МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЦЕНТРАЛЬНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Босько В.В., Константинова Л.В., Поліщук Л.І., Конопліцька-Слободенюк О.К.

БАЗИ ДАНИХ

Навчальний посібник

м. Кропивницький
2024

УДК 004.65
ББК 32.97
Б 17

Рекомендовано вченою радою ЦНТУ як навчальний посібник для студентів закладів вищої освіти
Протокол №9 від 27 травня 2024 р.

Рецензенти:

Мелешко Є. В. доктор технічних наук, професор, доцент кафедри кібербезпеки та програмного забезпечення Центральноукраїнського національного технічного університету;

Мацуй А. М. доктор технічних наук, професор, доцент кафедри автоматизації виробничих процесів Центральноукраїнського національного технічного університету, академік Академії технічних наук України, член-кореспондент Академії Прикладних Наук

Босько В.В., Константинова Л.В., Поліщук Л.І., Конопліцька-Слободенюк О.К.
Б 17 Бази даних: Навчальний посібник. – Кропивницький: ЦНТУ, 2024. – 226 с.

Навчальний посібник пропонує теоретичний та практичний матеріал для роботи з базами даних. Розглядаються основні поняття баз даних. Приділяється увага проектуванню баз даних. Описується архітектура баз даних, розповідається про моделювання даних. Розглядається концепція розподілених баз даних. Також описано основні прийоми роботи з БД за допомогою мови SQL.

Призначений для студентів, що навчаються за спеціальностями: «Комп'ютерна інженерія», «Комп'ютерні науки», «Кібербезпека та захист інформації» й «Електронні комунікації та радіотехніка». Для полегшення засвоєння наведеного матеріалу надаються ілюстрації та приклади.

УДК 004.65
ББК 32.97

Зміст

Вступ

Бази даних стають невід’ємною частиною життя сучасної людини. З швидкими технологічними та соціальними змінами у світі об’єми інформації, які підлягають зберіганню значно зростають. Сучасне інформаційне поле складається з маси подій, об'єктів і явищ. Воно охоплює такі обсяги, що без чітко діючої певної системи, зберігання всіх цих даних могло бути хаотичним і некерованим. У сучасному світі жодна велика фірма не обходиться без бази даних і спеціалістів, що з нею працюють. Тому знання цієї області важливі у навчанні,а фахівці, що знаються на роботі з базами даних в ціні завжди.

Цей посібник призначений для студентів, що навчаються за спеціальностями «Комп’ютерна інженерія», «Кібербезпека та захист інформації», «Комп’ютерні науки» й «Електронні комунікації та радіотехніка» для користувачів та розробників інформаційних систем, для тих, хто цікавиться реляційними базами даних та просто людей, які бажають поглибити знання в області баз даних. На відміну від інших довідкових джерел, посібник "Бази даних" містить легкий в освоєнні матеріал, який допоможе студентам у виконанні завдань з навчальної дисципліни Бази даних а також, який, можливо використовувати в практичній діяльності. Тут даються загальні та ключові поняття, практичні приклади роботи, завдання та запитання для самоконтролю.

Матеріал посібника складається з 10 частин і 2 додатків:

Частина 1. Загальні поняття баз даних. У першій частині розглядаються загальні поняття баз даних, основні терміни і їх визначення.

Частина 2. Архітектура баз даних. У другій частині надано інформацію про представлення даних, про моделювання даних, визначення моделей та їх різновиди й характеристики. Розглядається також архітектура інформаційних систем.

Частина 3. Загальна термінологія реляційної моделі даних. Більше уваги приділяється реляційній моделі даних, а саме на яких поняттях грунтується ця модель даних. Подано головні елементи реляційної моделі, про класи відношень, ключеві атрибути та їх види. Також розглядаються складові реляційних баз даних, поняття цілісності та ін.

Частина 4. Проектування й застосування баз даних. У четвертій частині можна розглянути методології проектування баз даних. Наводиться життєвий цикл баз даних та його етапи. Також є інформація про життєвий цикл інформаційних систем. Розглядаються різні підходи до проектування баз даних та задачі, що вирішуються на кожному з етапів.

Частина 5. Концептуальні моделі. В цій частині розглядається як побудувати концептуальну схему баз даних. Розглядається взагалі про концептуальне проектування, про концептуальні моделі даних та про два головних підходи до моделювання даних при концептуальному проектуванні, а саме про семантичні моделі та об'єктні. Наводиться модель «сутність-зв'язок», розповідається та на прикладах показується, як будують ER-діагрими. Про різні види зв'язків на діаграмі П. Чена й моделі «Пташина лапка».

Частина 6. Логічне проектування баз даних. В цій частині описано, яким чином концептуальні моделі й завдяки яким правилам трансформуються в логічні моделі даних. Розглядаються правила нормалізації, завдяки яким можливо впевнитись в структурній узгодженості, логічній цілісності й мінімальній збитковості прийнятої моделі даних. Також дана частина містить інформацію про поняття нормалізації та денормалізації. Розглядаються поняття аномалій, нормальних форм та транзакцій.

Частина 7. Фізична організація баз даних. Ця частина містить інфомацію про процес фізичного проектування баз даних. Тут розглядаються етапи виконання процесу пошуку й представлення

даних користувачу; такі поняття як кластеризація й пам'ять сторінок. Також описано, що зберігання даних організовано за певною ієрархією, З цієї частини можна дізнатись про індексацію з погляду користувача, технологію хешування та ін.

Частина 8. Засоби для автоматизації проектування баз даних. Ця частина знайомить з CASE-технологіями, розповідає які існують CASE-засоби. Тут можна дізнатись про переваги застосування CASE-технологій при проектуванні ІС. Познайомитись з поняттями такими, як репозиторій, метадані та ін. В цій частині подається також інформацію про RAD-технології та компонентно-орієнтовані технології.

Частина 9. Розподілена обробка даних. В цій частині посібника розповідається про багатокористувацькі БД. Тут можна познайомитись з поняттями такими як: розподілена обробка, розподілені бази даних, розподілені системи керування базами даних. Також в цій частині подається інформація про керування паралельною обробкою, механізм транзакцій; про поняття серіалізація транзакцій, механізм блокувань, тупикова ситуація. Тут відбувається знайомство з багаторівневою архітектурою, мультибазовими системами та методологією проектування розподілених баз даних.

Частина 10. SQL. Значна частина матеріалу, що міститься в цій частині присв’ячується структурованій мові запитів. Починається з історії мови, для яких цілей застосовується, на які підгрупи поділяється. Окремо описано про кожну групу команд. Розповідається також про сервер MySQL, його можливості та переваги. Також дається інформація про використання OpenServer.

Також у 10 частині описано основні прийоми роботи з БД за допомогою мови SQL, а саме, застосування таких підмов: мови вибірки даних, для визначення даних (DDL), мови для маніпулювання даними (DML). Надаються прийоми керування доступом користувачів до бази

даних та команди мови керування транзакціями (TCL). Розповідається про типи з’єднань в MySQL та надається візуальне представлення кожного з типів join-з'єднання.

Наостанок, представлено, як приклад, застосування SQL при розробці backend частини з написанням frontend. Розглядаються частини лістингів, SQL інструкції та дерево проекту, що отримали в результаті.

Додаток 1. У додатку 1 міститься інформація про роботу з БД за допомогою MySQL. Представлено приклад варіанту створення простої БД «Книги».

Додаток 2. У додатку 2 міститься інформація про роботу з БД за допомогою СКБД Microsoft Access, ознайомлення з основними об’єктами БД, робота з таблицями, створення зв’язків між таблицями й QBE-запити різного виду. Приводиться побудова інтерфейсу користувача за допомогою форм, звітів та ін.

SQL - це важлива мова програмування, яка стала дуже популярною завдяки можливості працювати з різними базами даних й ефективно опрацьовувати великі обсяги інформації. Запити на знання SQL постійно зростають серед аналітиків даних, адміністраторів баз даних, розробників, які працюють з даними, та інших спеціалістів, які мають справу з обробкою даних та розробкою моделей даних. Володіння знаннями SQL дозволяє безперебійно працювати з різними базами даних і забезпечує ефективну співпрацю з ними.

Кожен розділ посібника містить в кінці запитання для контролю знань, що покращує засвоєння матеріалу.

Поданий в навчальному посібнику матеріал надасть читачам теоретичні та практичні рекомендації й допоможе стати затребуваним та досвідченим фахівцем у сфері робіт з базами даних.

1. Загальні поняття баз даних

В даний час практично будь-яка задача, пов’язана з маніпуляцією інформації та даних. З цієї причини розроблено багато різних систем керування базами даних, що допомагають з цим, існують різні підходи та технології щодо керування даними. Для того, щоб легше розібратись з цими питаннями необхідно розібратись з деякими пов’язаними з базами даних термінами та поняттями.

Задачі інформаційного типу, на відміну від типу обчислювальних мають такі особливості [1]:

− збереження даних, що мають складну структуру;

− йде обробка інформації великих обсягів;

− застосовуються відносно прості алгоритми для обробки.

У вирішенні інформаційних задач допоможуть наступні поняття:

Предметна область (ПО) - це цілеспрямоване первинне перетворення картини зовнішнього світу в деяку умоглядну картину, визначена частина якої втілюється в інформаційній системі в якості алгоритмічної моделі фрагмента дійсності [2]. ПО - частина реального світу, що розглядається в межах даного контексту, це сфера застосування конкретної бази даних (наприклад: медицина, освіта, залізничний транспорт тощо).

База даних (БД) (DataBase (DB)) – сукупність спеціальним чином організованих даних, які зберігаються в пам’яті обчислювальної системи та відображають стан об’єктів та їх взаємозв’язки в даній ПО.

Система керування базами даних (СКБД) – комплекс мовних і програмних засобів, що призначені для створення, роботи та сумісного використання БД. (Наприклад: PostgreSQL, SQLite, MySQL, MS Access, Microsoft SQL Server, Oracle Database та ін.) [3][4].

В СКБД входять такі компоненти (рисунок 1.1): ядро СКБД, підсистема засобів проектування й підсистема засобів обробки [1].

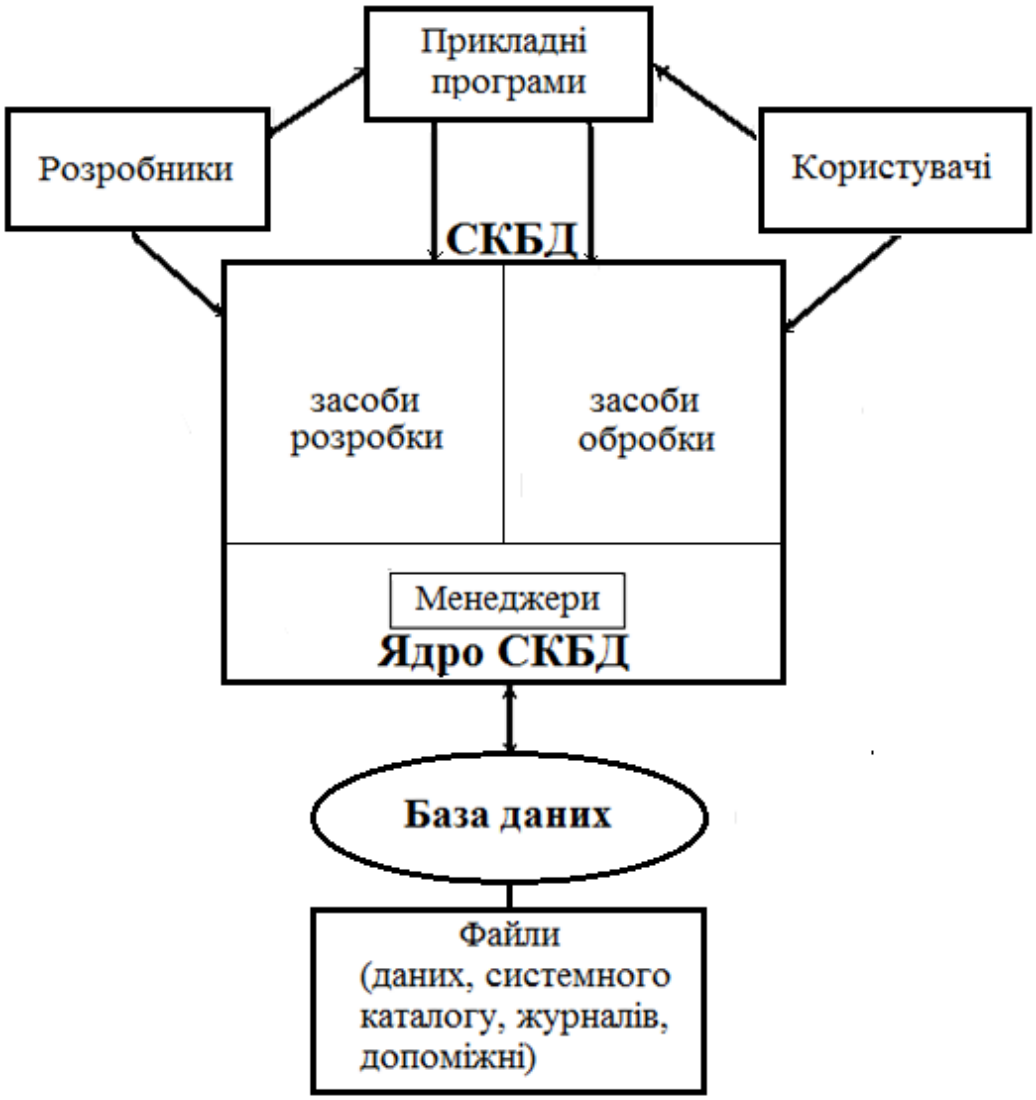


Рисунок 1.1 – Компоненти системи БД

Ядро СКБД – містить купу базових механізмів СКБД, які використовуються при будь-яких варіантах конфігурації системи. Ядро СКБД виконує функцію посередника між підсистемами засобів проектування й обробки та даними. Сучасні БД, у більшості, надають користувачу дані у вигляді таблиць. Ядро СКБД отримує запити від інших компонентів в термінах таблиць, стовпців, рядків і перетворює ці запити в команди операційної системи, які виконують запис і читання з фізичних носіїв інформації.

Крім того, ядро СКБД задіяне в управлінні транзакціями, блокуваннях, резервному копіюванні й відновленні. В ядро входять менеджери буферів, даних, транзакцій, журналів.

Менеджер буферів – призначений для вирішення задач ефективної буферізації оперативної пам'яті.

Менеджер даних – призначений для керування зовнішньою пам'ятью, забезпечує створення структур для даних, що зберігаються, та допоміжних структур (таких як індекси й т.ін.).

Менеджер транзакцій – відповідає за підтримку механізмів фіксації й відміни транзакцій. Він пов’язаний з менеджером буферів оперативної пам'яті й забезпечує зберігання інформації, що потрібна після збоїв програми.

Менеджер журналів – реєструє відомості про виконання транзакцій, про наявних користувачів. Також забезпечує реєстрацію виконання застосування, доступи до структур даних й т. ін.

Підсистема засобів проектування це сукупність інструментів, які спрощують проектування й реалізацію БД. Як правило, ця сукупність містить засоби для створення об’єктів БД (таблиць, запитів, форм, звітів). СКБД також мають мови програмування й інтерфейси для роботи.

Підсистема обробки - виконує обробку компонентів застосування, що створюються за допомогою засобів проектування.

Застосування БД може складатись з таких об’єктів як форми, запити, меню, звіти й прикладні програми. Форми, звіти й запити можна створювати за допомогою засобів, що постачаються у комплексі з СКБД. Прикладні програми можуть бути написані або на вхідній мові СКБД, або іншій стандартній мові, а потім за допомогою СКБД з'єднані з БД.

Інформаційна система (ІС (IS)) являє собою систему програмних, мовних, організаційних і технічних засобів, призначених для її централізованого зберігання та колективного використання даних.

Інформаційна система має функції збирання, зберігання, розповсюджує й оброблє інформацію.

Банк даних (БнД) – різновид ІС, в якій реалізовані функції централізованого зберігання та накопичення оброблюваної інформації, що об’єднується в одну або декілька БД.

Словник даних (СД) – це підсистема банку даних, що призначена для централізованого зберігання інформації про структури даних, взаємозв’язки файлів БД, типи, формати даних, про коди захисту та розмежування доступу. Функції СД викликаються з головного меню системи та виконуються СКБД.

Додаток – це програма або комплекс програм, яка автоматизує обробку інформації для деякої прикладної задачі. Додатки бувають зовнішні та внутрішні. Для створення зовнішніх додатків використовуються будь яка мова програмування, яка має засоби доступу до БД. Внутрішні додатки пишуться на вбудованій мові програмування (наприклад, в MS Access це Visual Basic).

Модель даних – це набір правил, що визначають, як створюються та змінюються структури даних згодом, які операції дозволені над ними та обмеження цілісності, включаючи допустимі зв'язки, значення даних і послідовності їх змін. **Модель представлення даних** – це логічна структура даних, за якою побудована БД. Існують наступні основні моделі [2][4]:

1. Ієрархічна модель;
2. Мережева (або в деяких джерелах - мережна) модель;
3. Реляційна модель;
4. Об’єктно-орієнтована модель.

В залежності від моделі представлених даних, СКБД відповідно підрозділяються на: ієрархічні, мережеві (мережні), реляційні, об’єктно-орієнтовані та багатозначні (multi-value).

Контрольні запитання:

1. Що собою являє інформаційна система, автоматизована інформаційна система?
2. Дайте визначення банку даних.
3. Що представляє собою БД?
4. Дайте визначення СКБД.
5. Дайте визначення поняттю «Предметна область».
6. Що собою являє словник даних?
7. Назвіть основні моделі даних.
8. Які існують основні моделі даних?
9. Дайте визначення додатку.
10. Які компоненти входять в СКБД?
11. Які функції виконує інформаційна система?

2. Архітектура баз даних

Робота з БД передбачає незалежність прикладних програм від даних. Це обумовлено тим, що у випадку зміни системи, а також з метою покращення обслуговування користувачів відбуваються постійні маніпуляції щодо зміни методів зберігання даних в БД, шляхів доступу до даних, редагування структури і форматів даних та зв'язків між ними.

Незалежність застосувань від даних забезпечується засобами СКБД. Цей підхід грунтується на тому, що користувачі застосовуючи БД, не знають внутрішнє подання даних.

Моделі даних

Модель даних можна уявити, як абстракцію, в якій знаходять своє відображення більш важливі елементи функціонування визначеної ПО, а другорядні моменти не враховуються.

Модель даних виступає в ролі цільової моделі ПО. Розрізняють три головні складові моделі даних:

− **структурна частина** (визначає правила породження допустимих для даної СКБД видів структур даних);

− **керуюча частина** (відповідає за визначення допустимих операцій над певними структурами);

− **групи обмежень цілісності даних**, що засоби системи спроможні реалізувать.

Моделювання даних – це дії створення логічного подання структури БД.

Кожний рівень представлення інформації надає певну модель.

Інфологічна модель – дає уявлення про інформацію ПО природною мовою, незалежно від СКБД, що застосовується [5]. Ця модель подає інформаційно-логічний рівень абстрагування, що

пов'язаний з визначенням об'єктів ПО, їх властивостей та взаємозв'язків між ними.

Даталогічна модель – відноситься до моделей логічного рівня, й дає уявлення про логічні зв'язки між елементами даних безвідносно до їх змісту й середовища зберігання даних. Ці моделі уподібнюють з логічними моделями.

Фізична модель – дає уявлення про спосіб зберігання даних на пристрої, представляючи інформацію про структуру записів, впорядкування, про шляхи доступу до інформації тощо.

Модель "сутність-зв'язок" (ER-модель) – визначає модель ПО й має у складі множину сутностей, множину зв’язків між сутностями, а також атрибутів сутностей. Модель також відображає обмеження цілісності даних, що пов'язано з двома множинами сутностей; та називається залежністю по існуванню. За допомогою ER-моделей можливо графічно подавати моделі ПО. Вони входять в більшість CASE-продуктів.

Семантична об'єктна модель – це модель ПО, являє собою модель даних. Вона складається з частин - семантичних об'єктів, які містять набір атрибутів, що групуються за класами. Модель даних має більш розвинені засоби відображення семантики у порівнянні з іншими моделями (теоретико-множинними, теоретико-графовими).

Теоретико-графова модель – це модель даних, в якій дозволені структури даних можуть бути описані у вигляді графа. Наприклад, це може бути дерево. Необхідну допустиму групу операцій на мові маніпулювання даними, які грунтуються на цій моделі, представляють навігаційні операції.

Теоретико-множинна модель – це модель даних, в якій використовуються математичні засоби реляційної алгебри, реляційного числення, а також здійснюється маніпулювання таблицями як операції над даними.

Фактографічні моделі – це моделі, що містять відомості, що подаються як ретельно організовані групи формалізованих записів даних.

Ієрархічна модель – модель даних, що грунтуэться на ієрархічній, деревоподібній структурі даних. Якщо розглядать вершини цієї структури, то це - є записи, які складаються з простих елементів даних різного типу. Запису предку відповідає довільне число екземплярів нащадків записів кожного типу.

Документальні моделі – це моделі, у яких окремим елементом інформації є документ, що не ділиться, а не структурована інформація про нього, яка часто має обмежений формат, або не структурується взагалі. Ці моделі, здебільшого, представляють тексти, надані природною мовою, і працюють з документами у вільних форматах.

Мережна (або в деяких джерелах мережева) **модель** – це така модель, де структури даних надаються у вигляді графа, вершинами якого можуть бути дані різних типів (від атомарних елементів даних та до записів складної структури). Відрізняючись від ієрархічної моделі нащадок в цих моделях може наслідувати будь-яку кількість предків.

Реляційна модель – модель даних, що грунтується на понятті відношення (математичному) й поданні відношень у вигляді таблиць.

Постреляційна модель – це реляційна модель, що розширена та без обмежень неподільності даних, що накладаються на записи таблиць. Допускається застосовувати багатозначні поля (тобто, значення таких складаються з підзначень). Набір значень в багатозначних полях розглядається, як самостійна таблиця, яка вбудовується в основну таблицю. Часто ці моделі порівнюють з об'єктно-реляційними моделями.

Об'єктно-орієнтована модель – це модель даних, що базується на понятті об'єкта (або сутності) що описує стан і поведінку. За допомогою атрибутів визначається стан об'єкта, а наявна сукупність

операцій чи методів визначають поведінку цього об'єкта. До того ж є можливість між типами об'єктів організовувати зв'язки.

Багатомірна модель. Ця модель даних дозволяє оперувати багатомірним представленням даних (у формі гіперкубу) та дозволяє підтримку аналіза даних. В цій моделі є можливим конструювання різних агрегатних операцій над даними у межах гіперкубу, побудова всяких його проекцій (підмножин гіперкубу), деталізація й обертання даних тощо.

Тезаурусна модель – це така модель, що подає документ за допомогою описувачів (дескрипторів) та вагомих відношень між лексичними одиницями (ціле-частина, рід-вид, клас-підклас тощо). Завдяки цим моделям дозволяється збільшити продуктивність дескрипторних моделей коштом кращого відображення ПО.

Дескрипторна модель – це модель, в якій описується кожен документ за допомогою деякого дескриптора. Сам дескриптор має певну структуру та являє собою множину якихось лексичних одиниць (термінів, слів, словосполучень), що необхідні при роботі з документами. При чому дескриптори не пов’язані між собою.

Гіпертекстова модель – це модель, яка грунтується на розмітці документа зі спеціальними навігаційними конструкціями, які відтворюють смислові зв'язки між різними документами або в межах окремих фрагментів одного документа.

Подібні конструкції створюють якусь семантичну мережу в документальній базі.

Трирівнева модель архітектури СКБД була заініційована спеціальною організацією, Комітетом планування стандартів і норм SPARC (Standarts Planning and Requirements Committee) Американського національного інституту стандартів ANSI (American National Standarts Institute) в 1971 р. [1][2] (концептуальний, зовнішній та внутрішній рівні).

Опис структури даних на будь-якому рівні називається **схемою**. Розрізняється три типи схем БД, що визначені відповідно до рівнів абстракції архітектури СКБД.

Архітектура ІС

Рівні абстракцій, що відмічають в перебігу аналізу та при проектуванні ІС:

1. Локальне подання даних – уявлення кінцевих користувачів про предметну область.

2. Концептуальне подання даних – представляє собою інформаційні потреби системи та відображає особливості предметної області. Концептуальне уявлення про ПО – не має зв'язку із засобами реалізації ІС. Це рівень адміністратора БД та прикладних програмістів.

