Оксана Мулеса
Яна Варга

ІНФОРМАЦІЙНІ СИСТЕМИ ТА РЕЛЯЦІЙНІ БАЗИ ДАНИХ

НАВЧАЛЬНИЙ ПОСІБНИК

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»

Оксана Мулеса
Яна Варга

ІНФОРМАЦІЙНІ СИСТЕМИ ТА РЕЛЯЦІЙНІ БАЗИ ДАНИХ

НАВЧАЛЬНИЙ ПОСІБНИК

Ужгород 2023

УДК 004.438
ББК 32.973.26-018.2

У пропонованому посібнику систематизовано теоретичні відомості та практичні прийоми проектування і роботи з реляційними базами даних. Розглянуто основи мови запитів SQL. На практичних прикладах детально описано основні конструкції мови, розглядаються прийоми створення запитів.

Навчальний посібник призначається для студентів вищих навчальних закладів, аспірантів та викладачів, які забезпечують викладання відповідних тематичних модулів.

Рецензенти:

О.Ю. Кучанський, д.т.н., доцент. (КНУ ім. Тараса Шевченка)

Ю.В. Андрашко, к.т.н, доц. (ДВНЗ «УжНУ»)

Інформаційні системи та реляційні бази даних. Навч. посібник. – Ужгород, 2023. – 132 с.

Рекомендовано до друку науково-методичною комісією факультету інформаційних технологій від 24 березня 2023 року, протокол №11.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

1NF – перша нормальна форма

2NF – друга нормальна форма

АІС – автоматизовані інформаційна система

БД – база даних

ЕОМ – електронно-обчислювальна машина

ІС – інформаційна система

ІТ – інформаційна технологія

РБД – реляційна база даних

1. СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ ЯК РІЗНОВИД ІНФОРМАЦІЙНИХ СИСТЕМ

1.1. Співвідношення понять «Інформація» та «Дані»

У сучасних наукових джерелах поняття «дані» та «інформація» і їх співвідношення трактуються по різному. Як правило, інтерпретація вказаних понять залежить від області дослідження.

Так, наприклад, в економічній теорії під даними розуміють невпорядковані спостереження, числа, слова, звуки, зображення; набір дискретних, об'єктивних фактів. В свою чергу, інформацію розглядають як сукупність даних, впорядкованих з певною метою, що додає їм сенс.

В контексті інформаційних систем під інформацією розуміють сукупність відомостей (даних), які сприймаються з навколишнього середовища (вхідна інформація), видаються у навколишнє середовище (вихідна інформація) або зберігаються в середині певної системи.

Звичайно, інформація надходить споживачеві саме у вигляді даних: таблиць, графіків, малюнків, фільмів, усних повідомлень, які фіксують у собі інформацію певної структури й типу. Таким чином, дані виступають як засіб подання інформації у певній, фіксованій формі, придатній для обробки, зберігання й передачі.

Дані можуть бути згруповані у документ. Документ може мати або не мати певної внутрішньої структури.

Інформація є даними, яким надається деякий зміст у конкретній ситуації у рамках деякої системи понять. Інформація подається за допомогою кодування даних і видобувається шляхом їхнього декодування та інтерпретації.

Отже, інформація – це продукт взаємодії даних та методів, який розглядається в контексті цієї взаємодії.

Дані – це інформація, подана у формалізованому вигляді, прийнятному для опрацювання автоматичними засобами за можливої участі людини.

Основними атрибутами інформації є наявність її носія, джерела і приймача, а також каналів зв'язку між ними.

Серед важливих властивостей інформації виділяють такі:

1. Достовірність – відображення істинного стану справ.

2. Повнота – достатність для розуміння і прийняття рішень

3. Зрозумілість – вираженість зрозумілою для адресата мовою.

4. Адекватність – відповідність образу реального об'єкта при її інтерпретації.

Інформацію ділять на види за різними ознаками.

За формою подання:

– текстова;

– числова;

– графічна;

– звукова.

За призначенням:

– масова — містить тривіальні відомості і оперує набором понять, зрозумілим більшій частині соціуму;

– спеціальна — містить специфічний набір понять, при використанні відбувається передача відомостей, які можуть бути не зрозумілі основній масі соціуму, але необхідні і зрозумілі в рамках вузької соціальної групи, де використовується дана інформація;

– особиста — набір відомостей про яку-небудь особистість, що визначає соціальний стан і типи соціальних взаємодій всередині популяції.

Під час інформаційного процесу дані перетворюються з одного виду в інший за допомогою певних методів. Обробка даних містить в собі множину різних операцій. Основними операціями є:

– збір даних – накопичення інформації з метою забезпечення достатньої повноти для прийняття рішення;

– формалізація даних – зведення даних, що надходять із різних джерел до однакової форми;

– фільтрація даних – усунення зайвої інформації, яка не потрібна для прийняття рішень;

– сортування даних – впорядкування даних за заданою ознакою з метою зручності використання;

– архівація даних – збереження даних у зручній та доступній формі;

– захист даних – комплекс дій, що скеровані на запобігання втрат, відтворення та модифікації даних;

– транспортування даних – прийом та передача даних між віддаленими користувачами інформаційного процесу. Джерело даних прийнято називати сервером, а споживача – клієнтом;

– перетворення даних – зміна форми даних, або зміна структури даних.

1.2. Поняття і види інформаційних систем

При самому загальному підході інформаційну систему (ІС) можна визначити як сукупність організаційних і технічних засобів для збереження та обробки інформації з метою забезпечення інформаційних потреб користувачів (абонентів). Таке визначення може бути задовільним тільки при самій узагальненій і неформальній точці зору і підлягає подальшому уточненню.

ІС здавна знаходять (в тому чи іншому вигляді) досить широке застосування в життєдіяльності людства. Це пов'язано з тим, що для існування цивілізації необхідним є обмін інформацією – передача знань, як між окремими членами і колективами суспільства, так і між різними поколіннями.

Найдавнішими і найпоширенішими ІС слід вважати бібліотеки. І, дійсно, здавна в бібліотеках збирають книжки (або їх аналоги), зберігають їх, дотримуючись певних правил, створюють каталоги різного призначення для полегшення доступу до книжкового фонду. Видаються спеціальні журнали та довідники, що інформують про нові надходження, ведеться облік видачі.

До цього моменту мова йшла про інформаційні системи без врахування способу їх реалізації. Найстаріші (у моральному і у фізичному розумінні) системи повністю базувалися на ручній праці. Пізніше їм на зміну прийшли різні механічні пристрої для обробки даних (наприклад, для сортування, копіювання, асоціативного пошуку, тощо). Наступним кроком стало впровадження автоматизованих інформаційних систем (АІС), тобто систем, де для забезпечення інформаційних потреб користувачів використовується ЕОМ зі своїми носіями інформації. В наш час,

розробляється і впроваджується велика кількість самих різноманітних АІСів з дуже широким спектром використання.

Як правило, в навчальній і науковій літературі, під інформаційною системою розуміють систему обробки даних в будь-якій предметній галузі із засобами накопичення, збереження, оновлення , пошуку та видачі інформації.

До ІС входять люди, різні обладнання, процеси, процедури, дані та методи їх обробки. ІС передбачають використання інформаційних технологій (ІТ). Під технологією в широкому аспекті розуміють науку про виробництво матеріальних благ, яка має три аспекти:

– інформаційний: опис принципів та методів виробництва;

– інструментальний: знаряддя праці, за допомогою яких реалізується виробництво;

– соціальний: кадри та їх організація.

Будь-яка ІС характеризується наявністю технології перетворення вихідних даних у результативну інформацію. Такі технології називають інформаційними. В інформаційній технології можна виділити дві складові:

– здатність генерувати за запитом інформаційний продукт;

– засоби доставки цього інформаційного продукту в зручний час і в зручній для користувача формі.

Кожна інформаційна технологія орієнтована на обробку інформації певних видів: даних, текстової інформації, статистичної графіки, знань, анімації, звуку і т. ін. Сучасний рівень розвитку інформаційної технології називається новою інформаційною технологією, ознаками якої є розвинута комп’ютерна техніка, сучасне програмне забезпечення, надійні комунікації, діалоговий режим спілкування користувача з комп’ютером, можливість колективного формування та заповнення документів.

Основною метою створення ІС є задоволення інформаційних потреб користувачів шляхом надання необхідної їм інформації на основі збережених даних. Потреба в інформації як такій не вичерпує поняття інформаційних потреб. Звичайно в поняття інформаційних потреб включають певні вимоги до якості інформаційного обслуговування й поводження системи в цілому (продуктивність, актуальність і надійність даних, орієнтація на користувача та ін.). Під інформаційною системою розуміється організаційна сукупність технічних засобів, технологічних процесів

і кадрів, що реалізують функції збору, обробки, зберігання, пошуку, видачі й передачі інформації. Необхідність підвищення продуктивності праці у сфері інформаційних технологій. Сучасні ІС – складні комплекси апаратних і програмних засобів, технології й персоналу, які ще називають автоматизованими інформаційними системами.

Загальноприйнятої класифікації ІС досі не існує, тому їх можна класифікувати за різними ознаками. Найбільш поширеними протягом тривалого часу були такі класифікації систем:

За рівнем або сферою діяльності:

– державні;
– територіальні (регіональні);
– галузеві;
– об'єднань;
– підприємств або установ;
технологічних процесів.

Залежно від засобів вирішення інформаційної проблеми:

– ручні;
– механізовані;
– автоматизовані;
– автоматичні.

За виконуючими функціями:

– інформаційно-пошукові (довідкові) системи;
– системи управління;
– системи моделювання (системи штучного інтелекту);
– навчальні та екзаменуючі системи;
– експертні системи.

Залежно від галузі застосування:

– медичні;
– економічні;
– соціальні;
– лінгвістичні.

За характером перетворення інформації:

– обчислювальні;
– імітаційні;
– підтримки прийняття рішень.

За математичною суттю:

– прямого розрахунку;
– інформаційно-пошукові;

– оптимізаційні.

За можливістю формалізованого опису:

– формалізовані

– неформалізовані.

Структурно ІС містять у собі апаратне (hardware), програмне (software), комунікаційне (netware), проміжного шару (middleware), лінгвістичне й організаційно- технологічне забезпечення.

Апаратне забезпечення ІС містить у собі широкий набір засобів обчислювальної техніки, передачі даних, а також цілий ряд спеціальних технічних пристроїв (пристрою графічного відображення інформації, аудіо- і відеопристрою, засобу мовного уведення тощо). Апаратне забезпечення є основою будь-якої ІС.

Комунікаційне (мережне) забезпечення містить у собі комплекс апаратних мережних комунікацій і програмних засобів підтримки комунікацій в ІС. Воно має істотне значення при створенні розподілених ІС й ІС на основі Інтернету.

Програмне забезпечення ІС забезпечує реалізацію функцій введення даних, їх розміщення на носіях, модифікації даних, доступ до даних, підтримку функціонування устаткування. Програмне забезпечення можна розділити на системне (яке завершує процес вибору апаратно-програмного рішення або платформи) і користувацьке (яке застосовується для вирішення завдань задоволення потреб користувача у комп'ютерному середовищі).

Лінгвістичне забезпечення ІС призначене для вирішення завдань формалізації змісту повнотекстової і спеціальної інформації для створення пошукового образу даних (профілю). У класичному змісті звичайно воно включає процедури індексування текстів, їхню класифікацію і тематичну рубрикацію. Найчастіше ІС, що містять складно-структуровану інформацію, містять у собі тезауруси термінів і понять. Сюди можна віднести й створення процесорів спеціалізованих формальних мов кінцевих користувачів, наприклад, мов для маніпулювання бухгалтерською інформацією, тощо. Найчастіше працям з розроблення лінгвістичного забезпечення не надається належного значення. Подібні недогляди найчастіше призводять до несприйняття користувачами самої інформації. Це стосується в першу чергу вузько спеціалізованих ІС. У міру зростання складності і масштабів ІС важливу роль починає відігравати організаційно-технологічне

забезпечення, що з'єднує різнорідні компоненти (апаратури, програми й персонал) у єдину систему й забезпечує процедури її керування й функціонування.

1.3. Бази даних та системи управління базами даних

Важливим завданням в процесі розробки сучасних ІС є забезпечення відокремленості програм і даних та їх розгляді як самостійних взаємодіючих об’єктів. На основі цього розвивалися різні підходи до представлення даних в ІС.

Перші інформаційні системи для збереження даних використовували звичайні файли і, відповідно, їх називали файловими системами. Файловим системам притаманні такі риси:

– дані ІС містяться у системі файлів, структура і взаємозв’язки яких визначаються розробниками ІС;

– відсутність домовленостей між розробниками ІС про єдиність підходу до побудови моделі даних;

– необхідність розробки спеціальних програмних засобів для інтерпретації структури даних при розробці кожної ІС;

– необхідність розробки програм для обробки кожного запиту.

Наступною за файловими системами була концепція баз даних (БД). На перших етапах широко розвивалися мережеві і ієрархічні бази даних. Пізніше виникли реляційні бази даних. Особливостями ІС на основі баз даних є те, що:

– дані ІС розміщуються в файлах операційних систем;

– фізична структура файлів уніфікована;

– існує єдиний підхід до побудови моделі даних;

– існує окремий інтерфейс для обробки даних.

Перевагами ІС на основі баз даних є швидкість виконання операцій з даними та ефективність використання пам’яті комп’ютера.

Наступним етапом в розвитку ІС стали об’єктно-орієнтовані бази даних. Їх логічна структура є поєднанням ієрархічної моделі даних з об’єктно-орієнтованими механізмами. Дані, при використанні цієї концепції, і далі розміщуються в файлах операційних систем, проте інформація наводиться у вигляді об’єктів. Всі операції з обробки даних та запитів реалізуються через

інтерфейс маніпулювання об’єктами. Об’єктно-орієнтовані БД, у порівнянні з реляційними БД, мають можливість відображати інформацію про складні взаємозв’язки об’єктів, визначати функції обробки окремих записів.

Також на сучасному етапі часто використовуються ІС з об’єктно-реляційними БД, інформаційними сховищами, розподіленими БД тощо.

Базою даних називають упорядкований набір логічно взаємопов’язаних даних, що використовуються спільно, та призначений для задоволення інформаційних потреб користувачів. Дані, які зберігаються в БД відображають стан об’єктів у заданій предметній області – тій частині реального світу, об’єкти якої описані в БД.

Дані в БД запам'ятовуються таким чином, щоб вони у міру можливості не залежали від програм. Для обробки даних застосовується загальний керуючий метод доступу. Якщо БД не перетинаються за структурою, то говорять про систему баз даних.

База даних повинна відповідати таким вимогам:

1. Відновлювальність – наявність механізмів забезпечення можливості відновлення даних після збою системи.

2. Цілісність – внутрішня єдність, пов’язаність усіх частин даних; стан при якому дані, що зберігаються в БД, відображають властивості реального світу і підпорядковуються правилам взаємної несуперечливості.

3. Безпека – виконається захист даних від санкціонованого і несанкціонованого доступу.

Для того, щоб ефективно управляти базами даних, використовують системи управління базами даних (СУБД).

В загальному випадку, СУБД – це комплекс програм, який створює, повертає за запитом та змінює дані.

СУБД є складними програмними системами, що працюють на різних операційних платформах. Саме СУБД повинна надати засоби визначення й маніпулювання даними, зробивши дані незалежними від прикладних програм, що їх використовують.

Таким чином, система управління базами даних – сукупність програмних та лінгвістичних засобів загального та спеціального призначення, які забезпечують управління створенням та використанням баз даних багатьма користувачами.

До основних функцій СУБД відносять:

– забезпечення мовних засобів опису та маніпуляції даними;
– забезпечення підтримки логічної моделі даних;
– забезпечення взаємодії логічної та фізичної структур даних;
– забезпечення захисту та цілісності даних;
– забезпечення підтримки БД в актуальному стані;
– адміністрування бази даних;
– управління даними у зовнішній пам’яті;
– управління транзакціями;
– журналізація.

До складу СУБД входять такі компоненти:

Ядро СУБД – містить сукупність базових механізмів СУБД, які використовуються при будь-яких варіантах конфігурації системи. Ядро СУБД виконує функцію посередника між підсистемами засобів проектування і обробки даними. Воно отримує запити в термінах таблиць, стовпців, рядків і перетворює ці запити в команди операційної системи. Також, ядро СУБД задіяне в управлінні транзакціями, блокуваннях, резервному копіюванні і відновленні.

В ядро СУБД входять такі складові:

– менеджер буферів – призначений для розв’язку задач ефективної буферизації оперативної пам’яті;

– менеджер даних – призначений для управління зовнішньою пам’яттю, забезпечення створення структур для даних, що зберігаються і допоміжних структур;

– менеджер транзакцій – підтримує механізми фіксації і відкату транзакцій, пов’язаний з менеджером буферів оперативної пам’яті і забезпечує зберігання всієї інформації, яка потрібна після збоїв системи;

– менеджер журналів – забезпечує реєстрацію відомостей про виконання транзакцій, про працюючих користувачів, про виконання додатків, про доступ до різних структур даних тощо.

Підсистема засобів проектування – набір інструментів, які спрощують проектування і реалізацію баз даних і застосувань. Це можуть бути засоби для створення таблиць, форм, запитів і звітів.

Підсистема обробки здійснює обробку компонентів застосування, які створені за допомогою засобів проектування.

СУБД має володіти такими характеристиками:

1. Відновлюваність бази даних після різних збоїв системи.

2. Безпека даних при використанні даних та захист від несанкціонованого доступу до даних.

3. Цілісність (повнота, несуперечливість, адекватність) даних, які містяться у базі даних.

4. Ефективність при обробці запитів користувачів.

2. ВСТАНОВЛЕННЯ ЛОГІЧНОЇ СТРУКТУРИ ДАНИХ В ПРОЦЕСІ ПРОЕКТУВАННЯ БАЗИ ДАНИХ

2.1. Визначення моделі даних як етап проектування систем управління базами даних

Подання інформації за допомогою даних вимагає уніфікованого підходу до поняття даних як незалежного об'єкта моделювання. Тому для розробника СУБД вибір відповідної моделі даних є однією з найважливіших задач. Вибір моделі даних спричиняє вибір засобів аналізу предметної області, як області реального світу, що підлягає вивченню й обробці. Модель даних обмежує можливість вибору СУБД, тому що, як правило, окремо взята модель підтримує певну модель даних. Таким чином, поняття моделі даних є одним з фундаментальних понять, від якого багато в чому залежать механізми реалізації ІС як програмно-апаратного комплексу. Модель даних (Data Model) є логічною структурою даних, що становить притаманні цим даним властивості, незалежні від апаратного й програмного забезпечення й не пов'язані з функціонуванням комп'ютера.

Можна розглянути кілька аспектів моделювання в обробці даних:

– інформаційне моделювання;

– концептуальне моделювання (моделювання семантики предметної області);

– логічне моделювання даних;

– фізичне моделювання:

– створення моделей доступу до даних;

– оптимізація фізичної організації даних в апаратному середовищі.

Наявність у СУБД певної структури даних приводить до поняття баз структурованих даних, тобто дані в таких БД повинні бути представлені як сукупність взаємозалежних елементів. Варто мати на увазі, що для кожного типу БД використовуються відповідні моделі даних. У цей час для баз структурованих даних розрізняють три основні типи логічних моделей даних залежно від характеру підтримуваних ними зв'язків між елементами даних – мережеву, ієрархічну й реляційну. Ознаками класифікації у цих моделях є: ступінь фіксації зв'язку, математичне подання структури моделі і припустимих типів даних. Так, мережева модель характеризується напівстійкими зв’язками між даними та представленням у формі графу, ієрархічна модель – стійкими зв’язками та деревоподібною структурою, реляційна – мінливими зв’язками та двовимірними таблицями.

При зіставленні моделей необхідно пам'ятати, що всі вони теоретично еквівалентні. Еквівалентність моделей полягає в тому, що вони можуть бути зведені одна до одної шляхом формальних перетворень.

2.2. Модель "сутність-зв'язок"

У реальному проектуванні структури бази даних застосовується так зване семантичне моделювання, в основі якого лежить аналіз змісту даних. Як інструмент семантичного моделювання використовуються різні варіанти діаграм сутність-зв'язок. На використанні різновидів даної моделі базуються більшість сучасних підходів до проектування моделей даних. Модель «сутність-зв’язок» є простою візуальною моделлю даних.

Модель "сутність-зв'язок" (ER-модель: Entity-relationship model або entity-relationship diagram) – модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків.

Модель "сутність-зв'язок" ґрунтується на якійсь важливій семантичній інформації про реальний світ і призначена для логічного представлення даних. Вона визначає значення даних в контексті їх взаємозв'язку з іншими даними. Важливим для нас є той факт, що з моделі "сутність-зв'язок" можуть бути породжені всі існуючі моделі даних (ієрархічна, мережева, реляційна, об'єктна), тому вона є найбільш загальною. Будь-який фрагмент наочної області може бути представлений як множина сутностей, між якими існує деяка множина зв'язків.

Модель «сутність-зв’язок» заснована на поняттях теорії множин, теорії відношень, логіки, теорії граток. Її розглядають у такі етапи:

– визначення структури даних: сутності, типу сутності, зв’язку, типу зв’язку, атрибутів тощо;

– задання обмежень цілісності: обмежень на типи зв’язків та атрибутів, обмежень на значення атрибутів, ключів тощо;

– задання операцій над даними.

Базовими елементами моделі є сутності, атрибути і зв’язки.

З об'єктами моделі "сутність-зв'язок" пов'язані поняття: тип – набір однорідних предметів, явищ, що виступають як єдине ціле; екземпляр – конкретний елемент набору, який (набір) визначає деякий тип; множина – конкретний набір екземплярів типу в деякий момент часу.

Сутність (entity) – це об'єкт, який може бути ідентифікований якимись способом, що відрізняє його від інших об'єктів.

Тип сутності визначає набір однорідних сутностей деякого типу.

Набір сутностей (entity set) – множини сутностей одного типу (що володіють однаковими властивостями). Сутність фактично є множиною атрибутів, які описують властивості всіх членів даного набору сутності.

Тип сутності визначає множину однорідних об'єктів, а «екземпляр сутності» – конкретний об'єкт з множини об'єктів.

Взаємовідношення сутностей визначається зв’язками.

Тип зв'язку – це осмислена асоціація між типами сутностей (зокрема, асоціація може бути задана на одному типі сутності). Зв'язок – це асоціація між сутностями, що належать відповідним типам сутностей, які беруть участь у даному типі зв'язку. Множина зв'язку – це множина всіх зв'язків типу зв'язку в деякий момент часу. Охоплені зв'язком сутності називаються учасниками даного зв'язку. Кількість учасників зв'язку називається степенем зв'язку (relationship degree), або його арністю.

Залежно від степеня зв'язку розрізняють:

– бінарні зв'язки (binary relationships);

– багатосторонні зв'язки (multiway relationships).

Бінарний зв'язок здатний з'єднати будь-яку сутність одного типу сутності з будь-якою сутністю іншого типу сутності, зокрема, з будь-якою сутністю того самого типу сутності.

Багатосторонній зв'язок охоплює більше двох типів сутностей або один тип сутності, в якому сутності беруть участь кілька разів у різних ролях.

Засобом, за допомогою якого визначають властивості типу сутності або типу зв'язку, є атрибут.

Атрибут – поіменована характеристика сутності. За допомогою атрибутів моделюються властивості сутностей. Основна роль атрибутів – опис властивостей сутності. Інша роль – ідентифікація екземпляра сутності. Тобто кожен екземпляр сутності повинен мати унікальну назву. Як назва виступає один або декілька атрибутів.

Множину значень, яких може набувати атрибут, називають доменом атрибуту. Домен визначає всі потенційні значення, які можуть бути присвоєні атрибуту.

Розрізняють такі класи атрибутів:

– прості (simple) і складені (composite);

– однозначні (single-valued) і багатозначні (multivalued);

– базові (stored) і похідні (derived);

– ключі (keys).

Простий атрибут складається з одного компонента з незалежним існуванням; складений – з кількох компонентів, кожен з яких характеризується незалежним існуванням. Прості атрибути іноді називають атомарними. Складений атрибут становить деяку композицію простих (або складених) атрибутів. Такі взаємозв'язки вказують на ієрархію атрибутів.

Ключ – атрибут або підмножина атрибутів, що унікально визначає сутність у складі типу сутності.

