Некластеризований індекс зберігає дані та індекси у різних місцях.

Індекси створюють за допомогою **CREATE.** Для видалення індексів застосовуємо **DROP**.

Синтаксис команди для створення індексу:

CREATE INDEX <Ім’я індексу> ON < Ім’я таблиці> (<назва поля1> [,<назва поля2>]…);

Приклад команда для створення індексу за полем, яке зберігає прізвище студента:

```
…CREATE INDEX SPRIIDX ON STUDENTS (SPRI);…
```

Для створення унікальних індексів використовується ключове слово **UNIQUE** у команді CREATE INDEX. Фактично такий індекс буде первинним ключем таблиці.

Для створення унікальних індексів можна зробити дії:

```
…CREATE UNIQUE INDEX
SNAMIDX ON STUDENTS (SNAM);…
```

Ця команда не виконається, якщо в полі SNAM трапляться неунікальні значення. Рекомендується створювати індекси до введення в таблиці даних.

Можливо в індексі застосовувати не одне поле, а комбінацію значень.

Приклад для видалення створеного індексу за прізвищем студента, можна скористатись командою:

```
…DROP INDEX SPRIIDX
ON STUDENTS;…
```

Видалення індексу ніяк не впливає на дані , що зберігаються в цих полях.

У MySQL не можна використати частковий індекс, якщо стовпці не утворять крайній ліворуч префікс цього індексу. Наприклад якщо розглянути запити:

Запит1: `SELECT * FROM tbl_name WHERE col1=val1;`

Запит2: `SELECT * FROM tbl_name WHERE col2=val2;`

Запит3: `SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;`

Якщо індекс існує по (col1, col2, col3), то тільки запит 1 використає цей індекс. Другий і третій, хоч і включають індексовані стовпці, але (col2) та (col2, col3) не є крайньою ліворуч частиною префіксів (col1, col2, col3).

MySQL застосовує індекси також для порівнянь LIKE, якщо аргумент у виразі LIKE є статичним рядом, що не починається із символу шаблона. Наприклад, такі запити з команди SELECT використовують індекси:

```
SELECT * FROM tbl_name WHERE key_col LIKE “Patrick%”;
SELECT * FROM tbl_name WHERE key_col LIKE “Pat%_ck%”;
```

У першому запиті розглядатимуться тільки записи з “Patrick» <= key_col<“Patrick», а в другій – тільки рядки з “Pat» <= key_col <“Pau»,