3. Формалізоване подання даних працює за підтримки СКБД, за допомогою якої буде розроблена дана система та представляє собою логічну організацію даних з погляду адміністратора БД, але на відміну від концептуального рівня здійснюється контроль і прив'язка конкретної СКБД.

4. Внутрішнє подання даних займається фізичним зберіганням даних, це рівень системних програмістів та адміністраторів БД. Цей рівень найбільше впливає на ефективність роботи ІС.

Крім того ці рівні обов'язково діють у взаємозв'язку (рисунок 2.1).

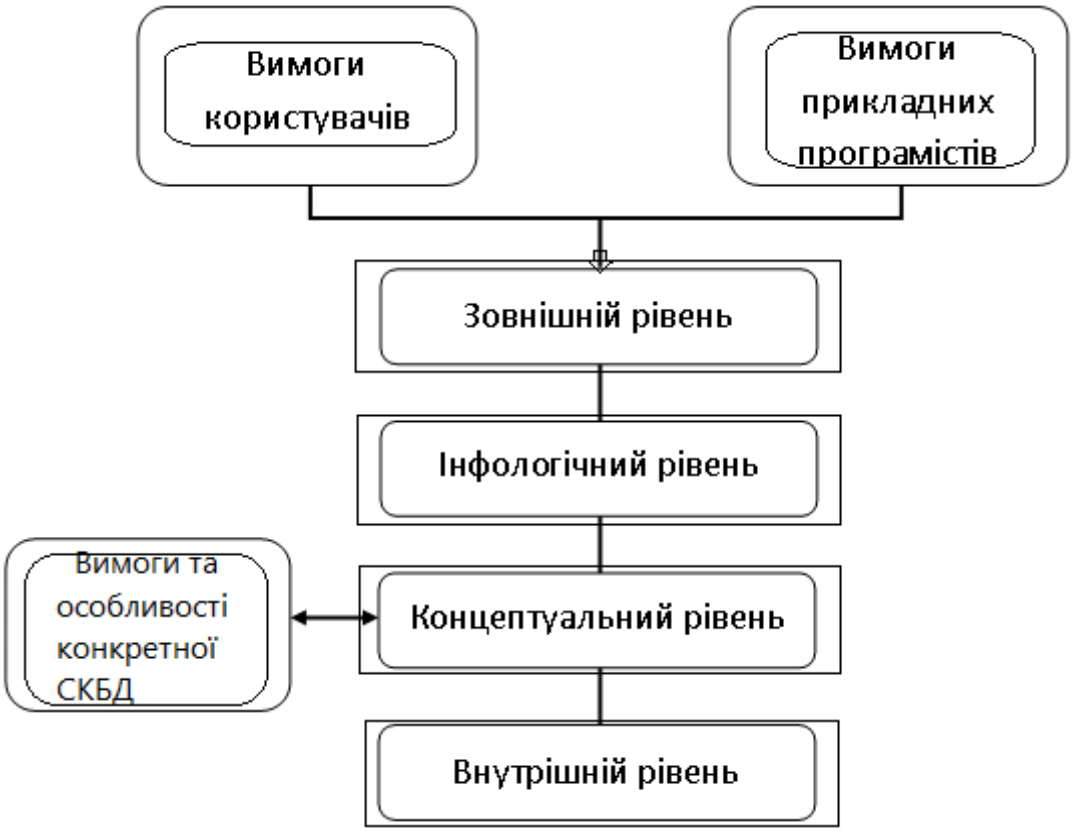


Рисунок 2.1 – Схема взаємозв’язку рівнів подання даних в БД

Зовнішній рівень – погляд на предметну область кінцевих користувачів та прикладних програмістів. На цьому рівні формується конкретний опис даних та їх взаємозв’язків.

Інфологічний рівень – це рівень, що відповідає погляду адміністратора на ПО. На цьому рівні можна спостерігати всю множину інформаційних об’єктів і зв’язки між ними. Сутність інфологічного моделювання полягає у виділенні інформаційних об’єктів, які підлягають зберіганню в БД, а також у визначенні атрибутів об’єктів і зв’язків між ними.

Концептуальний (датологічний) рівень відповідає уявленню про логічну організацію даних і формується з урахуванням специфіки конкретної СКБД. Цей рівень подібний з інфологічним, але має суттєву відмінність, яка полягає у прив’язці до засобів реалізації конкретної СКБД. На цьому рівні дані описуються за допомогою мови опису даних вибраної системи.

Внутрішній рівень - відповідає поданню й збереженню даних в пам’яті гаджета. Параметри внутрішнього рівня подання баз даних впливають на продуктивність функціонування ІС.

В мережі при інтеграції комп'ютерів є можливість розподілу додатків, що працюють з єдиною БД, та самої бази даних також. Найбільш поширеною є схема, при якій кожен користувач має свою персональну БД (КБД) і звертається до серверної БД (СБД) за інформацією, що спільно використовується багатьма користувачами.

Під **сервером** розуміється комп'ютер або програма, які керують певними ресурсами. **Клієнт** – це теж комп'ютер або програма, що користується цим ресурсами.

Таким чином є можливість поєднувати всі переваги централізованого зберігання даних та індивідуальної роботи користувачів.

Вибір моделі даних залежить від ряду факторів [5], таких як :

1. Обсяг інформації, який потрібно обробити і зберегти.
2. Складність задач, що вирішуються за допомогою моделі даних.
3. Наявність технічного та програмного оснащення для реалізації моделі даних.
4. Засоби маніпулювання даними, які використовуються для операцій з інформацією.
5. Цілісність та захист даних, що є важливими аспектами при обробці і зберіганні інформації.
6. Критерії якості, такі як надійність і точність моделі.
7. Можливість розвитку й масштабування моделі даних для відповіді на майбутні потреби.

Також варто зауважити, що вибір моделі даних часто визначає вибір системи керування базами даних, що буде використовуватися для роботи з даними.

Контрольні запитання:

1. Як називають процес побудови логічного подання структури БД?

2. Як можна визначити поняття модель даних?

3. Які основні складові моделі даних?

4. Що відображає інфологічна модель?

5. Що представляє собою даталогічна модель даних?

6. Що відображає фізична модель?

7. Що представляє семантична модель?

8. Що відображає модель «сутність-зв'язок»?

9. Що передбачають документальні моделі?

10. Яка модель називається теоретико–множинною, а яка теоретико-графовою?

11. Що можна сказать про фактографічну модель?

12. Що представляє собою ієрархічна модель?

13. Що відображає мережна модель?

14. На чому ґрунтується реляційна модель?

15. Що відомо про об'єктно-орієнтовану модель?

16. Яку модель називають багатомірною?

17. Що відомо про трирівневу модель архітектури СКБД?

18. Що називають схемою?

19. Що розуміють під локальним поданням даних?

20. Що розуміють під концептуальним поданням даних?

21. Що розуміють під формалізованим поданням даних?

22. Що розуміють під внутнішнім поданням даних?

3. Загальна термінологія реляційної моделі даних

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

В 1970 році Едгар Кодд запропонував концепцію реляційної моделі даних для розв’язування такої задачі: забезпечення незалежності подання опису даних від прикладних програм.

РМД ґрунтується на понятті «відношення» (relations), що подається у формі таблиці з певними умовами обмежень, яких потрібно дотримуватись.

Основні елементи РМД можна надати у вигляді деякої таблиці:

Таблиця 3.1 – Основні елементи РМД

Елементи РМД	**Форма подання**
Відношення	Таблиця
Схема відношення	Рядок заголовків стовпців таблиці
Кортеж	Рядок таблиці (запис)
Сутність	Опис властивостей об’єкту
Атрибут	Заголовок стовпця таблиці (поле)
Домен	Множина допустимих значень атрибутів
Значення атрибута	Значення поля в записі
Первинний ключ	Один або декілька атрибутів
Тип даних	Тип значень елементів таблиці

Відношення – це основне поняття БД, що являє собою двовимірну таблицю, в якій можуть міститись певні дані.

Відношення РБД можна розділити на дві групи:

- об’єктні відношення;

- зв’язні відношення.

Про **об’єктне відношення:** зберігає дані , що стосуються об’єкта (екземпляри сутності). В об’єктному відношенні є один або декілька атрибутів, що однозначно ідентифікують об’єкт. Їх називають **ключем** відношення (або первинним ключем). Основне обмеження в реляційній МД, що забезпечує цілісність даних, наступне: атрибути в об’єктному відношенні не повинні дублюватись.

Зв’язне відношення – це відношення, що зберігає ключі двох або більше об’єктних відношень. Завдяки цим ключам встановлюються зв’язки між об’єктами відношень.

Сутність – об’єкт будь-якого походження, дані про який зберігаються в БД. Інформація про певну сутність зберігаються у певному відношенні.

Атрибути – це властивості, завдяки яким можна характеризувати сутність.

В структурі таблиці кожен елемент має назву, яка відповідає заголовку певного стовпця у таблиці.

Кортеж являє собою рядок у таблиці, тобто один екземпляр таблиці (запис). Порядок кортежів у відношенні є не визначеним. Проте порядок атрибутів у відношенні є визначеним, і якщо поміняти їх місцями, отримаємо нове відношення.

Домен - це набір всіх можливих значень певного атрибута у відношенні.

Схема відношення або заголовок відношення – це перелік імен атрибутів.

Тіло відношення – так називають набір кортежів у відношенні.

На рисунку 3.1 надається приклад таблиці, яка відображає тип об’єкта реального світу, а кожен рядок – конкретний екземпляр об’єкта.

Стовпець таблиці – це набір значень певного атрибута об’єкта. Ці значення вибираються з набору всіх можливих значень атрибутів об’єкта, який називається доменом.

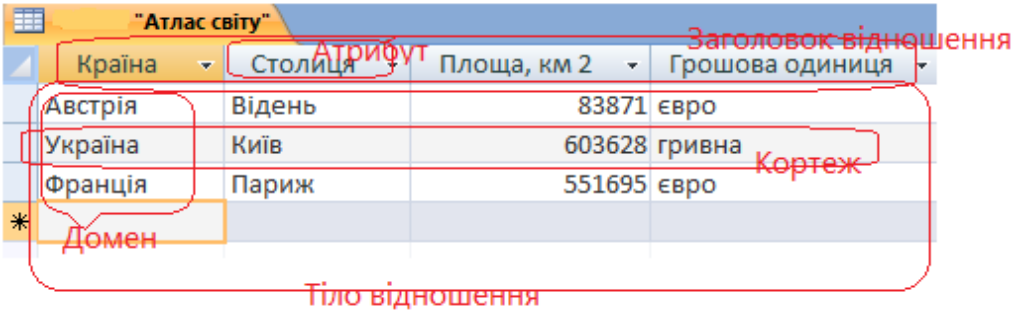


Рисунок 3.1 – Елементи РМД таблиці «Атлас світу»

Ключ – є важливим поняттям РМД. Існують наступні поняття пов’язані з ключем:

Потенційний ключ [1] – це мінімальний піднабір атрибутів відношення, який однозначно ідентифікує кортежі даного відношення.

Первинний ключ – це потенційний ключ, що обрали для унікальної ідентифікації кортежів відношення.

Вторинний ключ – ключ, кожному значенню якого може відповідати більше ніж один екземпляр індексованих даних, може бути унікальним для деякої підмножини даних, але може мати повторювані значення в інших частинах таблиці.

Зовнішній ключ – це сукупність атрибутів одного відношення, значення яких є одночасно також значеннями первинного або потенційного ключа іншого відношення.

В кожному відношенні обов’язково має бути атрибут або комбінація атрибутів, що може виконувати функцію ключа. Бувають відношення, які мають кілька комбінацій атрибутів, кожна з яких однозначно визначає всі кортежі відношення та претендують на роль

ключа відношення. Можна вибирать любий з можливих варіантів ключів, як первинний. Первинний ключ називають не надмірним, якщо він складається з мінімально необхідної множини атрибутів.

Ключі застосовують для досягнення наступних цілей:

1. Запобігають дублюванню значень в ключових полях [2] (інші атрибути не враховуються);

2. Впорядкування кортежів, що стає можливим при збільшенні або зменшенні значень всіх ключових полів;

3. Пришвидшення роботи з кортежами відношення;

4. Необхідні для операцій поєднання таблиць.

Нехай відношення R має не ключовий атрибут A, значення якого є значеннями ключового атрибута іншого відношення, тоді говорять, що атрибут A відношення R є **зовнішнім ключем** (рисунок 3.2).

В нашому прикладі є два відношення: R_Студенти з первинним ключем «ПІБ» і R_Предмети з первинним ключем «Код предмета», які пов’язані з відношенням R_Успішність, в якому атрибути «ПІБ» і «Код предмета» є зовнішніми ключами по відношенню до відношень R_Студент і R_Предмети відповідно.

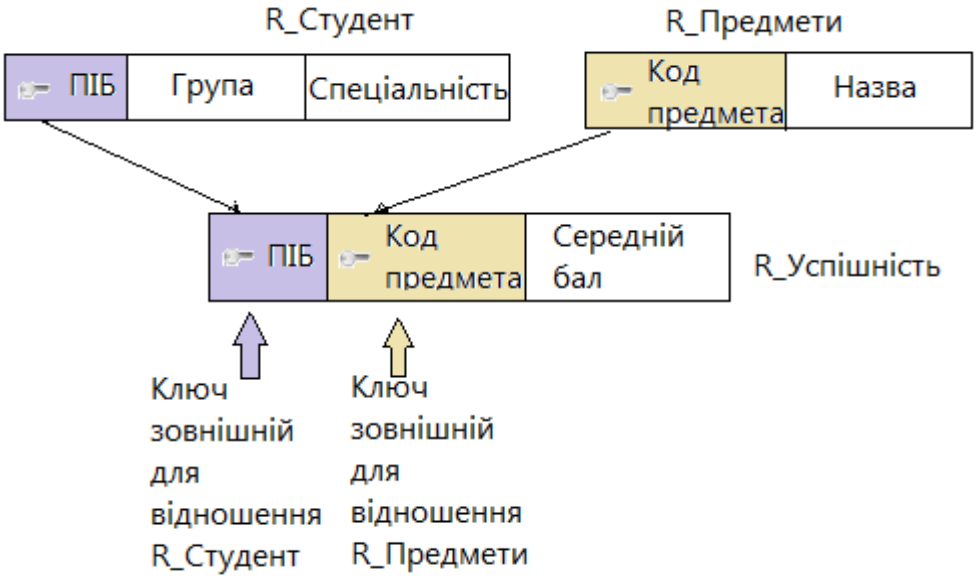


Рисунок 3.2 – Приклад зовнішніх ключів

У РМД є деякі властивості: це природність та простота структур даних, що використовуються й операцій маніпуляції даними; незалежність від середовища зберігання даних; підтримка віртуальних, а не фізичних зв'язків між даними.

Реляційна БД може складатись з наступних частин:

− **масиви інформації** (у вигляді таблиць, індексів);

− **інформація системи** (структура БД, обмеження цілісності, правила);

− **прикладні програми** (модулі, процедури, тригери, макроси, API, тощо).

Операційні можливості відношення базуються на **реляційній алгебрі** та **реляційному обчисленні**.

Існують мови запитів реляційної алгебри та мови реляційного обчислення, принципи побудови яких запропонував Е.Кодд. На відміну від теоретичних мов, реальні сучасні мови запитів, такі як SQL або QBE забезпечують не тільки функції відповідної теоретичної мови, але і реалізують деякі додаткові арифметичні операції, операції друку, тощо.

Цілісність баз даних

Цілісність баз даних – властивість даних, що визначає повноту і коректність інформації, яка вміщується в БД.

В підтримку цілісності БД входить такі складові:

- структурна цілісність;
- обмеження реальних значень даних;
- посилальна цілісність.

При **структурній цілісності** вимагається дотримання наступних умов:

- існування тільки реляційних відношень (серед структур даних);
- виключення кортежів, що дублюються;

– обов'язкове існування первинного ключа в кожному відношенні;

– обмеження доменів, яке передбачає визначення кожного атрибуту на своєму домені;

– можливість застосування значень NULL (невизначених значень) (позначає відсутність будь-якого значення атрибуту).

Обмеження реальних значень даних – це вимога, щоб значення поля відповідали деякому інтервалу значень, або задовільняли певне арифметичне співвідношення між значеннями певних полів таблиці. Обмеження значень можуть включати також визначення певних форматів для полів, задоволення значень полів певним статистичним умовам, бізнес правилам предметної області й т.ін.

Посилальна цілісність вказує, що зміни в таблицях повинні виконуватися синхронно, а зміст двох пов'язаних таблиць має відповідати таким правилам:

– кожному запису головної таблиці відповідає нуль або більше записів підпорядкованої таблиці;

– в підпорядкованій таблиці немає записів, які не відповідають записам в головній таблиці;

– Кожний запис підпорядкованої таблиці має тільки один батьківський запис головної таблиці.

З умовами цілісності даних визначається, які дані можуть бути записані в БД у результаті внесення, видалення або редагування даних (правила можна оглянути у таблицях 3.2 і 3.3).

Таблиця 3.2 - Правила видалення даних в БД

RESTRICT	Заборона вилучення рядка з батьківської таблиці, якщо він має нащадків в підлеглій таблиці.
CASCADE	При видаленні рядка з батьківської таблиці в підлеглій таблиці всі рядки-нащадки автоматично видаляться
SET NULL	При видаленні рядка з батьківської таблиці в підлеглій таблиці всім зовнішнім ключам рядків-нащадків автоматично привласнюється значення NULL
SET DEFAULT	При видаленні рядка з батьківської таблиці в підлеглій таблиці всім зовнішнім ключам рядків-нащадків автоматично привласнюється певне значення встановлене за замовчуванням

Таблиця 3.3 - Правила оновлення даних в БД

RESTRICT	Заборона зміни первинного ключа в рядку батьківської таблиці, якщо в підлеглій таблиці цей рядок має нащадків
CASCADE	При зміні первинного ключа в рядку батьківської таблиці в підлеглій таблиці відповідні значення зовнішнього ключа також автоматично змінюються у всіх рядках-нащадках для того, щоби відповідати новому значенню первинного ключа
SET NULL	При зміні первинного ключа в рядку батьківської таблиці в підлеглій таблиці відповідні значення зовнішнього ключа також автоматично змінюються у всіх рядках-нащадках і їм привласнюється значення NULL
SET DEFAULT	При зміні первинного ключа в рядку батьківської таблиці в підпорядкованій таблиці відповідні значення зовнішнього ключа також автоматично змінюються у всіх рядках-нащадках і їм привласнюється певне значення встановлене за замовчуванням

Також можливо виконання правила NONE – не виконуються ніякі дії і правила NULL ALLOWED – дозволяються невизначені значення.

При додаванні даних в таблицю (INSERT) необхідно дотримуватися такої послідовності введення: спочатку дані вводяться в батьківську таблицю, а потім в підпорядковану.

Алгеброю називають множину об'єктів із заданою на ній купою математичних операцій та правил, які замкнені відносно цієї множини.

Основною множиною в реляційній алгебрі є множина відношень. Варіант реляційної алгебри, запропонований Коддом, містить наступні **основні операції**: об'єднання, декартів добуток, різниця, перетин, проекція, селекція, ділення, з'єднання.

Контрольні запитання:

1. Як можна визначити реляційну модель даних (РМД) деякої ПО?
2. Що лежить в основі концепції РМД?
3. Як можна визначити поняття відношення для РБД?
4. Яке відношення називають об’єктним?
5. Яке відношення називають зв’язним?
6. Що означає поняття сутність?
7. Що означає поняття атрибат?
8. Що означає поняття кортеж?
9. Що означає поняття домен?
10. Що означає поняття схема відношення?
11. Із чого складається тіло відношення?
12. Що означає поняття первинний ключ?
13. Що означає поняття зовнішній ключ?
14. Що означає цілісність БД?
15. Що означає структурна цілісність?
16. Що передбачає посилальна цілісність?
17. Що означає поняття алгебра для БД?

4. Проектування й застосування баз даних

Про проектування баз даних

Якість і продуктивність роботи ІС знаходиться в залежності від таких складових:

- Проектування та розгортання БД;
- Проектування й запровадження додатків;
- супровід ІС.

База даних є основною частиною ІС, а проектування БД здійснюється при проектуванні інформаційної системи.

Інформаційна система має **життєвий цикл** (Systems Development Life Cicle, SDLC), що складається з наступних етапів:

- розпланування;
- визначення (збір, аналіз) вимог;
- проектування;
- втілення;
- тестування;
- супровід.

Ці етапи не є жорстко послідовними та передбачають повернення на попередні етапи за допомогою зворотних зв'язків. Як частина ІС, база даних існує за своїм життєвим циклом. **Життєвий цикл БД** складається з наступних етапів:

1. Планування БД;
2. Визначення вимог до БД;
3. Проектування БД (концептуальне, логічне, фізичне);
4. Розробка застосувань;
5. Реалізація й завантаження даних;
6. Тестування;

7. Експлуатація.

Кожний етап значною мірою залежить від складності програмного продукту, що розробляється. Для невеликих ІС кількість етапів може бути зменшена.

Що розглядається в кожному етапі проектування:

1. **Етап планування** БД передбачає створення загального плану, який дозволить ефективно реалізувати етапи життєвого циклу. Тут вирішуються такі питання:

-аналіз існуючих інформаційних систем;

-доцільність зміни існуючої ІС;

-обсяг робіт і ресурсів, вартість проекту;

-визначення технічного завдання для проекту БД;

-визначення технічних вимог;

-розробка методології збору даних, визначення їх формату;

-визначення необхідної документації;

-визначення послідовності проектування і реалізації застосувань.

2. На **етапі визначення вимог** до бази даних вирішуються такі задачі:

-визначення діапазону дії і меж застосувань БД;

-визначення складу користувачів і сфер застосування;

- визначення представлень користувачів, що підтримуються БД.

На цьому етапі також збираються та аналізуються вимоги користувачів:

-опис даних, що застосовуються (вхідні й вихідні документи);

- детальні відомості про транзакції;

-відомості про засоби застосування даних.

Грунтуючись на всій попередній інформації створюються специфікації вимог користувачів.

3. Процес **проектування БД** являє собою послідовність переходів від неформального мовного опису інформаційної структури предметної

області до формалізованого опису об'єктів ПО в термінах деякої моделі. Проектування БД складається з таких етапів:

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

Системний аналіз передбачає мовний опис реальних об'єктів ПО, визначення зв'язків між об'єктами, дослідження характеристик об'єктів і зв'язків. Результати дослідження використовуються при концептуальному проектуванні БД.

Для того щоб зробити визначення складу та структури ПО використовують наступні методи:

Функціональний підхід ґрунтується на діях "від задач" і застосовується у тих випадках, коли заздалегідь відомі функції майбутніх користувачів БД, а також відомі всі задачі, для інформаційних потреб яких створюються БД. В такій ситуації на основі виробничих документів та вимог замовників складється визначення мінімального набіру об'єктів ПО та їх взаємозв'язку.

Предметний підхід використовують якщо інформаційні потреби майбутніх користувачів чітко не визначені та не можна чітко описать мінімальний набір об'єктів ПО. В опис ПО включаються об'єкти та зв'язки, які є найбільш характерними та найбільш суттєвими. БД визначається як предметна й застосовується для розв’язку задач, що наперед не визначені.

Часто зустрічається *комплексний підхід*, який дозволяє розв’язувати конкретні інформаційні та функціональні задачі, й також враховує можливість додавання нових застосувань.

Загалом існує два підходи до проектування БД: **низхідне** проектування й **висхідне** проектування.

Низхідне проектування - це спочатку визначення наборів даних, потім визначаються елементи даних для кожного з таких наборів. Цей процес передбачає виявлення різних типів сутностей та розпізнавання атрибутів кожної сутності. Оглядаються операції *декомпозиції*, що мають на увазі заміну вихідного набору відношень, що складають схему БД, іншим набором відношень, що є проекціями цих відношень.

Такий підхід застосовують у тих ситуаціях, коли кількість, різноманітність та складність сутностей, зв'язків і транзакцій значна за розмірами. Найрозповсюдженішими моделями для цього проектування є моделі "сутність - зв'язок", інакше ER-моделі (Entity-Relationship model).

Висхідне проектування – це спочатку виявлення елементів даних, які потім групуються в набори даних. Визначаються атрибути, які потім об'єднуються в сутності. Висхідне проектування містить в собі операції синтезу, що означає виконання компоновки з заданого набору функціональних залежностей об'єктів ПО вихідних відношень в схемі БД. Застосовують такий підхід, якщо створюють невелику БД з незначною кількістю об'єктів, атрибутів і транзакцій.