Виділяють такі види ключів:

– потенційний (candidate) – атрибут або набір атрибутів, що унікально ідентифікують сутності типу сутності (зв'язки типу зв'язку);

– первинний (primary) – деякий вибраний потенційний ключ типу сутності (зв'язку) для спрощення роботи із сутностями (зв'язками);

– альтернативний (alternative) – потенційний ключ, що не є первинним.

Таким чином, логічну структуру в якій зберігаються дані в базі даних називають моделлю даних. До основних моделей представлення даних відносять наступні: мережеву, ієрархічну, реляційну, постреляційну, багатовимірну й об'єктно-орієнтовану. Деякі з них розглянемо нижче.

2.3. Мережева модель даних

Мережева модель – модель в основі якої лежить теорія графів.

Основними елементами мережевої бази даних є елемент даних, агрегат даних, запис, набір.

Елемент даних – найменша неподільна пойменована інформаційна одиниця, доступна користувачеві. Елемент даних може мати свій тип.

Агрегат даних – поіменована сукупність елементів даних усередині запису.

Запис – поіменована структура, що містить елементи даних.

Тип записів – це сукупність логічно пов'язаних примірників записів, моделює деякий клас об'єктів реального світу.

Набір – це поіменована дворівнева ієрархічна структура, яка виражає зв'язки між двома типами записів (один до одного, один до багатьох).

На формування типів зв'язку у мережевій моделі не накладаються особливі обмеження; можливі, наприклад, такі ситуації:

– даний тип запису може бути предком для будь-якого числа зв'язків;

– даний тип запису може бути нащадком в будь-якій кількості зв'язкі;

– може існувати будь-яке число зв'язків з одним і тим же типом запису предка і одним і тим же типом запису нащадка;

– типи записів X і Y можуть бути предком і нащадком в одному зв'язку і нащадком і предком – в іншому.

– предок і нащадок можуть належати до одного типу запису;

– між двома типами записів може бути будь-яка кількість зв’язків

Мережева модель даних може бути основою мережевої бази даних, яка в свою чергу є поіменованою сукупністю записів різного типу і наборів, що містять зв'язки між ними.

Відповідно, СУБД для БД такого типу має забезпечувати, наприклад, можливість виконання таких операцій:

– знаходження певного запису в наборі однотипних записів;

– перехід від предка до першого нащадка по деякому зв'язку;

– перехід до наступного нащадка в деякому зв'язку;

– перехід від нащадка до предка за деяким зв'язком;

– створення нового запису;

– знищення запису;

– модифікація запису;

– включення в зв'язок;

– виключення зі зв'язку;

– переміщення в інший зв'язок тощо.

Особливістю мережевої моделі даних є можливість ефективної реалізації за показниками витрат пам'яті й оперативності. Мережева модель надає великі можливості по створенню і моделюванню різних зв'язків між сутностями реального світу (предметної області). Недоліком мережевої моделі є висока складність і жорсткість схеми даних, складність для розуміння і виконання обробки інформації звичайним користувачем.

2.4. Ієрархічна модель даних

Перші СУБД використовували ієрархічну модель даних. Основними інформаційними одиницями цієї моделі є сегмент, поле, дерево.

Поле даних – найменша неподільна поіменована інформаційна одиниця; сегмент (запис) утворюється з конкретних значень полів даних; тип запису – набір взаємопов'язаних сегментів одного рівня; дерево – набір записів даного типу (таблиця).

Ієрархічна модель базується на теорії графів і являє собою деревоподібний граф із зв'язком «предок – нащадок». Ієрархічна модель через свою зовнішню форму часто називається деревом. У вершині ієрархії лежить корінь дерева, відгалуження – листя дерева. Між типами запису підтримуються зв'язки.

База даних з такою схемою називається ієрархічною БД.

Прикладами типових операторів маніпулювання ієрархічно організованими даними можуть бути наступні:

– знаходження вказаного дерева БД;

– перехід від одного дерева до іншого;

– перехід від одного запису до іншого всередині дерева;

– перехід від одного запису до іншої в порядку обходу ієрархії;

– вставка нового запису у вказану позицію;

– видалення поточного запису.

В ієрархічній моделі використовуються два методи доступу до даних: прямий порядок обходу дерева і зворотний порядок.

Прямий обхід здійснюється зверху вниз, починається з кореня, і так поступово робиться обхід всіх предків у напрямку зверху-вниз, зліва-направо.

Зворотний порядок починається з доступу до найнижчих сегментів з поступовим переходом знизу-вгору, зліва-направо.

У ієрархічних моделях вводяться обмеження цілісності. Цілісність зв'язку підтримується між предками і нащадками. Основним правилом є те, що жодний нащадок не може існувати без свого предка.

Крім того, ієрархічна модель має такі властивості:

1. Кожен нащадок має лише одного предка.
2. Предок може не мати нащадків.

Ієрархічна модель даних реалізує ефективне використання пам'яті комп'ютера і високі часові показники виконання операцій над даними. До недоліків ієрархічної моделі можна віднести її громіздкість для обробки інформації з досить складними зв'язками.

2.5. Реляційна модель даних

Реляційна модель даних – логічна модель даних, в основі якої лежить табличне представлення даних. Подання даних в цій моделі не залежить від способу їх фізичної організації. Це забезпечується за рахунок використання математичного поняття – відношення (таблиця).

Визначив три складові частини реляційної моделі даних:

- структурна;
- маніпуляційна;
- цілісна.

Структурна частина моделі визначає, що єдиною структурою даних є нормалізоване *n*-арне відношення. Відношення зручно представляти у формі таблиць, де кожен рядок називається кортежом, а кожен стовпець – атрибутом, визначений на деякому

домені. Даний неформальний підхід до поняття відношення дає більш звичну для розробників і користувачів форму представлення, де реляційна база даних являє собою скінченний набір таблиць.

Маніпуляційна частина моделі визначає два фундаментальних механізми маніпулювання даними – реляційну алгебру і реляційне числення. Основною функцією маніпуляційної частини реляційної моделі є забезпечення заходів реляційності будь-якої конкретної мови реляційних БД: мова називається реляційною, якщо вона має не меншу виразність і потужність, ніж реляційна алгебра або реляційне числення.

Цілісна частина моделі визначає вимоги цілісності сутностей і цілісності посилань. Перша вимога полягає в тому, що будь-який кортеж будь-якого відношення відмінний від будь-якого іншого кортежу цього відношення, тобто іншими словами, будь-яке відношення має володіти первинним ключем. Вимога цілісності щодо посилань, або вимога зовнішнього ключа полягає в тому, що для кожного значення зовнішнього ключа, що з'являється у відношенні, на яке йде посилання, повинен знайтися кортеж з таким же значенням первинного ключа, або значення зовнішнього ключа повинно бути невизначеним (тобто ні на що не вказувати).

Можна провести аналогію між елементами реляційної моделі даних і елементами моделі «сутність-зв'язок». Реляційні відношення відповідають наборам сутностей, а кортежі – сутностям. Тому, як і в моделі «сутність-зв'язок», стовпці в таблиці, що представляє реляційне відношення, називають атрибутами.

Кожен атрибут визначений на домені, тому домен можна розглядати як множина допустимих значень даного атрибуту. Кілька атрибутів одних відношень і навіть атрибути різних відношень можуть бути визначені на одному і тому ж домені.

Іменована множина пар «ім'я атрибута – ім'я домену» називається схемою відношення. Потужність цієї множини – називають ступенем чи «арністю» відношення. Набір іменованих схем відносин являє собою схему бази даних.

Атрибут, значення якого однозначно ідентифікує кортежі, називається ключовим (або просто ключем). Якщо кортежі ідентифікуються тільки зчепленням значень декількох атрибутів, то говорять, що відношення має складовий ключ. Ставлення може містити кілька ключів. Завжди один із ключів оголошується первинним, його значення не можуть оновлюватися. Всі інші ключі відносини називаються можливими ключами.

На відміну від ієрархічної і мережної моделей даних в реляційної відсутнє поняття групових відношень. Для відображення асоціацій між кортежами різних відносин використовується дублювання їх ключів.

Переваги реляційної моделі:

– простота і доступність для розуміння користувачем. Єдиною використовуваною інформаційною конструкцією є «таблиця»;

– суворі правила проектування, які базуються на математичному апараті;

– повна незалежність даних. Зміни в прикладній програмі при зміні реляційної БД мінімальні;

– для організації запитів і написання прикладного ПЗ немає необхідності знати конкретну організацію БД у зовнішній пам'яті.

Недоліки реляційної моделі:

– далеко не завжди предметна область може бути представлена у вигляді «таблиць»;

– в результаті логічного проектування з'являється множина «таблиць». Це призводить до труднощів розуміння структури даних;

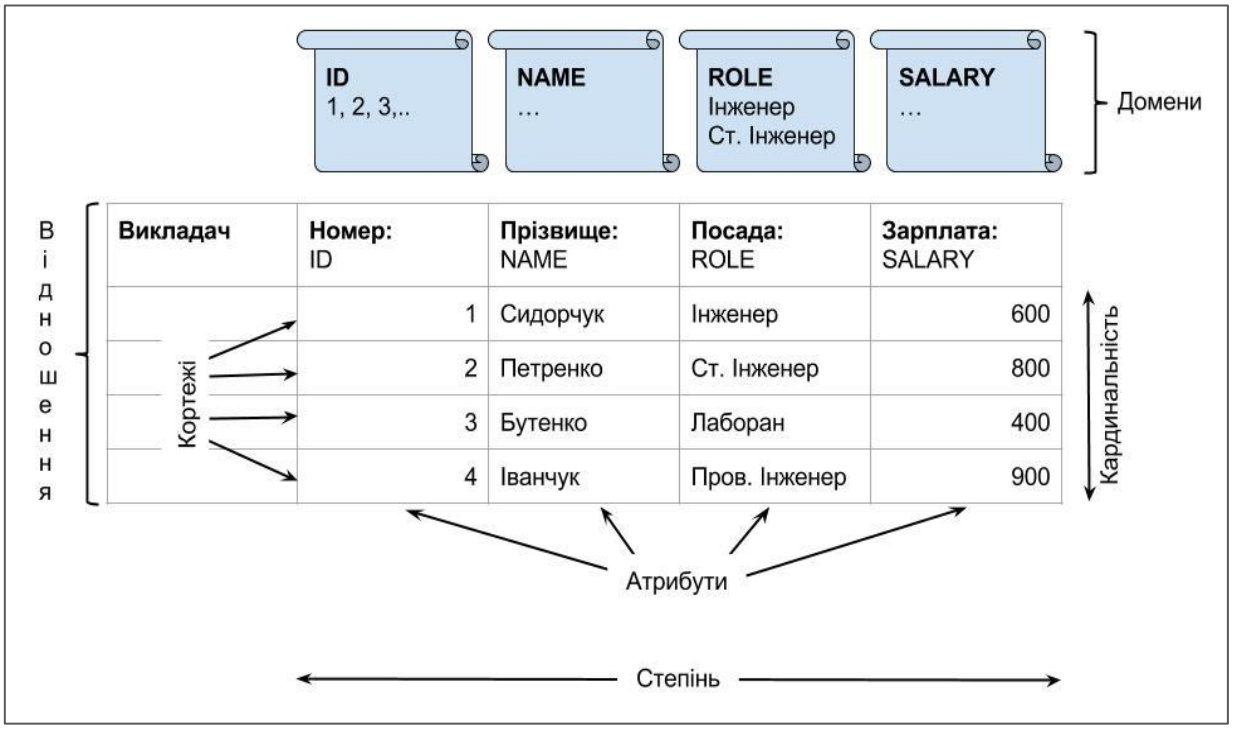
– БД займає відносно багато зовнішньої пам'яті;

– відносно низька швидкість доступу до даних.

3. РЕЛЯЦІЙНІ БАЗИ ДАНИХ

3.1. Основні поняття реляційних баз даних

Основними поняттями реляційних баз даних є тип даних, домен, атрибут, кортеж, первинний ключ, відношення. Співвідношення між цими поняттями показані на рисунку.



Відношення – фундаментальне поняття реляційної моделі даних, яке задається переліком своїх елементів та перерахунком їх значень. Математично відношення може бути визначене як множина кортежів, що є підмножиною декартового добутку фіксованого числа областей D_i (доменів).

Таким чином, *n*-арним відношенням R називають підмножину декартового добутку доменів $D_1, D_2, \ldots, D_n$.

Домени $D_1, D_2,\ldots, D_n$ є допустимими потенційними множинами значень даного типу і характеризуються такими властивостями:

– наявність унікальної назви;

– визначеність на іншому домені або на типі даних;

– наявність певного смислового навантаження.

Типи даних в теорії реляційних баз даних відповідають типам даних у мовах програмування. Серед них можуть бути:

– числові;

– символьні;

– дата і час;

– спеціалізовані числові дані;

– малюнки та медіа-файли;

– інші прості типи даних.

Відношення графічно можна представити у вигляді таблиці, стовпці якої називаються атрибутами і відповідають входженням доменів у відношення. Кількість атрибутів відношення називають його степенем. Кількість кортежів – кардинальністю.

Схемою (заголовком) відношення називають множину пар (*A*, *D*), де *A* – назва атрибуту, *D* – відповідний домен.

Рядки таблиці називаються кортежами і є впорядкованими наборами з *n* значень (*A*, *D*, v), де v – відповідне значенням атрибуту.

Тілом відношення називається невпорядкована множина різних кортежів.

3.2. Властивості відношень. Обмеження цілісності в реляційних базах даних

На таблиці, які є графічним представленням відношень у теорії реляційних баз даних накладаються такі обмеження:

1. Відсутність кортежів-дублікатів.

2. Атомарність значень атрибутів: відсутність повторюваних структур даних.

3. Довільність порядку кортежів відношення.

4. Унікальність імен атрибутів відношення.

Для унікальної ідентифікації кожного кортежу відношення розглядають поняття реляційного ключа.

У реляційній БД може існувати декілька видів ключів:

1. Потенційні (Candidate Key) – CK:
   а) Первинні (Primary Key) – PK;
   б) Альтернативні (Alternate Key) – AK
   або Вторинні (Secondary Key) – SK,
   або Унікальні (Unique Key) – UK.
2. Зовнішні (Foreign Key) – FK.

Потенційний ключ – це атрибут чи набір атрибутів, який можна використати для унікальної ідентифікації кортежів таблиці.

Ключ, що складається з декількох атрибутів, називається складним ключем. У таблиці може бути декілька потенційних ключів.

Первинний ключ – це особлива форма потенційного ключа. У таблиці може існувати лише один первинний ключ.

Зовнішній ключ – це атрибут чи набір атрибутів певної таблиці, значення якої (яких) відповідає значенню потенційного ключа деякої іншої таблиці. Зовнішній ключ організовує, як правило, зв’язок між двома таблицями та забезпечує цілісність посилань.

Для того, щоб база даних була реляційною, необхідно, щоб вона відповідала вимогам цілісності даних.

Термін цілісність даних відноситься до повноти інформації, яка міститься в базі даних. При зміні вмісту бази даних може виникнути порушення цілісності даних, що в ній містяться. Для збереження несуперечності та правильності інформації, що зберігається в реляційній СУБД встановлюється одна або декілька умов цілісності даних. Ці умови визначають які значення можуть бути записані в базу даних в результаті додавання чи обновлення даних. Як правило, в реляційних базах даних використовують такі умови цілісності даних:

1. Обов’язкова наявність даних. Деякі стовпці в базі даних повинні містити значення кожному рядку; рядки в таких стовпцях

не можуть включати значення NULL або не містити ніякого значення.

2. Умова на значення. У кожного стовпця є свій домен, тобто набір значень, які можна зберігати в даному стовпці. Можна вказати СУБД, що запис значень, які не входять в певний діапазон, в такі стовпці є недопустимим.

3. Цілісність сутності. Первинний ключ має в кожному рядку мати унікальне значення, яке відрізняється від значень у всіх інших рядках. Повтор значень в таких стовпцях є недопустимим, так як СУБД не зможе відрізняти один запис від іншого.

4. Цілісність за посиланням. Якщо у таблиці існує зовнішній ключ, то його значення повинні або співпадати зі значеннями потенційного ключа (первинного чи унікального) деякої базової таблиці, або містити пусті значення (NULL).

Налаштування механізму цілісності посилань відбувається за однією із таких сценаріїв:

1. Заборона. Згідно цієї стратегії накладається заборона на усі зміни потенційного ключа, якщо існують зовнішні ключі, що посилаються на нього. Тобто, видаляти стрічки чи змінювати значення у стовпці ID таблиці 1 можна лише для тих значень ID, для яких не існує відповідних їм елементів у таблиці 2.

2. Каскадні зміни. Виконання операцій над вихідною стрічкою базової таблиці з потенційним ключем «каскадним» чином розповсюджується на усі стрічки інших таблиць, зовнішні ключі яких посилаються на базову стрічку. Тобто, видалення стрічки з базової таблиці 1 спонукає видаленню усіх стрічок з таблиці 2, для яких значення зовнішнього ключа співпадає зі значеннями потенційного ключа таблиці 1. Зміна ж значення поля ID таблиці 1 генерує зміну відповідного значення поля ID таблиці 2.

3. Присвоєння NULL-значень (або значення за замовчуванням). У цьому випадку, при видаленні стрічок базової таблиці чи зміні значень потенційного ключа, відповідні значення зовнішніх ключів замінюються значеннями NULL.

Таким чином, правило цілісності посилань вимагає, щоб кожне значення зовнішнього ключа посилалося на реально існуюче значення потенційного ключа, в іншому випадку, для нього повинно бути встановлене значення `NULL`.

3.3. Зв’язки між відношеннями у реляційних базах даних

Зв'язок між таблицями встановлює відношення між співпадаючими значеннями в ключових полях.

У більшості випадків із ключовим полем однієї таблиці (головної таблиці), що є унікальним ідентифікатором кожного запису, зв'язується зовнішній ключ іншої таблиці (підпорядкованої таблиці).

Зовнішній ключ – одне (або декілька) полів у таблиці, що містять посилання на поле (або поля) первинного ключа в іншій таблиці.

Поле зовнішнього ключа визначає спосіб зв'язування таблиць. Вміст поля зовнішнього ключа повинен збігатися зі вмістом ключового поля, хоча імена полів можуть при цьому не збігатися.

Міжтабличний зв'язок – це відношення, встановлені між полями (стовпцями) двох таблиць.

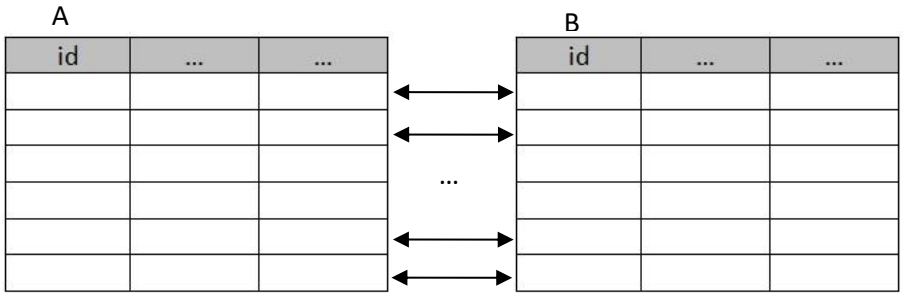
В моделі «Сутність-зв'язок» використовуються бінарні зв'язки, яких може бути три види:

– один до одного 1:1;

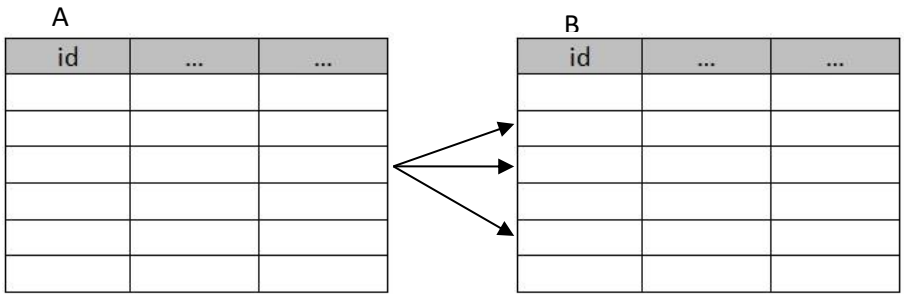
– один до багатьох 1:М;

– багато до багатьох М:М.

При відношенні «один-до-одного» запис у таблиці А має рівно один зв'язаний запису в таблиці В і навпаки. Схематично зв’язок можна зобразити так:



При відношенні «один-до-багатьох» кожному запису в таблиці А можуть відповідати кілька записів у таблиці В (в тому числі і нуль записів), але запис у таблиці В має рівно один відповідний йому запис в таблиці А.



При відношенні «багато-до-багатьох» одному запису в таблиці А можуть відповідати кілька записів у таблиці В (в тому числі і нуль записів), а одному запису в таблиці В кілька записів у таблиці А.

Також розглядаються зв’язки між трьома і більше відношеннями.

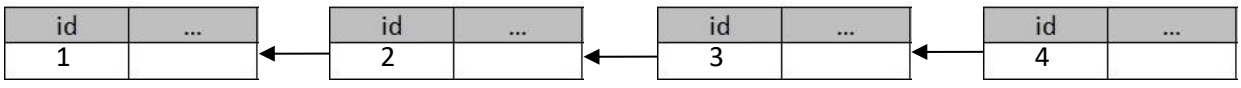
Окремим видом зв’язків у реляційних база даних є рекурсивні зв’язки, які також є декількох типів:

– рекурсивний зв’язок «один-до-одного»;

– рекурсивний зв’язок «один-до-багатьох»;

– рекурсивний зв’язок «багато-до-багатьох».

Рекурсивний зв’язок «один-до-одного» відображає структуру даних типу «черга», яку для прикладу зобразимо так:



Наведений зв’язок можна зобразити у вигляді таблиці наступним чином:

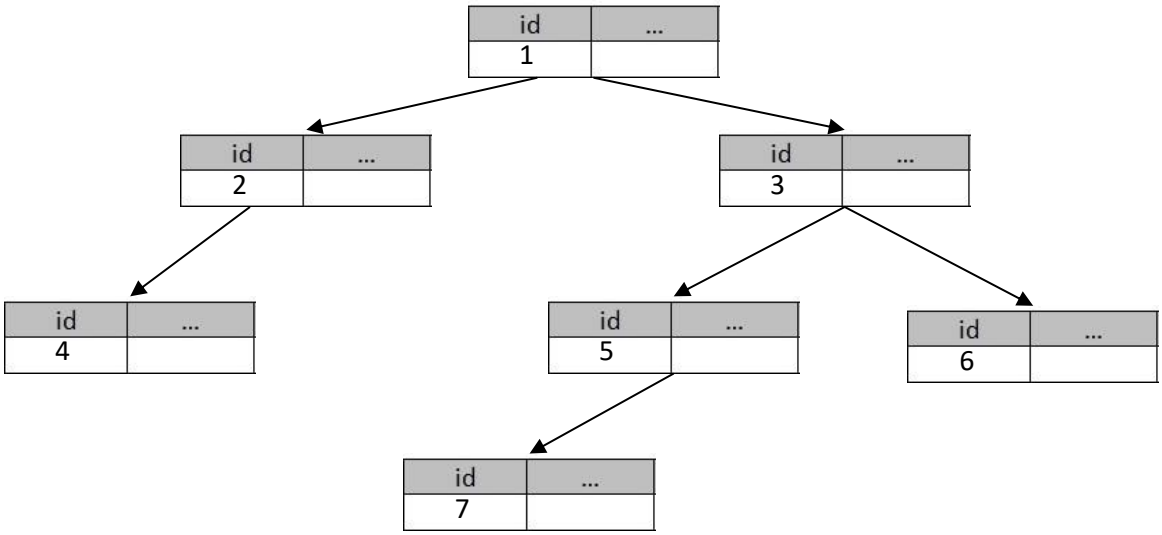
id	…	id_1
1		?
2		1
3		2
4		3

Тут у полі id зазначається ідентифікатор запису, а у полі id_1 – ідентифікатор запису-попередника для заданого.

При побудові таблиці виникає дилема першого запису, тобто питання як заповнювати поле id_1 для першого запису. Можливим є декілька варіантів вирішення цього питання серед яких присвоєння порожнього значення (`NULL`) або значення, яке співпадає з номером першого запису (id).