А наступні запити не будуть враховувати індекси.

```
SELECT * FROM tbl_name WHERE key_col LIKE “%Patrick%”;
SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

В першому запиті параметр Like починається з шаблонного символу, а у другій команді цей параметр не є константою.

Якщо індекс створюється на текстовому полі, то важливо вдало підібрати довжину індексу. Довжина індексу визначає кількість початкових символів у рядку по яких відбувається індексація. Більша довжина може спричинити збільшення розміру індексу, а отже і бази даних, а індекси недостатньої довжини будуть неефективно використовуватись в тих запитах, де довжина рядку-константи, що в запиті, перевищує довжину індексу.

Якщо в таблиці визначене поле первинного ключа, то генерується індекс, і немає необхідності визначати звичайний індекс по цьому полю. Як вже відомо, створення зайвих індексів може спричинити сповільнення роботи в БД.

Краще дотримуватися наступних **рекомендацій** під час роботи з індексами:

- Не застосовувати індекси у невеликих таблицях.

- Не застосовувати індекси в таблицях, які часто піддаються додаванню або зміні нових даних.

- Не застосовувати індекс з тими стовпцями БД, з якими будуть часті дії (наприклад, виконання складних запитів на вибірку даних).

- Не застосовувати індекси в стовпцях зі значенням NULL

- Індекси бажано застосовувати до тих стовпців, якими найчастіше ведеться пошук. Планувати застосування індексів потрібно не раніше, ніж коли в таблиці з'явиться щонайменше 10000 тис. записів.

Інакше помітного приросту швидкості обробки даних не буде.

Контрольні запитання до лабораторної роботи 6:

1. Що таке індекси?
2. В яких випадках застосовують індекси?

3. Які є недоліки у використанні індексів?
4. Якою командою створюють індекси?
5. Яких видів бувають індекси?
6. Коли не можна створити унікальний індекс в таблиці з даними?
7. Яким чином видаляють індекс?
8. Яких рекомендацій треба притримуватись по роботі з індексами?

Лабораторна робота № 7.

Тема: Робота з семантичними помилками у запитах БД.

Мета: Навчитися визначати найвідоміші семантичні помилки та виправляти запити.

Завдання:

1. Створити в БД (свій варіант) ситуацію згідно запиту, що на рисунку 7.1, визначити та виправити семантичну помилку в новому запиті.
2. Створити в БД (свій варіант) ситуацію згідно запиту, що на рисунку 7.3, визначити та виправити семантичну помилку в новому запиті.
3. Створити в БД (свій варіант) ситуацію згідно запиту, що на рисунку 7.4, визначити та виправити семантичну помилку в новому запиті.
4. Створити в БД (свій варіант) ситуацію згідно запиту, що на рисунку 7.5, визначити та виправити семантичну помилку в новому запиті.
5. Створити в БД (свій варіант) ситуацію згідно запиту, що на рисунку 7.6, визначити та виправити семантичну помилку в новому запиті.
6. Створити в БД (свій варіант) ситуацію згідно запиту, що на рисунку 7.8, визначити та виправити семантичну помилку в новому запиті.

Теоретичні відомості:

Хоча SQL-механізми здатні виявляти значну кількість синтаксичних помилок, зустрічаються семантичні помилки, що можуть призвести до серйозних проблем з продуктивністю програмного забезпечення або навіть до вразливостей у безпеці системи [36].

Семантична помилка означає, що було введено допустимий SQL-запит, але запит не завжди дає очікувані результати і, отже, невірний для цієї задачі.

Семантичні помилки часто з’являються в запитах новачків, що не оволоділи мовою SQL достатньо, до того ж можуть зустрічатись в реальних програмах, що робить їх більш небезпечними. Оскільки ці проблеми зазвичай не генерують жодних попереджень від механізмів БД, тому недосвідченим розробникам важче їх виявити, особливо через те, що в деяких випадках запити

можуть справді давати правильні результати. Саме з цієї причини можна стверджувати, що семантичні помилки в SQL є більш небезпечними, ніж синтаксичні помилки.

Крім того, семантичні помилки часто можуть впливати на загальну продуктивність запиту, що призводить до збільшення часу обчислення або неефективного використання ресурсів. Це знову ж таки дуже важливо в програмах, де швидкість є критичною, і виконання запитів, які містять семантичні помилки, може призвести до вузьких місць.

У таблиці 7.1 перелік деяких семантичних помилок, що можуть трапитись у запитах. Кожна з них розглядається окремо детальніше.

Таблиця 7.1 – Семантичні помилки в запитах [36]

Список семантичних помилок

Семантична помилка	Опис
Порівнювання з NULL	Використання звичайних операторів порівнювання з NULL замість виразу IS NULL
Зайвий DISTINCT в агрегаціях	Для деяких функцій агрегації ключове слово DISTINCT не впливає на результат
Ділення на нуль	Можливе ділення на нуль через порядок виконання операцій
Відсутня умова JOIN	Відсутнє посилання для об'єднання таблиць
Непотрібна команда GROUP BY	Зайвий атрибут у групі GROUP BY, який не відображається в групі SELECT або HAVING поза агрегаціями.
Неефективне використання команди HAVING	Умова всередині виразу HAVING, яку можна перемістити всередину виразу WHERE, що робить запит більш ефективним

Порівнювання з NULL

У деяких СКБД синтаксично допустимо використовувати A =NULL, однак цей вираз завжди має постійне значення істинності або NULL, або невідоме. Щоб уникнути таких ситуацій, завжди слід використовувати IS NULL або IS NOT NULL. Наступний приклад на рисунку 7.1 повинен бути виявлений як такий, що має семантичну помилку.

```
SELECT t1.id, t1.surname , t1.name FROM table t1
WHERE t1.phone <> NULL;
```

Рисунок 7.1 - Семантична помилка порівнювання з NULL

У цьому прикладі t.phone порівнюється з NULL. Для деяких механізмів баз даних це може не вважатися синтаксичною помилкою, однак умова поверне NULL або unknown. Щоб уникнути подібних ситуацій, коли використовується значення NULL, ми завжди повинні використовувати вирази IS NULL або IS

NOT NULL замість операторів порівняння. Потенційне виправлення для прикладу запиту наведено на рисунок 7.2.