Концептуальне проектування передбачає створення концептуальної моделі, яку надає концептуальна схема БД. На цьому етапі виконується визначення об'єктів, зв'язків між об'єктами, атрибутів, ключових атрибутів.

Логічне проектування передбачає створення логічної моделі на основі вибраної моделі даних. Цей етап потребує визначення СКБД, що буде застосовуватись в системі (реляційна, ієрархічна, мережна, об'єктно-орієнтована). Для перевірки вірності логічної моделі застосовується **нормалізація**. Окрім цього логічну модель обстежують на можливість забезпечення всіх транзакцій користувачів в БД.

Фізичне проектування передбачає визначення засобів фізичної реалізації отриманого логічного проекту для БД. Фізичні моделі

виконують визначення засобів розміщення даних в середовищі зберігання й засобів доступу до цих даних, які підтримуються на фізичному рівні.

4. **Застосування** – це програма, яка призначена для вирішення деякої сукупності задач в даній ПО, або яка являє собою типовий інструментарій, що застосовується в різних областях.

На цьому етапі вирішуються такі задачі:

– проектування транзакцій;

– проектування інтерфейсів користувачів.

Транзакція являє собою послідовні декілька операцій по роботі з БД, які трансформують БД з одного цілісного стану в інший. Зустрічаються транзакції: для отримання вибраної інформації з БД; транзакції для зміни даних в БД (додавання, оновлення, видалення) та змішані транзакції.

Інтерфейс користувача (UI або user interface) **–** це засіб зручної взаємодії користувача з системою, що складається з сукупності функціональних компонентів.

5. Для етапу **реалізації** характерні вирішення наступних питань**:**

- встановлюється технічне і програмне забезпечення СКБД;
- реалізується проект БД;
- реалізуються прикладні програми;
- реалізуються форми введення/виведення даних і звіти;
- наповнення БД даними;
- захист БД від несанкціонованого втручання;
- підтримка цілісності БД.

Реалізація БД відбувається за кошт створенню опису на мові визначення даних конкретної СКБД або з використанням графічного інтерфейса користувача. Застосування створюються за допомогою мов третього та четвертого покоління або за рахунок розширень мов БД.

Реалізація може виконуватися за допомогою інструментів автоматизованого проектування.

6. Для етапу **тестування характерне** вирішення наступних питань:

- діагностується на коректність роботи окремих модулів або функціональних компонентів (альфа-тестування);

- перевіряється продуктивніть роботи системи, визначаються потреби в ресурсах;

- виконується дослідницька експлуатація (тобтоо бета-тестування), обстежується на відповідність розробленої системи її специфікаціям.

Щоб покращити роботу системи розглядається модифікація логічного й фізичного проектів, зміна чи оновлення програмного забезпечення СКБД, заміна технічного забезпечення. Також виконується налаштування системних параметрів і параметрів СКБД.

7. На етапі **експлуатації** вирішуються такі задачі:

– контроль продуктивності роботи системи, а в разі потреби, збільшення продуктивності (наприклад через створення додаткових індексів);

– супроводження й модернізація застосувань БД;

– обслуговування для профілактики (наприклад, резервне копіювання);

– коригувальне обслуговування (відновлення БД);

– надання прав доступу для нових користувачів;

– організація статистики доступу до БД для підвищення ефективності роботи системи;

– періодичне інспектування безпеки;

– зведення використання системи.

Контрольні запитання:

1. З яких етапів складається життєвий цикл БД?

2. Що спільного у життєвих циклів інформаційної системи і бази даних?

3. Яка різниця між функціональним і предметним підходами в проектуванні БД?

4. У чому полягає спільність і різниця трьох етапів проектування БД?

5. Назвати етапи проектування БД.

6. З чого складається етап планування БД?

7. У чому полягає аналіз вимог до БД?

8. Яку роль відіграють застосуваня в проектуванні БД?

9. Які задачі вирішуються на етапі експлуатації?

5. Концептуальні моделі

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

- семантичні моделі;
- об'єктні моделі.

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

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

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

Модель "сутність-зв'язок"

ER-моделювання являє собою низхідний підхід до проектування БД, який починається з визначення найбільш важливих даних, які називаються *сутностями* (entities), і *зв'язків* (relationships) між даними, які повинні бути подані в моделі. Наступною дією в модель заноситься інформація про властивості сутностей та зв'язків, що називають *атрибутами*, а також обмеження, що відносяться до сутностей, зв'язків

й атрибутів. Завдяки ER-моделі маємо графічне подання логічних об'єктів і їх відношень в структурі бази даних. Послідовність виконання ER-моделювання показана на рисунку 5.1.

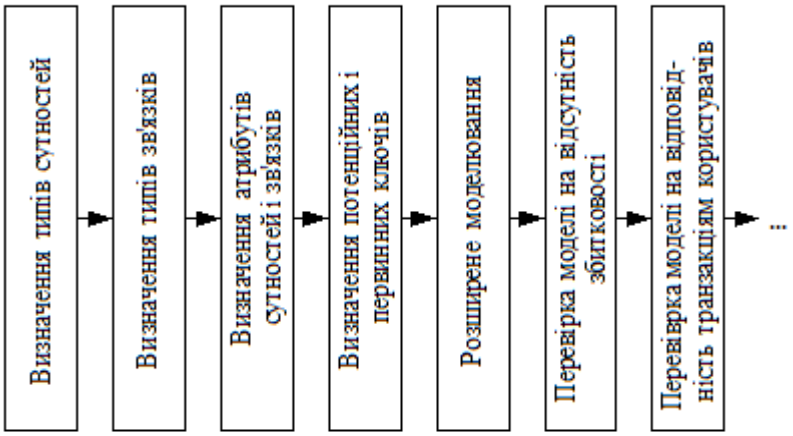


Рисунок 5.1 - Побудова моделі "сутність – зв'язок" поетапно [1]

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

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

...

Сутності

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

Набір атрибутів, який однозначно визначає конкретний екземпляр сутності, називають **ключем**. У прикладі, що надано, для сутності *Викладач* ключем є *Табельний номер*, бо табельні номери всіх викладачів різні. Екземпляром сутності *Викладач* буде опис конкретного викладача. Загальноприйняте позначення сутності - прямокутник (рисунок 5.2).

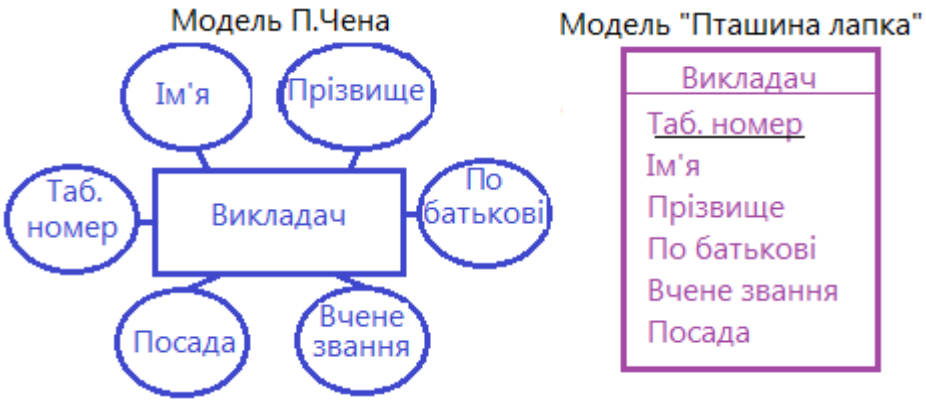


Рисунок 5.2 - Подання сутностей та атрибутів у ER-діаграмах П. Чена й «Пташина лапка»

Зв'язки

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

- Бінарні, тобто між двома сутностями;
- Тернарні, тобто між трьома сутностями ;
- N-арний зв'язок , тобто між N сутностями;
- Рекурсивні, тобто між однією сутністю.

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

- 1:1 (один до одного);
- 1:M (один до багатьох);

– N:M (багато до багатьох).

На рисунках 5.3 та 5.4 подаються зв'язки у різних ER-моделях.

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

Рисунок 5.3 - Зв’язок 1:1 на діаграмі Чена

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

Рисунок 5.4 - Зв’язок 1:M на діаграмі Чена

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

Рисунок 5.5 - Зв’язок N:M на діаграмі Чена

На рисунку 5.6 зображено представлення зв'язків (*а* - 1:1; *б* - 1:M; *в* - N:M) між відношеннями на діаграмі «Пташина лапка».

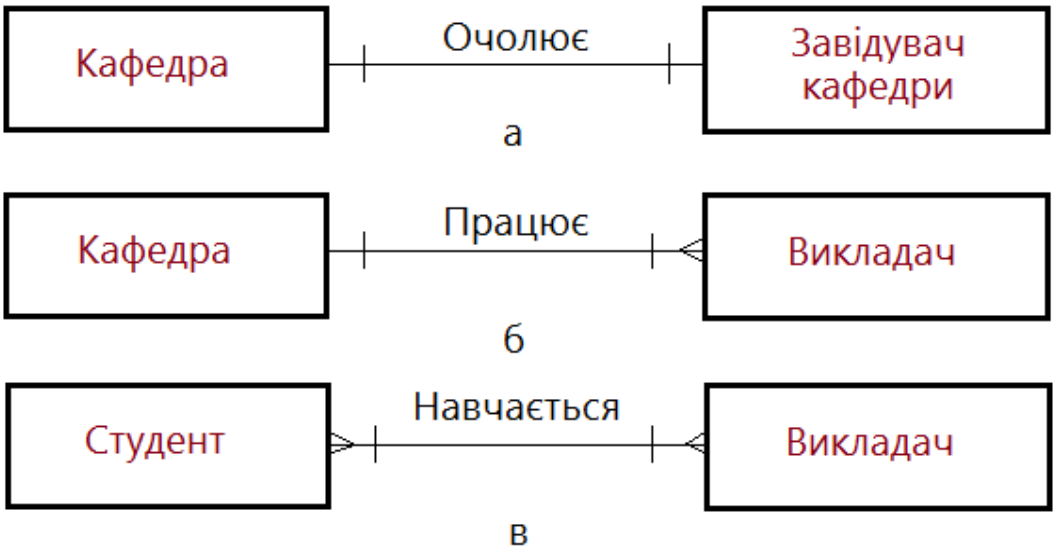


Рисунок 5.6 – Зв’язки 1:1, 1:M та N:M на діаграмі «Пташина лапка»

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

Атрибути залежно від складності значень, які вони можуть приймати поділяються на категорії або типи (Таблиця 5.1). Це прості, складові, однозначні, багатозначні й похідні атрибути. Властивості й приклади видно в таблиці.

Таблиця 5.1 – Типи атрибутів [1]

Тип	**Властивість**
Простий	Атрибут, який не може бути поділений на інші атрибути. Приклад: Прізвище; посада
Складовий	Атрибут, який можна поділити на інші атрибути. Приклад: Адреса; Прізвище, Ім'я, По батькові

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

Для приклада розглядається сутність **Студент** (рисунок 5.7).

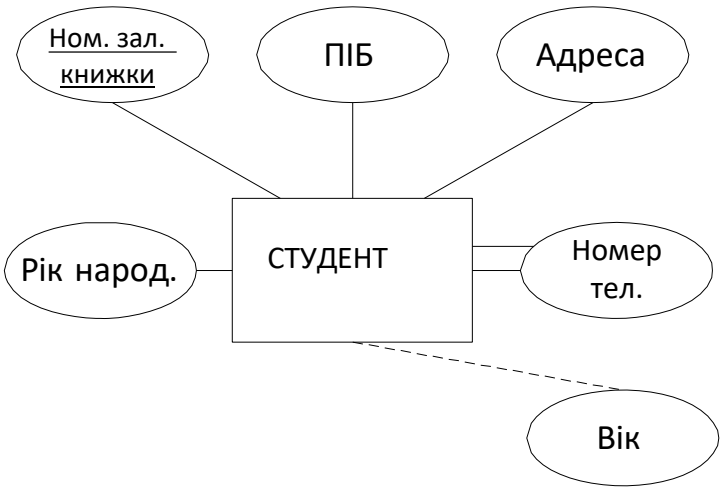


Рисунок 5.7 - ER-діаграма - сутність **Студент**

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

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

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

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

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

Потенційний ключ - це атрибут або набір атрибутів сутності, які застосовуються для ідентифікації екземпляра сутності. Сутність може мати декілька потенційних ключів. В прикладі на рисунку потенційними ключами можуть бути такі атрибути: **Номер залікової книжки, Прізвище Ім'я По батькові**.

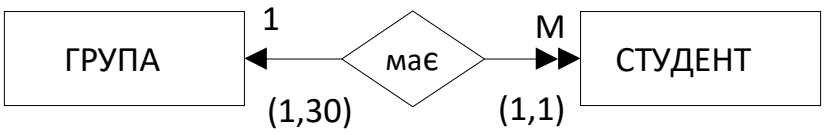
Потенційний ключ, що вибрано для однозначної ідентифікації кожного екземпляра сутності певного типу, називається **первинним ключем**. Первинний ключ визначається з умовою, що є гарантії унікальності його значень, а також мінімальної довжини атрибутів, які входять в його склад. В прикладі - первинний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

Потужність зв'язків

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

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

Для приклада розглядається зв'язок **Група-Студент** (рис. 5.8).



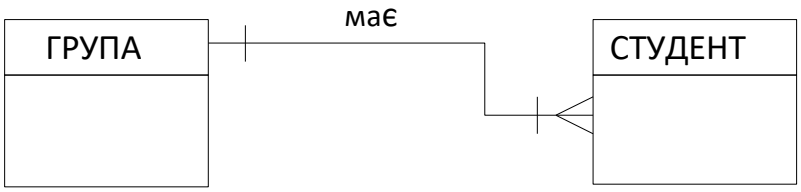


Рисунок 5.8 - ER-діаграми показують зв'язок і потужність [1]

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

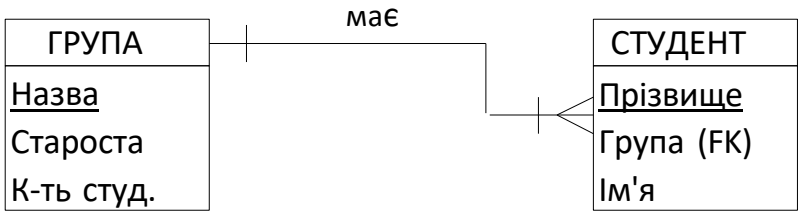
Сильні та слабкі зв'язки

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

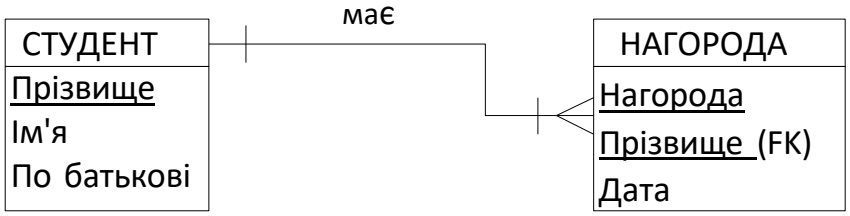
Якщо сутність незалежна від існування іншої, то зв'язок між ними називається **неідентифікаційним зв'язком** або **слабким зв'язком**. На ER-діаграмах "пташина лапка" слабкий зв'язок відображається штриховою лінією.

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

Для прикладу розглянемо зв'язки **Група-Студент** і **Студент-Нагорода** (рисунок 5.9).



а



б

Рисунок 5.9 - На ER-діаграмах неідентифікаційний (*а*) та ідентифікаційний (*б*) зв'язки [1]

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

Якщо сутність **Нагорода** залежна від сутності **Студент**, то лінія зв'язку між сутностями позначається суцільною, а **Прізвище студента,** атрибут сутності **Нагорода,** є одночасно частиною первинного ключа (Primary Key, PK) та Foreign Key (FK).

Атрибути зв'язків

Подібно сутностям, зв'язки можуть мати атрибути.

Для прикладу атрибути **День** і **Аудиторія** належать до зв'язку між сутностями **Студент** і **Дисципліна** (рисунок 5.10).

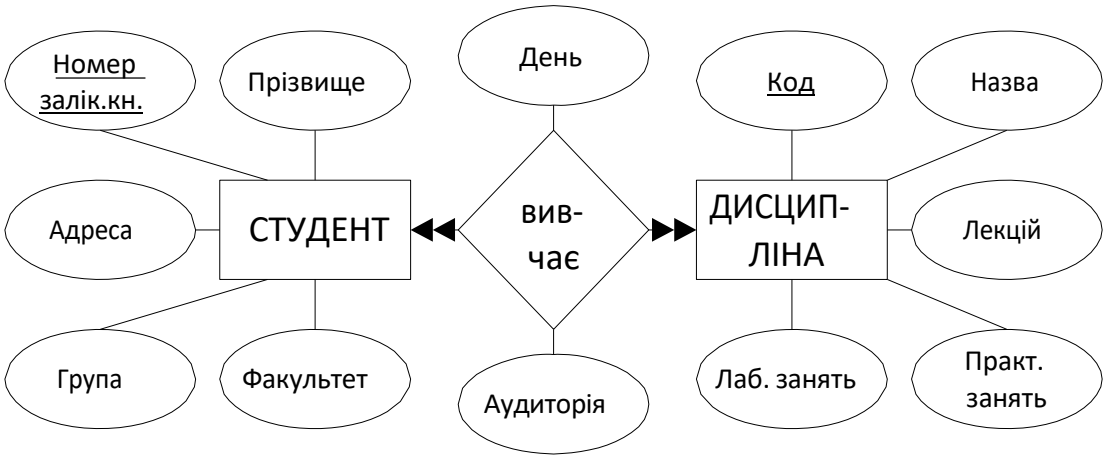


Рисунок 5.10 - Атрибути зв'язку **День** і **Аудиторія** на ER-діаграмі

Обов'язкові / необов'язкові зв'язки

Участь сутності у зв’язку називають необов’язковою, у випадку, якщо один з екземплярів сутності не має потреби в наявності відповідного екземпляра сутності в окремому зв'язку. І навпаки. У зв’язку сутностей, наприклад, **Студент-Нагорода,** не всі студенти мають нагороди. Це означає, що у таблиці **Студент** не кожен екземпляр мусить мати екземпляр сутності в таблиці **Нагорода**. В такому випадку сутність **Нагорода** визначається як необов'язкова відносно до сутності **Студент.** Необов'язкова сутність схематично позначається у вигляді невеликого кола з боку необов'язкової сутності (рисунок 5.11). Якщо є необов'язковість, то це означає, що мінімальне значення потужності зв'язку для необов'язкової сутності буде дорівнювати 0.

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

Наприклад, зв'язок між сутностями **Студент** і **Нагорода** є необов'язковим (рисунок 5.11). Може бути така ситуація, що не у коженого студента є нагорода, але якщо вона є, то нагорода обов'язково пов'язана з певним студентом.

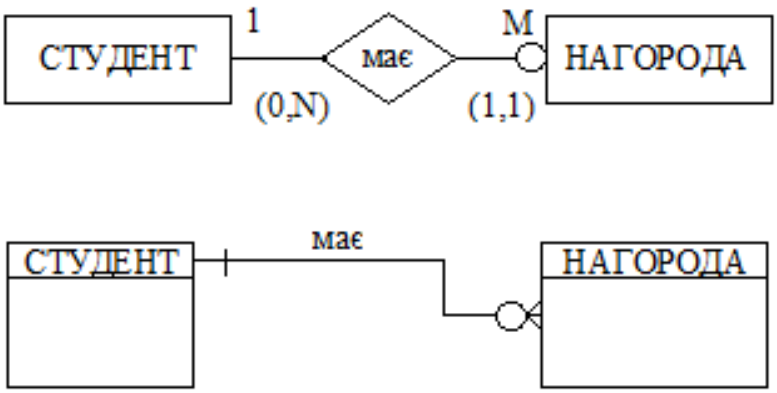


Рисунок 5.11 - Необов'язкова сутність в ER-діаграмах [1]

Слабкі сутності

Слабкою сутністю називається сутність, яка задовільняє таким умовам:

- залежності від існування сутності з якою вона зв'язана;
- первинний ключ цієї сутності частково або повністю отримано з іншої сутності.

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

Для прикладу на рисунку 5.12 Сутність **Нагорода** є слабкою по відношенню до сутності **Студент**: вона залежить від існування цієї сутності і в її первинний ключ входить первинний ключ сутності **Студент**.

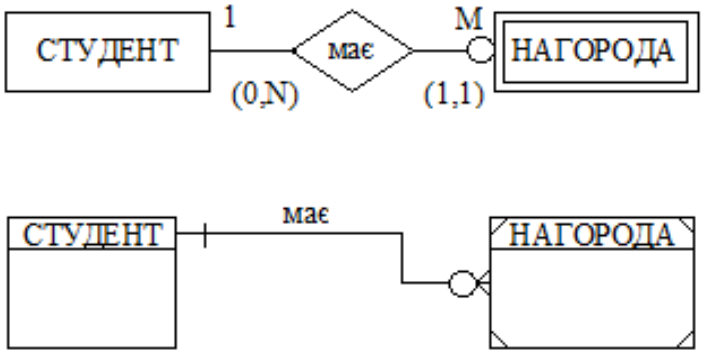


Рисунок 5.12 Позначення слабких сутностей на діаграмах П. Чена і "Пташина лапка"

Складні зв'язки

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

Для прикладу розглянуто сутності **Викладач**, **Дисципліна** й **Екзамен**, що утворюють тернарний зв'язок (рисунок 5.13).

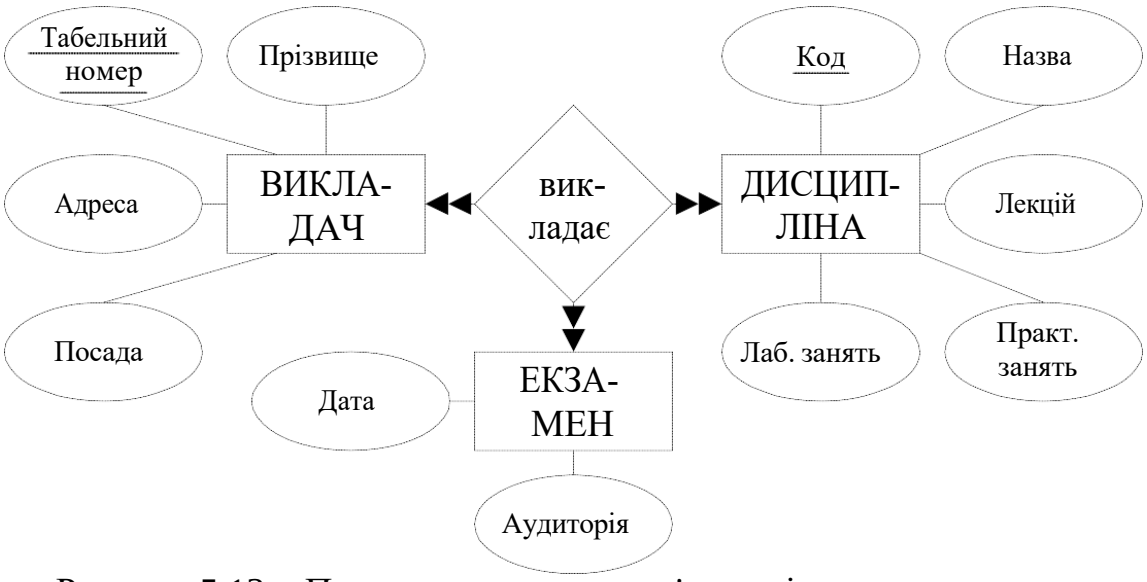


Рисунок 5.13 – Приклад тернарного зв'язку між трьома сутностями на діаграмі П. Чена

Рекурсія у зв'язках

Рекурсія у зв'язку відбувається у випадку, якщо є зв'язок між екземплярами того самого набору сутностей.

Для прикладу розглядаються можливі варіанти рекурсивних зв'язків (рисунок 5.14).