Таким чином, у побудованій таблиці стовпець id та id_1 перебувають у зв'язку «один-до-одного».

Рекурсивний зв’язок «один-до-багатьох» відображає структуру даних типу «дерево», яку схематично зобразимо так:

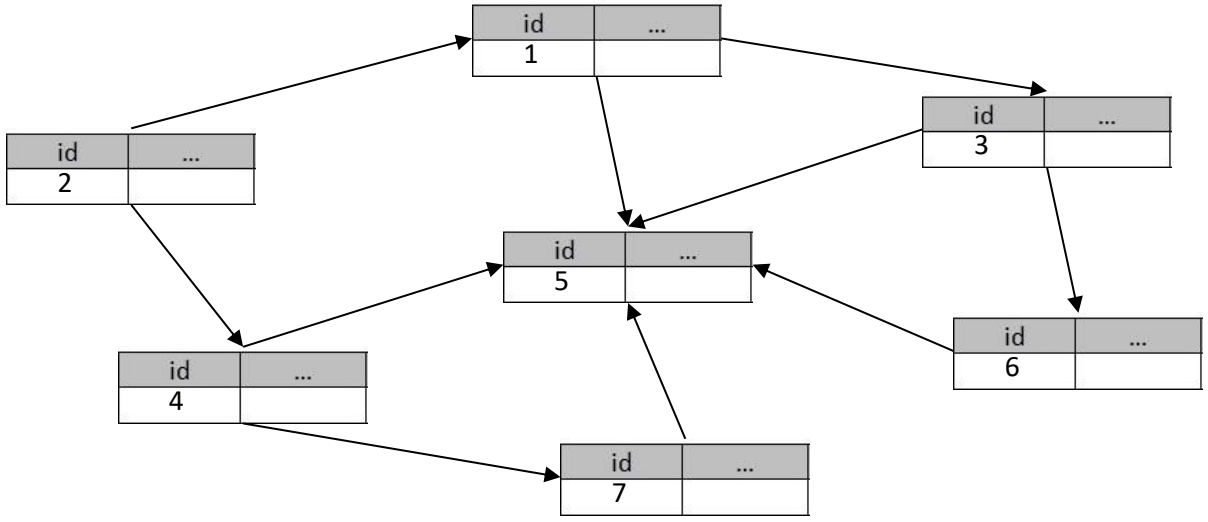


Інформацію про дані і зв’язок між ними у цьому випадку можна представити у вигляді таблиці наступним чином:

id	…	id_1
1		?

2		1
3		1
4		2
5		3
6		3
7		5

Рекурсивний зв’язок «багато-до-багатьох» відображає структуру даних типу «мережа»:



Вказану структуру даних зручно представити у вигляді двох таблиць:

id	…
1	
2	
3	
4	
5	
6	

id	id_1
1	5
1	3
2	1
2	4
3	5
3	6
4	5
4	7
6	5
7	5

Як бачимо з таблиць, у першій з них містяться відомості про вершини мережі, а в другій – про дуги.

3.4. Операції реляційної алгебри

Основні операції реляційної алгебри, відповідно до особливостей їх виконання, розділимо на декілька груп:

I. Об'єднання, перетин, різниця відношень.

II. Розширений прямий добуток відношень.

III. Проекція.

Розглянемо кожну групу окремо.

I. Для виконання операцій першої групи, необхідно, щоб відношення були сумісними за цими операціями. Розглянемо поняття сумісності.

Два відношення є сумісними за операціями об'єднання, перетину і різниці відношень, в тому і тільки тому випадку, якщо вони мають однакові заголовки, тобто, якщо таблиці мають однакову структуру.

Розглянемо відношення:

Відношення А

Код	Назва підприємства	Оплата
001	«Мрія»	1500
002	«Диво»	2400
003	«Каприз»	1450
006	«Берізка»	2510

Відношення В

Код	Назва підприємства	Оплата
002	«Диво»	2400
003	«Каприз»	1450
004	«Калина»	1250
005	«Дружба»	2100

Вони є сумісними за зазначеними операціями, так як мають однакові атрибути. Розглянемо операції над ними.

Відношення C є об'єднанням двох сумісних відношень A і B ($C = A \cup B$), якщо воно володіє тим же заголовком та містить всі кортежі, які входять в хоча б одне з відношень A або B. Якщо у

відношеннях A і B є кортежі-дублікати, то вони включаються у відношення C один раз.

Виконаємо об'єднання двох вище наведених відношень. В результаті отримаємо:

Відношення $C = A \cup B$

Код	Назва підприємства	Оплата
001	«Мрія»	1500
002	«Диво»	2400
003	«Каприз»	1450
006	«Берізка»	2510
004	«Калина»	1250
005	«Дружба»	2100

Відношення D є перетином двох сумісних відношень A і B ($D = A \cap B$), якщо воно володіє тим же заголовком та містить всі кортежі, які входять одночасно в обидва відношення A і B. Якщо у відношеннях A і B нема однакових кортежів, то відношення D буде порожнім.

Виконаємо перетин двох вище наведених відношень. В результаті одержуємо:

Відношення $D = A \cap B$

Код	Назва підприємства	Оплата
002	«Диво»	2400
003	«Каприз»	1450

Відношення E є різницею двох сумісних відношень A і B ($E = A \setminus B$), якщо воно зберігає заголовки відношень і містить всі кортежі, які входять у відношення A, але не входять у відношення B. Якщо всі кортежі відношення A містяться у відношенні B, то відношення E буде порожнім.

Виконаємо перетин двох вище наведених відношень. В результаті отримаємо:

Відношення $E = A \setminus B$

Код	Назва підприємства	Оплата
001	«Мрія»	1500
006	«Берізка»	2510

II. В теорії реляційної алгебри розглядають операцію розширеного прямого добутку відношень. Для виконання цієї операції необхідним є забезпечення сумісності відношень.

Два відношення сумісні відносно операції прямого добутку в тому і тільки тому випадку, якщо множини назв атрибутів цих відношень не перетинаються.

Відношення P є прямим добутком двох сумісних відношень R_1 і R_2 ($P = R_1 \times R_2$), якщо воно складається з усіх таких кортежів p таких, що p є конкатенацією кортежу $r_1 \in R_1$ і кортежу $r_2 \in R_2$, тобто $p = (r_1, r_2)$.

Розглянемо приклад. Нехай дано два відношення:

Відношення R_1

Код	Прізвище	Оклад
01	Іванов	5700
02	Сидоров	6400
03	Петров	5900
04	Кузьма	6250

Відношення R_2

Квартал	Розмір премії
1	1500
2	1700

Тоді відношення, яке є прямим добутком буде мати вид:

Відношення $P = R_1 \times R_2$

Код	Прізвище	Оклад	Квартал	Розмір премії
01	Іванов	5700	1	1500
02	Сидоров	6400	1	1500
03	Петров	5900	1	1500

04	Кузьма	6250	1	1500
01	Іванов	5700	2	1700
02	Сидоров	6400	2	1700
03	Петров	5900	2	1700
04	Кузьма	6250	2	1700

III. При виконанні проекції відношення на заданий набір його атрибутів отримується відношення, кортежі якого створюються шляхом взяття відповідних значень з кортежів операнда-відношення без повторів для заданих атрибутів.

Наприклад, проекцією відношення P на набір атрибутів <Код, Прізвище, Оклад> є відношення R_1.

3.5. Нормалізація в реляційних базах даних

Як відомо, дані в реляційних базах даних представляються у формі таблиць. Від способу організації даних та структури таблиць залежить ефективність утвореної бази даних та зручність роботи з нею. Розглянемо таблицю з відомостями про замовлення:

Таблиця «Замовлення»

№	Дата	Замовник	Товар
1.	12.03.2014	«Автолюкс», м.Хмельницький	«Папір офісний» – 2 шт.х 100грн , 200 грн.
2.	12.03.2014	«Автолюкс», м.Хмельницький	«Бланки для податкової звітності» – 1 набір х 45 грн, 45 грн.
3.	13.03.2014	«МріяПлюс», м.Луцьк	«Набір канцелярський» – 40 шт. х 90 грн, 3600 грн.
4.	13.03.2014	«МріяПлюс», м.Луцьк	«Папір офісний» – 10 шт. х 100 грн, 1000 грн.
5.	14.03.2014	«Автолюкс», м.Хмельницький	«Набір канцелярський» – 10 шт. х 90 грн, 900 грн.
6.	15.03.2014	«МріяПлюс», м.Луцьк	«Папір офісний» – 5 шт. х 120 грн, 600 грн.

Наведена таблиця має декілька особливостей:

– у таблиці, всі дані про замовників повторюються стільки разів, скільки замовлень робив відповідний замовник. Використання такого способу подачі даних ускладнює введення даних в таблицю при великих кількостях замовлень, підвищує вірогідність виникнення помилок введення, а також призводить до дублювання інформації в таблиці;

– у стовпчику «Товар» вказано назву товару, величину замовлення, вартість одиниці товару та загальну вартість товару. Тобто, в кожній комірці даного стовпця міститься цілий набір даних, що значно ускладнює обробку кожного конкретного елементу, наприклад – загальної вартості замовлення.

Таким чином, представлення даних у вигляді наведеної таблиці призводить до ряду складнощів, і для зручності організації та використання таких даних необхідним є її переформатування.

Для того, щоб уникнути вказаних труднощів та спростити організацію даних використовують метод нормалізації таблиць.

Нормалізація – це формальний метод аналізу таблиць на основі їх первинних чи потенційних ключів та існуючих функціональних залежностей. Він включає ряд правил, які можуть використовуватися для переформатування окремих таблиць таким чином, щоб уся БД могла бути нормалізованою до необхідного степеню. Якщо деяка вимога не виконується, тоді таблиця, що суперечить цій вимозі, повинна бути розбита на таблиці, кожна з яких окремо задовольняє усім вимогам нормалізації.

В теорії реляційних баз даних звичайно виділяється наступна послідовність нормальних форм:

- перша нормальна форма (1NF);
- друга нормальна форма (2NF);
- третя нормальна форма (3NF);
- нормальна форма Бойса-Кодда (BCNF);
- четверта нормальна форма (4NF);
- п'ята нормальна форма, або нормальна форма проекції-з'єднання (5NF або PJ / NF).

Основні властивості нормальних форм:

– кожна наступна нормальна форма в деякому сенсі краще попередньої;

– при переході до наступної нормальної форми властивості попередніх нормальних форм зберігаються.

В основі процесу проектування лежить метод нормалізації – декомпозиція відношення (таблиці), що знаходиться в попередній нормальній формі, в два або більше відношення, що задовольняють вимогам наступної нормальної форми.

Найбільш важливі на практиці нормальні форми відношень ґрунтуються на фундаментальному в теорії реляційних баз даних понятті функціональної залежності. Для подальшого розгляду нам будуть потрібні кілька визначень.

1) *Функціональна залежність.* У відношенні R атрибут Y функціонально залежить від атрибута X (X і Y можуть бути

складовими) в тому і тільки в тому випадку, якщо кожному значенню *X* відповідає в точності одне значення *Y: R.X (r) R.Y*.

2) *Повна функціональна залежність*. Функціональна залежність *R.X (r) R.Y* називається повною, якщо атрибут *Y* не залежить функціонально від будь-якої точної підмножини *X*.

3) *Транзитивна функціональна залежність*. Функціональна залежність *R.X (r) R.Y* називається транзитивною, якщо існує такий атрибут *Z*, що є функціональні залежності *R.X (r) R.Z* і *R.Z (r) R.Y* і відсутня функціональна залежність *R.Z (r) R.X* (При відсутності останньої вимоги ми мали б «нецікаві» транзитивні залежності в будь-якому відношенні, яке має декілька ключів).

4) *Неключовий атрибут*. Неключовим атрибутом називається будь-який атрибут відношення, який не входить до складу ключа (зокрема, первинного).

5) *Взаємно незалежні атрибути*. Два або більше атрибута взаємно незалежні, якщо жоден з цих атрибутів не є функціонально залежним від інших.

З урахуванням наведених понять і означень охарактеризуємо нормальні форми реляційних баз даних.

Перша нормальна форма. Відношення знаходиться у першій нормальній формі (1NF), якщо усі дані, що зберігаються в таблиці є атомарними (не містять повторюваних груп – масивів даних), а таблиця має первинний ключ.

З означення першої нормальної форми випливають такі властивості відношення:

- кожен рядок таблиці має унікальний ключ;
- рядки не впорядковані;
- атрибути не впорядковані (можна переставляти стовпці);
- відсутні структурні атрибути - всі атрибути «атомарні»;
- нема рядків, які повторюються.

1NF передбачає строге забезпечення вимоги цілісності сутностей, тобто усі рядки таблиці повинні бути різними.

Друга нормальна форма. Відношення R знаходиться у другій нормальній формі (2NF) в тому і тільки в тому випадку, коли

перебуває в 1NF, і кожен її неключовий атрибут пов’язаний повною функціональною залежністю з первинним ключем, тобто у ній не існує залежності від частини композитного (складного) ключа.

Третя нормальна форма. Відношення R знаходиться в третій нормальній формі (3NF) в тому і тільки в тому випадку, якщо перебуває в 2NF і кожен з неключових атрибутів нетранзитивно залежить від первинного ключа.

На практиці третя нормальна форма схем відношень достатня в більшості випадків, і приведенням до третьої нормальної форми процес проектування реляційної бази даних зазвичай закінчується. Однак іноді корисно продовжити процес нормалізації.

Нормальна форма Бойса-Кодда.

Детермінант – будь-який атрибут, від якого повністю функціонально залежить деякий інший атрибут.

Відношення R знаходиться в нормальній формі Бойса-Кодда (BCNF) в тому і тільки в тому випадку, якщо кожен детермінант є можливим ключем.

Четверта нормальна форма.

У відношенні *R (A, B, C)* існує багатозначна залежність *R.A (r) (r) R.B* в тому і тільки в тому випадку, якщо множина значень *B*, відповідна парі значень *A* і *C*, залежить тільки від *A* і не залежить від *C*.

Відношення *R* знаходиться в четвертій нормальній формі (4NF) в тому і тільки в тому випадку, якщо відношення знаходиться у 3NF та в разі існування багатозначної залежності *A (r) (r) B* всі інші атрибути *R* функціонально залежать від *A*.

П'ята нормальна форма.

Залежність з'єднання. Відношення *R (X, Y, ..., Z)* задовольняє залежності з'єднання *(X, Y, ..., Z)* в тому і тільки в тому випадку, коли *R* відновлюється без втрат шляхом з'єднання своїх проекцій на *X, Y, ..., Z*.

Відношення *R* знаходиться в п'ятій нормальній формі (нормальній формі проекції-з'єднання – PJ / NF) в тому і тільки в

тому випадку, коли будь-яка залежність з'єднання в *R* випливає з існування деякого можливого ключа в *R*.

П'ята нормальна форма – це остання нормальна форма, яку можна отримати шляхом декомпозиції. Її умови досить нетривіальні, і на практиці 5NF практично не використовується.

Наведемо приклад нормалізації таблиць.

Приклад нормалізації таблиць

Розглянемо наступну таблицю:

Таблиця «Замовлення»

Дата	Замовник	Товар
12.03.2014	«Автолюкс», м.Хмельницький, вул. Миру, 10	«Папір офісний» – 2 шт.х 100грн , 200 грн.
12.03.2014	«Автолюкс», м.Хмельницький, вул. Миру, 10	«Бланки для податкової звітності» – 1 набір х 45 грн, 45 грн.
13.03.2014	«МріяПлюс», м.Луцьк, вул. Волі, 25	«Набір канцелярський» – 40 шт. х 90 грн, 3600 грн.
13.03.2014	«МріяПлюс», м.Луцьк, вул. Волі, 25	«Папір офісний» – 10 шт. х 100 грн, 1000 грн.
14.03.2014	«Автолюкс», м.Хмельницький, вул. Миру, 10	«Набір канцелярський» – 10 шт. х 90 грн, 900 грн.
15.03.2014	«МріяПлюс», м.Луцьк, вул. Волі, 25	«Папір офісний» – 5 шт. х 120 грн, 600 грн.

Як бачимо з таблиці, вона не знаходиться у першій нормальній формі, так як у стовпцях «Замовник» та «Товар» є неатомарні значення. Приведемо дану таблицю у 1NF, шляхом розбиття вказаних стовпчиків:

Таблиця «Замовлення»

Дата	Замовник	Адреса замовника	Назва товару	Ціна за одиницю	Кількість	Вартість
12.03.2014	«Автолюкс»	м.Хмельницький, вул. Миру, 10	«Папір офісний»	100	2	200
12.03.2014	«Автолюкс»	м.Хмельницький, вул. Миру, 10	«Бланки для податкової звітності»	45	1	45
13.03.2014	«МріяПлюс»	м.Луцьк, вул. Волі, 25	«Набір канцелярський»	90	40	3600
13.03.2014	«МріяПлюс»	м.Луцьк, вул. Волі, 25	«Папір офісний»	100	10	1000
14.03.2014	«Автолюкс»	м.Хмельницький, вул. Миру, 10	«Набір канцелярський»	90	10	900
15.03.2014	«МріяПлюс»	м.Луцьк, вул. Волі, 25	«Папір офісний»	120	5	600

Таким чином, після розбиття стовпців, таблиця «Замовлення» відповідає всім вимогам 1NF.

Перевіримо, чи виконуються для неї вимоги другої нормальної форми. Для цього визначимо первинний ключ. Як бачимо, він є складним і складається з таких атрибутів: Дата, Замовник, Назва товару, Ціна за одиницю. Очевидно, що поле «Адреса замовника» залежить від поля «Замовник», тобто від частини композитного ключа. Таким чином, функціональна залежність не є повною, а, отже база даних не є в 2NF. Приведемо її в другу нормальну форму, шляхом введення таких змін:

1. Створимо таблицю «Замовники», в яку внесемо відомості про замовників.
2. Створимо таблицю «Товари», де кожному товару присвоїмо свій унікальний ідентифікатор.
3. Створимо таблицю «Замовлення», яка логічно пов'яже всі таблиці бази даних.

Таблиця «Замовники»

Замовник	Адреса замовника
«Автолюкс»	м.Хмельницький, вул. Миру, 10
«МріяПлюс»	м.Луцьк, вул. Волі, 25

Таблиця «Товари»

Код товару	Назва товару	Ціна за одиницю
001	«Папір офісний»	100
101	«Бланки для податкової звітності»	45
201	«Набір канцелярський»	90
002	«Папір офісний»	120

Таблиця «Замовлення»

Код замовлення	Дата	Замовник	Код товару	Кількість	Вартість
1	12.03.2014	«Автолюкс»	001	2	200
2	12.03.2014	«Автолюкс»	101	1	45
3	13.03.2014	«МріяПлюс»	201	40	3600
4	13.03.2014	«МріяПлюс»	001	10	1000
5	14.03.2014	«Автолюкс»	201	10	900
6	15.03.2014	«МріяПлюс»	002	5	600

База даних, яка складається з трьох вказаних таблиць, знаходиться в другій нормальній формі. Між таблицями встановлено такі зв’язки:

– таблиця «Замовники» та таблиця «Замовлення» перебувають у зв’язку «один-до-багатьох» за атрибутом «Замовник»;

– таблиця «Товари» та таблиця «Замовлення» перебувають у зв’язку «один-до-багатьох» за атрибутом «Код товару».

Для того, щоб утворена база даних перебувала у 3NF, необхідно, щоб неключові атрибути нетранзитивно залежали від ключа. Проте, у таблиці «Замовлення» значення атрибуту «Вартість» залежить не тільки від значень первинного чи потенційного ключа, а й від значень неключового атрибуту «Кількість». Таким чином, існує транзитивна залежність і база даних не перебуває у третій нормальній формі. Для того, щоб звести її у 3NF видалимо з неї атрибут «Вартість». Слід зазначити, що видалення атрибуту не призведе до втрати інформації, тому що, маючи всі три таблиці бази даних, обчислити вартість можна за допомогою простого багатотабличного запиту.

Таким чином, утворена база даних матиме таку структуру:

Таблиця «Замовники»

Замовник	Адреса замовника
«Автолюкс»	м.Хмельницький, вул. Миру, 10
«МріяПлюс»	м.Луцьк, вул. Волі, 25

Таблиця «Товари»

Код товару	Назва товару	Ціна за одиницю
001	«Папір офісний»	100
101	«Бланки для податкової звітності»	45
201	«Набір канцелярський»	90
002	«Папір офісний»	120

Таблиця «Замовлення»

Код замовлення	Дата	Замовник	Код товару	Кількість
1	12.03.2014	«Автолюкс»	001	2
2	12.03.2014	«Автолюкс»	101	1
3	13.03.2014	«МріяПлюс»	201	40
4	13.03.2014	«МріяПлюс»	001	10
5	14.03.2014	«Автолюкс»	201	10
6	15.03.2014	«МріяПлюс»	002	5

Використання інших нормальних форм не завжди призводить до оптимізації бази даних, тому даний приклад не міститиме їх розгляду.

3.6. Правила Кодда для реляційних систем управління базами даних

«12 правил Кодда» — набір 13 правил (пронумерованих від нуля до дванадцяти) запропонованих Едгаром Коддом, піонером реляційної моделі для баз даних, спроектовані для визначення того чи є СУБД реляційною.

Правила настільки суворі, що всі популярні так звані «реляційні» СУБД не відповідають багатьом критеріям. Особливо складні 6, 9, 10, 11 і 12 правила.

0. Фундаментальне правило (Foundation Rule)

Реляційна СУБД має бути здатною повністю керувати базою даних, використовуючи зв'язки між даними.

1. Інформаційне правило (Information Rule)

Інформація має бути представлена у вигляді даних, що зберігаються в осередках. Дані, що зберігаються у комірках, мають бути атомарними. Порядок рядків у реляційній таблиці не повинен впливати на зміст даних.

2. Правило гарантованого доступу (Guaranteed Access Rule)

Доступ до даних має бути вільним від двозначності. До кожного елементу даних має бути гарантований доступ за допомогою комбінації імені таблиці, первинного ключа рядку й імені стовпця.

3. Систематична обробка `NULL`-значень (Systematic Treatment of NULL Values)

Невідомі значення `NULL`, відмінні від будь-якого відомого значення, мають підтримуватись для всіх типів даних при виконанні будь-яких операцій. Наприклад, для числових даних невідомі значення не повинні розглядатись як нулі, а для символьних даних — як порожні рядки.

4. Правило доступу до системного каталогу на основі реляційної моделі (Dynamic On-line Catalog Based on the Relational Model)

Словник даних має зберігатись у формі реляційних таблиць, і СУБД повинна підтримувати доступ до нього за допомогою стандартних мовних засобів, тих самих, що використовуються для роботи з реляційними таблицями, які містять дані користувача.

5. Правило повноти підмови маніпулювання даними (Comprehensive Data Sublanguage Rule)

Система управління реляційними базами даних має підтримувати хоча б одну реляційну мову, яка

а) має лінійний синтаксис,

б) може використовуватись інтерактивно і в прикладних програмах,

в) підтримує операції визначення даних, маніпулювання даними (інтерактивні та програмні), обмежувачі цілісності, управління доступом та операції управління транзакціями (begin, commit і rollback).

6. Правило модифікації (View Updating Rule)

Кожне подання має підтримувати усі операції маніпулювання даними, які підтримують реляційні таблиці: операції вибірки, вставки, модифікації і видалення даних.

7. Правило високорівневих операцій модифікації даних (High-level Insert, Update, and Delete)

Операції вставки, модифікації і видалення даних мають підтримуватись не тільки щодо одного рядку реляційної таблиці, але й щодо будь-якої множини рядків.

8. Правило фізичної незалежності даних (Physical Data Independence)

Додатки не повинні залежати від використовуваних способів зберігання даних на носіях, від апаратного забезпечення комп'ютерів, на яких знаходиться реляційна база даних.

9. Правило логічної незалежності даних (Logical Data Independence)

Представлення даних в додатку не повинно залежати від структури реляційних таблиць. Якщо в процесі нормалізації одна реляційна таблиця розділяється на дві, подання має забезпечити об'єднання цих даних, щоб зміна структури реляційних таблиць не позначалась на роботі додатків.

10. Правило незалежності контролю цілісності (Integrity Independence)

Вся інформація, необхідна для підтримки цілісності, має бути у словнику даних. Мова для роботи з даними має виконувати перевірку вхідних даних і автоматично підтримувати цілісність даних.