```
SELECT t1.id, t1.surname, t1.name FROM table t1
WHERE t1.phone IS NOT NULL;
```

Рисунок 7.2 - Виправлення помилки порівнювання з NULL

Стратегія виявлення: щоб виявити цю семантичну помилку, щоразу, коли в запиті використовуються оператори рівний або не рівний, ми перевіряємо, чи є один із термінів ключовим словом NULL. Якщо це так, тоді оператор порівняння слід замінити виразом IS NULL або IS NOT NULL відповідно.

Зайвий DISTINCT в агрегаціях

Для функцій агрегування MIN і MAX ключове слово DISTINCT ніколи не потрібне, оскільки воно не впливає на базові результати запиту.

Крім того, у більшості випадків наявність дублікатів є, ймовірно, суттєвою при обчисленні результатів функцій агрегування SUM і AVG, тому дублікати не слід виключати, якщо для цього немає вагомих причин. Таким чином, присутність ключового слова DISTINCT у цих функціях агрегації є дивною і має викликати попередження. Для інших функцій агрегування неможливо визначити, чи потрібен DISTINCT у команді, не маючи додаткової інформації про завдання запиту, тому інші функції агрегування виключаються з цієї перевірки. У наступному прикладі на рисунку 7.3 слід виявити семантичну помилку.

```
SELECT SUM(DISTINCT price) AS income FROM
orders
```

Рисунок 7.3 - Семантична помилка з зайвим DISTINC в агрегації

У цьому прикладі ключове слово DISTINCT використовується всередині функції SUM. Якщо для цього немає вагомих причин, присутність тут DISTINCT вважається дивною, оскільки дублікати, швидше за все, значні в цьому випадку, однак, не маючи додаткової інформації про завдання, для якого був написаний запит, наш інструмент видасть попередження про потенційну семантична помилку. Не маючи додаткової інформації про основне завдання,

для якого був написаний запит, ми не можемо надати потенційне виправлення для цього запиту.

Стратегія виявлення: стратегія виявлення перевіряє, чи присутні в запиті будь-які функції агрегування MIN, MAX, SUM або AVG. Для кожного з них він потім визначає, чи використовувалося ключове слово DISTINCT у команді функції, і якщо так, виникає попередження.

Ділення на нуль

Однією з поширених проблем, пов'язаних із операторами типів даних, є ділення на нуль.

Оскільки в SQL немає гарантій на послідовність оцінки операторів у команді WHERE, розробникам важко уникнути таких ситуацій. Наступний приклад на рисунку 7.4 вважається небезпечним і має бути виявлений як семантична помилка

```
SELECT items FROM orders
WHERE (amount / count) > 500 AND count > 0;
```

Рисунок 7.4 - Семантична помилка з діленням на нуль

У цьому прикладі, хоча умова count > 0 присутня в запиті WHERE, оскільки порядок операцій не виконується, цей запит усе ще небезпечний. Таким чином, попередження повинно бути викликано, коли в запиті є ділення, за винятком ситуацій, коли ділення знаходиться в операторі SELECT, а стовпець дільника перевіряється на нерівність нулю в команді WHERE, оскільки в цих випадках WHERE буде оцінено перед SELECT будь-якою системою бази даних.

Стратегія виявлення: реалізація цього правила аналізує запит і зберігає всі знайдені оператори ділення. Окрім цього, також створюється логічний прапорець, який вказує, чи був термін поділу після SELECT чи ні. Крім того, усі стовпці, які перевіряються на нерівність нулю в команді WHERE, також зберігаються. Нарешті, ділення, яке знаходяться в запиті SELECT, але для яких стовпець дільника перевірено в операторі WHERE, видаляються з початкового набору, а решта членів ділення повертаються як результат для цього правила, яке генерує попередження про можливе ділення на нуль.

Порівнювання з NULL

Ця помилка з’являється в запитах із об’єднаними таблицями, для яких джерела об’єднаних таблиць не мають жодного стовпця, на який посилаються дані в умові JOIN, ані в WHERE. Якщо предикат JOIN відсутній, запит включатиме декартовий добуток усіх рядків, також відомий як перехресний добуток, що, безсумнівно, призведе до збільшення продуктивності для запитів і потенційно неправильних результатів. Тому важливо, щоб об’єднані таблиці посилалися в командах JOIN ON або WHERE, щоб уникнути подібних проблем. Є один виняток, коли використовується CROSS JOIN. У цих випадках немає потреби, щоб залучені джерела таблиці мали посилання на будь-які стовпці, оскільки для цих запитів очевидною метою є отримання перехресного добутку всіх рядків, тому попередження не повинно створюватися. У наступному прикладі, на рисунку 7.5, слід виявити семантичну помилку.