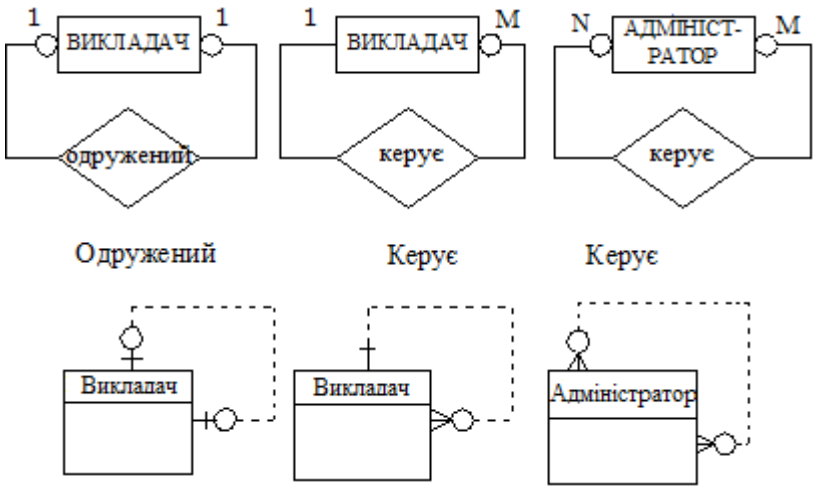


Рисунок 5.14 - Подання рекурсивних зв'язків на діаграмі П. Чена й моделі «Пташина лапка» [1]

На рисунку зв'язок 1:1 означає: якщо викладач є завідувачем кафедри, то керує декількома викладачами, а викладачі мають тільки одного керівника завідувача кафедри. M:N зв’язок означає, що адміністратор може мати декілька підлеглих-адміністраторів, і в свою чергу адміністратор має декілька керівників-адміністраторів.

Можна розглядати кілька зв'язків з різними змістовними навантаженнями між двома сутностями.

Якщо для прикладу розглянути сутності **Студент** та **Викладач**, то є сенс у створенні зв'язків **Викладає** та **Керівництво дипломним проектом**. Викладає викладач для багатьох студентів. Якщо взять кожного студента, то їм викладає багато викладачів. Для кожного студенту обов'язково повинен бути один керівник дипломного проекту, але не кожен викладач керує студентами (розглянуто на рисунку 5.15).

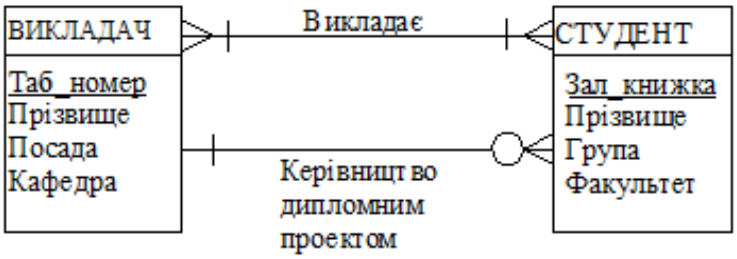


Рисунок 5.15 - Зв'язки з різними змістовними навантаженнями на моделі «Пташина лапка»

Розширена модель "сутність – зв'язок"

Для більш складних застосувань, що розширюють можливості, в семантичне моделювання були введені додаткові концепції у вигляді **розширеної ER-моделі** (Enhanced Entity Relationship, EER-модель). Окрім всіх загальних концепцій ER-моделі вона містить концепції узагальнення, уточнення, агрегування та композиції. Ці додаткові концепції за підґрунтя мають понятя **суперклас** та **підклас**. Суперклас може утримувати кілька підкласів. Як у прикладі, де підкласи **Керівник, Викладач** і **Лаборант** є учасниками суперкласу **Співробітник**. Тобто, кожен екземпляр підкласу одночасно є екземпляром суперкласу. Зв'язок між суперкласом та підкласом відноситься до зв’язку типу 1:1.

Поняття суперклас і підклас допомагають визначити для підкласів власні атрибути і атрибути, що наслідуються від суперклассу, наприклад, підклас **Викладач** повинен мати однакові атрибути, що і всі **Співробітники**. Але він має і власні атрибути, які не визначені для інших категорій працівників ВУЗу. Серед атрибутів можуть бути наступні: вчене звання, науковий ступінь, номер диплому про вчене звання, кількість наукових робіт, кількість навчально-методичних праць тощо. Якщо підкласи для об'єкту **Співробітник** відсутні, то треба було б створити атрибути, що мали б невизначене значення для певних співробітників (наприклад для інженерів). Підклас може мати власні зв'язки, що не підходять для всіх учасників суперкласу. Як приклад, **Викладач** може мати підкласи **Професор**, **Доцент**, **Старший викладач**, **Асистент**. Підклас наслідує не тільки атрибути, але й зв'язки суперкласу.

Термін **уточнення** означає процес збільшення різниці між екземплярами об'єкта завдяки визначенню їх характеристик, що не мають подібності. Цей процес - низхідний. Наприклад, кроки від об'єкта **Співробітник** до об'єктів **Викладач та Керівник**.

Узагальнення означає процес зменшення відмінностей між об'єктами до мінімуму завдяки виділенню їх спільних характеристик. Цей процес - висхідний. Як у прикладі, кроки від об'єктів **Керівник** і **Викладач** до об'єкту **Співробітник**.

Під час виконання процесів узагальнення або уточнення можуть застосовуватися обмеження не перетинання й ступеню участі.

Може бути що підкласи з наборів сутностей перетинаються, а може й не перетинаються. У випадку коли підкласи суперкласу не перетинаються, то це вказує, що кожен екземпляр сутності може бути елементом тільки для одного з підкласів (Or). Зв'язки, що не перетинаються, мають позначку "G". До прикладу, співробітник має можливість працювати або на посаді професора, або на посаді доцента, й не може бути одночасно на обох посадах.

У випадку коли перетинаються підкласи суперкласу, то це вказує, що будь-який екземпляр сутності може бути одночасно елементом кількох підкласів (And). А зв’язки, що перетинаються, мають позначку "Gs". Зустрічається, наприклад, що завідувач кафедри проводить заняття та, одночасно, виконує обов'язки керівника та викладача. На рисунку 5.16 подано ієрархію сутностей з підкласами, що не перетинаються та ті, що перетинаються.

Рисунок 5.16 - Приклад діаграми з використанням понять суперклас та підклас

Не обов’язково кожен елемент суперкласу має бути елементом одного з підкласів.

Але є зв’язки суперкласу з підкласом з *обов'язковою участю* (*Mandatory*). На рисунку 5.17 видно, що Студент обов'язково навчається або на денній, або на заочній, або на вечірній формі навчання, або екстернатом. В цьому випадку кожен елемент суперкласу повинен бути одночасно й елементом підкласу.

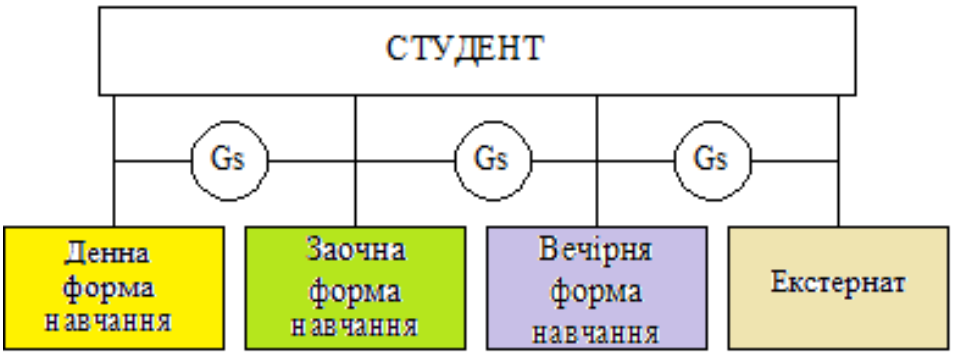


Рисунок 5.17 – Приклад діаграми зв'язку суперкласу з підкласом з обов'язковою участю

Є зв'язок суперкласу з підкласом з *необов'язковою участю*, який означає, що деякі елементи суперкласу можуть не належати жодному з підкласів (*Optional*). Для прикладу, якщо існують викладачі, які не обіймають посади асистента, професора чи доцента (наприклад старший викладач).

Поняття термінів суперкласів та підкласів стає в нагоді при виключенні опису різних екземплярів сутності, що володіють різними атрибутами. Це може сприяти появі набору незаповнених атрибутів. Крім того персональні атрибути можуть виявити зв'язки, які властиві тільки певним екземплярам сутностей, а не кожному.

В прикладі екземпляри сутності **Викладач**, що відносяться до викладачів на посаді професора, в порівнянні з викладачами на посаді асистента, володіють наступними додатковими атрибутами: вчений

ступінь, вчене звання, тощо. Окрім того, вони можуть мати зв’язок з сутністю **Аспірант** індивідуальними атрибутами.

Один підклас може бути пов'язаний з кількома суперкласами, які також є підкласами спільного для них суперкласу.

Якщо розглянути для прикладу підкласи **Викладач** і **Керівник** наслідують суперклас **Співробітник**. **Завідувач кафедри** - підклас є одночасно екземпляром суперкласу **Викладач** і також суперкласу **Керівник.**

Якщо підклас зв'язаний одразу з декількома суперкласами різних типів, то він вважається **категорією**. Буває категорія з повною участю і з частковою участю екземплярів. Повна участь передбачає, що кожен екземпляр всіх суперкласів зобов’язаний бути поданим у даній категорії. Часткова участь означає - присутність в категорії всіх екземплярів всіх суперкласі може бути необов'язковою.

Застосування EER-діаграм допомагають, якщо структура даних є занадто складною, для того, щоб її можна було з легкістю подати з використанням ER-діаграм.

Агрегування - являє собою зв'язок подібний на "входить у склад" чи "включає" між двома сутностями, одна з яких виконує роль "ціле", а інша -"частини".

Композиція - це особливий вид агрегування, що описує залежність між сутностями, які характеризуються повною приналежністю й співпадінням термінів існування між "цілим" та "частиною".

Проблеми при побудові моделей "сутність – зв'язок"

При невірній інтерпретації змісту деяких зв'язків виникають **дефекти з'єднання** *(*дефекти розгалуження і дефекти розриву) [1].

Дефекти розгалуження виникають, коли модель вірно подає зв'язки між сутностями, але шлях між окремими сутностями – є неоднозначно визначеним. Цей дефект з’являється, якщо два або більше зв'язків типу 1:M виникають з однієї сутності.

Розберемо деякі зв'язки: на факультеті навчається багато студентів, багато груп входять у склад факультету. Наприклад, ці зв'язки вірно відображають зміст предметної області, але при спробі ідентифікувати, в яких групах займаються конкретні студенти, виникають помилки. На рисунку 5.18 зображено: з сутності **Факультет** виходять два зв'язки типу 1:M.

Якщо надати коректну взаємодію цих сутностей, то можна усунути цю проблему (рисунок 5.19).

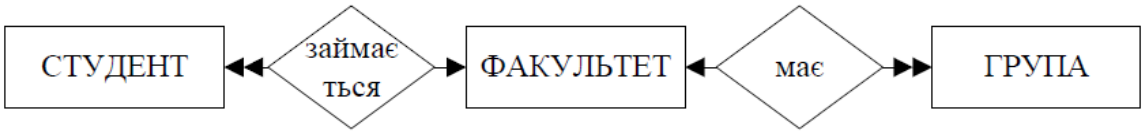


Рисунок 5.18 - ER-модель з дефектом розгалуження

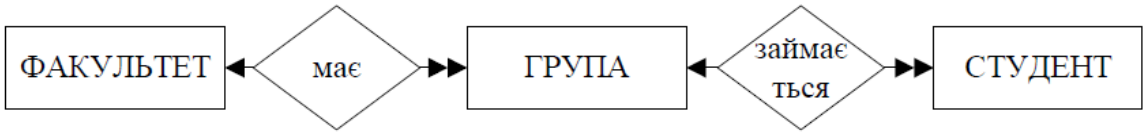


Рисунок 5.19 - ER-модель після усунення дефекту розгалуження [1]

Дефекти розриву – це такі дефекти, що виникають у тому випадку, коли в моделі передбачається наявність зв'язку між декількома сутностями. Таке стається у разі, коли існує один або декілька зв'язків з мінімальною потужністю, що дорівнює нулю, що визначає необов'язкову участь. Ці зв'язки становлять частину путі між взаємозв'язаними сутностями.

Для прикладу розглядаються наступні зв'язки: в склад факультету входять кафедри, кожна з яких може відповідати за кілька комп'ютерних лабораторій чи класів (від 0 до M). Є кафедри, що не мають відповідати за комп'ютерні класи. У той час комп'ютерна лабораторія може бути підпорядкованою певною кафедрою, а може

безпосередньо факультетом. Ці зв'язки коректно відображають зміст ПО, але при спробі з'ясувати, які саме комп'ютерні класи підпорядковані певному факультету, виникають запитання. Зв'язок між сутностями **Комп'ютерний клас та Кафедра** прогнозує участь сутностей - необов'язкову, і він є частиною напрямку між сутностями **Факультет та Кафедра**.

Введення додаткового зв'язку між сутностями **Факультет** і **Комп'ютерний клас** допоможе усунути цю проблему (рисунок 5.20).

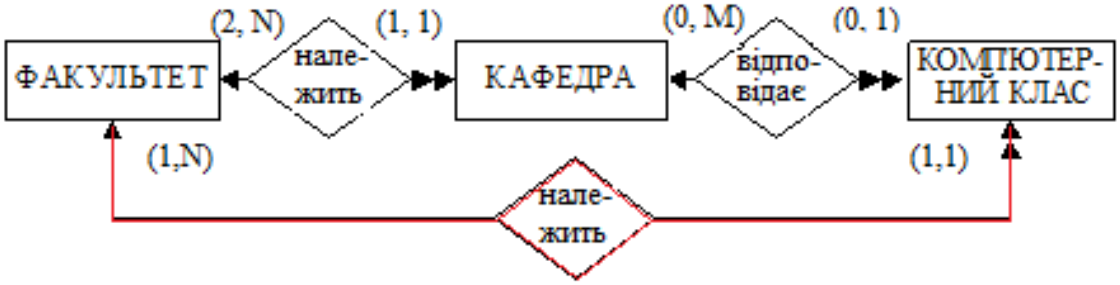


Рисунок 5.20 - ER-модель без дефекту розриву

Побудова моделі "сутність – зв'язок"

Процес концептуального проектування БД є ітеративним та заснований на діях, що повторюються. Першим ділом будується базова ER-модель деякої ПО. Під час дослідження моделі, здебільшого, виникнуть додаткові сутності, зв'язки й атрибути. ER-модель буде модифікуватись до тих пір, поки концептуальна модель не буде відображати певну ПО.

Грунтуючись на опитуваннях фахівців розробниками БД визначаються сутності, атрибути, зв'язки в предметній області, що розглядається, досліджуються документи, що застосовуються в роботі на підприємстві.

На прикладі розглянемо побудову ER-моделі ПО «Заклад вищої освіти (ЗВО)».

Визначення задач інформаційної системи

Мета створення БД визначається в наступному:

– забезпечення ЗВО довідковою інформацією за факультетами, за кафедрами, за спеціальностями, за викладачами, за студентами й за дисциплінами, що складають навчальні плани;

– ведення контролю успішності студентів.

Аналіз предметної області

У результаті дослідження й аналізу ПО були визначені сутності, атрибути і первинні ключі, що складають таблицю 5.2.

На основі бізнес-правил, які побудовані з урахуванням організаційної структури та операцій, що виконуються в системі визначаються зв'язки сутностей:

– університет має декілька факультетів;

– факультет має групи студентів, що навчаються за певними спеціальностями;

– групи складають студенти;

– кафедри об’єднуються в факультети;

– на кафедрах працюють викладачі;

– кожна спеціальність складається з ряду дисциплін, що викладаються викладачами з різних кафедр;

– кожна дисципліна спеціальності має підсумковий контроль (іспит чи залік), що складається студентами;

– не кожен викладач читає дисципліни (асистент), не за кожною дисципліною призначено викладача (нова дисципліна).

Таблиця 5.2 - Сутності, атрибути й первинні ключі ПО ЗВО [1]

Сутність	**Атрибути**	**Первинний ключ**
ФАКУЛЬТЕТ	Код факультету Назва Декан	Код факультету

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

Щоб не ускладнювати концептуальну модель БД цілий ряд об'єктів і бізнес-правил ЗВО не розглядаються.

Дослідження ПО виявило, наступне: всі сутності є сильними, а зв'язки між сутностями є неідентифікуючими. Зв'язок між сутностями **Дисципліна** й **Студент** має наступний атрибут: **Оцінка**.

ER-діаграма ЗВО

На основі задач, які були поставлені перед інформаційною системою, і на основі аналізу ПО, побудована ER-діаграма ЗВО (рисунок 5.21).

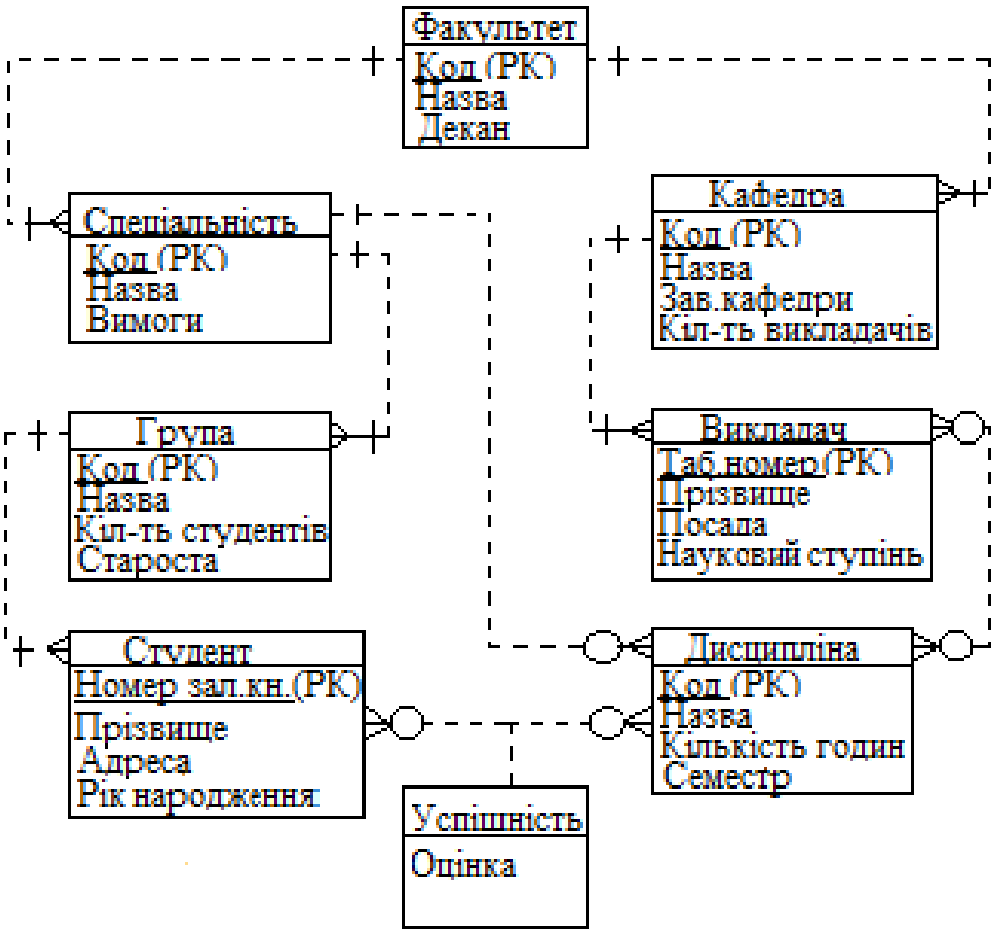


Рисунок 5.21 - ER-діаграма ПО ЗВО [1]

Отриманий концептуальний проект треба перевірити на збитковість і на відповідність транзакціям користувачів.

При *перевірці на збитковість* передбачається перевірка ER- моделі з метою визначення та видалення збиткових даних. Збиткові зв'язки проявляються в тому, якщо між двома сутностями є декілька шляхів і вони дублюються (але це не про зв'язки, що подають різні асоціації).

Підходи до *перевірка моделейі на відповідність транзакціям користувачів*:

– перевіряється, чи відображає модель всю інформацію, яка обов’язкова для кожної транзакції (сутності, зв'язки і атрибути);

– перевіряється маршрут кожної транзакції по ER-діаграмі.

Концептуальний проект відповідає всім необхідним вимогам у тому випадку, якщо він пройшов перевірку моделі на збитковість та на відповідність транзакціям користувачів.

Слід зауважити, що розроблений концептуальний проект не є єдиним проектом, що відповідає поставленій задачі. Існують інші варіанти розробки системи із застосуванням інших зв'язків між сутностями, або із використанням розширеної ER-моделі.

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

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

Контрольні запитання:

1. Що представляє собою сутніть.
2. Чим відрізняється сильна сутність від слабкої?
3. Що представляє собою атрибут?
4. Які бувають атрибути?
5. Що представляє собою поняття ступінь зв'язку?
6. Що таке кардинальне число?
7. Який зв’язок називають ідентифікаційний?
8. Який зв’язок називають неідентифікаційним?
9. Які символьні позначення застосовуються в діаграмах "сутність – зв'язок"?
10. Що означає поняття підтипи сутностей?(з прикладами)
11. Як відображається наслідування на діаграмах "сутність – зв'язок"?

12. Навести приклади зв'язків 1:N для таких різновидів зв'язків: необов'язково-необов'язково, необов'язково-обов'язково, обов'язково-необов'язково, обов'язково- обов'язково.

13. Що означає поняття рекурсивний зв'язок?

14. Навести приклади рекурсивних зв'язків 1:1, 1:N, M:N.

15. Які є переваги і недоліки ER-моделі?

6. Логічне проектування баз даних

Етапи логічного проектування

Логічне проектування для реляційної моделі даних полягає у створенні реляційної схеми, визначенні кількості та структури таблиць, формуванні запитів до БД, визначенні типів звітних документів, розробці алгоритмів обробки інформації, створенні форм для введення й редагування даних в БД і вирішенні інших задач. Таким чином концептуальні моделі завдяки певним правилам трансформуються в логічні моделі даних.

Задачею логічного етапу проектування є відображення об'єктів ПО в об'єкти моделі даних, що використовується, щоб це відображення не суперечило семантиці предметної області й було за можливістю зручним, ефективним, найкращим.

За допомогою *правил нормалізації* можливо впевнитись в структурній узгодженості, логічній цілісності й мінімальній збитковості прийнятої моделі даних. Також модель проходить перевірку на виявлення можливостей виконання транзакцій, які будуть задаватися користувачами. Проектування являє собою циклічний процес. Етапи логічного проектування наведені на рисунку 6.1.

Рисунок 6.1 - Логічне проектування БД поетапно

Етап спрощення концептуальної моделі

Спочатку спрощення концептуальної моделі є попередні перетворення з метою видалення зв'язків, які є несумісними з реляційною моделлю.

Цей етап передбачає виконання операцій вилучення:

- двосторонніх зв'язків M:N;
- складних зв'язків;

– багатозначних атрибутів;

– рекурсивних зв'язків;

– зв'язків з атрибутами.

Вилучення двосторонніх зв'язків "багато до багатьох":

Зв’язок "багато до багатьох"(M:N) замінюється двома зв'язками 1:N з новою проміжною сутністю. Наприклад, якщо Викладач може викладати багато Дисциплін, одну Дисципліну викладає багато Викладачів (рисунок 6.2).

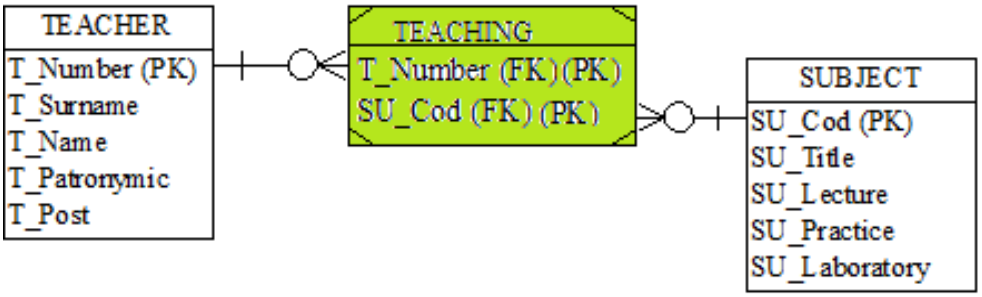


Рисунок 6.2 - Результат перетворення зв'язку M:N на два зв’язки 1:N

Після маніпуляцій отримали нову сутність, що є слабкою та залежною від сутностей TEACHER та SUBJECT. Первинний ключ нової сутності складний (складається з первинних ключів двох сутностей). Кожен атрибут TEACHING окремо є вторинним ключем.

Позбавлення від складних зв'язків:

Для заміни складних зв'язків виконуються наступні дії:

– в модель включають нову сутність;

– складний зв'язок замінюють бінарними зв'язками "один до багатьох" з новою сутністю;

– ступінь складності зв'язку визначає кількість бінарних зв'язків.