11. Правило незалежності від розміщення (Distribution Independence)

База даних може бути розподіленою, може перебувати на кількох комп'ютерах, і це не повинно впливати на додатки. Перенесення бази даних на інший комп'ютер не повинне впливати на додатки.

12. Правило узгодженості мовних рівнів (The Nonsubversion Rule)

Якщо використовується низькорівнева мова доступу до даних, вона не повинна ігнорувати правила безпеки і правила цілісності, які підтримуються мовою більш високого рівня.

4. ОСНОВИ МОВИ ЗАПИТІВ SQL

4. 1. Інструкції SQL

В SQL існує близько 40 інструкцій. Кожна з них звертається до СУБД за виконанням конкретної дії. Відповідно до призначення, інструкції поділяються на види:

– інструкції обробки даних;

– інструкції визначення даних;

– інструкції управління доступом;

– інструкції управління трансакціями;

– програмні інструкції.

Основні інструкції з обробки та представлення даних наведені в таблиці:

Інструкції з обробки та представлення даних SQL

Інструкція	Опис
`SELECT`	Отримує дані з однієї або декількох таблиць
`INSERT`	Додає нові рядки в таблицю
`DELETE`	Видаляє рядки з таблиці
`UPDATE`	Обновляє дані, які вже існують в таблиці
`CREATE TABLE`	Додає нову таблицю в БД
`DROP TABLE`	Видаляє таблицю з БД
`ALTER TABLE`	Змінює структуру існуючої таблиці

Кожна інструкція SQL починається з команди, тобто ключового слова, яке описує дію, що виконується інструкцією. Після команди йде одне або декілька речень. Речення описують дані, з якими працює інструкція, або містять інформацію про дію,

яку виконує інструкція. Кожне речення також починається з ключового слова, наприклад `WHERE` (де), `FROM` (звідки), `INTO` (куди). Деякі речення в інструкціях є обов’язковими, інші – ні. Велика частина речень містить імена таблиць чи стовпців, деякі з них можуть містити додаткові ключові слова, константи, вирази.

У кожного об’єкта в БД є унікальне ім'я: імена таблиць, імена стовпців тощо.

4.2. Типи даних в SQL

Найбільш поширені типи даних, які використовуються в SQL:

– цілі числа;

– десяткові числа;

– числа з плаваючою точкою;

– рядки символів сталої довжини;

– рядки символів змінної довжини;

–грошові величини;

– дата та час;

– булеві величини;

– довгий текст;

– неструктуровані потоки байтів;

– нелатинські символи.

Типи даних стандарту ANSI/ISO наведені в таблиці:

Типи даних в SQL

Тип даних	Опис
CHAR (довжина)	Рядки символів сталої довжини
CHARACTER (довжина)	
VARCHAR (довжина)	Рядки символів змінної довжини
CHAR VARYING (довжина)	
CHARACTER VARYING (довжина)	
NCHAR (довжина)	Рядки локалізованих символів сталої довжини
NATIONAL CHAR (довжина)	
NATIONAL CHARACTER (довжина)	
NCHAR VARYING (довжина)	Рядки локалізованих символів змінної довжини
NATIONAL CHAR VARYING (довжина)	
NATIONAL CHARACTER VARYING (довжина)	
INTEGER	Цілі числа
INT	
SMALLINT	Малі цілі числа
BIT (довжина)	Рядки бітів сталої довжини
BIT VARYING (довжина)	Рядки бітів змінної довжини
NUMERIC (точність, степінь)	Числа з плаваючою точкою
DECIMAL (точність, степінь)	
DEC (точність, степінь)	
FLOAT (точність)	
REAL	Числа з плаваючою крапкою низької точності
DOUBLE PRECISION	Числа з плаваючою крапкою високої точності
DATE	Дата
TIME (точність)	Час
TIMESTAMP (точність)	Дата і час
INTERVAL	Часовий інтервал

4.3. Вбудовані функції SQL

Найбільш корисні функції, які підтримуються в різних СУБД перераховані в таблиці:

Деякі вбудовані функції стандарту SQL2

Функція	Значення
`BIT_LENGTH(рядок)`	Кількість біт в рядку
`CAST (значення AS тип даних)`	Перетворює у вказаний тип даних
`CHAR_LENGTH(рядок)`	Довжина рядку символів
`CONVERT(рядок USING функція)`	Рядок, перетворений у відповідності до вказаної функції
`CURRENT_DATE`	Поточна дата
`CURRENT_TIME(точність)`	Поточний час із вказаною точністю
`CURRENT_TIMESTAMP(точність)`	Поточні дата і час із вказаною точністю
`DAY(дата)`	Повертає день з дати
`EXTRACT (частина FROM значення)`	Вказана частина (DAY, HOUR,…) із значення DATETIME
`LOWER(рядок)`	Рядок, переведений в нижній регістр
`MONTH(дата)`	Повертає місяць з дати
`OCTET_LENGTH(рядок)`	Кількість байт в рядку символів
`POSITION(підрядок IN рядок)`	Позиція, з якої починається входження підрядка в рядок
`SUBSTRING(рядок FROM n FOR довжина)`	Частина рядка, починаючи з n-го символу, вказаної довжини

Функція	Значення
`TO_CHAR(дата, специфікація)`	Перетворює дату відповідно до заданої специфікації
`TRANSLATE(рядок USING функція)`	Рядок, трансльований за допомогою вказаної функції
`TRIM(BOTH символ FROM рядок)`	Рядок, з якого видалені перші і останні вказані символи
`TRIM(LEADING символ FROM рядок)`	Рядок, з якого видалені перші вказані символи
`TRIM(TRAILING символ FROM рядок)`	Рядок, з якого видалені останні вказані символи
`UPPER(рядок)`	Рядок, перетворений у верхній регістр
`YEAR(дата)`	Повертає рік з дати

4.4. Константи дати і часу

В реляційних СУБД значення дати, часу та інтервалів часу представлені у вигляді рядкових констант. Формати цих констант в різних СУБД відрізняються один від одного. Крім того, способи запису дати і часу змінюються в залежності від країни. Деякі формати дати і часу наведені нижче:

Формати дати і часу в деяких реляційних СУБД

Формат	Формат значень типу DATE	Приклад значення типу DATE	Формат значень типу TIME	Приклад значення типу TIME
Американський	mm/dd/yyyy	5/19/1960	hh:mm am/pm	2:18 PM
Європейський	dd.mm.yyyy	19.5.1960	hh:mm.ss	14:18.08
Японський	yyyy-mm-dd	1960-5-19	hh:mm:ss	14:18:08
ISO	yyyy-mm-dd	1960-5-19	hh:mm:ss	14:18:08
TIMESTAMP	yyyy-mm-dd-hh.mm.ss.nnnnnn			

4.5. Створення таблиць. Інструкція CREATE TABLE

Інструкція `CREATE TABLE` автоматично визначає нову таблицю (структуру) та готує її до запису даних. Різні блоки інструкції задають елементи визначення таблиці. Синтаксична структура інструкції є такою:

```
CREATE TABLE ім'я_таблиці (визначення стовпців
або визначення обмежень таблиці,…)
```

Після виконання інструкції створюється нова таблиця. Створена таблиця є порожньою; додавати до неї записи можна за допомогою інструкції `INSERT`.

Визначення стовпців

Визначення стовпців являє собою розміщений в дужках список, елементи якого відокремлені один від одного комами. Порядок слідування визначень стовпців в списку відповідає

порядку стовпців в таблиці. Кожне визначення містить наступну інформацію:

– ім’я стовпця;

– тип даних стовпця;

– вказівка на те, чи обов’язково стовпець має містити дані: якщо вказано обмеження NOT NULL, то значення NULL не може міститися в стовпці;

– значення за замовчуванням, яке заноситься в таблицю у тому випадку, якщо інструкція INSERT не містить значення даного стовпця.

Значення за замовчуванням та відсутні значення

У визначенні кожного стовпця вказується, чи допускається збереження в ньому значень NULL. Для задання значень елементів стовпців за замовчуванням, у їх визначенні використовується ключова інструкція DEFAULT таким чином:

```
ім'я_стовпця тип DEFAULT значення
або
ім’я_стовпця тип NOT NULL DEFAULT значення
```

Визначення первинного та зовнішнього ключів

В інструкції CREATE TABLE вказується також інформація про первинний ключ та її зв’язках з іншими таблицями бази даних. Ця інформація міститься в частині PRIMARI KEY та FOREIGN KEY.

В частині PRIMARY KEY задається стовпець чи стовпці, які утворюють первинний ключ таблиці. Цей стовпець чи стовпці є унікальними ідентифікаторами рядків таблиці. СУБД автоматично слідкує за тим, щоб первинний ключ кожного рядка таблиці містив унікальне значення. Крім того, у визначенні рядків первинного ключа має бути вказано, що вони не можуть містити значення NULL.

В частині FOREIGN KEY задається зовнішній ключ таблиці і визначається зв'язок, який задається. В ньому вказуються:

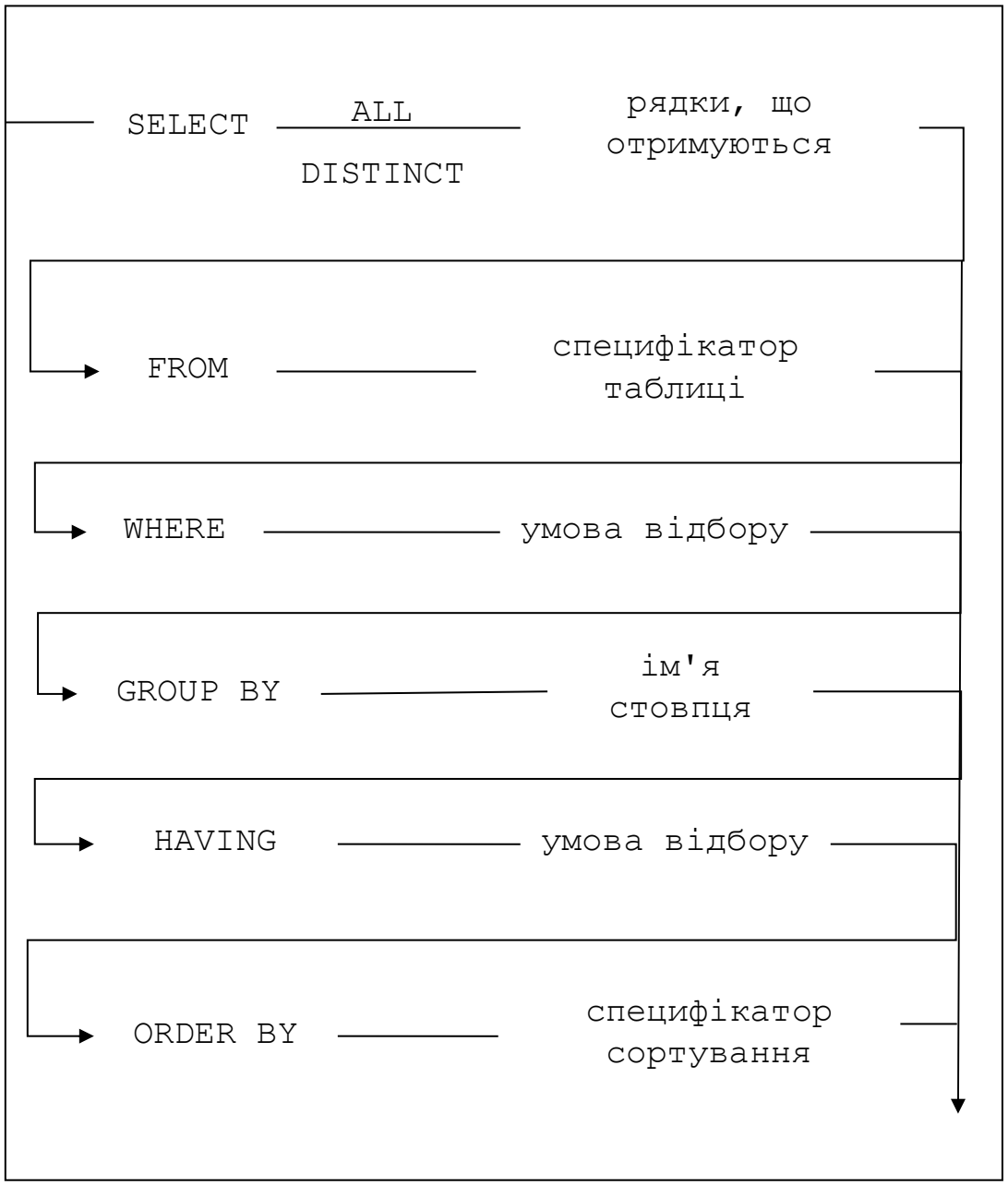
– стовпець чи стовпці створюваної таблиці, які утворюють зовнішній ключ;

– таблиця, зв'язок з якою створюється.

4.6. Прості запити. Інструкція SELECT

Інструкція SELECT отримує інформацію з бази даних та повертає її у вигляді таблиці результатів запиту.

Синтаксична діаграма інструкції SELECT:



В частині SELECT вказується список стовбців, які мають бути отримані у результаті виконання запиту. Стовпці можуть містити значення, отримані з стовпців таблиць бази даних, або можуть обчислюватися під час виконання запиту.

В частині FROM вказується список таблиць, які місять елементи даних, до яких звертається запит.

В частині WHERE міститься умова для відбору рядків, які будуть включені у результат запиту.

Блок GROUP BY дозволяє створити підсумковий запит. Звичайний запит включає в результати запиту по одному запису для кожного рядка із таблиці. Підсумковий запит, в свою чергу, спочатку групує рядки бази даних за визначеною ознакою, а потім включає в результат запиту один підсумковий рядок для кожної групи.

Блок HAVING показує, що в результаті запиту, необхідно включити тільки деякі групи, створені за допомогою GROUP BY. В цій частині для відбору груп використовується умова відбору.

Блок ORDER BY впорядковує результати запиту на основі даних, що містяться в одному чи декількох стовпцях.

Блок SELECT

В частині SELECT необхідно вказати елементи даних, які будуть отримані в результаті виконання запиту. Ці елементи задаються у вигляді списку стовпців, розділених комами. Для кожного елемента із цього списку в таблиці результатів буде створений стовпець. Стовпець результуючої таблиці може являти собою:

– ім'я стовпця, яке відповідає стовпцю однієї з таблиць, які перераховані в частині FROM;

– константу, яка показує, що в кожному рядку результату запиту має бути одне і те ж значення;

– вираз, який показує, що СУБД має обчислити значення за формулою, визначеною у виразі.

Вирази для обчислення значень певних стовпців можуть містити операції додавання, віднімання, множення та ділення. Тут також можна використовувати дужки.

Для того, щоб отримати всі стовпці таблиці, замість списку стовпців можна використовувати символ зірочки (*).

Якщо із таблиці-результату запиту необхідно прибрати рядки, які містять однакові значення, то в частині SELECT перед списком стовпців необхідно вказати предикат DISTINCT, що забезпечить уникнення повторів при виводі результату.

Блок FROM

Блок FROM містить список специфікаторів таблиць, розділених комами. Кожен специфікатор таблиці ідентифікує таблицю, що містить дані, які отримує запит.

Блок WHERE

Для того, щоб вказати які саме рядки необхідно відібрати при виконанні запиту, використовується Блок WHERE. У ньому записують умову відбору рядків. Для кожного з рядків умова відбору може мати одне з трьох значень:

– якщо умова має значення TRUE, то рядок включається в результат відбору;

– якщо умова приймає значення FALSE, то рядок виключається з результатів запиту;

– якщо умова має значення NULL, то рядок виключається із результатів відбору.

Існує багато умов відбору, які дозволяють ефективно створювати різні типи запитів. Основними умовами відбору є:

1. *Порівняння*. Значення одного виразу порівнюється із значенням іншого виразу для кожного рядка даних. Існує шість різних способів порівняння виразів:

= , < > , < , <= , > , >=.

Результатом виконання СУБД порівняння двох виразів може бути:

– якщо порівняння істинне, то результат перевірки має значення TRUE;

– якщо порівняння хибне, то результат перевірки має значення FALSE;

– якщо хоча б один з двох виразів має значення NULL, то результатом перевірки буде NULL.

2. *Перевірка на належність діапазону значень.* Перевіряється чи потрапляє вказане значення в визначений діапазон. Схематично таку форму умови відбору можна зобразити так:

```
        вираз, що перевіряється
  BETWEEN нижня межа AND верхня межа
                або
        вираз, що перевіряється
NOT BETWEEN нижня межа AND верхня межа
```

При такій перевірці верхня та нижня межі вважаються частиною діапазону.

В деяких СУБД визначені такі правила обробки значення NULL в перевірці BETWEEN:

– якщо вираз, що перевіряється має значення NULL або якщо обидва виразів, які визначають діапазон, рівні NULL, то і перевірка BETWEEN повертає NULL;

– якщо вираз, що визначає нижню межу діапазону, має значення NULL, то перевірка BETWEEN повертає FALSE, коли значення, що перевіряється більше, ніж верхня межа діапазону, і NULL в протилежному випадку;

– якщо вираз, що визначає верхню межу діапазону, має значення NULL, то перевірка BETWEEN повертає FALSE, коли значення, що перевіряється менше, ніж нижня межа діапазону, і NULL в протилежному випадку.

3. *Перевірка на входження до множини.* Перевіряється, чи співпадає значення виразу з одним із значень заданої множини. Схематично таку форму умови відбору можна зобразити так:

```
вираз, що перевіряється IN (список констант
відокремлених комами)

або

вираз, що перевіряється NOT IN (список констант
відокремлених комами)
```

4. *Перевірка на відповідність шаблону.* Перевіряється чи відповідає рядкове значення, яке міститься в стовпці певному шаблону. Схематично таку форму умови відбору можна зобразити так:

```
ім'я стовпця LIKE шаблон

або

ім'я стовпця NOT LIKE шаблон
```

Шаблон являє собою рядок, в який може входити один або більше підстановочних знаків. В SQL використовуються такі підстановочні знаки:

1) % – співпадає з будь-якою послідовністю з нуля чи більше символів;

2) _ (символ підкреслення) – співпадає з будь-яким окремим символом.

У випадку, коли підстановочний знак може виявитися елементом рядка, для побудови шаблону використовуються символи пропуску. Коли, в шаблоні зустрічається такий символ, символ, який слідує за ним, вважається не підстановочним. Структура умови в такому випадку є наступною:

```
ім'я стовпця LIKE шаблон ESCAPE символ пропуску
або
ім'я стовпця NOT LIKE шаблон ESCAPE символ пропуску
```

Наприклад, якщо шаблон містить символ %, то умова буде такою:

```
WHERE name LIKE 'A$%BC' ESCAPE '$'
```

У цьому випадку '$' є символом пропуску і символ %, який слідує після нього є простим елементом рядка.

5. *Перевірка на рівність значенню* NULL. Значення NULL дозволяє застосовувати тризначну логіку в умовах відбору. У випадку, коли необхідно явно перевірити значення стовпців на рівність NULL використовується така структура умови:

```
ім'я стовпця IS NULL
або
ім'я стовпця IS NOT NULL
```

Дана перевірка завжди повертає значення TRUE або FALSE.

Перераховані прості умови відбору, після застосування до деякого рядка повертають значення TRUE, FALSE або NULL. За допомогою правил логіки ці прості умови можна об'єднувати в більш складні, використовуючи при цьому логічні операції AND, OR, NOT. Їх таблиці істинності наведені нижче:

Таблиця істинності оператора AND

AND	**TRUE**	**FALSE**	**NULL**

TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Таблиця істинності оператора OR

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Таблиця істинності оператора NOT

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Оператор NOT володіє найвищим пріоритетом, наступний пріоритет має оператор AND, найнижчий – OR.

Блок ORDER BY

Для впорядкування результатів запиту використовується блок ORDER BY. Структура блоку є такою:

```
ORDER BY ім'я/порядковий номер стовпця ASC/DESC
```

При впорядкуванні можна обирати зростаючий (ASC) або спадний (DESC) порядок. За замовчуванням дані сортуються по зростанню.

4.7. Підсумкові запити на вибірку

Статистичні (агрегатні) функції

Для проведення підсумків по інформації, яка міститься в базі даних, застосовуються статистичні (агрегатні) функції. Статистична функція приймає в якості аргументу будь-який стовпець даних, а повертає одне значення. Існують такі статистичні функції:

Статистичні функції

Функція	Значення
`sum(стовпець/вираз)`	обчислює суму всіх значень
`avg(стовпець/вираз)`	обчислює середнє всіх значень
`min(стовпець/вираз)`	знаходить найменше серед всіх значень
`max(стовпець/вираз)`	знаходить найбільше серед всіх значень
`count(стовпець)`	підраховує кількість значень, що містяться в стовпці
`count(*)`	підраховує кількість рядків в таблиці - результату запиту

Аргументом статистичної функції може бути ім'я стовпця або арифметичний вираз, що містить імена стовпців, арифметичні операції, числові константи.

Структура статистичних функцій наведена нижче:

```
sum (вираз) або sum (DISTINCT ім'я_стовпця)
avg (вираз) або avg (DISTINCT ім'я_стовпця)
min (вираз)
max (вираз)
count (вираз) або count (DISTINCT ім'я_стовпця)
count (*)
```

Для видалення рядків, що повторюється використовується предикат DISTINCT. Проте, в одному запиті цей предикат можна використовувати не більше одного разу.

Запити з групуванням (блок GROUP BY)

Запит, який включає в себе блок GROUP BY, називається запитом з групуванням, оскільки він об'єднує рядки початкових таблиць в групи і для кожної групи рядків генерує один рядок в таблиці результатів запиту. Стовпці, вказані в цьому блоці, називаються стовпцями групування.

На запити, в яких використовується групування, накладаються додаткові обмеження. Стовпці з групуванням мають бути реальними стовпцями таблиць, перерахованими в блоці FROM. Не можна групувати рядки на основі виразу, значення якого обчислюється.

Умови відбору груп (блок HAVING)

Так само як блок WHERE використовується для відбору окремих рядків, які беруть участь у запиті, блок HAVING можна використовувати для відбору груп рядків. Його формат відповідає формату блоку WHERE.

Блок HAVING майже завжди використовується з блоком GROUP BY, проте синтаксис запиту SELECT цього не вимагає. Якщо блок HAVING використовується без блоку GROUP BY, то СУБД розглядає результати запиту як одну групу.

4.8. Об'єднання результатів запитів

Для об'єднання результатів запитів використовують службове слово UNION за такою схемою:

```
<Запит1>
```

```
UNION [ALL]
<Запит2>
UNION [ALL]
<Запит3>
…
```

Оператор UNION об'єднує вихідні рядки кожного з запитів в один результуючий набір. Якщо визначений параметр ALL, то зберігаються всі дублікати рядків, в іншому випадку, в результуючому наборі зберігаються тільки унікальні рядки. В загальному випадку, можливим є об'єднання будь якої кількості запитів.

Умови застосування оператора є такими:

– Кількість вихідних стовпців кожного з запитів має бути однакова.

– Вихідні стовпці кожного з запитів мають бути порівнянні між собою (в порядку їх слідування) по типам даних.

– В результуючому наборі використовуються імена стовпців, задані в першому запитів.

– Блок ORDER BY застосовується до результату об'єднання, тому може бути вказаний тільки в кінці всього запиту.

4.9. Додавання нових даних. Інструкція INSERT

Існує декілька способів додавання нових рядків в БД, серед них:

– однорядкова інструкція INSERT, яка дозволяє додати в таблицю один новий рядок;

– багаторядкова інструкція INSERT, забезпечує отримання рядків із одної частини БД та додавання їх в іншу частину.

Інструкція INSERT додає в таблицю новий рядок або групу рядків. При цьому, значення стовпців можуть являти собою константи, а можуть бути результатом виконання підзапиту. У

першому випадку для вставки кожного рядка виконується окремий оператор, а в другому – буде додано стільки рядків, скільки повертає підзапит.