```
SELECT name , surname, email , phone , customer_id
FROM organizations AS t1 INNER JOIN customers as t2;
```

Рисунок 7.5 - Семантична помилка з порівнюванням з NULL

У цьому прикладі є дві об’єднані таблиці, organizations і customers, які не мають жодних посилань на стовпці ні в умовах JOIN, ні в WHERE. Це означає, що результат запиту включатиме перехресний добуток усіх рядків. Незалежно від того, чи був це намір автора запиту чи ні, відсутні умови з’єднання є потенційною причиною зайвих витрат на продуктивність і повинні бути повідомлені як семантичні помилки. Не маючи додаткової інформації про основне завдання, для якого був написаний запит, ми не можемо надати потенційне виправлення для цього запиту.

Стратегія виявлення: Виявлення цієї семантичної помилки здійснюється шляхом аналізу запиту та відстеження всіх використаних джерел таблиці. Якщо використовується менше двох таблиць або використовується CROSS JOIN, немає потреби продовжувати перевірку відсутності умов з’єднання, тому попередження не виникатимуть. Якщо використовуються принаймні два джерела таблиць, тоді перевіряються командах ON і WHERE відповідного підзапиту, щоб визначити, чи мають ці джерела таблиці будь-які стовпці, на які

посилаються. Для будь-якої таблиці, на яку немає посилання, буде створено попередження про відсутність умови об’єднання.

Непотрібна команда GROUP BY

Кожен раз, коли атрибут, який з’являється в комнаді GROUP BY, функціонально визначається іншими атрибутами, і якщо він не відображається в командах SELECT або HAVING поза функціями агрегації, тоді його можна взагалі видалити з GROUP BY. Для нашого інструменту ця умова послаблена, оскільки ми розглядаємо лише запит, не знаючи схеми БД, тому неможливо визначити, чи атрибут функціонально визначається іншими атрибутами чи ні. Тому ми перевіряємо лише атрибут, який з’являється в команді GROUP BY, але не використовується ні в командах SELECT, ні в HAVING поза будь-якими функціями агрегації. У цьому випадку буде викликано попередження про цю семантичну помилку. У наступному прикладі на рисунку 7.6 слід виявити семантичну помилку.

```
SELECT COUNT(*) AS counter FROM customers
WHERE amount = 122342
GROUP BY amount HAVING counter > 1 ORDER BY
counter;
```

Рисунок 7.6 - Семантична помилка з непотрібною командою GROUP BY

У цьому прикладі атрибут amount присутній у команді GROUP BY, але не використовується ні в команді SELECT, ні в команді HAVING поза функціями агрегації, тому його можна видалити з GROUP BY.

Це ще більше спростить запит, оскільки amount є єдиним атрибутом, наявним у GROUP BY. Крім того, цей запит містить ще одну семантичну помилку, оскільки amount може мати лише одне значення, оскільки воно прирівнюється до деякої константи в команді WHERE, тому групування не матиме впливу на результат.

Потенційне виправлення для прикладу запиту наведено на рисунку 7.7.