Наприклад, в університеті один **Викладач** може викладати багато **Дисциплін**, але одну **Дисципліну** може викладати багато **Викладачів**. **Викладач** проводить **Екзамен з Дисципліни** (рисунок 6.3).

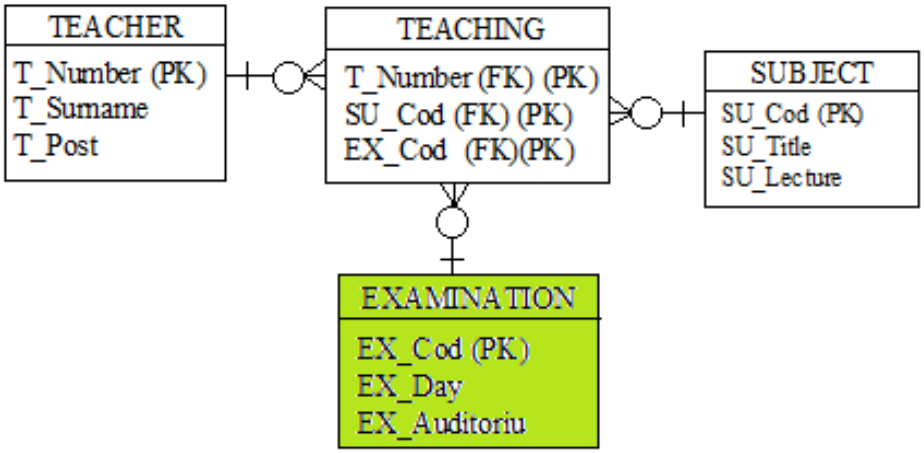


Рисунок 6.3 – Складний зв'язок перетворився на три двосторонні зв'язки й нову сутність **Екзамен**

Вилучення багатозначних атрибутів:

Багатозначні атрибути в концептуальній моделі даних можна виправити шляхом декомпозиції цього атрибуту для визначення деякої сутності.

Наприклад, сучасний **Студент**, може мати декілька телефонних номерів (рисунок 6.4, 6.5).

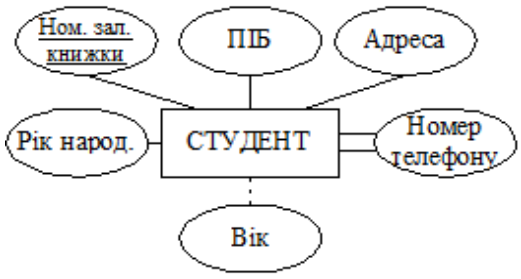


Рисунок 6.4 - Модель з сутністю, що має багатозначний атрибут [1]

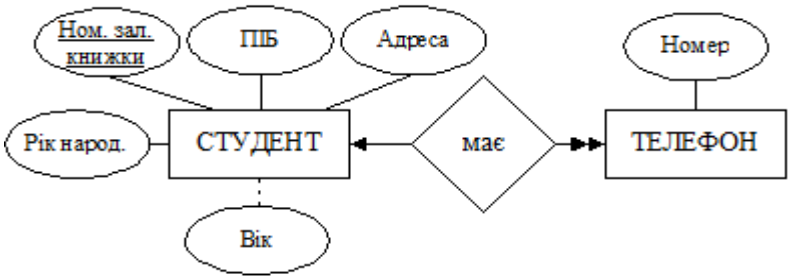


Рисунок 6.5 - Вилучено багатозначний атрибут та додано нову сутність **Телефон** [1]

Вилучення рекурсивних зв'язків:

На етапі спрощення концептуальної моделі рекурсивні зв'язки 1:1 і 1:M (рисунок 6.6) можуть бути перетворені у одне відношення. У випадку, коли є необов'язкова сутність з боку "багато" для зв'язку 1:M для зменшення пустих значень створюється нове відношення. Зв'язок M:N перетворюється на дві сутності (рисунок.6.7).

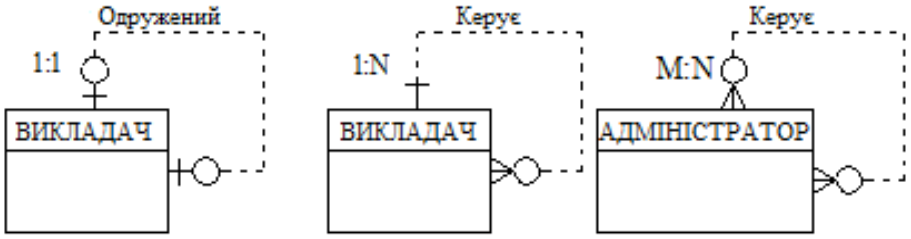


Рисунок 6.6 - Типи рекурсивних зв'язків

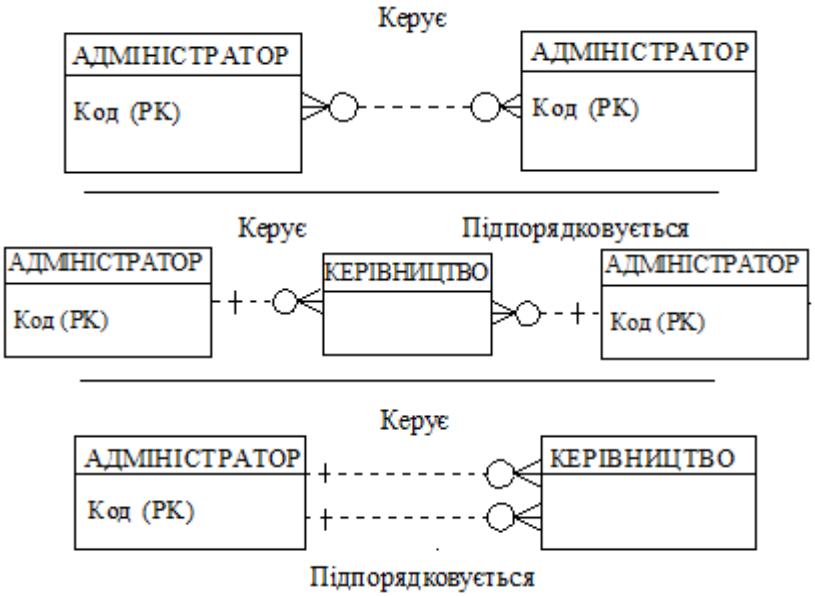


Рисунок 6.7 - Послідовне перетворення рекурсивного зв'язку M:N

Вилучення зв'язків з атрибутами

Для того, щоб позбутися зв'язків з атрибутами необхідно додати у модель нову сутність для відношення M:N з атрибутами зв'язку. Атрибути зв'язку для відношення 1:M передаються в сутність "багато" без створення нової сутності.

Для прикладу розглядаються наступні сутності **Студент** *і* **Дисципліна**. Після перетворень зв'язку з атрибутами можна отримати реляційну схему як на рисунку 6.8.

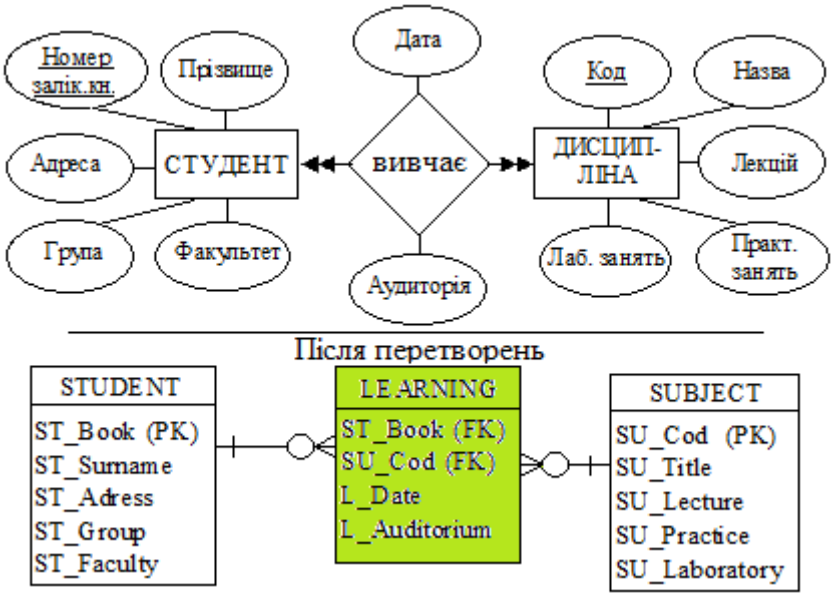


Рисунок 6.8 – Реляційна схема після перетворення з новим відношенням **Learning**

Спрощення концептуальної моделі передбачає також вилучення збиткових зв'язків. Збиткові зв'язки характеризуються тим, що одна і та ж інформація може бути отримана не тільки через них, але і через інші зв'язки.

Після спрощення концептуальна модель має містити тільки наступні елементи:

- об'єкти й атрибути;
- зв'язки типів 1:1 і 1:M;
- суперклас-підклас тип зв'язків.

Перетворення ER-діаграм в реляційні структури

Щоб перетворити ER-модель в реляційну модель даних необхідно слідувати алгоритму певних дій та додержуватись відповідних **правил**:

1. Сутності та атрибути

Сутностям відповідають відношення, а кожен атрибут сутності є атрибутом відповідного відношення.

Первинний ключ сильних сутностей стає PRIMARY KEY (PK) відповідного відношення.

На прикладі розглядається сутність **Студент**, що перетворюється на відношення Student (рисунок 6.9).

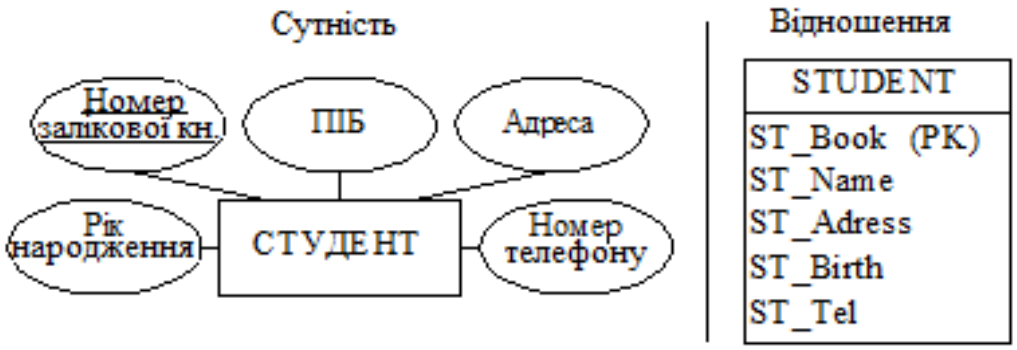


Рисунок 6.9 - Перетворення сутності **Студент** у відношення **Student** [1]

Первинний ключ слабких сутностей частково або повністю залежить від ключа сутності власника (декількох власників). Спочатку треба визначити всі PK сутностей власників, а потім визначається PK слабкої сутності.

Для прикладу розглядаєься трансформація сильної сутності **Студент** і слабкої сутності **Нагорода** (рис. 6.10).

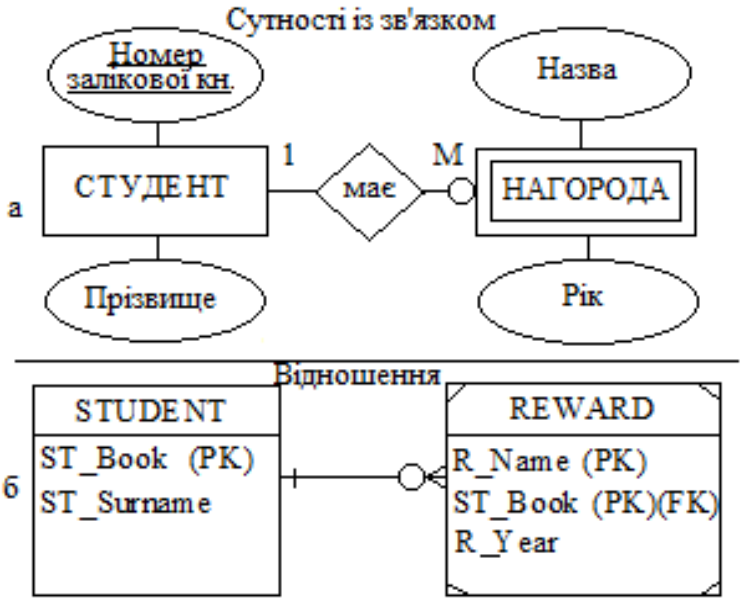


Рисунок 6.10 - Перетворення сутностей (а) у зв’язані відношення **Student** і **Reward** (б)

2. Зв'язки

Після перетворення концептуальної моделі залишаються такі типи зв'язків:

1. "один до одного";

2. "один до багатьох";

3. рекурсивні зв'язки;

4. суперклас – підклас.

В реляційній моделі зв'язки між відношеннями реалізуються шляхом застосування первинних і зовнішніх ключів.

1. "Один до одного"

В концептуальних моделях даних визначають наступні **обмеження ступеня участі сутностей**:

- обов'язкова участь для обох сутностей;
- обов'язкова участь для однієї сутності;
- необов'язкова участь для обох сутностей.

Перетворення до реляційної моделі будуть різнитись в залежності від визначених обмежень.

В прикладі розглядаються можливі варіанти перетворення зв'язку між сутностями **Викладач** і **Дисципліна** (рисунок 6.11).

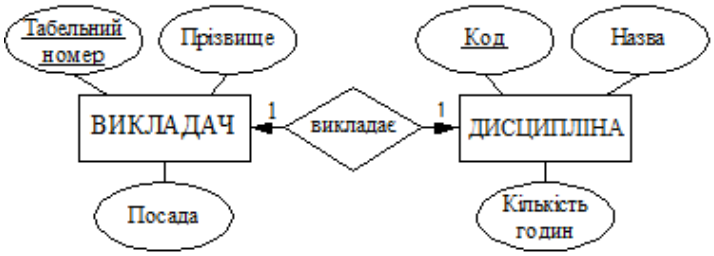


Рисунок 6.11 - Сутності **Викладач** та **Дисципліна** і зв'язок 1:1

Обов'язкова участь для обох сутностей

Якщо вважати, що кожен викладач повинен обов'язково викладати одну дисципліну і кожна дисципліна обов'язково викладається одним викладачем, то реляційна структура може представлятись одним відношенням і мати вигляд один з наступних (рисунок 6.12).

TEACHING
T_Number (PK)
T_Name
T_Post
SU_Cod
SU_Name
SU_Hours

а

TEACHING
SU_Cod (PK)
T_Number
T_Name
T_Post
SU_Name
SU_Hours

б

Рисунок 6.12 - Зв’язок 1:1 і варіанти відношень для перетворення з обов'язковою участю для обох сутностей [1]

Обов'язкова участь для однієї сутності

Якщо розглядати варіант, що кожен викладач обов'язково викладає одну дисципліну, а дисципліна необов'язково закріплена за викладачем. В такому варіанті сутність **Дисципліна**, яка є необов'язковою, є в якості батьківської сутності, а обов'язкова сутність **Викладач** - є дочірньою. На рисунку 6.13 зображено реляційну структуру.

Зовнішній атрибут SU_Cod також може бути ключем для сутності **Викладач**.

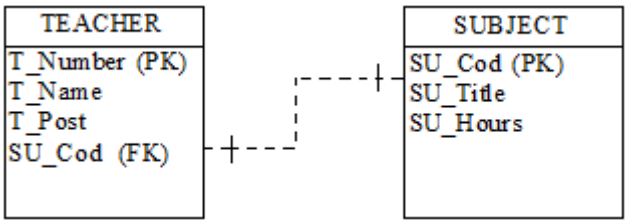


Рисунок 6.13 - Обов'язкова участь сутності **Викладач** і необов'язкова участь сутності Дисципліна у зв’язку 1:1

Необов'язкова участь для обох сутностей

Якщо припустити, що необов'язково кожен викладач повинен викладати дисципліну та необов'язково кожна дисципліна закріплена за викладачем, то можливі три варіанти реляційних структур (рисунок 6.14).

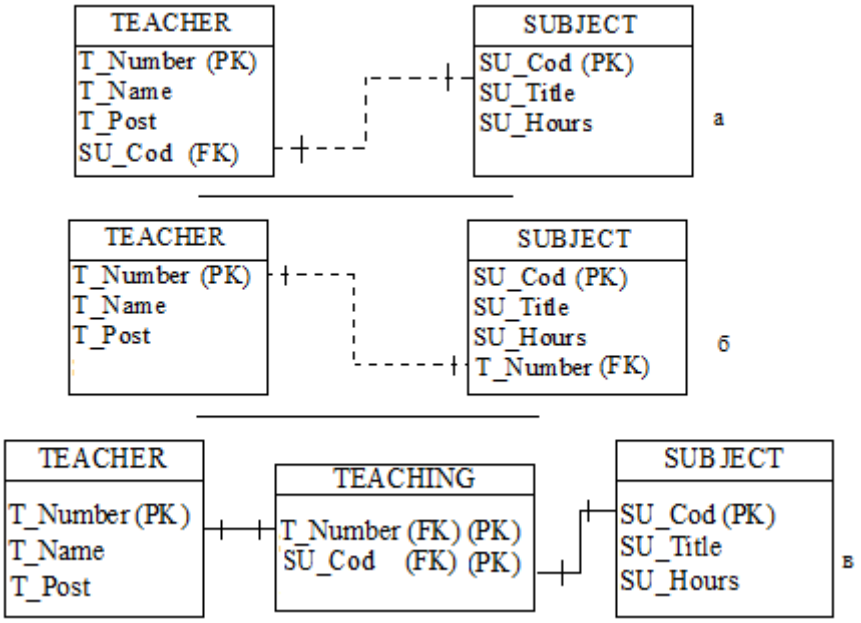


Рисунок 6.14 - Варіанти реляційних схем відношень для перетворення зв'язку 1:1 з необов'язковою участю для обох сутностей

2. "Один до багатьох"

Кожне відношення, що відповідає підлеглій (дочірній) сутності, має набір атрибутів з основної (батьківської) сутності, це - первинний ключ основної сутності. У відношенні, що відповідає дочірній

сутності, цей набір атрибутів перетворюється в зовнішній ключ (FOREIGN KEY, FK).

Атрибути, що відповідають зовнішньому ключу набувають властивості допустимості невизначених значень (тобто NULL) при моделюванні необов'язкового типу зв'язку. У разі обов'язкового типу зв'язку атрибути набувають властивості відсутності невизначених значень (NOT NULL).

Для прикладу розглядаються варіанти при перетворенні зв'язку між сутностями **Викладач** і **Дисципліна** (рисунок 6.15).

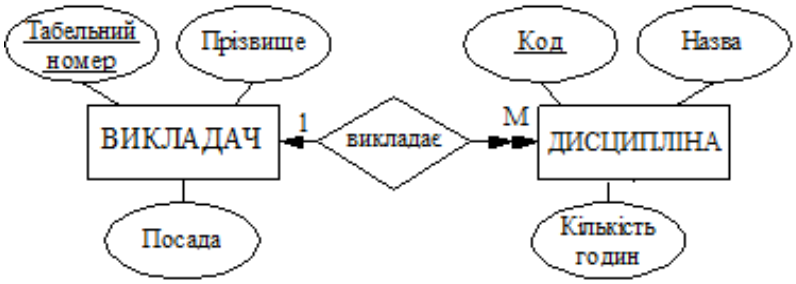


Рисунок 6.15 - Зв'язок 1:M між сутностями **Викладач** і **Дисципліна**

Якщо необов'язкова участь сутності **Викладач** і обов'язкова участь сутності **Дисципліна,** то можна представити зв’язок наступним чином (рисунок 6.16).

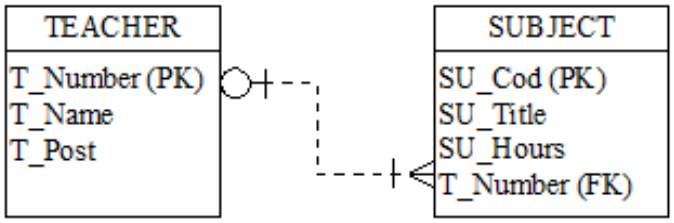


Рисунок 6.16 - Перетворення зв'язку 1:M з необов'язковою участю сутності **Викладач** і обов'язковою участю сутності **Дисципліна**

Варіант необов'язкової участі сутностей **Викладач** і **Дисципліна** розглядається на рисунку 6.17.

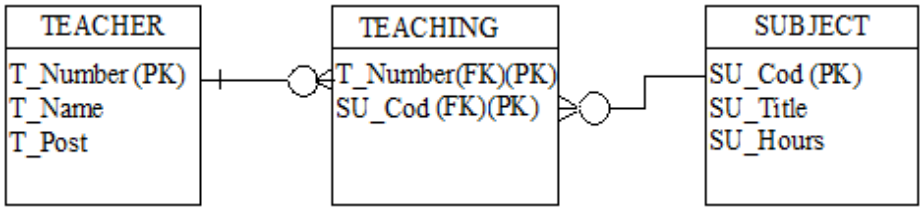


Рисунок 6.17 - Перетворення зв'язку 1:M з необов'язковою участю сутностей **Викладач** і **Дисципліна**

3. "Багато до багатьох"

Кожний зв'язок M:N потребує створення додаткового відношення, яке представляє цей зв'язок та додати в нього всі атрибути, які складають цей зв'язок. В якості зовнішніх ключів у новому відношенні беруться копії атрибутів первинного ключа сутностей, які беруть участь у зв'язку. Ці зовнішні ключі представляють також первинний ключ нового відношення.

Наприклад, розглянемо варіанти перетворення зв'язку між сутностями **Викладач** і **Дисципліна** що на рисунку 6.18.

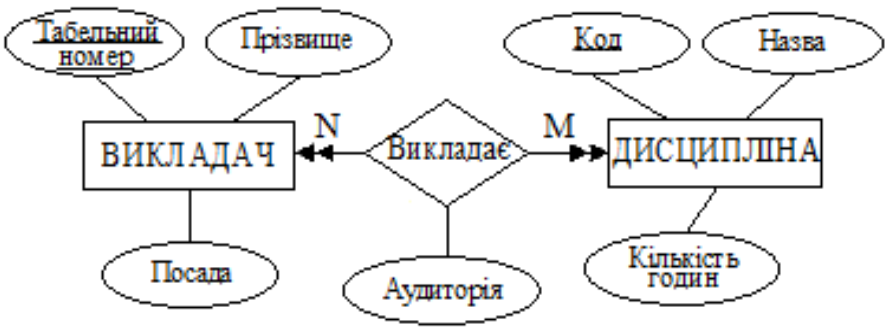


Рисунок 6.18 - Зв'язок N:M між сутностями **Викладач** і **Дисципліна**

В ситуації, що розглядається, існує єдина схема перетворення, яка зображена на рисунку 6.19.

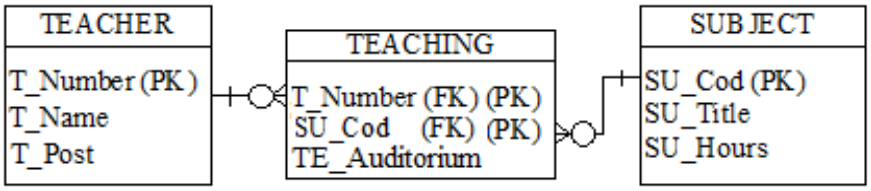


Рисунок 6.19 - Перетворення зв'язку N:M у реляційну схему

Інші види зв'язків

4. Рекурсивні зв’язки 1:1 виконуються згідно правил визначених раніше для зв'язку між двома сутностями 1:1. Для рекурсивного зв'язку 1:1 з обов'язковою участю двох сторін, реляційна схема представляється у вигляді одного відношення з двома копіями первинного ключа (рисунок 6.12). Одна копія відповідає зовнішньому ключу. Для рекурсивного зв'язку 1:1 з обов'язковою участю тільки однієї сторони створюється або одне відношення, або нове відношення, яке відображає цей зв'язок (рисунок 6.13). Для рекурсивного зв'язку 1:1 з необов'язковою участю обох сторін створюється нове відношення (рисунок 6.14) [1].

Для **складних типів зв'язків** вводиться нове відношення, яке відображає цей зв'язок і складається з атрибутів, які входять в цей зв'язок. Для використання як зовнішні ключі, копії атрибутів первинного ключа сутностей, що зв'язуються, передаються у нове відношення. Ці зовнішні ключі утворюють також первинний ключ нового відношення (рисунок 6.3).

Для багатозначного атрибуту створюється нове відношення, яке відповідає багатозначному атрибуту, і в це нове відношення передається первинний ключ сутності для використання в якості зовнішнього ключа (рисунок 6.4, 6.5).