Синтаксична структура інструкції INSERT наведена нижче:

```
INSERT INTO ім'я_таблиці (ім'я_стовпця1,
          ім'я_стовпця2,… )
    VALUES (значення1, значення2,…)
                або
INSERT INTO ім'я_таблиці (ім'я_стовпця1,
          ім'я_стовпця2,… )
              підзапит
                або
INSERT INTO ім'я_таблиці (ім'я_стовпця1,
          ім'я_стовпця2,… )
            DEFAULT VALUES
```

Варто зазначити, у тому випадку, коли список стовпців запису співпадає з структурою таблиці, тобто запис забезпечує значення для всіх стовпців, зберігаючи їх порядок, – перераховувати назви стовпців необов'язково. Тоді структура інструкції матиме вид:

```
INSERT INTO ім'я таблиці VALUES (значення 1,…)
                або
    INSERT INTO ім'я таблиці підзапит
                або
INSERT INTO ім'я таблиці DEFAULT VALUES
```

У випадку багаторядкової інструкції INSERT, на вкладений запит, як правило, накладаються деякі обмеження:

– в запит не можна включати частину ORDER BY;

– таблиця результатів запиту має містити таку ж кількість стовпців, як і список стовпців в інструкції INSERT.

4.10. Видалення існуючих даних. Інструкція DELETE

Інструкція DELETE видаляє вибрані рядки з одної таблиці. Синтаксична структура інструкції є такою:

```
DELETE ім'я таблиці WHERE умова відбору
```

У блоці DELETE вказується таблиця, що містить рядки, які потрібно видалити. У блоці WHERE вказують критерії відбору рядків, які будуть видалені. У випадку, коли блок WHERE відсутній – інструкція видаляє всі рядки таблиці.

Також, допустимим є використання вкладеного запиту для задання умови відбору.

4.11. Обновлення існуючих даних. Інструкція UPDATE

Інструкція UPDATE обновляє значення одного чи декількох стовпців у вибраних рядках однієї таблиці. Синтаксична структура інструкції є такою:

```
UPDATE ім'я_таблиці SET ім'я_стовпця1=вираз1,
    ім'я_стовпця2=вираз2…
WHERE умова відбору
```

В інструкції вказується цільова таблиця, яка має бути модифікована. В блоці SET вказується, які стовпці мають бути модифікованими. Блок WHERE відбирає рядки таблиці для обновлення. Якщо цей блок відсутній, то обновляться всі рядки таблиці.

Також, допустимим є використання вкладених запитів для задання умов відбору рядків.

4.12. Видалення таблиці. Інструкція DROP TABLE

Видалення таблиці виконується за допомогою інструкції DROP TABLE, синтаксична структура якої є наступною:

```
DROP   TABLE   ім'я таблиці
```

Інструкція містить ім'я таблиці, яка видаляється. Після виконання інструкції, визначення бази даних та весь її вміст втрачаються і не можуть бути відновленими автоматично.

4.13. Зміна визначення таблиці. Інструкція ALTER TABLE

За допомогою інструкції ALTER TABLE можна виконати такі дії:

- додати в таблицю стовпці;
- видалити стовпці з таблиці;
- змінити значення за замовчуванням для стовпця;
- додати чи видалити первинний ключ таблиці;
- додати чи видалити зовнішній ключ таблиці;
- додати чи видалити умову унікальності;
- додати чи видалити умову на значення.

Синтаксична структура інструкції є такою:

```
ALTER TABLE ім'я_таблиці змістовний блок
```

Додавання стовпця

Для додавання стовпця до існуючої таблиці до інструкції `ALTER TABLE` дописують блок `ADD` та частину з визначенням стовпця в такому ж форматі, як в інструкції `CREATE TABLE`. Новий стовпець додається в кінець таблиці. Синтаксична структура інструкції в цьому випадку має вид:

```
ALTER TABLE ім'я_таблиці ADD визначення стовпця
```

Видалення стовпця

За допомогою інструкції `ALTER TABLE` для видалення стовпця необхідно використати блок `DROP`. Синтаксична структура інструкції є такою:

```
ALTER TABLE ім'я_таблиці DROP ім'я_стовпця
```

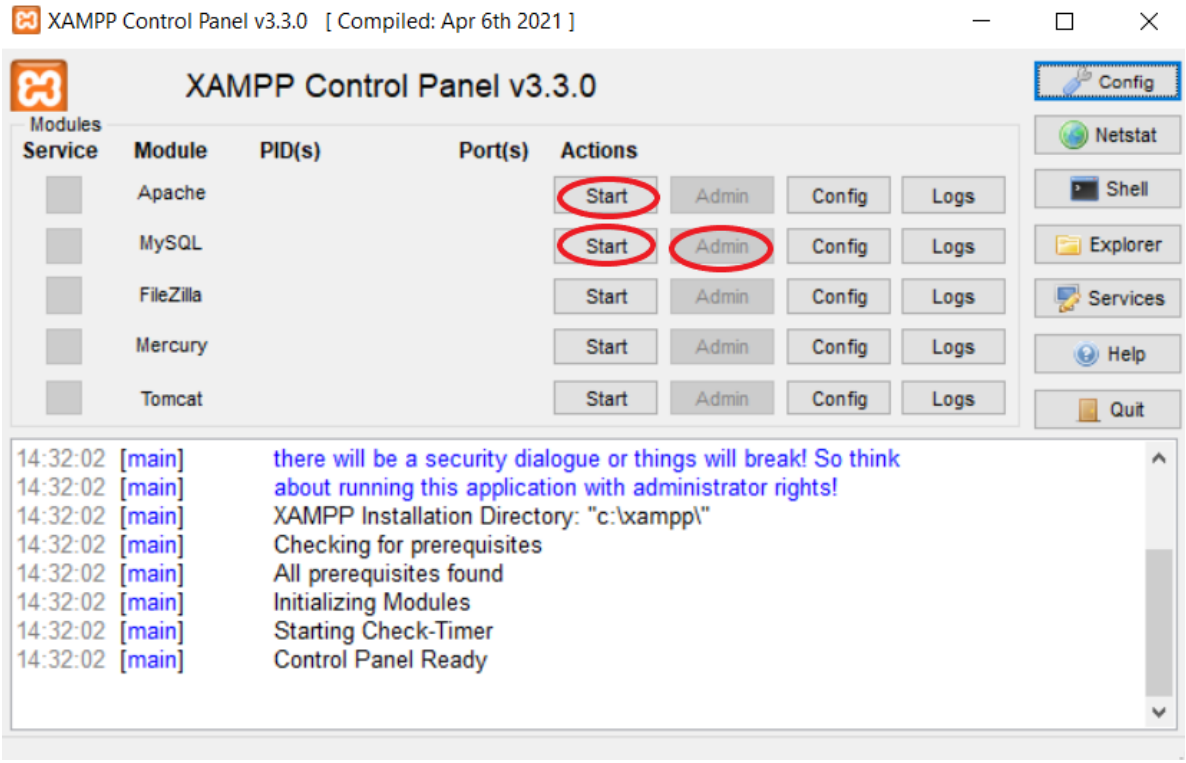
5. СТВОРЕННЯ ТА ОБРОБКА РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ В СИСТЕМІ КЕРУВАННЯ БАЗАМИ ДАНИХ MYSQL

Для спрощення установки та подальшого використання системи управління базами даних (СУБД) MySQL рекомендується встановити один з вільно розповсюджуваних WAMP (Windows, Apache, MySQL, PHP) або LAMP (Linux, Apache, MySQL, PHP) серверів, наприклад OpenServer або XAMPP. В подальшому буде використовуватися сервер XAMPP.

5.1. Веб-додаток phpMyAdmin. Створення бази даних

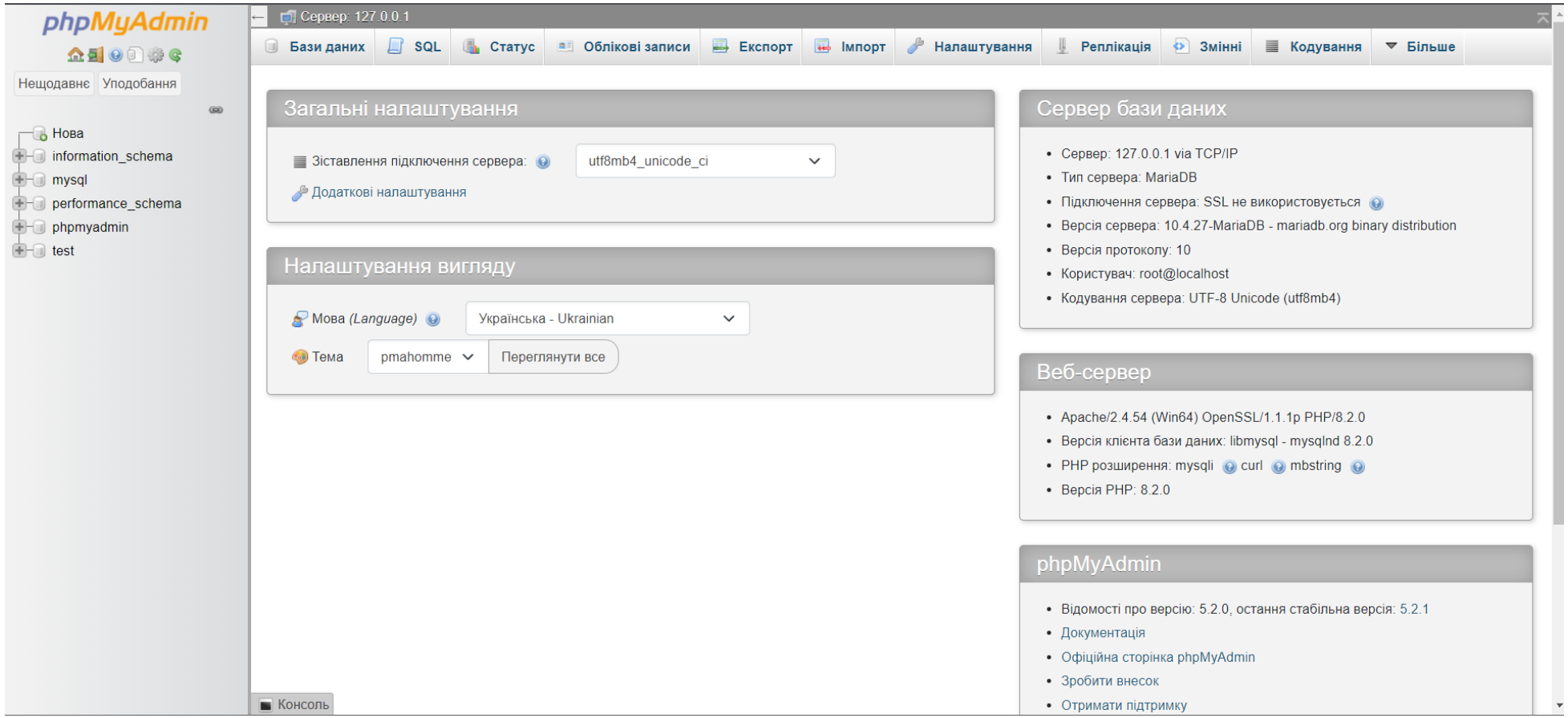
Розглянемо основні прийоми роботи з реляційними базами даних у веб-дадатку phpMyAdmin на сервері XAMPP, який розповсюджується за ліцензією GNU General Public License. Варто зауважити, що MySQL – це вільна система керування реляційними базами даних компанії Oracle, яка також поширюється за ліцензією GNU GPL.

Для запуску додатку phpMyAdmin, необхідно запустити на комп’ютері серверний додаток, а саме:

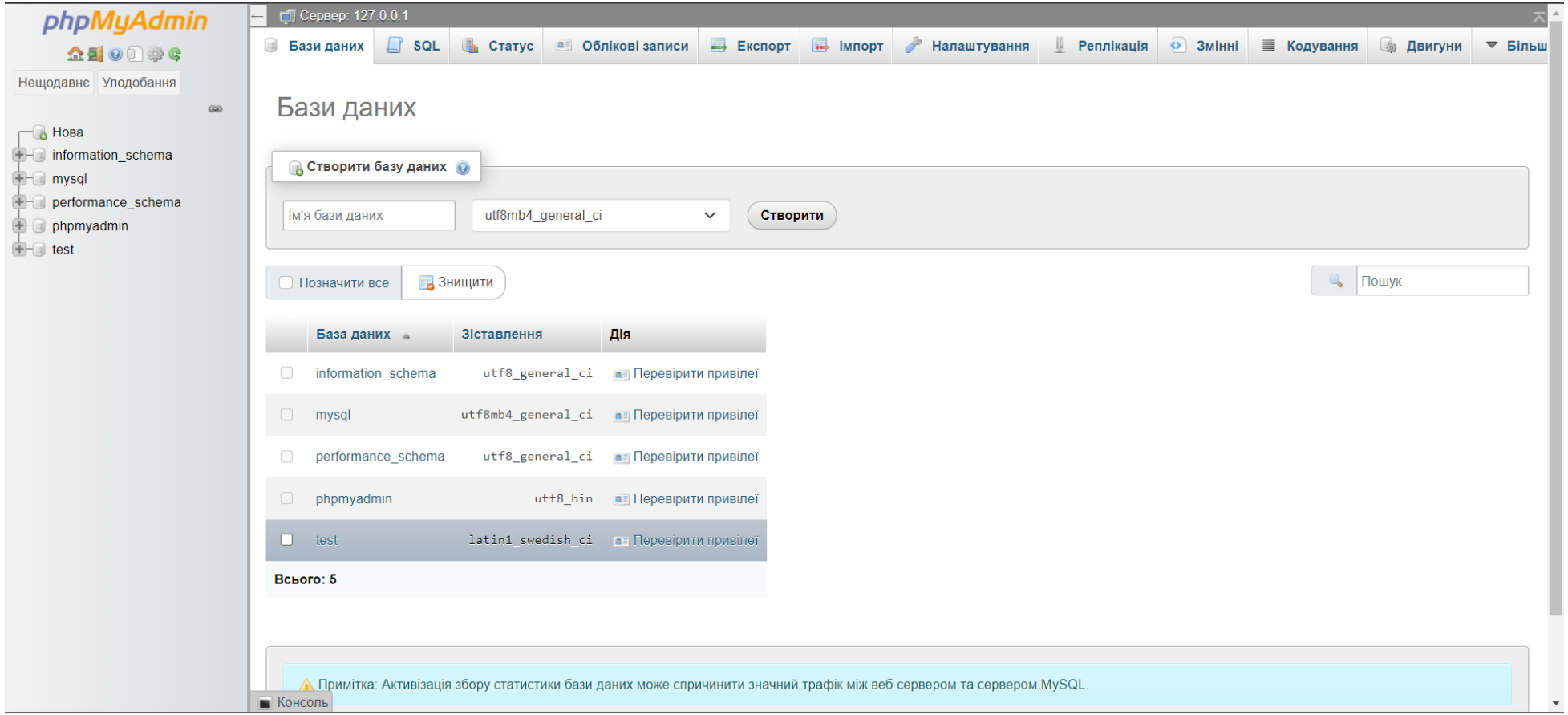


Локальний сервер можна скачати за адресою:
https://www.apachefriends.org/

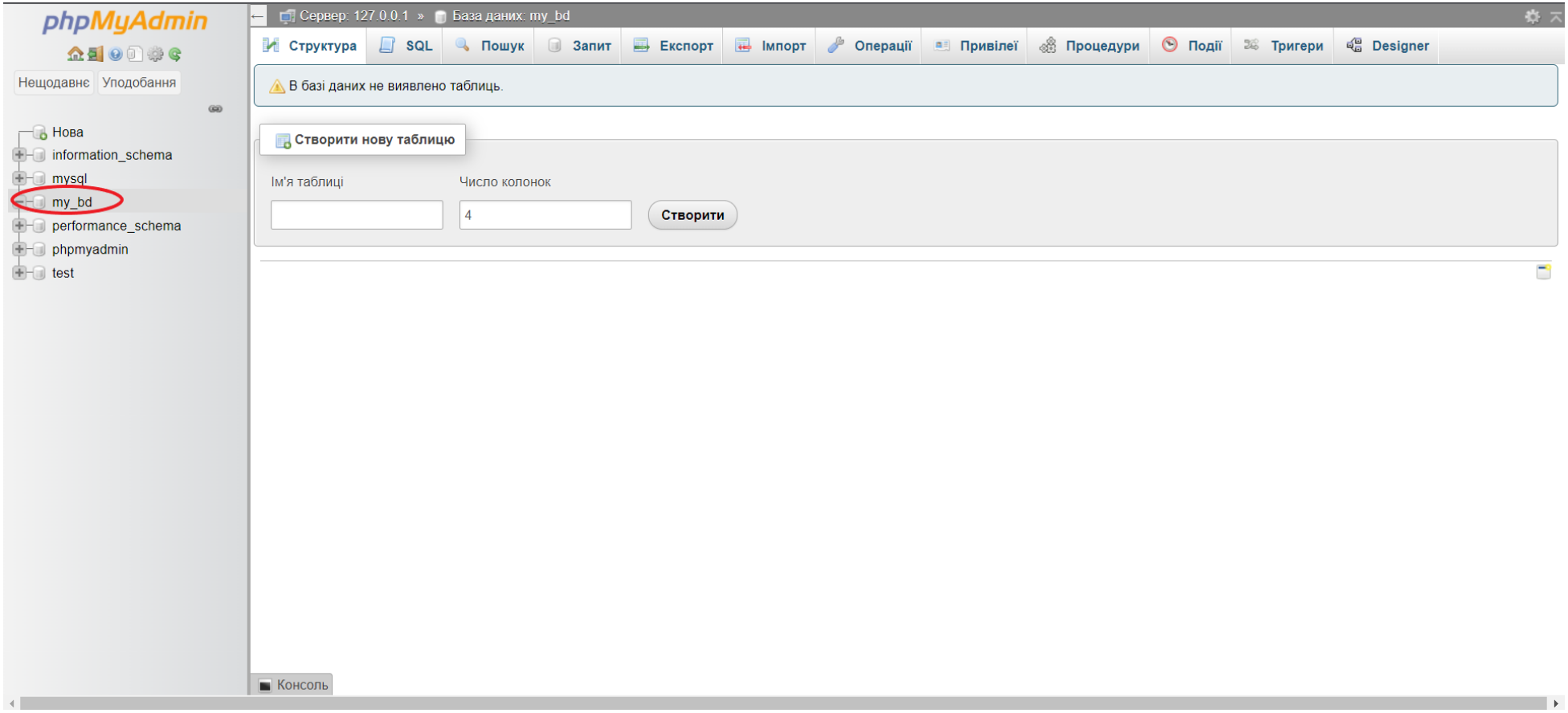
При запуску веб-додатку phpMyAdmin з’являється таке вікно:



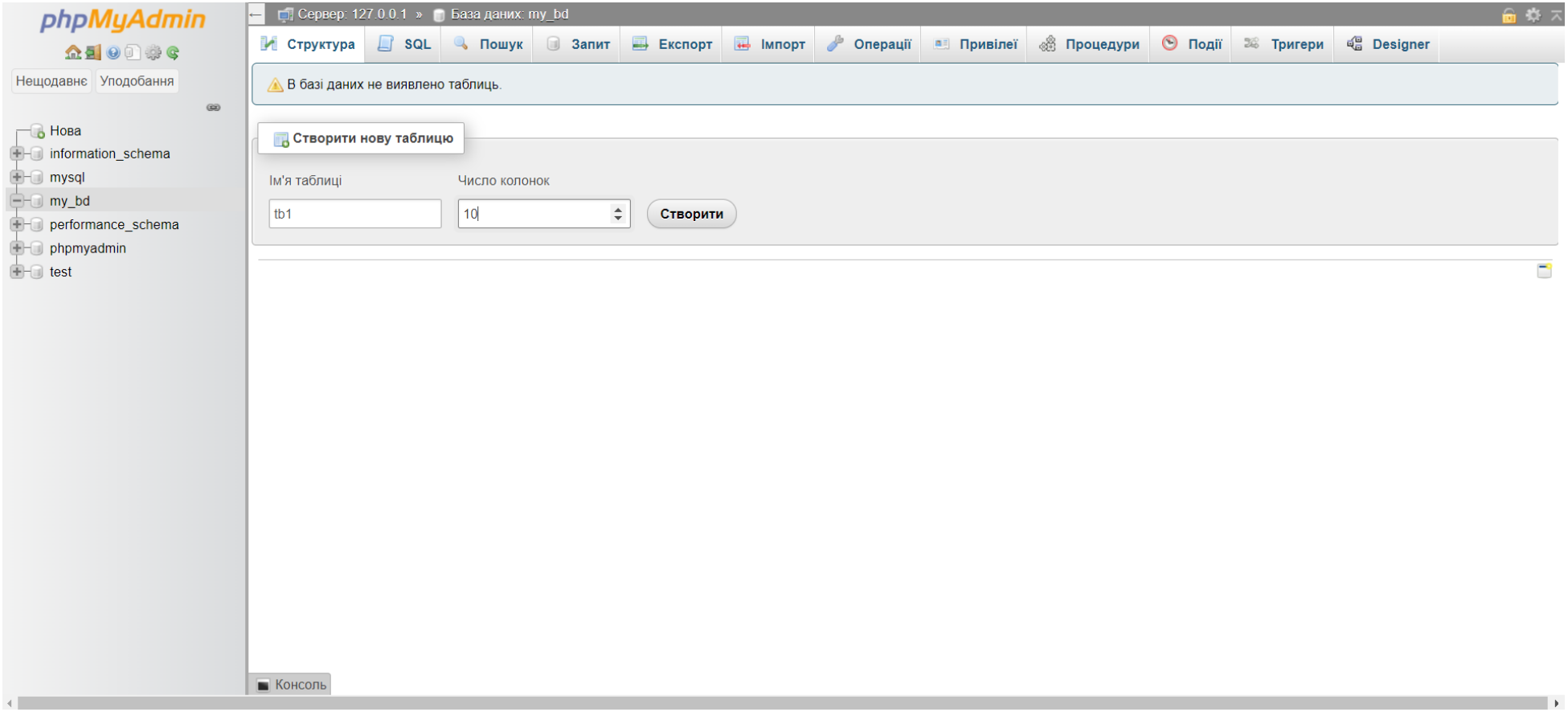
На першому етапі необхідно створити власну базу даних, де і будуть в подальшому розміщуватися таблиці з даними. Для цього потрібно перейти на вкладку «Бази даних» і ввести назву власної БД:



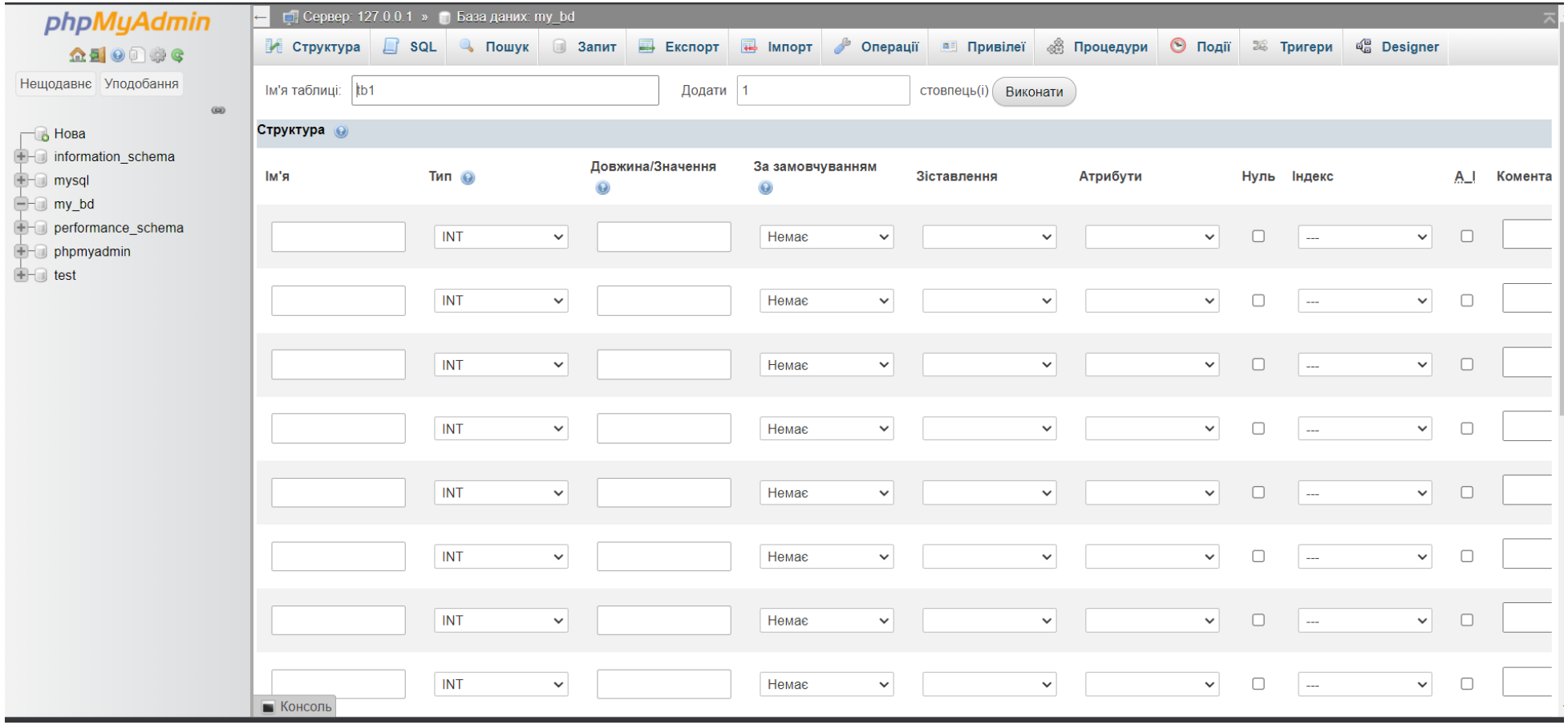
Після натиснення на кнопку «Створити», створена база даних з’являється у переліку баз даних, які є в СУБД. Для подальшого створення таблиць в базі даних, необхідно перейти на неї:



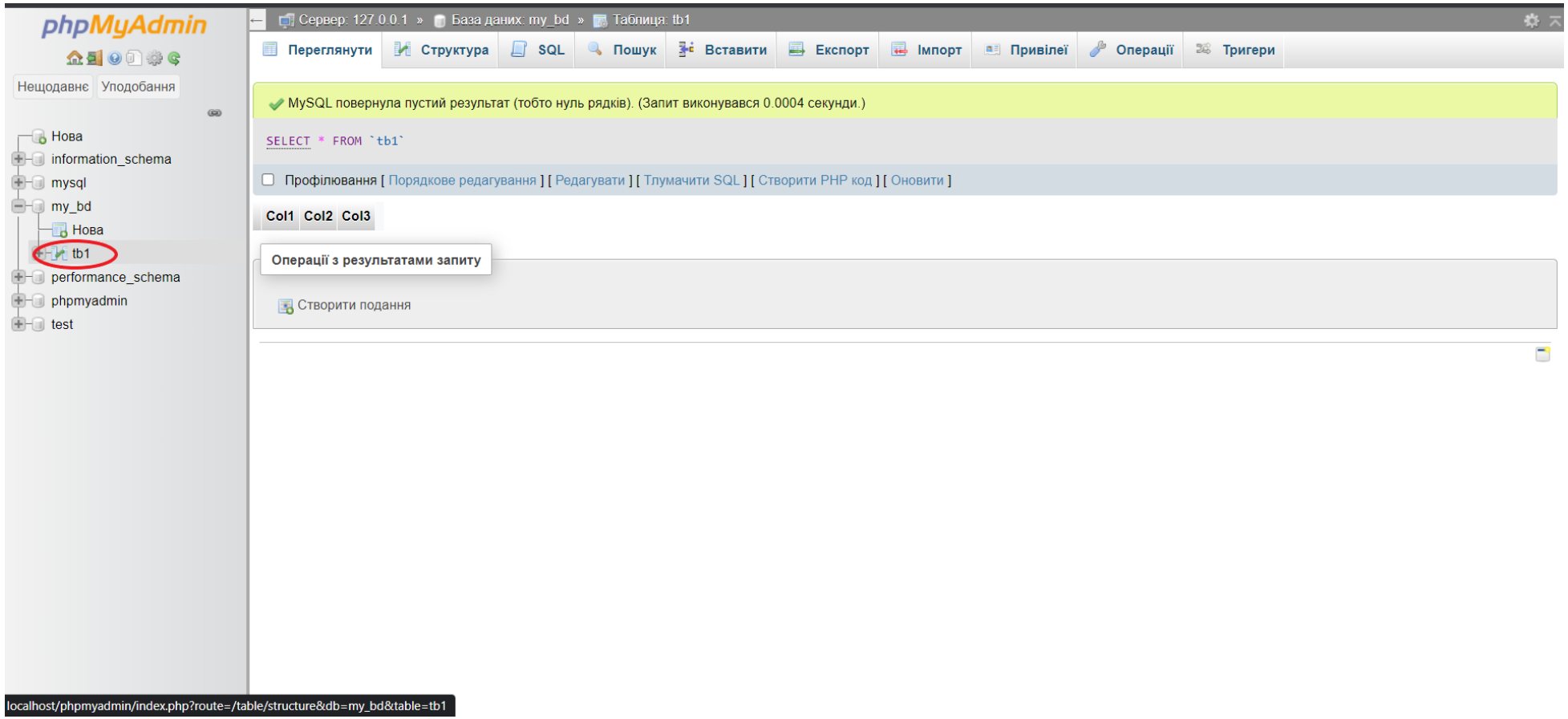
Далі, процес створення таблиць починається з зазначення їх назви та визначення кількості стовпців:



Наступний етап полягає у визначенні структури таблиці: назв її стовпчиків, типів полів, задання ключових атрибутів тощо.

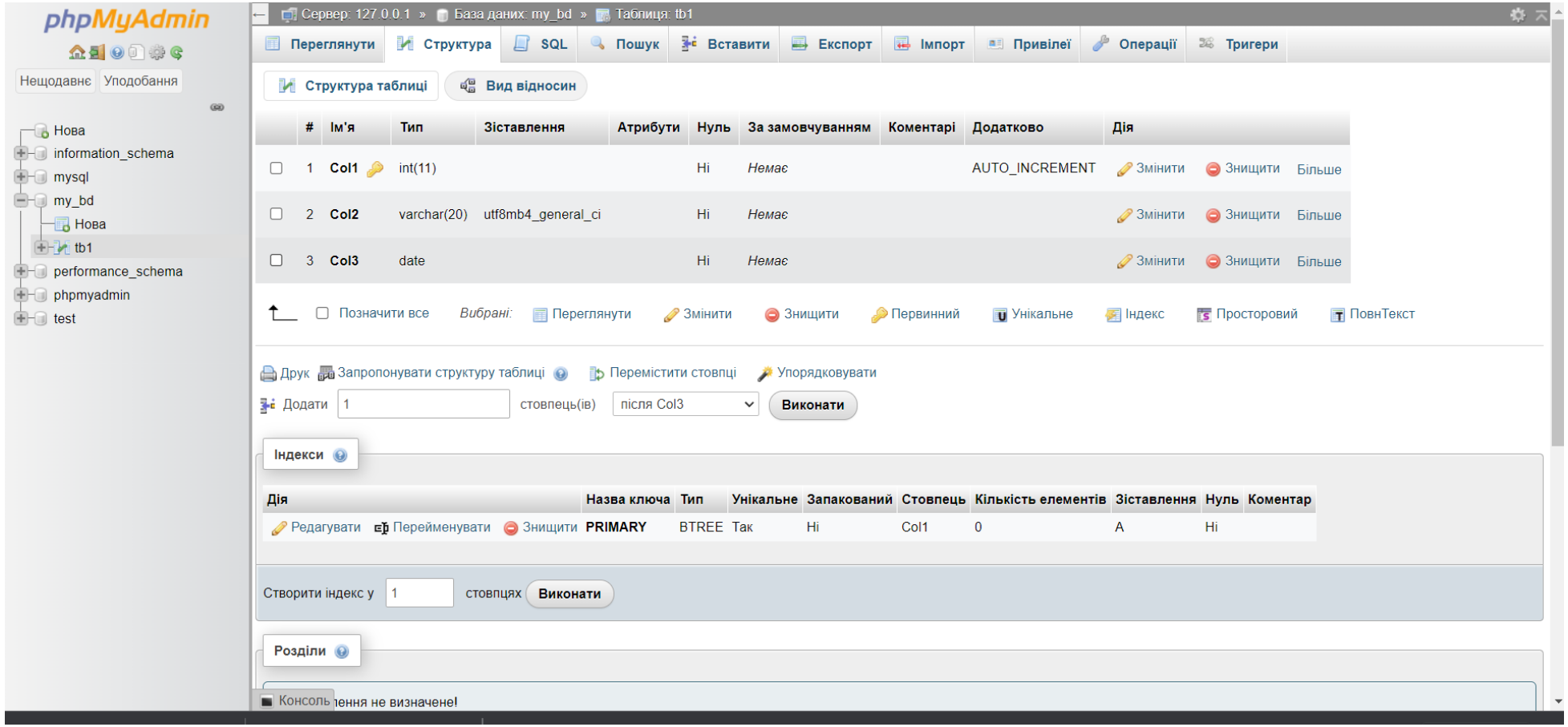


Створена таблиця з’являється у переліку таблиць бази даних:

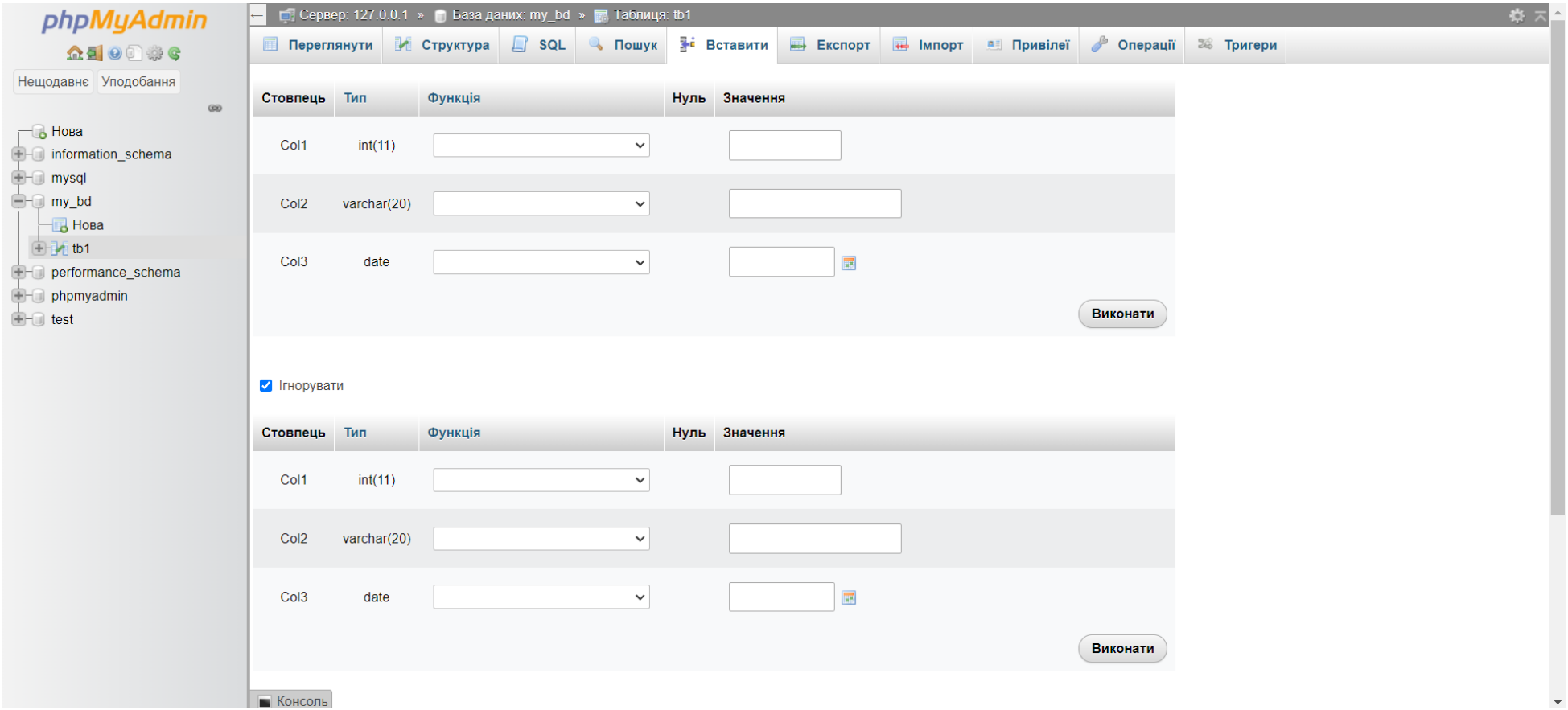


Аналогічним чином можна створити всі необхідні таблиці.

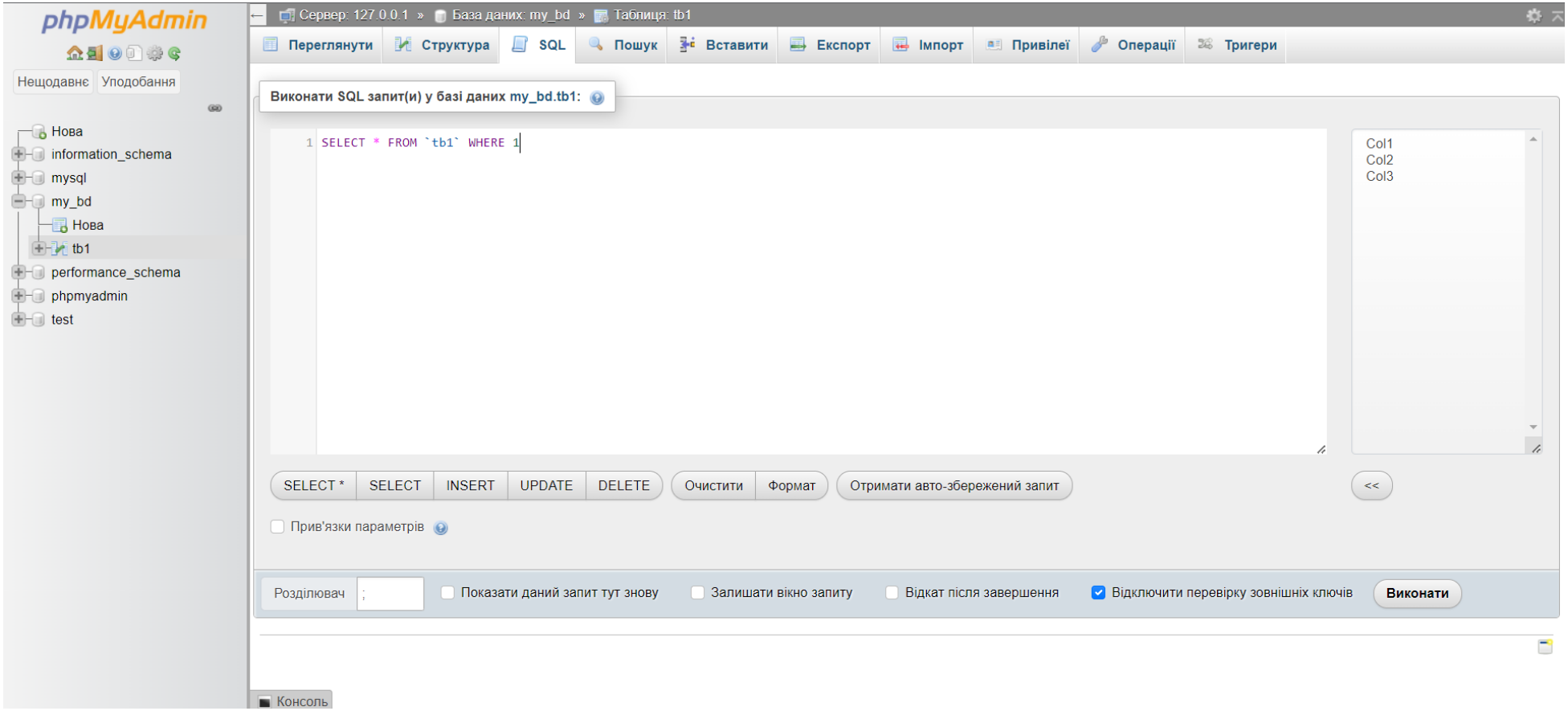
Після переходу на створену таблицю з’являється режим редагування та роботи з таблицями:



Додавання записів в таблицю можливо в інтерактивному режимі при переході на вкладку «Вставити»:



Для запису та виконання запиту SQL необхідно перейти на вкладку «SQL»:



5.2. Структура робочої реляційної бази даних

На далі в посібнику всі задачі будуть стосуватися робочої бази даних, яка складатиметься з трьох таблиць:

1. Таблиця **`Stud`**:

Назва поля	Зміст	Тип даних
`NumS`	Порядковий номер студента	Ціле число
`Name`	ПІП студента	Текстова/рядкова величина
`Data`	Дата народження	Дата
`Fakult`	Факультет	Текстова/рядкова величина
`Kurs`	Курс	Ціле число
`E1`	Оцінка1	Ціле число
`E2`	Оцінка2	Ціле число
`E3`	Оцінка3	Ціле число
`Stip`	Стипендія	Дійсне число

2. Таблиця **`Gurt`**:

Назва поля	Зміст	Тип даних
`NumG`	Порядковий номер гуртка	Ціле число
`NameG`	Назва гуртка	Текстова/рядкова величина
`NameK`	Керівник гуртка	Текстова/рядкова величина
`DateZ`	Дата заснування гуртка	Дата

3. Таблиця **`Rob_Gurt`** з такою структурою**:**

Назва поля	Зміст	Тип даних
`NumS`	Порядковий номер студента	Ціле число
`NumG`	Порядковий номер гуртка	Ціле число
`DateP`	Дата прийому в гурток	Дата
`Robota`	Оцінка роботи студента	Текстова/рядкова величина

При створенні та заповненні вказаних таблиць необхідно забезпечити, щоб вони перебували між собою у таких відношеннях:

1. `Stud` і `Rob_Gurt` у відношенні один до багатьох: `Stud.NumS` – первинний ключ, `Rob_Gurt.NumS` – зовнішній ключ.

2. `Gurt` і `Rob_Gurt` у відношенні один до багатьох: `Gurt.NumG` – первинний ключ, `Rob_Gurt.NumG` – зовнішній ключ.

Тобто, при заповненні таблиць необхідно дотримуватися вимог цілісності сутностей та цілісності посилань.

5.3. Створення користувача MySQL і налаштування прав доступу

За замовчанням у MySQL буде лише один користувач — root. Це адміністратор з доступом до всіх баз даних та таблиць усередині цих баз. Щоб створити користувача MySQL спочатку потрібно підключитися до сервера SSH. Після підключення до сервера авторизуйтесь як root користувач MySQL за допомогою команди:

```
mysql -u root –p
```

Після цього необхідно ввести пароль. Зазвичай пароль користувача root є порожнім, тому можна просто натиснути Enter.

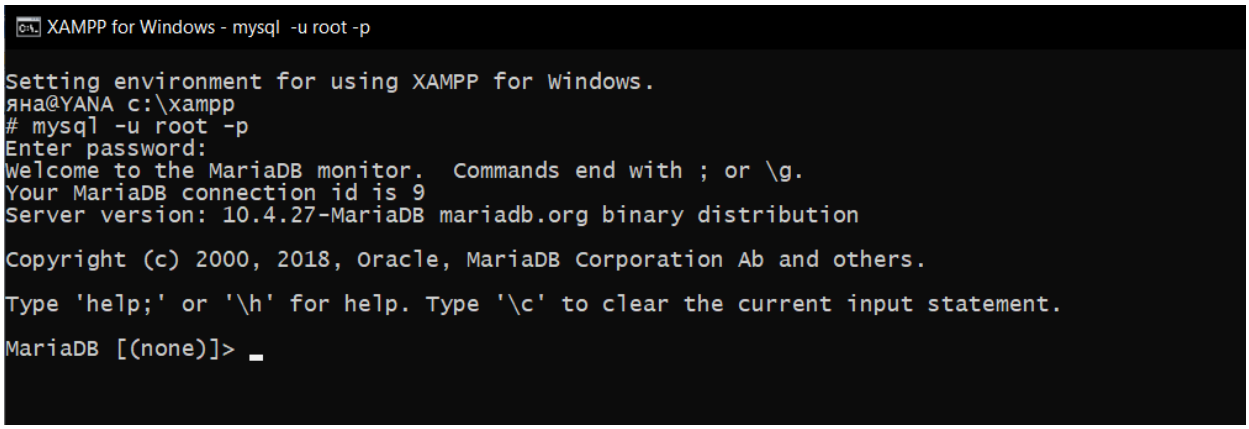


Рисунок – Підключення до MySQL

Усі команди MySQL повинні закінчуватися символом “ ; ”. Цей знак означає закінчення запиту.

Безпосередньо для створення користувача MySQL використовується команда `CREATE USER` з таким синтаксисом:

CREATE USER “ім’я_користувача“@”хост” IDENTIFIED BY “пароль“;

Параметр «хост» у команді означає пристрій, з якого користувач зможе підключатися до сервера MySQL. При підключенні вказують одне з наступних значень:

- **localhost** – означає, що доступ до MySQL з’явиться у користувача лише після того, як він підключиться до сервера;
- **IP-адреса пристрою в мережі**, з якого буде здійснюватися підключення до MySQL;
- **%** – означає, що користувач матиме доступ до MySQL з будь-якого пристрою.

В останніх двох варіантах додатково підключатися до сервера не потрібно, але доведеться спочатку дозволити віддалені підключення в налаштуваннях MySQL.

Найбезпечнішим варіантом підключення є підключення з параметром localhost.

Приклад 5.3.1.

Створити користувача MySQL *my_bd_administrator* з хостом *localhost* та паролем *Hgy467Hkgd*

```
1 CREATE USER
2 'my_bd_administrator'@'localhost'
3 IDENTIFIED BY
4 'Hgy467Hkgd';
```

Для того щоб користувач зміг щось зробити з БД потрібно призначити йому привілеї. Для цього використовується команду GRANT з таким синтаксисом:

GRANT ПРАВО, ПРАВО ON база_даних.таблиця TO “ім’я_користувача“@”хост“;

Право — це, наприклад, можливість редагувати базу даних або створювати нового користувача.

Список прав, які часто використовуються:

- **ALL** — дати всі права до бази даних, за винятком GRANT OPTION. Якщо ви не вкажете назву конкретної бази даних, користувач отримає повний доступ до всього сервера MySQL;
- **CREATE** — право створювати нові бази даних та таблиці;
- **DELETE** — право видаляти рядки з таблиці;
- **DROP** — право видаляти бази даних або таблиці;
- **GRANT OPTION** — право призначати чи відбирати права. Але вдасться дати або відібрати тільки ті права, якими володіє користувач, який використовує команду;
- **INSERT** — право створювати рядки у таблиці;
- **SELECT** — право переглядати рядки в таблицях;
- **UPDATE** — право змінювати зміст рядків у таблицях.

Приклад 5.3.2.

Надати користувачеві *my_bd_administrator* право на створення таблиць у базі даних *my_bd*. А також дозволити додавати записи до цих таблиць.

```
1 GRANT CREATE, INSERT
2 ON my_bd.* TO
3 'my_bd_administrator'@'localhost';
```

Зірочка у прикладі означає «всі таблиці всередині бази даних my_bd».

Приклад 5.3.3.

Надати користувачеві *my_bd_administrator* право тільки на читання таблиць у базі даних *my_bd*.

```
1 GRANT
2 SELECT
3 ON my_bd.* TO
4 'my_bd_administrator'@'localhost';
```

Для того щоб привілеї набули чинності, потрібно перезавантажити їх командою `FLUSH PRIVILEGES`.

```
1 FLUSH PRIVILEGES;
```

Приклад 5.3.4.

Перевірити, які доступи до БД є у користувача m*y_bd_administrator*.

```
1 SHOW GRANTS
2 FOR 'my_bd_administrator'@'localhost';
```

5.4. Зміна прав користувача MySQL

Забрати привілеї, можна за допомогою команди `REVOKE`:

REVOKE ПРАВО ON база_даних.таблиця FROM “ім’я_користувача“@”хост“;

Приклад 5.4.1.

Забрати у користувача *my_bd_administrator* право видаляти таблиці в базі даних *my_bd*.

```
1 REVOKE DROP ON
2 ON my_bd.* FROM
3 'my_bd_administrator'@'localhost';
```

Після чого, знову потрібно перезавантажити привілеї командою `FLUSH PRIVILEGES`.