```
SELECT COUNT(*) AS counter FROM customers
WHERE amount = 122342
HAVING counter > 1 ORDER BY counter;
```

Рисунок 7.7 - Виправлення помилка з непотрібною командою GROUP BY

Стратегія виявлення: стратегія виявлення цієї помилки аналізує запит і відстежує атрибути, що з'являються в командах SELECT і HAVING поза функціями агрегації. Крім того, атрибути, які з'являються в команді GROUP BY, також зберігаються. Після завершення аналізу запиту ми перевіряємо атрибути, знайдені в команді GROUP BY, яких немає в жодному з двох списків з атрибутами, виявленими в командах SELECT або HAVING. Для кожного з цих атрибутів виникає попередження, яке вказує на те, що в запиті виявлено семантичну помилку.

Неефективне використання команди HAVING

Проблема: якщо запит має точно такі самі атрибути в команді SELECT, перелічених у команді GROUP BY, і якщо в жодному з двох пунктів не використовуються функції агрегації, тоді команд GROUP BY можна замінити на SELECT DISTINCT. У більшості випадків оптимізатор SQL створює однакові або схожі плани виконання для двох запитів, тому в цих випадках не завжди є виграш у продуктивності, однак, переписавши запит, це стає коротшим і зрозумілішим. У більшості цих випадків використання команди GROUP BY по суті видаляє дублікати з набору результатів, тому оператор DISTINCT краще підходить для використання, на відміну від ситуацій, коли використовуються функції агрегації, і в цьому випадку GROUP BY справді потрібна. У наступному прикладі на рисунку 7.8 слід виявити семантичну помилку.

```
SELECT t1.genre FROM film t1 LEFT JOIN cartoon t2
ON t1.actor = t2. actor
WHERE t2. actor IS NOT NULL GROUP BY t1. genre;
```

Рисунок 7.8 - Семантична помилка з неефективним використанням команди HAVING

У цьому прикладі всі атрибути SELECT перераховані під командою GROUP BY, а також не використовуються функції агрегації, тому кманду GROUP BY можна викинути та замінити на SELECT DISTINCT, зробивши запит коротшим і зрозумілішим. Потенційне виправлення для прикладу запиту наведено на рисунку 7.9.