5. Зв'язки "суперклас – підклас"

Щоб виконати трансформацію зв'язку типу суперклас – підклас у реляційну модель потрібно звернути увагу на обмеження ступеня участі у зв'язку (Optional, Mandatory) та обмеження неперетинання (Or, And). Є можливим виконати чотири сполучення, перетворення яких дає чотири реляційні схеми. Має вплив на схему також те, чи беруть участь підкласи в різних зв'язках, число сутностей у зв'язку тощо. Діапазон можливих варіантів рішення є достатньо великим і конкретна

схема вибирається в кожному конкретному випадку з урахуванням множини факторів.

Для прикладу розглядається суперклас **Викладач**, який має атрибути **Табельний номер, Прізвище, Посада**. Об'єкти **Професор**, **Доцент**, **Асистент** (рисунок 6.20) виступають підкласами суперкласу. Кожен екземпляр підкласу може також бути й екземпляром суперкласу, тобто у суперкласа можуть бути свої екземпляри (Optional). А кожен викладач обов'язково відноситься до тільки одного підкласу (Or). Перетворення цієї діаграми в реляційну схему відношень розглядається на рисунку 6.21. Завдяки ключу суперкласу (Табельний номер) виконується зв'язок між відношеннями.

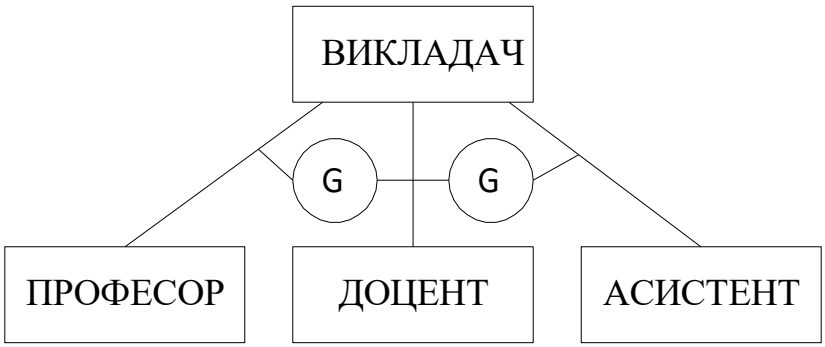


Рисунок 6.20 - Зв'язок суперклас – підклас з обмеженнями Or і Optional на діаграмі

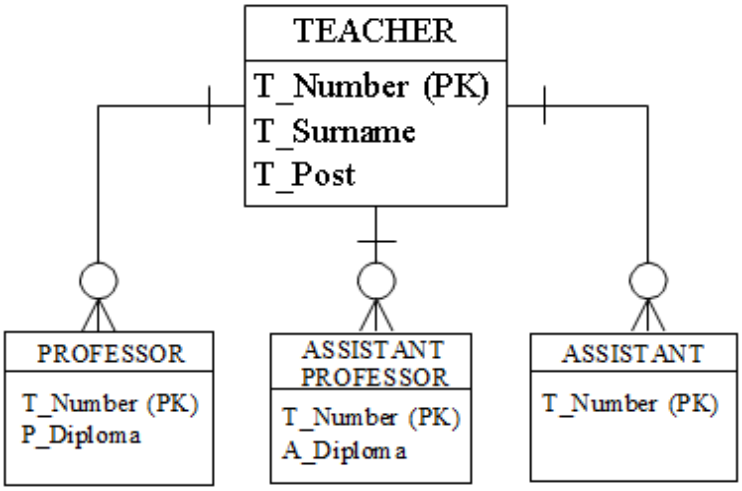


Рисунок 6.21 – Перетворена діаграма із зв'язком суперклас – підклас на реляційну схему

Для прикладу розглядається суперклас **Студент**, який має атрибути **Номер залікової книжки, Прізвище, Група**. Підкласами суперкласу виступають об'єкти **Очна, Заочна і Дистанційна форми навчання** (рисунок 6.22). Кожен екземпляр підкласу є одночасно і екземпляром суперкласу (Mandatory). Кожен студент може навчатись на будь-якій формі навчання одночасно, тобто належати до кількох підкласів (And). Ця діаграма модифікується в наступну реляційну схему (рисунок 6.23).

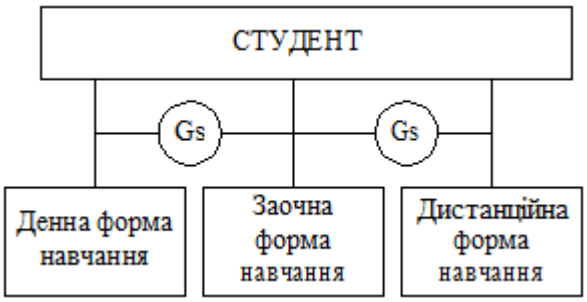


Рисунок 6.22 - Зв'язок суперклас-підклас з обмеженнями Mandatory та And

STUDENT
ST_Book (PK)
ST_Surname
ST_Group
ST_Confront
ST_Correspond
ST_Distance

Рисунок 6.23 - Реляційна схема, що відповідає попередньому зв'язку суперклас–підклас

Нормалізація та денормалізація

Набір відношень логічної моделі БД, що отримали на попередніх етапах, повинен бути перевірений на коректність об'єднання атрибутів у кожному відношенні. Шляхом застосування процедури послідовної

нормалізації до кожного відношення. Допомога нормалізації в тому, що модель, що отримується не буде містити протиріч, а збитковість буде мінімальною. В результаті нормалізації атрибути будуть згруповані відповідно до існуючих між ними логічних зв'язків. Якщо виявились відношення, які не відповідають вимогам нормалізації, необхідно повернутися на попередні етапи проектування й перебудувати помилково створені елементи моделі, щоб забезпечити коретність логічної моделі.

Нормальна форма - властивість зв'язку в реляційній моделі даних, що характеризує його з точки зору надмірності, потенційно призводить до логічно помилкових результатів вибірки або зміни даних. Нормальна форма визначається як сукупність вимог, яким має задовільняти відношення [3].

Під **нормалізацією** розуміють механізм трансформації відносин БД до того виду, що відповідає нормальним формам. Вона призначається для приведення структури БД до стану мінімальної логічної надмірності, та не має на меті збільшення чи зменшення продуктивності роботи або фізичного обсягу бази даних. Нормалізація виконується з метою зменшення потенційної суперечливості збереженої в БД інформації. Як зазначає К. Дейт, загальне призначення процесу нормалізації полягає в наступному:

-виключення деяких типів надмірності;

-усунення деяких аномалій оновлення;

-розробка проекту бази даних, який є досить «якісним» поданням реального світу, інтуїтивно зрозумілий і може служити хорошою основою для подальшого розширення;

-спрощення процедури застосування необхідних обмежень цілісності.

Позбавлення надлишковостей відбувається,зазвичай, за допомогою декомпозиції відносин таким чином, щоб в кожному відношенні

зберігалися тільки первинні факти, що не виводяться з інших збережених фактів.

Процес проектування БД з використанням методу нормальних форм є ітераційним і полягає в послідовному переведені відношень з першої нормальної форми до бінормальних форм більш високого порядку за певними правилами. Кожна наступна нормальна форма обмежує певний тип функціональної залежності, усуває відповідні аномалії при виконанні операцій над відношеннями БД і зберігає властивості попередніх нормальних форм [2].

Надлишковість даних призводить до збільшення об'єму пам'яті й уповільнення роботи БД. Тому є небажаною обставиною для БД і є наслідком, передусім, дублювання даних. Розрізняють *збиткове* та *незбиткове* дублювання даних. Зовсім лишатись надлишковості немає необхідності, бо в такому випадку буде неможливо підтримувати базу даних як щось ціле. Можна тільки зробити мінімум надлишковості та залишити необхідне дублювання даних.

Дублювання даних викликає проблеми під час виконання операцій (редагування, додавання інформації або вилучення) з базою даних.

Ситуація, що виникає в БД, що призводить до протиріч та/або суттєво ускладнює обробку даних називається **аномаліями.**

Зустрічаються аномалії додавання, вилучення та модифікації.

Аномалія додавання може виникнути при додаванні інформації, наприклад, нового студента, тобто необхідно вводити інформацію, яка вже існує в БД: назва факультету, прізвище декана. Крім того неможливо створити нову групу поки не введено студентів, що в ній навчаються.

Аномалія модифікації може статись при спробі змінити значення, наприклад, прізвище декана; в такому випадку необхідно переглянути всі кортежі. Якщо осяг БД великий, то це затребує значного часу, крім

того можливі помилки при модифікації прізвища, які можуть порушити цілісність БД.

Аномалія вилучення виникає при спробі вилучити дані про студента, що в групі доданий один, наприклад. В такому разі зникне інформація про групу.

Застосування декомпозиції відношень допомагає позбутися вищезгаданих аномалій.

Існує наступна послідовність нормальних форм [2][3]:

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

Перша нормальна форма (1НФ)

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

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

Про повну й неповну функціональні залежності

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

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

Друга нормальна форма (2НФ)

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

Третя нормальна форма (3НФ)

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

Нормальна форма Бойса-Кодда (BCNF)

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

Четверта нормальна форма (4НФ)

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

П’ята нормальна форма (5НФ)

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

Доменно-ключева нормальна форма (ДКНФ)

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

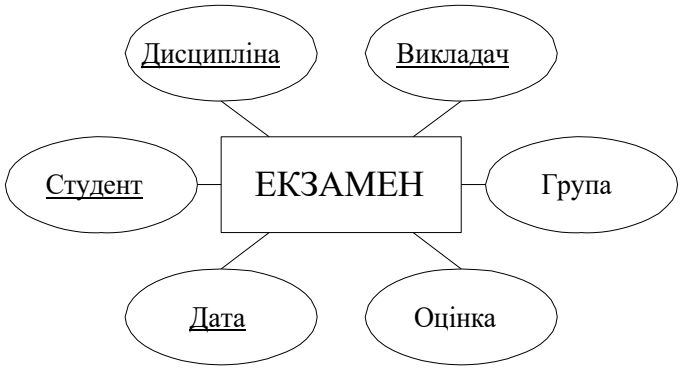
Шоста нормальна форма (6НФ)

Змінна відношення знаходиться в шостий нормальній формі тільки тоді, якщо вона задовільняє всім нетривіальним залежностям з'єднання. З визначення випливає, що змінна знаходиться в 6НФ тільки тоді, якщо вона не приводима, тобто не може бути піддана подальшій декомпозиції без втрат. Кожна змінна відношення, яка знаходиться в 6НФ, також знаходиться й в 5НФ.

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

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

Для прикладу розглянемо відношення, отримане в результаті проектування, яке видно на рисунку 6.24.



EXAMINATION
EX_Subject (PK)
EX_Student (PK)
EX_Teacher (PK)
EX_Date (PK)
EX_Group
EX_Mark

а *б*

Рисунок 6.24 - Сутність **Екзамен** (*а*) перетворюється у відношення **Examination (б)**

Досліджуючи дане відношення було виявлено такі функціональні залежності:

Дисципліна, *Викладач*, *Студент*, *Дата* → *Оцінка*

Студент → *Група*

Через те, що було виявлено аномалії необхідно продовжити процес нормалізації. Виконуємо декомпозицію вихідного відношення і отримуємо схему, що подається на рисунку 6.25.

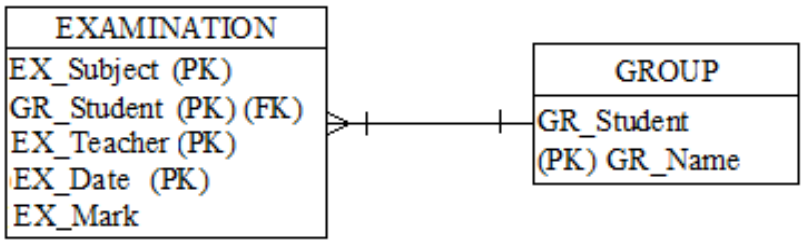


Рисунок 6.25 - Реляційна схема, яка відповідає сутності **Екзамен**

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

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

– при об'єднанні таблиць зі зв'язками "один до одного";

– при дублюванні неключових атрибутів у зв'язках "один до багатьох" (для зменшення кількості з'єднань);

– при дублюванні атрибутів зовнішнього ключа у зв'язках "один до багатьох" (для зменшення кількості з'єднань);

– при дублюванні атрибутів "багато до багато" (для зменшення кількості з'єднань);

– при створенні таблиць з даних, які містяться в інших таблицях;

– при введенні груп полів, які повторюються.

Перед виконанням денормалізації слід визначитись з користю її й негативними наслідками, та якщо переваг більше, то з обережністю

застосовувати. До переваг відносять у тому числі збільшення пропускної спроможності, рівня задоволеності клієнтів та продуктивності. Слід пам’ятати, про негативні наслідки цього процесу:

- поява аномалій в БД;
- знижується гнучкість системи;
- може зменшитись час на відповіді до БД, проте уповільнюються операції оновлення даних;
- ускладнення фізичної реалізації системи.

Перевірка відповідності відношень вимогам транзакцій користувачів

Транзакція (*transaction*) – це група послідовних операцій з БД, яка є логічною одиницею роботи з даними. Транзакція може бути або виконана цілком й успішно, дотримуючись цілісності даних та незалежно від інших транзакцій, що йдуть паралельно, або відмінена зовсім, тоді вона не має ніякого ефекту. В процесі роботи створюється історія транзакцій. Транзакції з БД використовують для:

1. Правильного відновлення роботи БД у випадку збоїв, системних відмов, коли виконання операцій зупиняється і більшість операцій над БД залишаються незавершеними з нез'ясованим статусом.
2. Забезпечення роздільного доступу для процесів, що одночасно звертаються до БД. При відсутності ізоляції операцій результати, отримані процесами, можуть бути помилковими.

Перевірка відбувається в зазначенні безпосередньо на ER- діаграмі всіх шляхів, які потрібні для виконання кожної з транзакцій. У випадку, якщо вдається виконати всі транзакції, то перевірка закінчується. Інакше необхідно повернутися до попередніх етапів і перевірити, а якщо потрібно, внести зміни в ті фрагменти моделі, які не задовільняють необхідну роботу транзакцій.

Під час перевірки можуть бути виявлені області, що не беруть безпосередньої участі у роботі транзакцій. В таких випадках їх вилучають з моделі.

Перевірка на підтримку цілісності

Накладені обмеження цілісності в БД запобігають появі суперечливих даних. Це можна вирішити ще на стадії проектування БД, а саме:

– визначити наявність обов'язкових і необов'язкових значень даних для атрибутів (NULL, NOT NULL);

– визначити наявність обмежень для доменів атрибутів (визначення області значень або діапазону значень);

– застосування правил цілісності сутностей (обов'язкова наявність Primary Key в кожному відношенні);

– правила посилкової цілісності (зв'язування таблиць за допомогою Foreign Key);

- встановити обмеження предметної області (бізнесправила), які реалізуються як засобами БД, так і на рівні застосувань.

У таблиці 6.1 наведені правила зовнішнього ключа для відношення "один до багато" для сильної сутності.

Таблиця 6.1 - Підтримка посилкової цілісності для сильної сутності

Тип зв'язку	Вимоги до зовнішнього ключа
Обов'язкова наявність значень відповідних екземплярів у батьківській і залежній таблицях	NOT NULL ON DELETE RESTRICT ON UPDATE CASCADE
Необов'язкова наявність значень відповідних екземплярів у батьківській і залежній таблицях	NULL ALLOWED ON DELETE SET NULL ON UPDATE CASCADE
Обов'язкова наявність значень відповідних екземплярів у залежній таблиці і **необов'язкова** наявність значень в батьківській таблиці	NULL ALLOWED ON DELETE SET NULL ON DELETE RESTRICT ON UPDATE CASCADE
Обов'язкова наявність значень відповідних екземплярів у батьківській таблиці та **необов'язкова** наявність значень в залежній таблиці	NOT NULL ON DELETE RESTRICT ON UPDATE CASCADE

Якщо розглядать слабкі сутності, то використовуються однакові правила, за винятком обмежень на зовнішній ключ: NOT NULL, ON DELETE CASCADE та ON UPDATE CASCADE.

Приклад створення логічної моделі БД

На розглянутому раніше прикладі розроблено концептуальний проект БД для ПО ЗВО. На ER-діаграмі видно всі бізнес правила, які визначають сутності, атрибути, зв'язки й т.ін.

Наступний етап проектування БД - це створення логічної моделі БД на основі існуючої ER-моделі (рисунок 5.21). Логічну модель БД

будують застосовуючи правила перетворення ER-діаграми в логічну модель (рисунок 6.26).

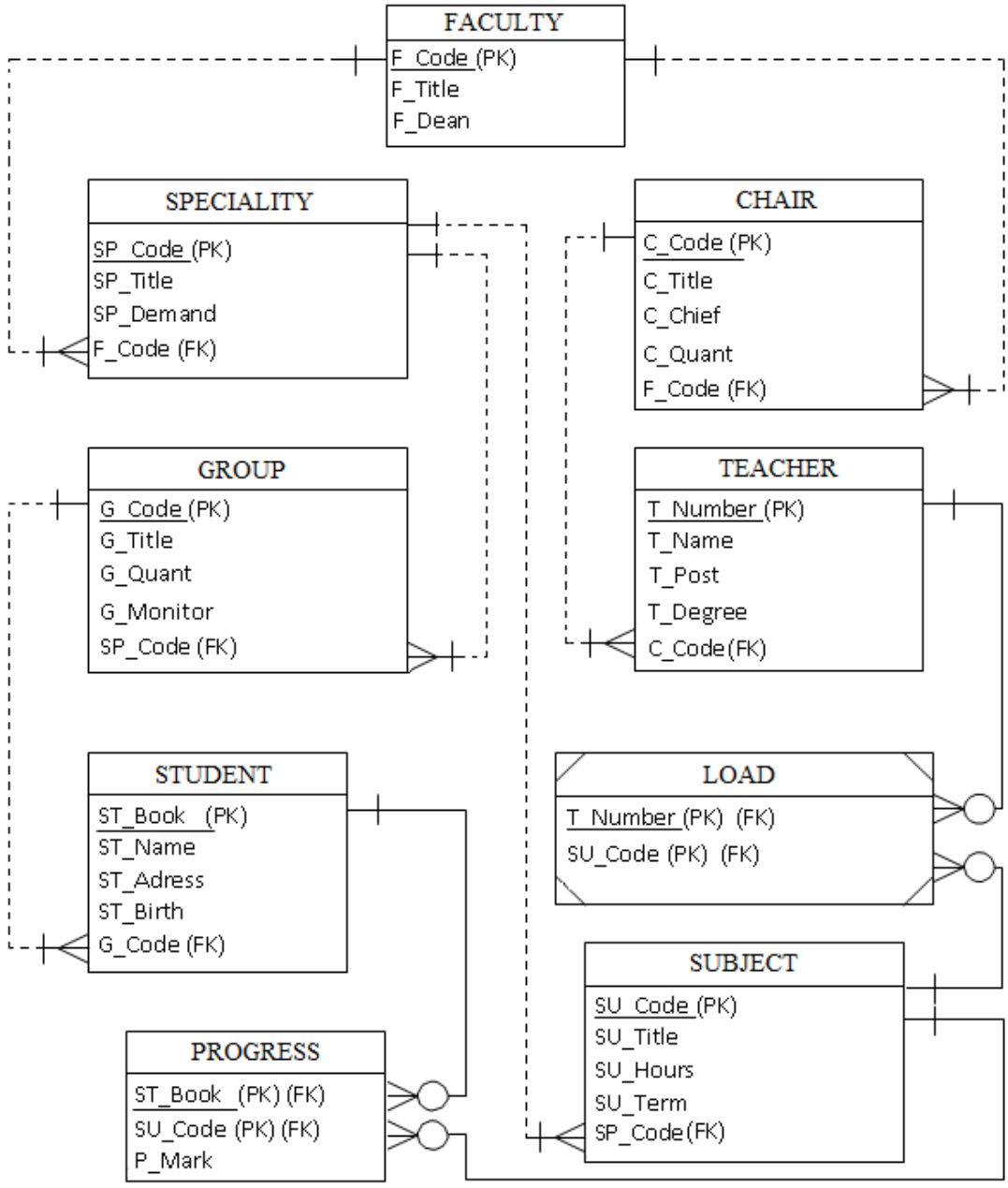


Рисунок 6.26 - Логічна модель БД для ПО ЗВО

Отримана логічна модель даних, реляційна схема аналізується на коректність об'єднання атрибутів в одному відношенні. Перевірка здійснюється із застосуванням послідовної нормалізації до кожного з відношень. Якщо схема бази даних не знаходиться хоча б в 3НФ або в БКНФ, то необхідно повернутися на попередні етапи проектування і змінити помилково створені частини моделі. В даному випадку

перевірка логічної моделі БД ЗВО показує, що реляційна схема знаходиться в 4НФ, а це означає, що корегувати модель не потрібно.

Після перевірки логічної моделі за допомогою правил нормалізації система аналізується на предмет виконання транзакцій користувачів, які задаються на початкових етапах проектування. У разі неможливості виконання певних транзакцій необхідне корегування моделі БД (рисунок 6.1).

Подальша перевірка моделі вимагає перевірки підтримки правил цілісності даних. Грунтуючись на розглянутих матеріалах до прикладу можна тільки визначити, що правила посилкової цілісності підтримуються. Всі інші перевірки, включаючи й перевірку транзакцій користувачів, вимагають більш детально опрацьованого проекту БД.

Контрольні запитання:

1. Що називається логічним проектуванням?
2. Яка інформація є вихідною для логічного проектування?
3. Перелічити етапи логічного проектування.
4. Які зв'язки ER-діаграм не підтримуються в реляційній схемі?
5. Як відображаються сильні й слабкі сутності та їх атрибути на реляційній схемі?
6. Як відображається необов'язковість зв'язку між сутностями на реляційній схемі?
7. Як відображається зв'язок "багато до багатьох" на реляційній схемі?
8. Як відображаються складні зв'язки і зв'язки з атрибутами на реляційній схемі?
9. Як відображаються рекурсивні зв'язки в реляційній схемі?
10. Яким чином подається зв'язок "один до одного" залежно від рівня участі сутностей на реляційній схемі?
11. Як відображається зв'язок "один до багатьох" залежно від

рівня участі сутностей на реляційній схемі?

12. Як відображається зв'язок суперклас-підклас залежно від ступеня участі сутностей і обмеження неперетинання на реляційній схемі?

13. Навіщо потрібно виконувати перевірку реляційної схеми на відповідність правилам нормалізації?

14. Що означає термін функціональна залежність?

15. У чому полягає збиткове і незбиткове дублювання даних?

16. Що таке аномалії додавання, оновлення, вилучення?

17. Дати визначення 2НФ.

18. Дати визначення 3НФ.

19. Дати визначення НФБК.

20. Що означає термін багатозначна залежність?

21. Дати визначення 4НФ.

22. Дати визначення 5НФ.

23. Яке місце займає нормалізація в процесі проектування бази даних?

24. Що означає денормалізація БД і які її переваги і недоліки?

25. Як виконується перевірка реляційної схеми на відповідність вимогам транзакцій користувачів?

26. Як перевіряється цілісність реляційної бази даних?

7. Фізична організація баз даних

Організація зберігання інформації

В БД фізична організація даних відповідає за їх зберігання, керування, форми подання й структури даних.

Фізичне проектування - це процес визначення характеристик сховища даних та доступу до них в БД. Властивості сховища даних залежать від пристроїв зберігання, засобів доступу до даних, що підтримуються системою та від СКБД. Етап фізичного проектування має на увазі визначення розташування даних на пристроях зберігання та загальної продуктивності системи.

В реляційних БД складні фізичні процеси організації даних приховані від користувача, але вони важливі для продуктивності роботи з БД.

Є декілька етапів виконання процесу пошуку й подання даних користувачу (рисунок 7.1).