Щоб видалити користувача MySQL, використовується команда `DROP USER`:

DROP USER “ім’я_користувача“@”хост“;

Ця команда не вимагає додаткового підтвердження. Користувач бази даних видаляється з першого разу і скасувати це не можна.

Приклад 5.4.2.

Видалити користувача *my_bd_administrator*.

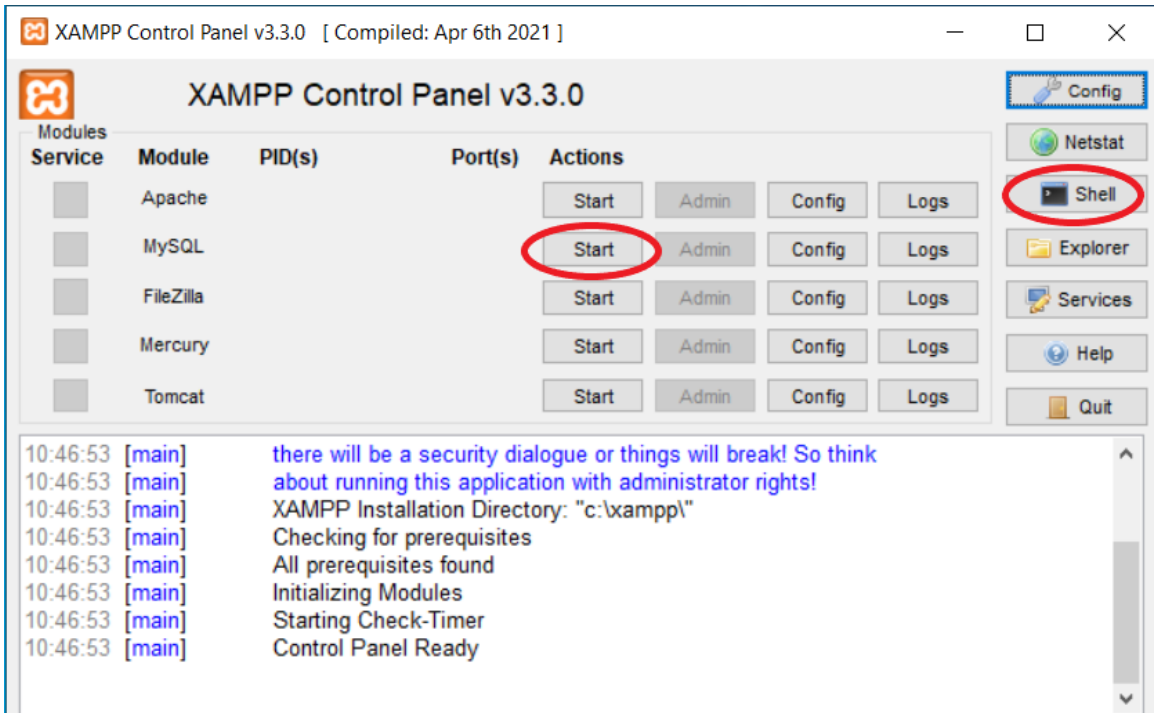
```
1 DROP USER
2 'my_bd_administrator'@'localhost';
```

Наприкінці роботи потрібно відключитися від MySQL-сервера за допомогою команди:

```
\q
```

5.5. Підключення до MySQL та створення бази даних

Для запуску додатку MySQL, необхідно запустити на комп'ютері серверний додаток, а саме:



Після підключення до сервера авторизуйтесь як користувач MySQL:

Mysql –u ім'я_користувача -p

А саме авторизуйтесь як my_bd_administrator користувач MySQL за допомогою команди:

```
mysql -u my_bd_administrator -p
```

Після чого вводимо пароль.

```
# mysql -u my_bd_administrator -p
Enter password: **********
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 47
Server version: 10.4.27-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

Рисунок – Підключення до MySQL

Для того щоб перевірити список всіх бд на MySQL-використовуємо команду SHOW DATABASES:

```
MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| my_bd              |
| mysql              |
| performance_schema |
| phpmyadmin         |
| test               |
+--------------------+
6 rows in set (0.073 sec)

MariaDB [(none)]>
```

Рисунок – Список усіх бд на MySQL

Щоб створити базу даних MySQL, використовується команда CREATE DATABASE:

Приклад 5.5.1.

Створити БД з назвою *my_bd.*

```
1  CREATE DATABASE my_bd;
```

Щоб працювати, з відповідною базою даних потрібно вибрати її окремою командою: USE назва_бази_даних;

```
USE my_bd;
```

Рисунок – Вибір відповідної БД

Інформація в базі даних MySQL зберігається в таблицях. Їх потрібно створювати окремо за допомогою CREATE TABLE.

Приклад 5.5.2.

Створити таблицю з назвою Stud (структура таблиці Stud на ст.84).

```
1  CREATE TABLE Stud(
2  NumS INT AUTO_INCREMENT PRIMARY KEY,
```

```
3   Name VARCHAR(50) NOT NULL,
4   Data DATE NOT NULL,
5   Fakult VARCHAR(50) NOT NULL,
6   Kurs INT NOT NULL,
7   E1 INT NOT NULL,
8   E2 INT NOT NULL,
9   E3 INT NOT NULL,
10  Stip REAL
11  );
```

Якщо в таблиці унікальні значення мають бути в кількох стовпцях, то потрібно записати PRIMARY KEY окремим рядком та додати назви стовпців у дужках:

PRIMARY KEY (назва_стовпця, назва_стовпця, назва_стовпця)

Приклад 5.5.3.

Заповнити таблицю Stud.

Стовпець *NumS* не вказуємо, тому що значення первинного ключа MySQL визначає автоматично завдяки параметру AUTO_INCREMENT.

Підставимо значення для першого рядка:

```
1  INSERT INTO Stud (Name, Data, Fakult,
2                    Kurs,E1,E2,E3,Stip)
3  VALUES('Vovkiv Jana','2005-04-12',
4         'FIT','3','90','90','96','3050');
```

Підставимо значення для другого рядка:

```
1  INSERT INTO Stud (Name, Data, Fakult,
2                    Kurs,E1,E2,E3,Stip);
```

```
3 VALUES('Varha Diana','2003-10-24',
4         'FIT','5','60','70','90','0');
```

Результат

```
MariaDB [my_bd]> SELECT * FROM Stud;
+------+-------------+------------+--------+------+----+----+----+------+
| NumS | Name        | Data       | Fakult | Kurs | E1 | E2 | E3 | Stip |
+------+-------------+------------+--------+------+----+----+----+------+
|    1 | Vovkiv Jana | 2005-04-12 | FIT    |    3 | 90 | 90 | 96 | 3050 |
|    2 | Varha Diana | 2003-10-24 | FIT    |    5 | 60 | 76 | 90 |    0 |
+------+-------------+------------+--------+------+----+----+----+------+
```

Рисунок – Записи, створені у таблиці Stud

Оновлення даних (зміна значень полів у існуючих записах) у БД виконується за допомогою оператора UPDATE.

UPDATE назва_таблиці SET назва_стовпця= "нове_значення" WHERE назва_стовпця= "значення";

Після SET вказується назву стовпця, де потрібно змінити значення. А після WHERE — назва якогось іншого стовпця з цього рядка.

Приклад 5.5.4.

Виправити помилку у даті народження студента під номером 1.

```
1 UPDATE Stud
2 SET Data='2006-04-12'
3 WHERE Nums='1';
```

Приклад 5.5.5.

Видалити з таблиці Stud студента під номером 2.

```
1 DELETE FROM Stud
2 WHERE Nums='2';
```

Приклад 5.5.6.
Додати у таблицю Stud стовпець email.

```
1 ALTER TABLE Stud
2 ADD email VARCHAR (30);
```

Приклад 5.5.7.
Видалити з таблиці Stud стовпець email.

```
1 ALTER TABLE Stud
2 DROP COLUMN email;
```

Приклад 5.5.8.
Змінити у таблиці Stud тип колонки Fakult на varchar(40).

```
1 ALTER TABLE Stud
2 MODIFY COLUMN Fakult VARCHAR(40);
```

Приклад 5.5.9.
Видалити таблицю Stud з БД.

```
1 DROP TABLE Stud;
```

Якщо потрібно очистити всі рядки відразу, але зберегти структуру таблиці, то використовується команда TRUNCATE:

TRUNCATE TABLE назва_таблиці;

Приклад 5.5.10.
Очистити всі рядки таблиці Stud.

```
1 TRUNCATE TABLE  Stud;
```

Приклад 5.5.11. Видалити БД my_bd.

```
1 DROP DATEBESE my_bd;
```

5.6. Прості запити в SQL. Впорядкування результату

Простим назвемо запит, у якому необхідно зробити вибірку з однієї таблиці бази даних без необхідності виконання додаткових обчислень. Структура такого запиту, як правило, прямо випливає з задачі. Розглянемо способи розв'язування деяких задач такого типу.

Приклад 5.6.1.

Вивести інформацію про всіх студентів, які зареєстровані в таблиці `Stud`.

```
1 SELECT *
2 FROM Stud
```

Приклад 5.6.2.

Вивести інформацію про всіх студентів, які зареєстровані в таблиці `Stud`, впорядкувавши дані за прізвищами студентів в алфавітному порядку.

```
1 SELECT *
2 FROM Stud
3 ORDER BY Name
```

Приклад 5.6.3.

Вивести прізвища, факультети та курси всіх студентів, які зареєстровані в таблиці `Stud`.

```
1 SELECT Name, Fakult, Kurs
2 FROM Stud
```

Приклад 5.6.4.

Вивести прізвища керівників гуртків та назви гуртків, якими вони керують, впорядкувавши записи за датою створення гуртка від найновішого до найстарішого.

```
1 SELECT NameK, NameG
2 FROM Gurt
3 ORDER BY DateZ DESC
```

При роботі з полями типу дата необхідно пам'ятати, що мінімальна дата належить найстаршому об'єкту, максимальна – найновішому.

Приклад 5.6.5.

Вивести прізвища студентів, факультети на яких вони навчаються та розмір стипендії, яку вони отримують, впорядкувавши записи спочатку за назвою факультету в алфавітному порядку, а потім за розміром стипендії в порядку спадання.

```
1 SELECT Name, Fakult, Stip
2 FROM Stud
3 ORDER BY Fakult, Stip DESC
```

Приклад 5.6.6.

Вивести прізвища студентів, які отримують ненульову стипендію.

```
1 SELECT Name
2 FROM Stud
3 WHERE Stip>0
```

Приклад 5.6.7.

Вивести номери курсів та прізвища тих студентів математичного факультету, середній бал яких не перевищує 60 балів. Результат впорядкувати за курсами, а потім за прізвищами студентів.

```
1 SELECT Name, Kurs
2 FROM Stud
3 WHERE (Fakult='Математичний') AND
4 ((E1+E2+E3)/3<=60)
  ORDER BY Kurs, Name
```

Приклад 5.6.8.

Вивести прізвища та сумарний бал всіх студентів-першокурсників математичного факультету.

```
1 SELECT Name, E1+E2+E3
2 FROM Stud
3 WHERE (Fakult='Математичний') AND (Kurs=1)
```

Приклад 5.6.9.

Вивести прізвище та оцінки першого за алфавітом студента математичного факультету.

```
1 SELECT Name, E1, E2, E3
2 FROM Stud
3 WHERE Fakult='Математичний'
4 ORDER BY Name
5 LIMIT 1
```

Для того, щоб вивести декілька перших записів з результату вибірки, необхідно застосувати директиву `LIMIT`*, після якої вказати кількість записів для виводу.*

Приклад 5.6.10.

Вивести відомості про наймолодшого студента університету.

```
1 SELECT *
2 FROM Stud
3 ORDER BY Data DESC
4 LIMIT 1
```

5.7. Основні прийоми для роботи з полем типу DATE в SQL

Основними вбудованими функціями для роботи з полем типу date в SQL є такі:

`Year(поле або вираз типу Date)`, `Month(поле або вираз типу Date)`, `Day(поле або вираз типу Date)`, `CURRENT_DATE`, `CURRENT_TIME`, `Now()`. Проілюструємо їх дію на деяких прикладах.

Приклад 5.7.1.

Вивести прізвища студентів, які народилися у 1999 році.

```
1 SELECT Name
2 FROM Stud
3 WHERE Year(Data)=1999
```

Приклад 5.7.2.

Вивести прізвища, факультети та курси тих студентів, які святкують день народження в поточному місяці. Результат впорядкувати спочатку за факультетом, потім за курсом.

```
1 SELECT Name, Fakult, Kurs
2 FROM Stud
3 WHERE Month(Data)=Month(CURRENT_DATE)
4 ORDER BY Fakult, Kurs
```

Приклад 5.7.3.

Вивести назви та прізвища керівників тих гуртків, які були засновані в поточному році.

```
1 SELECT NameG, NameK
2 FROM Gurt
3 WHERE Year(DateZ)=Year(CURRENT_DATE)
```

Приклад 5.7.4.

Вивести прізвища студентів математичного факультету, впорядкувавши їх за датами народження, від найстаршого до наймолодшого.

```
1 SELECT Name
2 FROM Stud
3 WHERE Fakult='Математичний'
4 ORDER BY Data
```

5.8. Використання агрегатних функцій в простих запитах

Розглянемо такі агрегатні функції SQL: `count()`, `sum()`, `avg()`, `min()`, `max()`. Аргументом функції може виступати як деяке поле, так і арифметичний вираз. Розглянемо приклади застосування цих функцій.

Приклад 5.8.1.

Визначити кількість студентів, які зареєстровані в таблиці `Stud`.

```
1 SELECT count(*)
2 FROM Stud
```

У даному випадку аналогічно спрацює такий запит:

```
1 SELECT count(NumS)
2 FROM Stud
```

Приклад 5.8.2.

Обчислити сумарну стипендію, яка виплачується студентам математичного факультету.

```
1 SELECT sum(Stip)
2 FROM Stud
3 WHERE Fakult='Математичний'
```

Приклад 5.8.3.

Знайти максимальний середній бал студентів-першокурсників математичного факультету.

```
1 SELECT Max((E1+E2+E3)/3)
2 FROM Stud
3 WHERE (Fakult='Математичний') AND (Kurs=1)
```

Дану задачу можна також розв'язати без використання функції `max()`. У такому випадку є можливість також вивести дані студента, який отримав цей максимальний бал. Проте, на даному етапі, у випадку, якщо декілька студентів набрали однаковий бал, який і є максимальним, то виведе тільки першого з них.

```
1 SELECT Name, (E1+E2+E3)/3 AS MAX_BAL
2 FROM Stud
3 ORDER BY MAX_BAL DESC
4 LIMIT 1
```

У цьому запиті використаний механізм надання псевдонімів стовпцям (SQL Alias) через сполучник `AS`.

Приклад 5.8.4.

Обчислити середню стипендію, яку отримують студенти-відмінники.

```
1 SELECT avg(Stip)
2 FROM Stud
3 WHERE (E1>=90) AND (E2>=90) AND (E3>=90) AND
4 (Stip>0)
```

В задачах на обчислення середньої грошової виплати, як правило, беруться до уваги тільки ті записи, у яких ця виплата здійснюється, тобто перевищує 0.

5.9. Обробка унікальних значень стовпця. Оператор DISTINCT

Оператор `DISTINCT` використовується у тому випадку, коли необхідно здійснити обробку тільки унікальних значень стовпця. Як правило, в таких випадках в умові задачі або прямо, або опосередковано звучить словосполучення «без повторів».

Приклад 5.9.1.

Вивести назви факультетів університету, впорядкувавши їх за алфавітом.

Розглянемо два різні запити:

```
1 SELECT Fakult
2 FROM Stud
3 ORDER BY Fakult
```

та

```
1 SELECT DISTINCT Fakult
2 FROM Stud
3 ORDER BY Fakult
```

Результатом роботи першого запиту будуть впорядковані за алфавітом назви факультетів, які містяться в таблиці `Stud`, при чому кожен факультет буде виведено стільки разів, скільки разів він зустрічається в даній таблиці. В свою чергу, результатом роботи другого запиту буде впорядкований за алфавітом список факультетів без дублікатів.

Приклад 5.9.2.

Обчислити кількість факультетів в університеті.

```
1 SELECT count(DISTINCT Fakult)
2 FROM Stud
```

Приклад 5.9.3.

Знайти скільки різних студентів беруть участь у гуртках.

```
1 SELECT count(DISTINCT NumS)
2 FROM Rob_Gurt
```

5.10. Пошук значень за зразком. Оператор LIKE

Оператор `LIKE` використовується з умовою `WHERE` для пошуку значень за зразком. Для цього використовують відповідні маски, які описані в розділі 4.

Приклад 5.10.1.

Вивести інформацію про студентів, прізвища яких починаються літерою ‘А’.

```
1 SELECT *
2 FROM Stud
3 WHERE Name LIKE 'A%'
```

Приклад 5.10.2.

Вивести відомості про всіх студентів з іменем ‘Іван’. Результат впорядкувати по факультетам і курсам.

```
1 SELECT *
2 FROM Stud
3 WHERE Name LIKE '% Іван %'
4 ORDER BY Fakult, Kurs
```

Приклад 5.10.3.

Вивести відомості про успішність всіх студентів, крім тих ім'я яких 'Іван'.

```
1 SELECT Name, E1, E2, E3
2 FROM Stud
3 WHERE Name NOT LIKE '% Іван %'
```

Приклад 5.10.4.

Вивести відомості про гуртки, назви яких складаються рівно з восьми літер.

```
1 SELECT *
2 FROM Gurt
3 WHERE NameG LIKE '________'
```

(8 символів підкреслення)

Приклад 5.10.5.

Знайти гуртки, назви яких закінчуються на 'я', але не на 'ля'.

```
1 SELECT nameg
2 FROM Gurt
3 WHERE (NameG LIKE '%я') AND
4       (NameG NOT LIKE '%ля')
```

Приклад 5.10.6.

Знайти гуртки, назви яких містять знак підкреслення ('_').

Пояснення: так як, знак підкреслення є підстановочним символом, то необхідно використати символ пропуску.

```
1 SELECT NameG
2 FROM Gurt
3 WHERE NameG LIKE '%#_%' ESCAPE '#'
```

5.11. Запити з групуванням

Групування дозволяє розділити всі дані на логічні набори, завдяки чому стає можливим виконання статистичних обчислень в кожній окремі групі. Групи утворюються за допомогою блоку `GROUP BY` оператора `SELECT`. Розглянемо приклади.

Приклад 5.11.1.

Для кожного факультету університету підрахувати скільки студентів на ньому навчається.

```
1 SELECT Fakult, count(NumS)
2 FROM Stud
3 GROUP BY Fakult
```

Приклад 5.11.2.

Знайти скільки студентів-відмінників навчається на кожному курсі математичного факультету.

```
1 SELECT Kurs, count(NumS)
2 FROM Stud
3 WHERE Fakult='Математичний'
4 GROUP BY Kurs
```

Приклад 5.11.3.

Для кожного курсу кожного факультету визначити максимальну стипендію, яка на ньому виплачується студентам. Результати впорядкувати спочатку по курсам, а потім по факультетам.

```
1 SELECT Fakult, max(Stip)
2 FROM Stud
3 GROUP BY Fakult, Kurs
4 ORDER BY Kurs, Fakult
```

Приклад 5.11.4.

Для кожного курсу математичного факультету підрахувати кількість студентів, які не склали принаймні один іспит. Результат впорядкувати за номером курсу.

```
1 SELECT Kurs, count(NumS)
2 FROM Stud
3 WHERE (Fakult='Математичний') AND
4       ((E1<60) OR (E2<60) OR (E3<60))
5 GROUP BY Kurs
6 ORDER BY Kurs
```

Приклад 5.11.5.

Вивести назву факультету на якому навчається найбільша кількість студентів-відмінників.

```
1 SELECT Fakult, count(NumS) AS cnt
2 FROM Stud
3 WHERE (E1>=90) AND (E2>=90) AND (E3>=90)
4 GROUP BY Fakult
5 ORDER BY cnt
6 LIMIT 1
```

5.12. Групування та відбір в блоці HAVING

У випадку, коли необхідно здійснити відбір груп за умовою на результат агрегатних функцій використовують блок HAVING. Блок HAVING працює схоже до блоку WHERE, за винятком того, що рядки в ньому будуються зі значень стовпців вказаних в GROUP BY та агрегатних функцій.

У випадку, якщо GROUP BY відсутній, то HAVING працює аналогічно до WHERE.

Приклад 5.12.1.

Для факультетів, кількість студентів у яких перевищує 300 вивести мінімальний середній бал його студентів. Результат впорядкувати за назвою факультету.

```
1 SELECT Fakult, min((E1+E2+E3)/3)
2 FROM Stud
3 GROUP BY Fakult
4 HAVING count(NumS)>300
```

Приклад 5.12.2.

У випадку, якщо кількість студентів університету перевищує 10000, обчислити кількість факультетів університету та сумарну стипендію, яка виплачується в університеті.

```
1 SELECT count(DISTINCT Fakult), sum(Stip)
2 FROM Stud
3 HAVING count(NumS)>10000
```

Приклад 5.12.3.

Для тих факультетів, у яких кількість студентів-відмінників більша 50 обчислити сумарну стипендію, яка виплачується відмінникам. Результат впорядкувати по значенням суми.

```
1 SELECT Fakult, sum(Stip) AS sum_Stip
2 FROM Stud
3 WHERE (E1>=90) and (E2>=90) and (E3>=90)
4 GROUP BY Fakult
5 HAVING count(NumS)>50
6 ORDER BY sum_Stip
```

Приклад 5.12.4.

Вивести назву останнього за алфавітом факультету та мінімальний середній бал тих студентів, які мають принаймні одну п’ятірку, серед тих факультетів у яких є принаймні один студент, який отримав 100 балів з кожного предмету.

```
1 SELECT Fakult, min((E1+E2+E3)/3)
2 FROM Stud
3 WHERE (E1>=90) OR (E2>=90) OR (E3>=90)
4 GROUP BY Fakult
5 HAVING max(E1+E2+E3)=300
6 ORDER BY Fakult DESC
7 LIMIT 1
```

5.13. Багатотабличні запити в SQL

У випадку, коли для розв’язання задачі необхідно використати інформацію з декількох таблиць бази, використовують

багатотабличні запити. При цьому логічне об'єднання таблиць відбувається по відповідним значенням первинного та зовнішнього ключів таблиць, які розглядаються. Поля для співставлення таблиць вказуються у блоці `WHERE`. Розглянемо приклади.

Приклад 5.13.1.

Вивести прізвища студентів та номери гуртків, які вони відвідують. Результати впорядкувати спочатку за номерами гуртків, а потім за прізвищами.

```
1 SELECT Name, NumG
2 FROM Stud, Rob_Gurt
3 WHERE Stud.NumS=Rob_Gurt.NumS
4 ORDER BY NumG, Name
```

Слід звернути увагу, що в даному прикладі необхідно отримати інформацію з двох таблиць, назви яких вказані у блоці `FROM`.

Умова `Stud.NumS=Rob_Gurt.NumS` вкзаує за якими полями відбувається логічне об'єднання таблиць.

Приклад 5.13.2.

Вивести прізвища студентів та назви гуртків, які вони відвідують. Результати впорядкувати за назвами гуртків та прізвищами.

```
1 SELECT Name, NameG
2 FROM Stud, Gurt, Rob_Gurt
3 WHERE (Stud.NumS=Rob_Gurt.NumS) AND
4       (Gurt.NumG=Rob_Gurt.NumG)
5 ORDER BY NameG, Name
```

Приклад 5.13.3.

Знайти скільки різних студентів-відмінників беруть участь принаймні в одному гуртку.

```
1 SELECT count(DISTINCT NumS)
2 FROM Stud, Rob_Gurt
3 WHERE (Stud.NumS=Rob_Gurt.NumS) AND (E1=5) AND
4 (E2=5) AND (E3=5)
```

Приклад 5.13.4.

Знайти кількість студентів математичного факультету, які беруть участь у гуртку «Волейбол».

```
1 SELECT count(DISTINCT NumS)
2 FROM Stud, Rob_Gurt, Gurt
3 WHERE (Stud.NumS=Rob_Gurt.NumS) AND
4 (Gurt.NumG=Rob_Gurt.NumG) AND
5 (Fakult='Математичний') AND (NameG='Волейбол')
```

Приклад 5.13.5.

Знайти наймолодшого студента математичного факультету, який бере участь у гуртку «Волейбол».