```
SELECT DISTINCT t1.genre FROM film t1 LEFT JOIN
cartoon t2 ON t1.actor = t2. actor
WHERE e2. actor IS NOT NULL;
```

Рисунок 7.9 - Виправлення помилки з неефективним використанням команди HAVING

Стратегія виявлення: реалізація цієї стратегії спочатку перевіряє, чи використовуються будь-які функції агрегації в командах SELECT або GROUP BY. Якщо це так, то немає необхідності продовжувати подальшу перевірку запиту на цю семантичну помилку. В іншому випадку ми продовжуємо, виявляючи всі змінні, що використовуються в команді SELECT, а також ті, що використовуються в команді GROUP BY.

Якщо існує ідеальна відповідність між цими двома наборами термінів, це означає, що команду GROUP BY можна замінити на SELECT DISTINCT і спрацьовує попередження про цю семантичну помилку.

Для збільшення надійності програмного забезпечення важливо вчасно виявляти семантичні помилки в процесі розробки програмного забезпечення.

Контрольні запитання до лабораторної роботи 7:

1. Що таке семантичні помилки в запитах?
2. Чим загрожують семантичні помилки в запитах?
3. Які є семантичні помилки?
4. Чим загрожує порівнювання з NULL?
5. Чим загрожує зайвий DISTINCT в агрегаціях?
6. Чому треба бути обережним з діленням на нуль?
7. Чим може загрожувати відсутня умова JOIN?
8. Чим загрожує непотрібна команда GROUP BY?
9. Як виправити неефективне використання команди HAVING?

Рекомендовані джерела інформації

1. Бази даних: навч. посіб. / Босько В.В., Константинова Л.В., Поліщук Л.І., Конопліцька-Слободенюк О.К.; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - Кропивницький : ЦНТУ, 2024. – 226 с.

2. Гайна Г.А. Основи проектування баз даних: Навчальний посібник. − К.: КНУБА, 2005. – 204 с.

3. Сидоренко В.В., Константинова Л.В., Смірнов С.А. Організація баз даних Навчальний посібник. - Кропивницький: ЦНТУ, 2018. – 274 с. [Електронний ресурс] - Режим доступу: http://dspace.kntu.kr.ua/jspui/bitstream/123456789/10527/1/NavPosOBD.pdf (дата звернення: 7.11.2023).

4. Пасічник В.В., Резніченко В.А. Організація баз даних та знань.-К: Видавнича група BHV, 2006.-384с.:іл.

5. Балик Н.Р., Мандзюк В.І. MySQL: лабораторний практикум: Посібник для студентів.-Тернопіль:Навчальна книга-Богдан, 2008.-88с.

6. 07 - Моделювання даних та маніпулювання даними /Навчальні ресурси СумДУ University online learning ecosystem. СумГУ, 2015 [Електронний ресурс] - Режим доступу: https://elearning.sumdu.edu.ua/free_content/lectured:a8104441b8e00905159c1ff04257b014dd456247/20151109195846/162252/index.html (дата звернення: 20.03.2024).

7. Що таке хешування [Електронний ресурс] - Режим доступу: https://academy.binance.com/uk/articles/what-is-hashing (дата звернення: 7.01.2024).

8. Пєтух А.М., Романюк О.В., Романюк О.Н. Бази даних. Мови запитів, управління транзакціями, розподілена обробка даних, ВНТУ, 2016 [Електронний ресурс] - Режим доступу: https://web.posibnyky.vntu.edu.ua/fitki/11petuh_bazdanyh_movy_zalitiv/index.htm (дата звернення: 12.03.2024).

9. Клієнт-серверна архітектура. QATestLab. 28.05.2020. [Електронний ресурс] - Режим доступу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення: 20.03.2024).

10. Database Replication 101: Everything You Need To Know. January 24th, 2024. [Електронний ресурс] - Режим доступу: https://www.astera.com/type/blog/database-replication-101/ (дата звернення: 20.03.2024).
11. ISO/IEC 9075-1:2023(en). Information technology - Database languages SQL [Електронний ресурс] - Режим доступу: https://www.iso.org/obp/ui/en/#iso:std:76583:en (дата звернення: 25.03.2024).
12. Features of SQL databases / LinkedIn. 27.05.2023. [Електронний ресурс]/ - Режим доступу: https://www.linkedin.com/pulse/features-sql-databases-database-designer-sql-mysql (дата звернення: 24.03.2024).
13. Stephane Faroult with Peter Robson «The Art of SQL». Sebastopol, Calif.: O'Reilly Media Inc. (2006).
14. 10 SQL Skills for Programmers and Developers / Indeed. 11.03.2023 [Електронний ресурс] - Режим доступу: https://www.indeed.com/career-advice/resumes-cover-letters/sql-skills (дата звернення: 25.03.2024).
15. Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни «Бази даних» (1 частина) : для студ. денної та заочної форми навч. за спец. : 123 «Комп'ютерна інженерія», 125 «Кібербезпека» / [уклад. : В. В. Босько, Л. В. Константинова] ; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т, каф. кібербезпеки та програмного забезпечення. - Кропивницький : ЦНТУ, 2020. - 77 с.