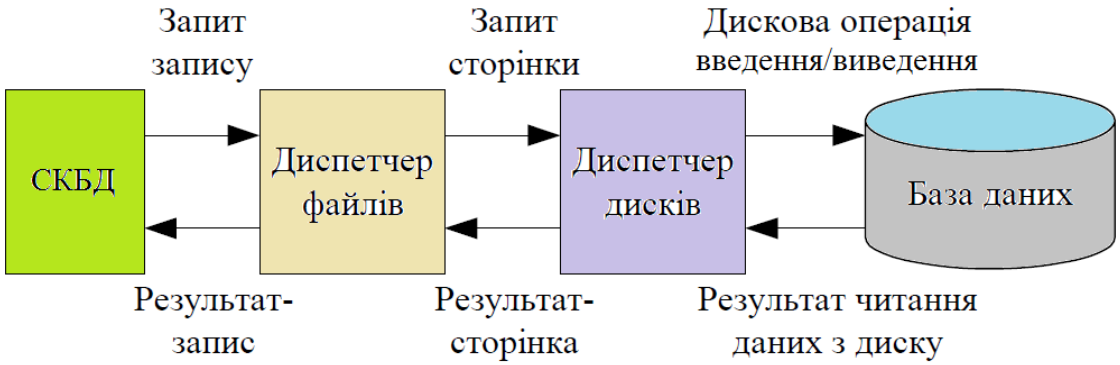


Рисунок 7.1 - Організація доступу до даних

Починається робота з визначення необхідного запису, для знаходження якого викликається диспетчер файлів. Диспетчер файлів визначає сторінку з записом та викликає диспетчер дисків для її отримання. Диспетчер дисків визначає фізичне розташування потрібної сторінки на диску та передає її диспетчеру файлів, а той – СКБД. Головними одиницями операцій обміну системи СКБД - БД є сторінки

даних. БД виглядає як набір записів з точки зору СКБД, та як набір сторінок - з точки зору диспетчера файлів. Кожна сторінка пам'яті має унікальний ідентифікатор. Кожен запис зберігається повністю на одній сторінці.

Для зберігання даних існує технологія **кластеризації** – фізично близьке розташування записів у просторі пам'яті середовища зберігання баз даних.

На продуктивність роботи системи впливають характеристики: розмір сторінок пам’яті, кількість буферів пам’яті, дії алгоритмів вибору буферів пам’яті для розташування сторінок. Простір чи зовнішньої , чи віртуальної пам’яті в БД поділяється на сторінки. Для розташування сторінок в оперативній пам’яті використовуються буфери (спеціальні області). Зміст сторінок оновлюється в буферах і повертається до зовнішньої пам’яті знову.

Призначені для зберігання даних елементи системи баз даних, різняться за ємкістю та швидкістю доступу до даних. Порядок організації видів пам'яті наведено на рисунку 7.2.

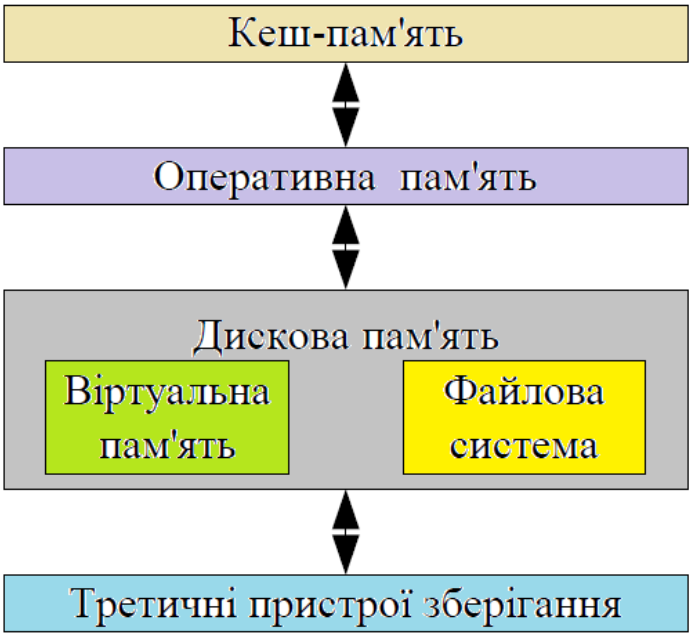


Рисунок 7.2 – Пристрої пам'яті БД в ієрархії

Кеш-пам'ять - найшвидша пам'ять відносно невеликого розміру, яка зберігає копії останніх даних, до яких виконувався доступ. Вона реалізується апаратними або програмно-апаратними засобами.

Віртуальна пам'ять - це вид пам'яті, що моделюється за допомогою апаратних засобів і ПЗ (в різних видах пам’яті). Вона дозволяє оперувати значно більшими об'ємами даних чим дійсний простір, що виділяється в оперативній пам'яті, таким чином імітуючи знаходження в оперативній пам'яті.

Пам'ять третична – це пристрій, що запам’ятовує, з великим об'ємом та низькою вартістю [1].

Зберігання даних організовано за певним порядком і правилами:

– атрибути відтворюються у байтовій послідовності постійної або змінної довжини, що називають полями;

– поля складають набори даних, що називають записом;

– збереження записів відбувається у фізичних блоках (сторінках);

– набори записів, що складають відношення, зберігаються у файлах.

Засоби фізичної організації файлів даних розрізняють за наступними: послідовний, прямий та індексно-послідовний.

Файл, при роботі з яким можливо до записів послідовний доступ у порядку фізичного розміщення в пам'яті називають **послідовним файлом**. Послідовна організація ефективна, коли застосування при кожному зверненні до БД обробляє значну кількість записів.

Файл прямого доступу – це файл, в якому можливий прямий доступ до його записів за їх безпосередньою або посередньою адресою у середовищі зберігання, або за заданим ключем за допомогою будь-якого методу відображення ключа в адресу. Якщо для більшості

застосувань потрібний прямий доступ до записів, то застосовується пряма організація.

Індексно-послідовний файл – це файл, до записів якого є можливість прямого доступу за допомогою створеного для нього індексу за заданим ключем, а також послідовний доступ згідно з їх впорядкуванням за цим ключем індексування. Якщо багато застосувань потребують послідовної обробки, але і багато застосувань потребують прямого доступу, то застосовують індексно-послідовну організацію, як більш ефективну.

Про індексацію

Індекс – це (з погляду користувача) перелік стовпців таблиці, за значеннями яких записи логічно впорядковуються. (З погляду СКБД) – механізм, що дає змогу значно підвищити швидкість доступу до записів та забезпечує ефективну перевірку унікальності значень індексованих полів [4].

Отже **індекс** - це структура даних, яка допомагає СКБД швидше знайти окремі записи в файлі та скоротити час виконання запитів користувачів.

Основне призначення індексів полягає в забезпеченні ефективного прямого доступу до записів таблиці за ключем. Файл, що містить індексні записи називається **індексним файлом**.

Залежно від організації індексної й основної області розрізняють два наступні типи файлів: зі щільним індексом і з розрідженим індексом. Індекс **щільний** (Dense index) містить індексні записи для всіх значень ключа пошуку в даному файлі. Індекс **розріджений** (Sparse index) вміщує індексні записи тільки для деяких значень ключа пошуку в файлі.

Основна область файлів зі щільним індексом містить послідовність записів однакової довжини, які розташовані у довільному порядку, а

структура індексного запису має таку форму: (значення ключа, номер запису).

Файл зі щільним індексом має таке правило: для кожного запису в основній області існує один запис з індексної області, а всі записи в індексній області впорядковуються за значенням ключа.

Індексно-прямі файли – це індексні файли зі щільним індексом.

Файли з розрідженим індексом на відміну від щільних так побудовані: основна область вміщує послідовність записів однакової довжини, що розташовані у впорядкованому порядку, а структура індексного запису має форму: (значення ключа першого запису блока, номер блока з цим записом).

Розріджений індекс створюєьться для впорядкованих файлів. **Індексно-послідовні** файли - це індексні файли з розрідженим індексом.

Доцільно створювати індекси за стовпцем або по групі стовпців у таких випадках:

– часто здійснюється пошук у БД за атрибутами, які перелічені в умові WHERE оператора SELECT;

– часто здійснюється об'єднання таблиць за певними атрибутами;

– часто здійснюється сортування таблиць (використання ORDER BY в операторі SELECT).

Не доцільно створювати індекси за стовпцем або групою стовпців у наступних випадках:

– зрідка використовуються для пошуку;

– містять значення змінного характеру, що призводить до частого оновлення індексу та уповільнення роботи;

– містять замалу кількість значень.

В БД визначають два різновиди індексів:

– індекси, що використовуються запитами для доступу до даних;

– індекси, що забезпечують посилкову цілісність між таблицями.

Значна частина БД підтримує B-дерева, окрім того деякі з них працюють із хеш-таблицями. Деякі системи представляють багатомірні структури даних, такі, як варіанти R-дерев і квадрадерев (*kd*-дерева). Деякі системи підтримують також бітові вектори та багататабличні індекси з'єднання. Системи, які розміщуються в оперативній пам'яті, окрім того підтримують структури даних, що мають невеликі коефіцієнти розгалуження, такі як бінарні дерева й Т-дерева.

Про хешування

Хешування – це технологія прямого доступу до записів БД, що дозволяє швидко знаходити та отримувати доступ до даних за ключем.

Хеш-функції застосовують в різних областях, таких як бази даних, криптографія, безпека мережі та програмування. Суть методу хешування полягає в тому, що за значеннями ключа *k*, або його характеристик, вираховується хеш-фукція *h*(*k*), значення, що отримали і є адреса початку пошуку. Серед недоліків хеш-функцій можна надати, що вони не гарантують в результаті унікальної адреси, з причини, що кількість можливих значень поля хешування може бути значно ширшим за кількість адрес, що доступні для запису. Зустрічається наступні обставини, при яких різним значенням ключів відповідає одне значення хеш-функції. Це називається **колізією**, а значення ключів, що мають однакову хеш-функцію називають **синонімами**. Для боротьби з колізіями застосовують спеціально призначені методи: послідовного перебору, багатократне хешування, введення області переповнення. На рисунку 7.3 надається вид хеш-структури.

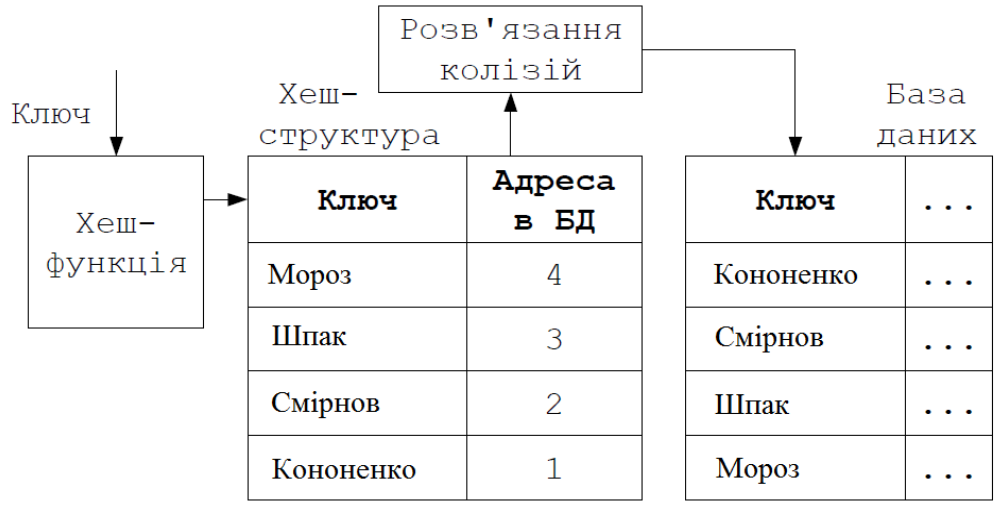


Рисунок 7.3 - Приклад організації хеш-структури

Хеш-структури – це такі структури, що можуть забезпечити відповідь на запит до БД за одне звернення до диску за умови відсутності синонімів. Хеш-структури мають не високу ефективність для запитів на визначення діапазонів, знаходження екстремумів тощо.

Звичайні хеш-функції мають широкий спектр варіантів використання, зокрема пошук у БД, аналіз великих файлів та управління даними [7]. З іншого боку, криптографічні хеш-функції широко використовуються в програмах забезпечення інформаційної безпеки, таких як аутентифікація повідомлень та зняття цифрових відбитків пальців. Щодо Bitcoin, криптографічні хеш-функції є невід'ємною частиною процессу майнінгу, та беруть участь у генерації нових адрес та ключів.

Про B-дерева

У багатьох системах керування БД застосовується для збереження індексів структура даних, яку називають деревом. Можна зустріти серед дерев більш поширені бінарні дерева, B-дерева, а також їх різновиди (B^+-дерева, B^*-дерева тощо). B-дерево – це дерево, що є збалансованим, впорядкованим за значенням ключа. Найбільш розповсюдженими серед B-дерев вважають B^+-дерева.

Про інвертовані файли

Інвертовані списки застосовуються для забезпечення прискореного доступу за вторинними ключами. **Інвертований список** - це дворівнева індексна структура, де на першому рівні знаходиться файл або частина файлу, в якій впорядковано розміщені значення вторинних ключів. Де кожний запис із вторинним ключем містить покажчик на номер першого блоку в ланцюжку блоків, що містять номери записів з даним значенням ключа вторинного. На другому рівні знаходиться ланцюжок блоків, що мають номери записів, які вміщують однакові значення вторинного ключа. До того ж блоки другого рівня впорядковані за значенням ключа вториного. На третьому рівні знаходиться цей файл даних. Послідовність пошуку даних:

– значення вторинного ключа визначають на першому рівні;

– за покажчиком, що знайшли, читається блок другого рівня, (містить номери всіх записів із заданим значенням вторинного ключа);

– зміст всіх записів із заданим значенням вторинного ключа завантажується у робочу область.

Контрольні запитання:

1. За що відповідає фізична організація даних?

2. Що представляє собою індекс?

3. Який файл називають індексним?

4. Чим відрізняється послідовна, індексно-послідовна й пряма організація файлів?

5. Чому розподіл записів по блоках впливає на швидкість роботи системи?

6. В чому різниця між організацією пошукових структур за первинним ключем і за вторинним ключем?

7. Що представляє собою сторінкова організація даних в СКБД?

8. Що собою являє технологія хешування та яке її застосування в БД?

9. У яких випадках необхідно застосовувати індекси, а у яких краще не використовувати?

8. Засоби для автоматизації проектування баз даних

CASE-технології

CASE-засоби (Computer Aided Software Engineering) - це засоби спеціального класу, що покликані вирішувати проблеми автоматизації проектування інформаційних систем. CASE-технологія являє собою сукупність методів проектування ПЗ, а також набір інструментів, що дозволяють візуально зручно моделювати ПО, аналізувати цю модель на всіх стадіях розробки й супроводження ПЗ та створювати додатки відповідно до інформаційних потреб користувачів.

Методологія CASE-технологій має за основу спадний підход до проектування й дає можливість відслідковувати кожен етап життєвого циклу ІС або її окрему задачу. Спадний підхід до проектування полягає в тому, що характеристики під час реалізації системи конкретизуються чим далі - більш детально.

Переваги застосування CASE-технологій при проектуванні ІС:

– прискорення та полегшення процесу розробки, підвищується якість ІС, що створюється;

– з'являється можливість переносу застосувань із середовища однієї СКБД в іншу за рахунок перетворення концептуальної моделі на фізичну і навпаки;

– є можливість на початкових стадіях розробки проводити більш детальне моделювання системи на початкових етапах розробки.

За допомогою таблиці 8.1 подано порівняння якісних змін процесу розробки ІС при застосуванні Computer Aided Software Engineering.

Таблиця 8.1 - Порівняння моментів традиційної розробки інформаційної системи й розробки з використанням CASE-засобів

Традиційна розробка (ручна)	Розробка застосовуючи CASE- засоби
Основна увага на кодування і тестування	Основна увага на аналіз і проектування
"Паперові" специфікації	Швидке ітераційне прототипування
Ручне кодування	Автоматична генерація коду
Ручне документування	Автоматична генерація документації
Тестування кодів	Автоматичний контроль проекту
Супроводження кодів	Супроводження специфікацій проектування

Інструментальні CASE-засоби складаються з таких компонентів:

– інструменти, що надаються для графічного створення структурних діаграм;

– компоненти дизайну й створення звітів для розробки форматів введення/виведення інформації та користувацьких інтерфейсів;

– єдиний архів (репозиторій) для зберігання даних проекту системи разом зі словником даних;

– інструмент для генерації програмної документації;

– засоби для автоматизованої перевірки несуперечливості системи, її синтаксису наповнення.

В базі даних CASE-системи зберігаються дані, які відносяться до різних етапів життєвого циклу розробки ПЗ: планування, збору й аналізу вимог, проектування, реалізації, тестування, супроводження і документування.

У середовищі розробки CASE проектувальники баз даних та додатків використовують CASE-інструменти, що допомагають із збереженням опису схеми БД, різних елементів даних, прикладних

процесів, звітів, екранів тощо. CASE-інструментарій об’єднує інформацію розробки системи в загальному репозиторії. Адміністратор БД може перевіряти цей репозиторій на точність та несуперечливість, а якщо є необхідність, вносити виправлення.

Репозиторій - застосування БД, яке забезпечує зберігання і обробку даних і метаданих, а також їх представлення за запитами. Репозиторії підтримують різні подання даних й метаданих, що зберігаються, множину їх версій, у них зберігаються всі документи разом з історією модифікацій та іншою службовою інформацією.

Метадані - дані, що описують їх склад і структуру, формат представлення, методи доступу й необхідні для цього повноваження користувачів, місце зберігання, семантику даних тощо.

Метадані виконують такі функції:

- описують властивості ІС, її механізми й інформаційні ресурси в CASE- середовищах;
- застосовуються при обміні даними між різними інструментами CASE й/або застосуваннями ІС;
- є джерелами відомостей про властивості й зміст інформаційних ресурсів для механізмів управління даними в інформаційних системах;
- забезпечення механізмів об’єднання інформаційних ресурсів з різних джерел відомостями про властивості цих ресурсів;
- роль джерела інформації, що необхідна для перетворення ІС;
- забезпечують подання відомостей про систему, її ресурси для різних застосувань і користувачів.

CASE-системи є структурними або об'єктно-орієнтованими.

Структурний підхід застосовує до аналізу та проектування з наступні види моделей:

– DFD (Data Flow Diagrams) - діаграми для потоків даних;

– SADT (Structured Analysis and Design Technique - метод структурного аналізу й проектування) - моделі й відповідні функціональні діаграми;

– ERD (Entity-Relationships Diagrams) - діаграми "сутність-зв'язок".

Об’єктно-орієнтований підхід грунтується на розв’язання завдань, що мають чіткий поділ системи на взаємодіючі між собою сутності (наприклад, імітаційне моделювання, моніторинг, управління технічними об’єктами або технологічними процесами). Частіше застосовується для розподілених систем. Найбільш відомими серед о**б'єктно-орієнтованих** моделей є моделі, побудовані за допомогою мови моделювання UML (Unified Modeling Language - уніфікованої мови моделювання). Словник UML утворюють предмети, відношення й діаграми. Предмети бувають структурні, групуючи, поведінки, й пояснюючі. Розрізняють відношення таких видів: залежності, узагальнення, асоціації й реалізації. Діаграми, з яких складається проект ІС за допомогою UML, зустрічаються таких видів: прецендентів використання, станів, класів, активності, компонентів, слідування, співробітництва й розміщення.

Серед основних компонентів об'єктно-орієнтованих CASE- систем розрізняють:

- репозиторій, що подає об'єктно-орієнтовану БД;

- графічний користувацький інтерфейс (GUI);

- засоби для перегляду проекту, завдяки яким можливо пересуватись елементами проекту (по ієрархії класів й підсистем), переключатись між видами діаграм;

- засоби для контролю проекту;

- засоби для ведення статистики;

- генератор документів, що дозволяє формувати тексти вихідних документів на основі даних репозиторія.

Є можливість інтегрування засобів моделювання даних з іншими моделями проектування на основі розширених можливостей репозиторіїв, та також принципів й середовищ інтеграції CASE.

RAD-технології та компонентно-орієнтовані технології

Для швидшої розробки застосувань використовуються **RAD-технології**. Головні ознаки RAD (Rapid Application Development, середовища швидкої розробки застосувань):

− наявність об'єктно-орієнтованої мови програмування;

− застосування візуальних засобів розробки;

− підтримка стандартних протоколів обміну даними між додатками, що дозволяє створювати багаторівневі, незалежні від джерела даних застосування.

Дана технологія призначена для створення максимально швидким методом версії програмного продукту. При виборі цього підходу систему поділяють на підсистеми, що є слабко пов'язаними за даними та функціями й точно визначають інтерфейси між ними.

Компонентно-орієнтовані технології – це технології, що грунтуються на застосування готових програмних компонентів, що попередньо були розроблені. В цій ситуації широко застосовуються бібліотеки класів, а застосування готового модуля виконується за допомогою його інтерфейсу. Специфікації, що визначають інтерфейс, відокремлені від модуля, а внутрішні деталі сховані від користувача. Компоненти постачаються в скомпільованій формі.

Запит від користувача на виконання процедури надсилається посереднику. Посередник має попередньо підготовлений каталог чи репозиторій інтерфейсів процедур з покажчиком на компоненти виконавці. Результати, що отримуються, повертаються користувачу після виконання.

Технології штучного інтелекту

Роль штучного інтелекту (ШІ), як технології для проектування БД дуже важлива. Існують різні інструменти на основі ШІ, що допомагають у проектуванні БД.

Розглянемо деякі з них: SuperDuperDB, Taskade AI, PostgreSQL, Airtable AI, Towhee, MongoDB Atlas та інші.

SuperDuperDB - це така структура Python, що інтегрує можливості ШІ в БД, що існують. Він також інтегрує API моделей штучного інтелекту та векторні пошукові системи з базами даних. Цей засіб включає повний ШІ та екосистему Python, таким чином має найкращу інфраструктуру даних. Він здатний забезпечувати ефективний спосіб розгортання, керування та створення програм ШІ.

MongoDB Atlas AI – це сервіс, що дозволяє розробляти програми, збагачені ШІ. Це сприяє розробці сучасних програм, які використовують ШІ для різних цілей, в тому числі для запобігання шахрайству, прогнозне обслуговування та керування персоналізацією. Це повністю керований хмарний сервіс БД, що надає потужну платформу для роботи з БД MongoDB.

PostgreSQL - це засіб розширення машинного навчання для баз даних PostgreSQL. Він може виконувати різноманітні завдання обробки природної мови (NLP), такі як відповіді на запитання, переклад, резюмування, аналіз настроїв та генерування тексту за допомогою простих рядів SQL. Він включає в себе машинне навчання в БД, має доступ до моделей, що попередньо навчені, а також налаштування, гнучкість й інтеграцію з існуючими екосистемами.

Taskade AI – це комплексний інструмент для проектування БД, що пропонує генератор схем БД. Він може керувати, розробляти проекти та генерувати ідеї в життя за допомогою ШІ. Він здатний підвищити продуктивність роботи при проектуванні БД.

Застосування технологій ШІ при проектуванні та дизайні баз даних допомагають досягати мети, при цьому, полегшує та пришвидшує роботу.Інновації просуваються вперед. Вони дозволяють оптимізувати створення схем БД, роблячи їх більш масштабованими та стійкими до потреб зазначених даних. Такі функції, як генерація на основі ШІ та можливості оптимізації власної схеми, підвищують ефективність робіт.

Контрольні запитання:

1. Які засоби та які технології називають CASE-засобам і CASE-технологіями?
2. Які основні переваги CASE-технологій для розробки інформаційних систем?
3. З яких компонентів складаються інструментальні CASE-засоби?
4. Що являє собою методологія функціонального моделювання?
5. Яка технологія називається RAD-технологією?
6. Яка технологія називається компонентно-орієнтованою?
7. Які відмінності CASE-систем, що базуються на структурному й об'єктно-орієнтованому підходах?
8. Що спільного у CASE-систем, що базуються на структурному й об'єктно-орієнтованому підходах?
9. Що означає поняття репозиторій?
10. Які засоби ШІ застосовують для проектування БД?

9. Розподілена обробка даних

За видами архітектур БД, загалом, можна розділити на однокористувацькі та на багатокористувацькі. Загальну схему режимів використання БД можна представити у вигляді, як на рисунку 9.1.

Якщо до БД мають доступ одночасно декілька користувачів, то СКБД повинна забезпечувати коректну паралельну роботу всіх користувачів з даними. Розрізняють розподілену обробку і розподілені БД.