```
1 SELECT NumS, Name, Fakult, Kurs
2 FROM Stud, Rob_Gurt, Gurt
3 WHERE (Stud.NumS=Rob_Gurt.NumS) AND
4 (Gurt.NumG=Rob_Gurt.NumG) AND
5 (Fakult='Математичний') AND (NameG='Волейбол')
6 ORDER BY Data DESC
7 LIMIT 1
```

Приклад 5.13.6.

Для кожного факультету знайти скільки його студентів відмінників беруть участь у гуртку «Волейбол». Результат впорядкувати за назвою факультету.

```
1 SELECT Fakult, count(DISTINCT NumS)
2 FROM Stud, Rob_Gurt, Gurt
3 WHERE (Stud.NumS=Rob_Gurt.NumS) AND
4 (Gurt.NumG=Rob_Gurt.NumG) AND
5 (NameG='Волейбол') AND (E1=5) AND (E2=5) AND
6 (E3=5)
7 GROUP BY Fakult
  ORDER BY Fakult
```

Приклад 5.13.7.

Знайти факультет на якому навчається найбільша кількість студентів-учасників гуртку «Волейбол».

```
1 SELECT Fakult, count(DISTINCT NumS) AS cnt
2 FROM Stud, Rob_Gurt, Gurt
3 WHERE (Stud.NumS=Rob_Gurt.NumS) AND
4 (Gurt.NumG=Rob_Gurt.NumG) AND
5 (NameG='Волейбол')
6 GROUP BY Fakult
7 ORDER BY cnt DESC
8 LIMIT 1
```

5.14. Використання вкладених запитів в SQL

Так як у блоці WHERE можливість здійснення обчислень за допомогою агрегатних функцій є обмеженою, то виникає необхідність використання вкладених запитів. Структура такого запиту є наступна

```
1 SELECT поля
2 FROM таблиці
3 WHERE поле_або_вираз логічний_знак (SELECT
4 поле FROM таблиці
5         ……)
6 ……
```

Як видно зі схеми, вкладений запит розміщується у блоці умов та записується в дужках. Його результатом має бути або одне атомарне значення, або множина атомарних значень, або порожнє значення NULL. Розглянемо приклад.

Приклад 5.14.1.

Вивести прізвища, факультети та курси тих студентів, які отримують стипендію, не меншу за середню стипендію всіх студентів університету. Результат впорядкувати за назвою факультету, курсом та прізвищем.

```
1 SELECT Name, Fakult, Kurs
2 FROM Stud
3 WHERE Stip>= (SELECT avg(Stip)
4                FROM Stud
5                WHERE Stip>0)
6 ORDER BY Fakult, Kurs, Name
```

Приклад 5.14.2.

Для кожного факультету підрахувати скільки його студентів не бере участь у жодному з гуртків.

```
1 SELECT Fakult, count(NumS)
2 FROM Stud
3 WHERE NumS NOT IN (SELECT NumS
4                    FROM Rob_Gurt)
5 GROUP BY Fakult
```

Приклад 5.14.3.

Для кожного факультету вивести кількість його студентів-відмінників, які приймають участь у гуртку, який був заснований найпізніше. Результат впорядкувати за кількістю від найбільшого до найменшого значення.

```
1 SELECT Fakult, count(DISTINCT NumS) AS cnt
2 FROM Stud, Rob_Gurt
3 WHERE (E1=5) AND (E2=5) AND (E3=5) AND
4 (Stud.NumS=Rob_Gurt.NumS) AND
5 NumG=(SELECT NumG
6       FROM Gurt
7       ORDER BY DateZ
8       LIMIT 1)
9 ORDER BY cnt DESC
```

Приклад 5.14.4.

Вивести прізвища тих студентів, сумарний бал яких більший за мінімальний сумарний бал студентів, що відвідують перший по алфавіту гурток. Результат впорядкувати за алфавітом.

```
1  SELECT DISTINCT Name
2  FROM Stud
3  WHERE (E1+E2+E3)>(SELECT min(E1+E2+E3)
4     FROM Stud, Rob_Gurt
5     WHERE (Stud.NumS=Rob_Gurt.NumS) AND
6     NumG = (SELECT NumG
7               FROM Gurt
8               ORDER BY NameG
9               LIMIT 1))
10 ORDER BY Name
```

5.15. Використання умови в функції COUNT ()

У випадку, якщо для розв'язання задачі необхідно знайти кількість записів за різними умовами, допустимим є застосування умови в середині функції `count()`. Синтаксис конструкції є таким:

```
1  SELECT count(CASE WHEN умова
2                 THEN 1 ELSE NULL END)
3  FROM tbl
4  ……
```

Тобто, якщо запис задовольняє умові, то його включають в обчислення, інакше – ні. Розглянемо приклади.

Приклад 5.15.1.

Знайти скільки студентів математичного факультету отримали за перший іспит менше 60 балів та скільки більше, ніж 90 балів.

```
1 SELECT count(CASE WHEN E1<60 THEN 1 ELSE NULL
2 END) AS cnt1, count(CASE WHEN E1>90 THEN 1
3 ELSE NULL END) AS cnt2
4 FROM Stud
5 WHERE Fakult='Математичний'
```

Приклад 5.15.2.

Для кожного факультету підрахувати скільки його студентів отримали середній бал більший ніж 90 та скільки студентів не отримують стипендії.

```
1 SELECT Fakult, count(CASE WHEN (E1+E2+E3)/3>90
2 THEN 1 ELSE NULL END) AS cnt1,
3 count (CASE WHEN STIP=0 THEN 1 ELSE NULL END)
4 AS cnt2
5 FROM Stud
6 GROUP BY Fakult
```

5.16. Створення нової таблиці як результату запиту SELECT

Для того, щоб результат запиту `SELECT` зберегти в окрему таблицю бази даних необхідно використати команду `CREATE TABLE` в такому форматі:

```
1 CREATE TABLE new_tbl
2 AS (SELECT column1, column2
3     From tbl
4     …..)
```

Приклад 5.16.1.

На основі наявних таблиць створити таблицю `Reit` з наступною структурою: Прізвище студента, факультет, курс, середній бал. Заповнити таблицю даними з таблиці `Stud`.

```
1 CREATE TABLE Reit
2 AS (SELECT Name, Fakult, Kurs, (E1+E2+E3)/3 AS
3     avgb FROM Stud)
```

Приклад 5.16.2.

На основі даних, які містяться в таблиці `Stud` створити таблицю з інформацією про студентів математичного факультету.

```
1 CREATE TABLE Math
2 AS (SELECT *
3     FROM Stud
4     WHERE Fakult='Математичний')
```

Приклад 5.16.3.

Створити звідну таблицю з інформацією про успішність студентів, які відвідують гуртки.

```
1 CREATE TABLE Info
2 AS (SELECT Name, E1, E2, E3
3     FROM Stud
4     WHERE NumS IN (SELECT NumS FROM Rob_Gurt))
```

5.17. Додавання та редагування записів в таблиці

Приклад 5.17.1.

Додати студента під номером 10 у гурток з номером 6 та студента з номером 12 у гурток з номером 3.

```
1 INSERT INTO Rob_Gurt (NumS, NumG)
2 VALUES (10, 6), (12, 3)
```

В результаті виконання даної інструкції в таблицю `Rob_Gurt` будуть додані два нових записи `<NumS=10, NumG=6>` та `<NumS=12, NumG=3>`.

Приклад 5.17.2.

Нарахувати всім студентам математичного факультету, середній бал яких не нижче 90 балів, стипендію 2000 грн.

```
1 UPDATE Stud
2 SET Stip=2000
3 WHERE (Fakult='Математичний') AND
4       ((E1+E2+E3)/3)>=90)
```

Приклад 5.17.3.

Студенту з номером 12 записати результати зданих іспитів: `E1=90, E2=85, E3=92`.

```
1 UPDATE Stud
2 SET E1=90, E2=85, E3=92
3 WHERE NumS=12
```

Приклад 5.17.4.

Підвищити всім студентам першого курсу стипендію на 10%.

```
1 UPDATE Stud
2 SET Stip=Stip*1.1
3 WHERE Kurs=1
```

Приклад 5.17.5.

Студентам, які відвідують принаймні один гурток встановити надбавку до стипендії – 200 грн.

```
1 UPDATE Stud
2 SET Stip=Stip+100
3 WHERE (Stip>0) AND
4       (NumS IN (SELECT NumS FROM Rob_Gurt))
```

Приклад 5.17.6.

Студентам, середній бал яких перевищує 90 балів встановити стипендію, яка рівна максимальній стипендії по університету.

```
1 UPDATE Stud
2 SET Stip=(SELECT max(Stip) FROM Stud)
3 WHERE (E1+E2+E3)/3>90
```

Приклад 5.17.7.

Студентам, які за перший іспит отримали менше 60 балів, відмінити виплату стипендії, всім іншим – встановити її в 2500 грн.

```
1 UPDATE Stud
2 SET Stip=CASE WHEN E1<60 THEN 0 ELSE 2500
```

5.18. Об'єднання запитів. Використання оператору UNION

Приклад 5.18.1.

Використовуючи оператор Union, вивести спочатку назви гуртків, якими керує керівник Іванов І.І., а потім назви тих гуртків, якими керує керівник Петров І.І.

```
1 SELECT NameG, NameK
2 FROM Gurt
3 WHERE NameK='Іванов I.I.'
4 UNION
5 SELECT NameG, NameK
6 FROM Gurt
7 WHERE NameK='Петров I.I.'
```

Приклад 5.18.2.

Вивести результати іспитів студентів математичного факультету у форматі <Прізвище, Курс, Оцінка>. Результат впорядкувати по курсам та прізвищам.

```
1 SELECT Name, Kurs, E1 AS bal
2 FROM Stud
3 WHERE Fakult='Математичний'
4 UNION ALL
5 SELECT Name, Kurs, E2
6 FROM Stud
7 WHERE Fakult='Математичний'
8 UNION ALL
```

```
9  SELECT Name, Kurs, e3
10 FROM Stud
11 WHERE Fakult='Математичний'
12 ORDER BY Kurs, Name
```

ЗАВДАННЯ ДЛЯ ПІДГОТОВКИ ДО ПЕРЕВІРОЧНИХ РОБІТ

1. Вивести інформацію про всіх студентів з таблиці `Stud`.
2. Вивести інформацію про всі гуртки з таблиці `Gurt`.
3. Вивести прізвища та дати народження всіх студентів.
4. Вивести інформацію про студентів математичного факультету.
5. Вивести прізвища та оцінки всіх першокурсників фізичного факультету.
6. Вивести інформацію про студентів, які не отримують стипендію, але мають принаймні одну п’ятірку.
7. Вивести прізвища, курс та факультет всіх відмінників, впорядкувавши їх за алфавітом.
8. Вивести інформацію про студентів математичного факультету, які отримують стипендію, впорядкувавши відомості спочатку по курсам, в порядку спадання, а потім по алфавіту.
9. Для кожного студента хімічного факультету вивести його прізвище та середній бал.
10. Вивести назви факультетів без повторів.
11. Вивести прізвища керівників гуртків без повторів, впорядкувавши їх в оберненому до алфавіту порядку.
12. Знайти найбільшу стипендію математичного факультету.
13. Знайти середню стипендію, яку отримують студенти, що мають принаймні одну двійку.
14. Знайти кількість студентів фізичного факультету, які отримали за іспити рівно одну п’ятірку.
15. Знайти середній сумарний бал студентів математичного факультету, які навчаються на першому курсі.
16. Знайти мінімальний середній бал студентів, що не отримують стипендію.
17. Знайти кількість різних факультетів університету.

18. Знайти кількість студентів, які записані в хоча б один з гуртків.
19. Знайти дату останнього запису в гуртки.
20. Знайти дату заснування найпершого гуртка.
21. Знайти кількість різних сум стипендій серед тих, які отримують студенти-фізики.
22. Знайти першого по алфавіту студента-математика.
23. Вивести відомості про наймолодшого студента фізичного факультету.
24. Вивести відомості про студента-математика, середній бал якого є найменшим.
25. Вивести прізвища перших трьох за успішністю студентів хімічного факультету.
26. Знайти номер студента, який першим записався в один з гуртків.
27. Знайти кількість студентів-математиків, які народилися у травні.
28. Знайти наймолодшого студента фізичного факультету, який святкує свій день народження зимою.
29. Вивести відомості про студентів, які народилися 1 вересня.
30. Вивести інформацію про гуртки, які були засновані в поточному місяці поточного року, впорядкувавши записи за прізвищами керівників.
31. Для кожного факультету знайти кількість відмінників, які на ньому навчаються.
32. Для кожного курсу математичного факультету знайти кількість студентів, які на ньому навчаються. Результат впорядкувати спочатку по кількості, а потім по алфавіту.
33. Для кожного факультету знайти мінімальну ненульову стипендію, яка на ньому виплачується.
34. Знайти факультет на якому навчається найбільша кількість студентів.
35. Для кожного керівника гуртка знайти якою кількістю гуртків він керує.

36. Знайти на якому курсі фізичного факультету виплачується найменша середня стипендія.
37. Знайти на якому факультеті навчається найбільша кількість студентів, які народилися весною.
38. Для кожного курсу математичного факультету підрахувати скільки студентів з іменем «Михайло» на ньому навчається. Результат впорядкувати за кількістю в порядку спадання.
39. Знайти номер гуртка, в якому бере участь найменша (ненульова) кількість студентів.
40. Для кожного курсу кожного факультету підрахувати кількість студентів, які не здали принаймні 1 іспит. Результат впорядкувати спочатку по факультетам, а потім по курсам.
41. Для кожного студента вивести номери гуртків, які він відвідує.
42. Для кожного керівника гуртків вивести прізвища студентів, з якими він займається. Результат впорядкувати спочатку за прізвищем керівника гуртка, а потім за факультетом.
43. Знайти найбільшу стипендію, яку отримують студенти-гуртківці.
44. Знайти мінімальний середній бал, який мають студенти-гуртківці.
45. Знайти наймолодшого студента-гуртківця та вивести назву гуртка, який він відвідує.
46. Знайти скільки студентів, що відвідують гурток «Волейбол» є відмінниками.
47. Вивести інформацію про всіх студентів-відмінників, які є учасниками гуртка «Танцювальний».
48. Для кожного факультету знайти кількість студентів-гуртківців.
49. Для кожного курсу математичного факультету знайти максимальну стипендію, яку отримують студенти цього факультету, які відвідують гуртки.
50. Для кожного гуртка знайти скільки у ньому є учасників.

Результат впорядкувати спочатку по кількості, а потім по алфавіту.

51. Знайти наймолодшого серед тих студентів-гуртківців, середній бал яких більший за середній бал першого за алфавітом студента-математика.
52. Знайти кількість студентів, які отримують стипендію, більшу за мінімальну ненульову стипендію студентів, прізвище яких починається на літеру «Л».
53. Знайти першого по алфавіту студента-гуртківця, який отримує мінімальну (ненульову) стипендію.
54. Вивести відомості про студентів, які народилися в один рік з першим по алфавіту студентом-учасником гуртка «Волейбол».
55. Вивести відомості про двох студентів, які отримують стипендію більшу за середню стипендію студентів-гуртківців, які вчаться на математичному факультеті.
56. Для кожного факультету підрахувати кількість студентів, середній бал яких більший за мінімальний середній бал студентів-гуртківців.
57. Вивести назви факультетів, студенти яких не беруть участь у гуртках.
58. Для кожного гуртка вивести скільки його учасників отримує максимальну стипендію по університету.
59. Знайти кількість студентів, які народилися пізніше за останнього по алфавіту студента-гуртківця.
60. Знайти скільки студентів університету не записані у гуртки.
61. Вивести інформацію про студентів, які народилися весною та отримують стипендію більшу за стипендію найстаршого студента математичного факультету.
62. Вивести прізвища студентів, оцінка за перший іспит яких не перевищує середньої оцінки студентів-учасників гуртка «Волейбол».
63. Знайти останнього за алфавітом студента, рік народження якого співпадає з роком народження наймолодшого

відмінника університету.

64. Знайти наймолодшого серед тих студентів-гуртківців, середній бал яких більший за середній бал першого за алфавітом студента-математика.
65. Для кожного факультету знайти кількість першокурсників, день народження яких у поточному місяці. Результат впорядкувати за кількістю.
66. Знайти кількість студентів, які отримують стипендію, більшу за мінімальну ненульову стипендію студентів, прізвище яких починається на літеру «Л».
67. Знайти першого по алфавіту студента-гуртківця, який отримує мінімальну (ненульову) стипендію.
68. Вивести відомості про студентів, які народилися в один рік з першим по алфавіту студентом-учасником гуртка «Волейбол».
69. Вивести відомості про двох студентів, які отримують стипендію більшу за середню стипендію студентів-гурткіців, які вчаться на математичному факультеті.
70. Для кожного факультету підрахувати кількість студентів, середній бал яких більший за мінімальний середній бал студентів-гуртківців.
71. Знайти на якому факультеті навчається найбільша кількість студентів-танцюристів.
72. Вивести назви факультетів, студенти яких не беруть участь у гуртках.
73. Для кожного гуртка вивести скільки його учасників отримує максимальну стипендію по університету.
74. Знайти кількість студентів, які народилися пізніше за останнього по алфавіту студента-гуртківця.
75. Знайти скільки студентів університету не записані у гуртки.
76. Вивести інформацію про студентів, які народилися весною та отримують стипендію більшу за стипендію найстаршого студента математичного факультету.
77. Вивести прізвища студентів, оцінка за перший іспит яких не

перевищує середньої оцінки студентів-учасників гуртка «Волейбол».

78. Знайти останнього за алфавітом студента, рік народження якого співпадає з роком народження наймолодшого відмінника університету.
79. Знайти факультети кількість студентів-відмінників, що відвідують гуртки у яких не менша, ніж 5.
80. Вивести прізвища студентів, які відвідують принаймні два гуртки.
81. Вивести прізвища керівників, які керують принаймні 10 студентами.
82. Вивести назви гуртків у яких беруть участь принаймні 5 студентів-відмінників.
83. Знайти на якому курсі математичного факультету кількість гуртківців більша за кількість учасників гуртка «КВК».
84. Знайти середній бал тих студентів, які відвідують більше, ніж три гуртки.
85. Знайти середню стипендію студентів, які отримали найбільший середній бал на своєму факультеті.
86. Для кожного курсу математичного факультету вивести наймолодшого студента, який отримує максимальну стипендію.
87. Для кожного з гуртків, кількість студентів у яких не перевищує 10, знайти студента, який останнім у нього записався, не використовуючи при цьому поле « кількість студентів».
88. Для кожного керівника, який керує більше, ніж одним гуртком, знайти наймолодшого студента, якого він навчає.
89. Для кожного факультету підрахувати скільки студентів отримали з першого іспиту 5, скільки з другого іспиту – 5, скільки з третього іспиту – 5.
90. Для кожного факультету вивести: середній бал за перший і за третій іспити.
91. Для кожного факультету вивести сумарні стипендії по

курсам.

92. Для кожного факультету вивести загальну кількість студентів факультету та кількість студентів відмінників.
93. Для кожного факультету вивести дати народження наймолодших студентів по курсам.

СПИСОК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1. Revesz, Peter. *Introduction to databases*. Springer, London, UK, 2010.
2. Abiteboul, S., Hull, R., & Vianu, V. (1995). Foundations of databases (Vol. 8). Reading: Addison-Wesley.
3. Beaulieu, A. (2009). *Learning SQL: master SQL fundamentals*. " O'Reilly Media, Inc.".
4. Berg, K. L., Seymour, T., & Goel, R. (2013). History of databases. International Journal of Management & Information Systems (IJMIS), 17(1), 29-36.
5. Date, C. J. (1989). *A Guide to the SQL Standard*. Addison-Wesley Longman Publishing Co., Inc..
6. Donahoo, M. J., & Speegle, G. D. (2010). *SQL: Practical guide for developers*. Elsevier.
7. Emerson, S. L., Darnovsky, M., & Bowman, J. (1989). *The practical SQL handbook: using structured query language*. Addison-Wesley Longman Publishing Co., Inc..
8. Garner, P., & Mariani, J. (2015). Learning SQL in steps. *Learning*, *12*, 23.
9. Grimm, Eric Christopher, et al. "Databases and their application." (2013).
10. Güting, R. H. (1994). An introduction to spatial database systems. *the VLDB Journal*, *3*, 357-399.
11. Kim, W. (1992). INTRODUCTION TO SQL/X. In *Future Databases' 92* (pp. 2-7).
12. Maimon, O., & Rokach, L. (2005). Introduction to knowledge discovery in databases. *Data mining and knowledge discovery*

handbook, 1-17.

13. McQuillan, M. (2015). *Introducing SQL Server*. Apress.
14. Melton, J., & Simon, A. R. (2001). *SQL: 1999: understanding relational language components*. Elsevier.
15. Migler, A., & Dekhtyar, A. (2020, February). Mapping the SQL learning process in introductory database courses. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 619-625).
16. Mistry, R., & Misner, S. (2010). *Introducing Microsoft SQL Server 2008 R2*. Microsoft press.
17. Mistry, R., & Misner, S. (2014). *Introducing Microsoft SQL Server 2014*. Microsoft Press.
18. Mitrovic, A. (1998, March). Learning SQL with a computerized tutor. In *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education* (pp. 307-311).
19. Official MySQL Database Site. URL: https://www.mysql.com/.
20. Pratt, P. J., & Last, M. Z. (2017). *A guide to SQL*. Cengage Learning.
21. Адамик, О. В., & Адамик, К. Б. (2018). Реляційні бази даних як сучасний стандарт накопичення інформації в комп'ютерній системі бухгалтерського обліку. *Сучасні проблеми обліку, аналізу, аудиту й оподаткування суб'єктів господарської діяльності: теоретичні, практичні та освітянські аспекти: Збірник наукових праць за матеріалами II Всеукраїнської науково-практичної конференції (29-30 березня 2018 р.).–Дніпро: НМетАУ, 2018.–747 с.*, 698.
22. Буй Д. Б., Сільвейструк Л. М. Формалізація моделі «сутність-зв'язок». Монографія. – К.: ВПЦ «Київський університет»., 2011.– 175 с.

23. БУЙНИЦЬКА, О. П. Інформаційні технології та технічні засоби навчання: навч. посіб. К.: Центр учбової літератури, 2012.

24. Вікіпедія. Вільна енциклопедія. Режим доступу: https://uk.wikipedia.org/

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

26. Завадський, І. О. (2011). Основи баз даних. *К.: Видавець ІО Завадський*.

27. Зінов'єва, І. С., & Артемчук, В. О. (2019). Сучасні підходи до подальшої еволюції концепції баз даних. In *III International scientific and practical conference «Dynamics of the development of world science», Canada* (pp. 34-44).

28. Коломейчук, В. В. (2009). Розробка та дослідження бази даних для систем обробки статистичної інформації. *Математичні машини и системи*, *1*(4), 89-95.

29. Коротун, О. В. (2018). *Використання хмаро орієнтованого середовища у навчанні баз даних майбутніх учителів інформатики* (Doctoral dissertation, ЖДУ).

30. Митні інформаційні технології. Режим доступу: http://pidruchniki.com/18411014/informatika/mitni_informatsiyni_tehnologiyi

31. Наумов А. Н., Вендров А. М. і ін., "Системи керування базами даних і знань", М.:Фінанси і статистика, 1991 р.

32. Неня А. В. Організація баз даних та знань: конспект лекцій для студентів заочної форми навчання. – Суми:Вид-во СумДУ, 2010. – 109 с.

33. Олійник В.М. Інформаційні системи і технології у фінансах. Конспект лекцій. http://elkniga.info/book_264.html

34. Пасічник В. В., Резніченко В. А. Організація баз даних та знань: підручник для ВНЗ. – К.: Видавнича група BHV, 2006. – 384с.

35. Романюк, О. Н., & Савчук, Т. О. (2003). Організація баз даних і знань.

36. Сахно, Ю. М., Ребенок, А. В., & Сахно, Є. Ю. (2005). Створення бази даних для розробки проектів.

37. Шаров, С. В., & Петровський, В. В. (2015). Огляд нереляційних баз даних. Всеукраїнська Internet-конференція «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення», (14), 16-18.