16. Paul DuBois, MySQL, 5th Edition, Mar 29, 2013 by Addison-Wesley Professional.
17. SQL Підручник [Електронний ресурс] – Режим доступу: https://w3schoolsua.github.io/sql/index.html#gsc.tab=0 (дата звернення: 7.04.2024).
18. Методичні вказівки до лабораторних робіт з дисципліни «Бази даних і знань» для студентів напряму підготовки «Управління інформаційною безпекою» / [уклад. : Ю. Є. Яремчук, Д. П. Присяжний, І. О. Дьогтєва, О. В. Салієва] ; ВНТУ [Електронний ресурс] - Режим доступу: https://web.posibnyky.vntu.edu.ua/fmib/37yaremchuk_metodvkaz_labrob_bazi_dani

h_znan_upravlinnya_informacijnoyu_bezpekoyu/07.html (дата звернення: 7.04.2024).

19. Documentation. MySQL, 2024. [Електронний ресурс] - Режим доступу: https://dev.mysql.com/doc/ (дата звернення: 21.03.2024).

20. Бази даних: метод. вказівки до виконання комп‘ютерного практикуму для студентів спеціальності "Електронні комунікації та радіотехніка " / Уклад.: Суліма С.В., Глоба Л.С., Скулиш М.А.. – К.: КПІ ім. Ігоря Сікорського, 2023. – 54 с.

21. Efficient MySQLPerformance: Best Practices and Techniques. 1st Ed.DanielNichter. O'Reilly, 2021. 276p.

22. Learning MySQL: Get a HandleonYourData. 2nd Ed.Vinicius M. Grippa, SergeyKuzmichev.O'Reilly, 2021. 550 p.

23. 97 Things Every Data Engineer Should Know: Collective Wisdom from the Experts. Tobias Macey. O'Reilly, 2021. 264 p.

24. Beginning Spring Data: Data Access and Persistence for Spring Framework 6 and Boot 3 1st ed. Edition. Andres Sacco.Apress, 2022. 439 p.

25. Learning PHP, MySQL & JavaScript. A Step-by-Step Guide to Creating Dynamic Websites. 6th Ed.Robin Nixon.O'Reilly, 2021. 826 p.

26. PHP & MySQL: Novice to Ninja 7th Edition. Tom Butler. SitePoint, 2022. 686 p.

27. PostgreSQL Query Optimization: The Ultimate Guide to Building Efficient Queries. Anna Bailliekova, Henrietta Dombrovskaya, Boris Novikov. Apress, 2021. 344 p.

28. Practical Fraud Prevention. Fraudand AML Analytics for Fintechande Commerce, Using SQL and Python. Gilit Saporta, Shoshana Maraney. O'Reilly, 2022. 394 p.

29. SQL Cookbook. Query Solutionsand Techniques for All SQL Users. 2nd Ed.Anthony Molinaro, RobertdeGraaf. O'Reilly, 2020. 500 p.

30. SQL for Data Analysis. Advanced Techniques for Transforming Data into Insights. 1st Ed. CathyTanimura. O'Reilly, 2021. 350 p.

31. SQL in a Nutshell. A DesktopQuickReference. 4th Edition. Leo Hsu, Regina Obe, Kevin Kline. O'Reilly, 2022. 838 p.

32. SQL PocketGuide: A Guideto SQL Usage. 4th Ed.Alice Zhao. O'Reilly, 2021. 250 p.

33. N. V. Sytnyk and I. S. Zinovieva, “MODERN NoSQL DATABASES FOR TRAINING BACHELORS OF ‘COMPUTER SCIENCE’ SPECIALTY ”, ITLT, vol. 81, no. 1, pp. 255–271, Feb. 2021, doi: 10.33407/itlt.v81i1.3098.

34. Surabhi Gupta, Karthik Ramachandra. Procedural Extensions of SQL: Understanding their usage in the wild. Proceedings of the VLDB Endowment, vol. 14, no. 8, pp. 1378-1391, May2021, doi: 10.14778/3457390.3457402.

35. Olga Poppe, Qun Guo, Willis Lang, Pankaj Arora, Morgan Oslake, Shize Xu, Ajay Kalhan. Moneyball: proactive auto-scaling in Microsoft Azure SQL database serverless. Proceedings of the VLDB Endowment,vol. 15, no. 6, pp. 1279-1287, Feb. 2022, doi: 10.14778/3514061.3514073.

36. Праворська Н.І., Яшина О.М., Нетреба І.В., Доміна А.Р., Кириченко О. М. Метод конструювання програмного забезпечення згідно аналізу помилок SQL-запитів. Вісник ХНУ: Технічні науки. – 2023. Вип. 3, 2023 (321). – С. 302-307. – URL: http://journals.khnu.km.ua/vestnik/wp-content/uploads/2023/06/vknu-ts-2023-n3321-302-307.pdf.

37. Zhekova M., Pashev G., Totkov G. An Algorithm for Translation of a Natural Language Question into SQL Query.15th International Conference Education and Research in the Information Society, ERIS 2022; Plovdiv; Bulgaria; 13 October 2022.CEUR WorkshopProceedings, vol. 3372, pp. 32-40, Oct. 2022. doi:10.21125/edulearn.2023.2016.

38. Klock R. Quality of SQL Code Security on Stack Overflow and Methods of Prevention. Honors Papers, p. 835, Aug. 2021, URL: https://digitalcommons.oberlin.edu/honors/835/.

39. Суліма С. В., Єрмолаєв О. Д. Метод оптимізації SQL запитів системи управління базами даних. Київ : КПІ ім. І. Сікорського. Системи управління навігації та зв’язку Збірник наукових праць – 2023. –Вип. 2 (72). – С. 151-157. – doi:10.26906/SUNZ.2023.2.151.