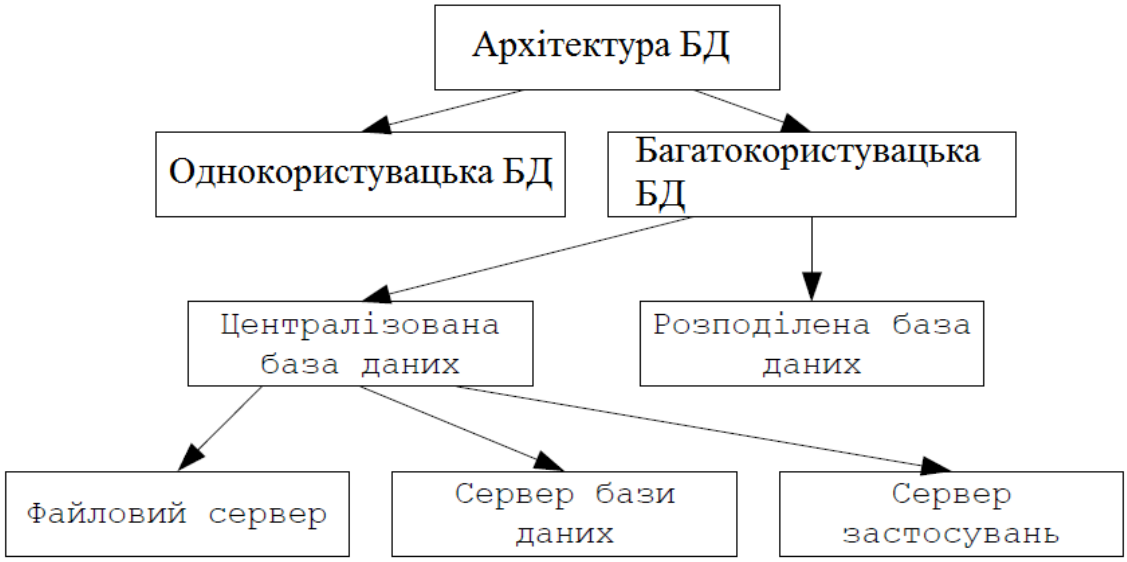


Рисунок 9.1 – Види режимів роботи з БД

Розподілена обробка - це робота з обробки даних з використанням централізованої БД, доступ до якої мають різні комп'ютери мережі. В цій системі одні вузли (комп’ютери) - клієнти, а інші - сервери.

Сервер - комп'ютер, що надає деякі послуги іншим комп'ютерам, обмін повідомленнями з якими відбувається за допомогою мережі.

Клієнт – це комп’ютер чи процес, що надсилає запит на обслуговування.

Розподілена БД - це набір логічно зв'язаних між собою роздільних даних і їх описів, які фізично розподілені в мережі.

Тобто, розподілена БД застосовується на різних просторово розосереджених обчислювальних засобах, разом з організаційними, технічними й програмними засобами її створення й обслуговування.

В дійсності розподілена БД є віртуальною БД, компоненти якої фізично зберігаються на декількох різних реальних БД на декількох різних вузлах [8].

Розподілена СКБД – це програмна система, яка призначена для керування розподіленими БД і яка забезпечує прозорий доступ користувачів до інформації.

Гомогенною (однорідною) називається розподілена система, в якої всі вузли використовують той самий тип СКБД.

Гетерогенною (різнорідною) називається розподілена система, в якої всі вузли використовують різні типи СКБД, які обробляють різні моделі даних.

Керування паралельною обробкою

В багатокористувацьких системах до БД одночасно мають доступ декілька користувачів або прикладних програм. Виникає необхідність захисту БД від можливих випадкових чи спланованих ситуацій, коли існує вирогідність втрати дані. Для збереження цілісності даних і забезпечення безпеки в цих умовах застосовуються транзакції, які забезпечують роботу кожного користувача з узгодженим станом БД. Підтримка механізму транзакцій – показчик рівня розвинутості СКБД.

Транзакція - послідовність операторів маніпулювання даними (читання, видалення, вставки, модифікації), яка розглядається СКБД, як одне ціле. Транзакція або успішно виконується, і СКБД фіксує зміни БД, які були виконані транзакцією, у зовнішній пам'яті, або, у разі невдачі, жодна зміна не відбувається у БД.

Щоб механізм транзакцій забезпечував цілісність даних й ізольованість користувачів, транзакція повинна мати такі властивості:

атомарність (Atomicity), узгодженість (Cosistency), ізольованість (Isolation), довготерміновість (Durability). Транзакції, які мають ці властивості називаються ACID-транзакціями [8]:

-Атомарність або нерозривність означає, що транзакція виконується, як єдина операція доступу до БД і виконується або повністю або не виконується зовсім (за забезпечення нерозривності відповідає підсистема відновлення СКБД);

-Узгодженість гарантує взаємну цілісність даних, тобто виконання обмежень цілісності БД після завершення роботи транзакції (відповідальність за забезпечення властивості узгодженості покладається на СКБД та на розробників додатку);

-Ізольованість можна означити так, що транзакції, які конкурують за доступ до БД, проходять послідовну фізичну обробку, ізольовано одна від одної, але мають виглядає для користувачів, начебто, паралельно виконуються;

-Довготерміновість або стійкість означає, що коли транзакція виконана успішно, то всі зміни, які вона зробила в даних, не будуть втрачені ні за яких обставин (за забезпечення стійкості відповідає підсистема відновлення).

Для паралельних транзакції застосовується серіалізація транзакцій і метод тимчасових міток.

Серіалізація транзакцій - це процедура, яка забезпечує підтримку незалежного виконання транзакцій. Іншими словами, дія двох паралельно діючих транзакцій буде як і їх послідовна дія: спочатку одна, а потім інша, або навпаки, спочатку друга, а потім перша. У ході виконання транзакції користувач видно тільки узгоджені дані, а неузгоджені проміжні залишаються непомітними. Для підтримки паралельної роботи складається спеціальний план.

Для реалізації серіалізації транзакцій застосовується **механізм блокувань.** Цей механізм має встановити режим доступу (сумісний або

монопольний) до певного ресурсу даних. Це дозволяє виключити доступ до цього об’єкту одночасно з даною транзакцією інших транзакцій, і як результат, може статись порушення логічної цілісністі даних БД. Блокувати можна всю базу даних, або окремі рядки, таблиці або сторінки.

Застосовують наступні види блокувань, щоб підвищити ступінь паралельності доступу до одної бази даних декількох користувачів:

– *нежорстке* блокування або роздільне блокування (Shared S-блокування); об'єкт блокується для виконання операції читання; об'єкти в цьому випадку не змінюються у ході виконання транзакції і доступні іншим транзакціям також, але тільки в режимі читання;

– *жорстке* блокування або монопольне (eXclusive X-блокування); об'єкт блокується для виконання операції запису, модифікації або вилучення. При такій ситуації здійснюється блокування об'єкта монопольно й об'єкт залишається недоступним іншим транзакціям поки не закінчиться робота даної транзакції.

Застосування різних типів блокувань призводить до тупиків. Виникає **тупикова ситуація** в таких випадках, коли дві й більше транзакцій знаходяться в режимі очікування одночасно, для продовження роботи кожної транзакції необхідно завершити роботу іншої транзакції.

Наприклад, якщо транзакція 1 у деякий момент часу t1 блокує ресурс A, а транзакція 2 у момент часу t2 блокує ресурс B. A в момент часу t3 транзакції 1 потрібен ресурс B та вона потребує його звільнення транзакцією 2. A в момент часу t4 транзакція 2 потрібен ресурс A та вона потребує його звільнення транзакцією 1. Транзакція 1 очікує транзакцію 2, а транзакція 2 очікує транзакцію 1. Алгоритм виходу із тупика передбачає визначення транзакції-жертви. Після вибору такої транзакції виконується її відкат.

Застосовується двофазне блокування для серіалізації транзакцій, що виконується таким чином:

– транзакція блокує деякий об'єкт перед виконанням операцій з ним (накопичення захватів);

– транзакція не накладає ніяких інших блокувань після того, як відбулось зняття блокування (вивільнення захватів).

Порядок виконання операцій транзакції

Будь-яка транзакція має у завершенні один із двох варіантів. У варіанті успішного завершення результат транзакції фіксується (commit) в БД. БД переходить в новий узгоджений стан. Інакше, якщо виконання транзакції не успішне, вона відміняється. В цьому варіанті БД повинна відновити той узгоджений стан, в якому вона знаходилась до початку транзакції. Цей процес називається відкатом (roll back), або відміною транзакції [6][8].

Зафіксовану транзакцію не можна відмінити. У разі якщо зафіксована транзакція була помилковою, то виконують іншу транзакцію, що відмінить дії попередньої транзакції. Таку транзакцію називаю **компенсуючою**. Однак, у випадку якщо транзакція, що завершилась аварійно, а для неї було застосовано відкат, може бути викликана на виконання пізніше та, залежно від причин попередньої відмови, цілковито успішно завершена й зафіксована в БД.

В більшості СКБД для визначення границь окремих транзакцій використовуються оператори BEGIN TRANSACTION, COMMIT і ROLLBACK (або щось еквівалентне). У разі, якщо не були використані, зазвичай, як єдина транзакція розглядається вся програма, що виконується. Автоматично виконується команда COMMIT у разі нормального завершення цієї програми та, у разі аварійного завершення програми, в БД автоматично буде застосована команда ROLLBACK.

Розподілені СКБД

В порівнянні з централізованою СКБД розподілена СКБД, повинна мати додатково наступні функціональні можливості:

- розширені служби встановлення з'єднань повинні забезпечувати доступ до віддалених вузлів і дозволяти передавати запити й дані між вузлами, які входять у мережу [8];

- розширені засоби ведення каталогу, що дозволяють зберігати інформацію про розподіл даних в мережі;

- засоби обробки розподілених запитів, включаючи механізми оптимізації запитів та організації віддаленого доступу до даних;

- функції управління захистом (розширені), що дозволяють забезпечити дотримання правил авторизації та прав доступу до розподілених даних;

- функції управління паралельним виконанням (розширені) , які дозволяють підтримувати цілісність копіювання;

- функції розширеного відновлення, які враховують вирогідність відміни роботи окремих вузлів та віказ у зв’язкових лініях.

Розподілені СКБД можуть складатись з **компонентів**:

- робочі станції (сайти або вузли);

- компоненти мережевого обладнання та ПЗ кожної робочої станції, які дозволяють усім вузлам взаємодіяти один з одним та обмінюватися даними;

- комунікаційні пристрої, завдяки яким переносяться дані з однієї робочої станції на іншу;

- процесор транзакцій (TP) – програмний компонент, що знаходиться на кожному комп'ютері, де виконується запит даних. TP отримує і обробляє запити;

- процесор даних (DP) – програмний компонент, розташований на кожному комп'ютері, де зберігаються й обробляються дані, розташовані на даному вузлі. DP може представляти централізовану СКБД.

Залежно від того, яким чином розташовані компоненти по відношенню один до одного розрізняють монолітне виконання, дворівневе і трирівневе.

Доступ до даних в розподіленій базі даних звичайно забезпечується в трирівневій моделі: клієнт - сервер додатків - вузли збереження даних. При інтернет-доступі до даних роль сервера додатків відіграє вебсервер або спеціальний додаток на боці клієнта.

Багаторівнева архітектура або N-рівнева архітектура

За останні роки, разом із зростанням популярності інтернет-додатків та збільшенням кількості користувачів, традиційна трирівнева клієнт-серверна архітектура була розширена шляхом додавання нових рівнів, що сприяло появі багаторівневих архітектур. Зазвичай такі архітектури складаються з чотирьох рівнів, де на сервері в мережі відбувається обробка зв'язку між клієнтським браузером і сервером додатків. Однією з переваг є можливість підключення кількох веб-серверів до одного сервера додатків, що сприяє обробці більшої кількості одночасно підключених користувачів. Схема такої архітектури подається на рисунку 9.2.

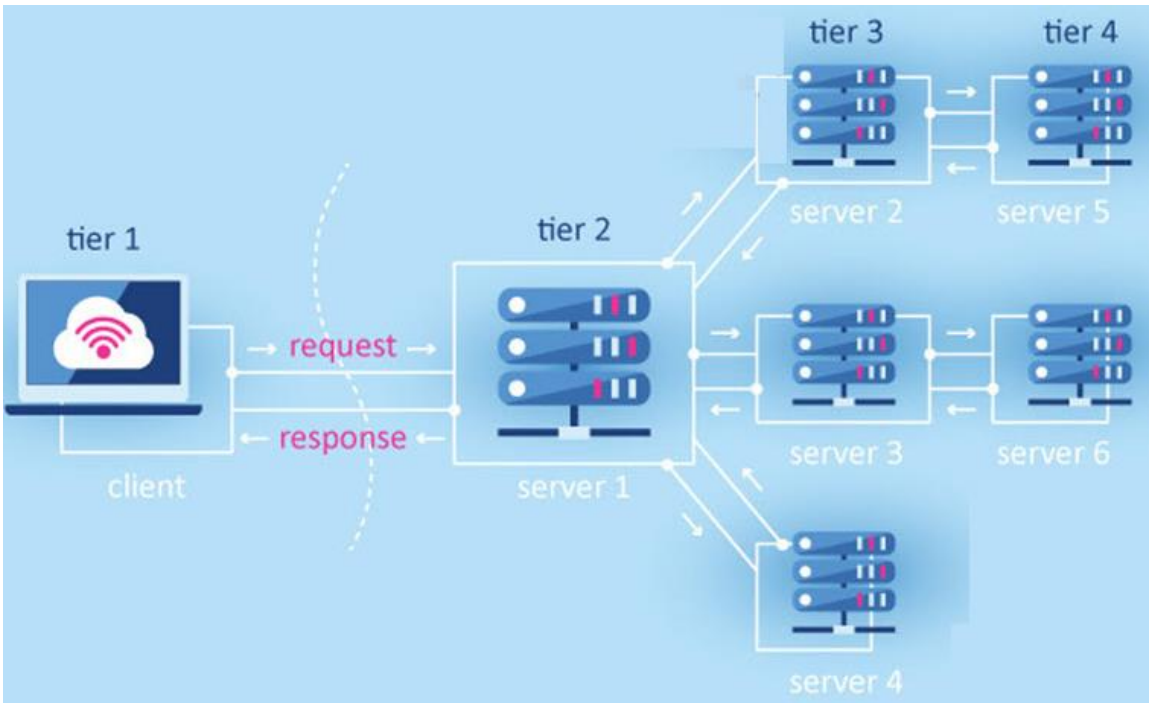


Рисунок 9.2 – Багаторівнева клієнт-серверна архітектура [9]

Багаторівнева архітектура дозволяє підвищити ефективність роботи ІС, а також оптимізувати розподіл її програмно-апаратних ресурсів.

Розподілена СКБД, в якій керування кожним з вузлів здійснюється автономно називається **мультибазовою системою**.

Існують **необ'єднані** (не мають локальних користувачів) і **об'єднані** мультибазові системи. Об'єднані системи – це системи гібриди, що складаються з розподіленої та централізованої систем, оскільки вона має вигляд як розподілена система для користувачів, що віддалені, а виглядає наче централізована система для користувачів локальних.

Один з прикладів мультибазової системи є система компанії Cincom Corporation – UniSQL. Ця система може створювати програми за допомогою єдиного глобального уявлення і єдиної мови доступу до БД для роботи з багатьма різнорідними реляційними і об'єктно-орієнтованими СКБД [8].

Відмінність між розподіленими системами БД і засобами розподіленої обробки даних. Розподілена обробка даних використовує централізовану БД, доступ до якої може здійснюватись з різних комп’ютерів мережі. Вузол, на якому розміщена БД, називають **сервером** БД. Важливо, що система працює з даними, які фізично розподілені мережею, тобто фрагменти розподіленої БД розміщені на різних вузлах.

При розподіленій обробці даних, логічні операції БД розподіляються між двома або більше фізично незалежними вузлами, які з'єднані в мережу. Наприклад, розподілена обробка може виконувати введення/виведення даних, вибірку і перевірку даних на одному комп'ютері, а випуск звіту на основі цих даних - на іншому. Кожен вузол має доступ до даних та може оновлювати БД.

Відмінності розподіленої обробки даних від розподіленої БД:

Перша не потребує розподіленої бази даних, але друга обов'язково вимагає розподіленої обробки інформації.

Обидва підходи потребують наявності локальної мережі для зв'язку між компонентами.

Також, варто **розрізняти розподілені та паралельні системи керування базами даних** (СКБД). Паралельні СКБД використовують декілька процесорів і дисків для підвищення продуктивності. Наприклад, вони можуть об'єднувати декілька слабкопотужних машин для досягнення такої ж продуктивності, що й у потужнішої машини, але при цьому забезпечують більшу масштабованість та надійність.

Паралельна СКБД повинна забезпечувати управління сумісним доступом до ресурсів, якщо надається декільком процесорам спільний доступ до однієї і тієї ж БД. В залежності від ресурсів, що поділяються, розрізняють три види паралельних СКБД: - системи з поділом пам'яті; - системи з поділом дисків; - системи без поділу обчислювальних ресурсів [8].

Паралельна система, **що не розділяє обчислювальні ресурси** (цю архітектуру інакше називають архітектурою з масовою паралельною обробкою), іноді порівнюється з розподіленою системою керування базами даних (СКБД). Проте у такій системі розподіл даних здійснюється для поліпшення продуктивності, а не з метою реального розподілу. Вузли розподіленої СКБД часто розташовані в різних місцях, керуються різними адміністраторами та підключені через повільні мережеві зв'язки. Натомість вузли паралельної СКБД зазвичай знаходяться на одному комп'ютері або в рамках одного виробничого майданчика.

Системи з **поділом пам'яті** включають тісно зв'язані компоненти, які мають кілька процесорів і ділять загальну системну пам'ять. Ця архітектура також відома як архітектура з симетричною

багатопроцесорною обробкою (SMP) і зараз широко застосовується на різних обчислювальних платформах. Вона охоплює персональні робочі станції з кількома паралельно працюючими мікропроцесорами, великі RISC-системи та найбільші мейнфрейми. Така архітектура дозволяє швидко отримувати доступ до даних для обмеженого числа процесорів, зазвичай не більше 64. У протилежному випадку, взаємодія через мережу може стати головним обмеженням всієї системи.

Системи **з поділом дисків** складаються з менш тісно пов'язаних компонентів. Вони є оптимальним вибором для додатків, які мають високу централізацію обробки і повинні гарантувати найвищу доступність і продуктивність. Кожен процесор має прямий доступ до всіх дискових пристроїв, але має власну оперативну пам'ять. Подібно до архітектури без поділу обчислювальних ресурсів, архітектура з поділом дисків усуває обмеження, пов'язані зі спільно використовуваною пам'яттю. Але на відміну від архітектури без поділу обчислювальних ресурсів, ця архітектура усуває ці обмеження без додаткових витрат на фізичний розподіл даних по окремим пристроям. Іноді такі колективні дискові системи називають кластерами.

Паралельні технології часто використовуються для дуже великих баз даних, розміри яких можуть сягати кількох терабайт або в системах, що обробляють тисячі транзакцій на секунду. Такі системи потребують швидкого доступу до великого обсягу даних і мають важливе значення з точки зору часу відповіді на запити.

Спосіб розташування даних визначає поділ розподілених БД на **зосереджені** і **розосереджені**.

Зосереджені (або **централізовані**) розподілені БД розташовуються в одному місці фізично. Для обміну даними між локальними частинами БД застосовуються канали зв'язку прямого доступу. Обмін інформацією між взаємопов'язаними підбазами відбувається без очевидних обмежень щодо обсягів і характеру передаваної інформації.

Децентралізовані (або **розосереджені**) БД фізично розташовані у різних місцях - вузлах обчислювальної мережі. Обмін інформацією між підбазами здійснюється за допомогою каналів зв'язку. У таких системах можуть використовуватись як централізовані БД, так і окремі локальні підбази.

Обмін інформацією між взаємопов'язаними підбазами переважно відбувається результатною (обробленою, узагальненою) інформацією. При виконанні запитів у таких системах запит розбивається на підзапити до локальних підбаз і виконується паралельно в різних вузлах обчислювальної мережі.

Децентралізовані БД мають беззаперечні переваги порівняно з централізованими БД.

Проектування розподілених баз даних

Рисунок 9.3 представляє схему архітектури розподіленої СКБД.

Розглятуті наступні схеми розміщення даних в системі: централізоване, фрагментоване, з повною реплікацією, з вибірковою реплікацією.

Розміщення **централізоване -** характерне тим, що на одному з вузлів мережі знаходиться єдина БД, доступ до якої мають всі користувачі мережі.

Розміщення **фрагментоване** - характерне тим, що в одному з вузлів системи розміщується один з фрагментів, на які поділяється БД.

Розміщення з **повною реплікацією** означає, що кожен вузол системи утримує повну копію БД.

Вибіркова реплікація це розміщення, що комбінує методи фрагментованого розміщення з методами, реплікації й централізованого розміщення.

Реплікація даних (Data Replication) [8] – це механізм синхронізації вмісту декількох копій фрагментів розподіленої БД. Під реплікацією слід розуміти асинхронне перенесення змін об’єктів

вихідної БД в БД, що знаходяться на різних вузлах розподіленої системи.

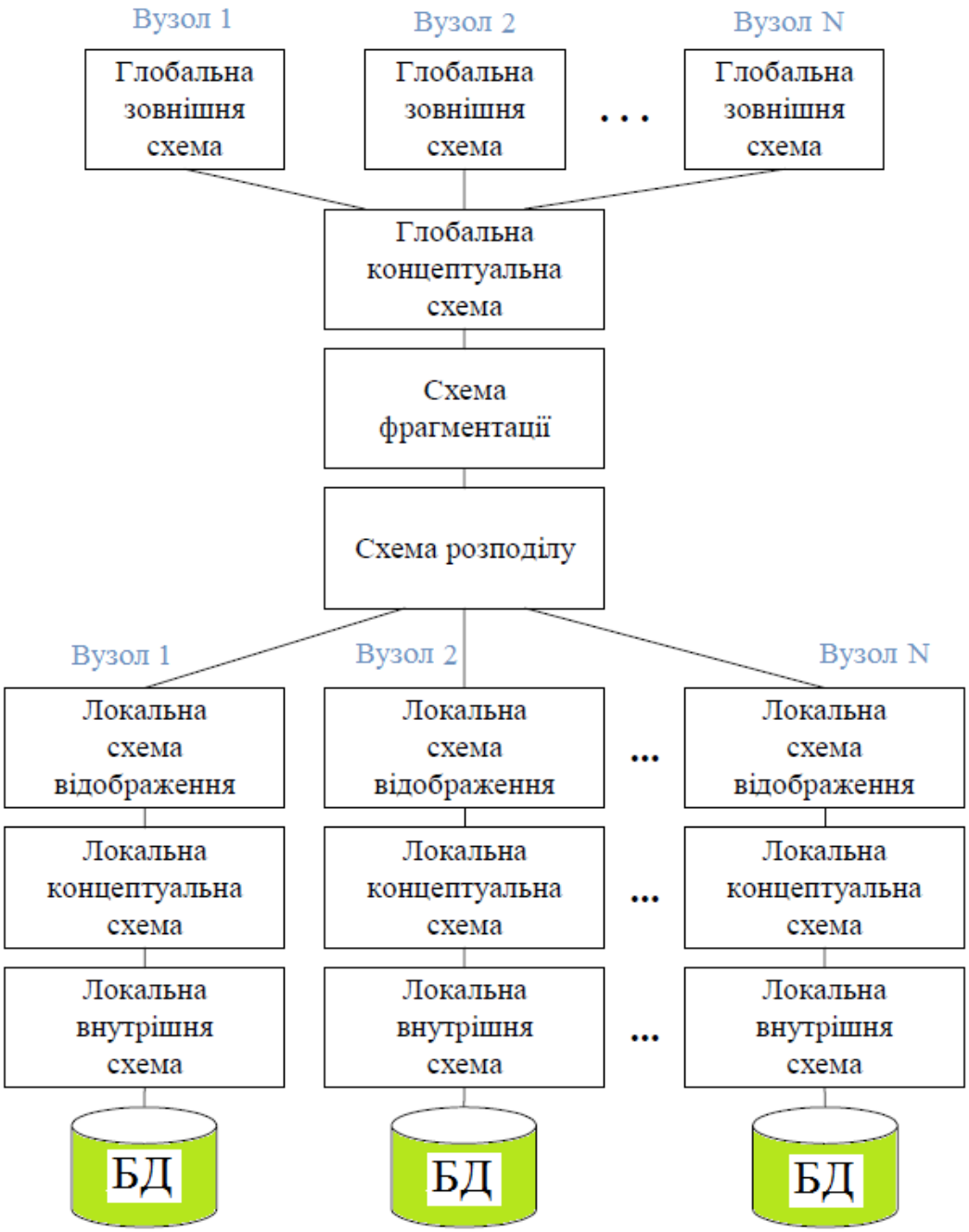


Рисунок 9.3 – Схема архітектури розподіленої СКБД

Використання реплікацій дозволяє досягнути багатьох переваг [1,8]. Реплікація бази даних відіграє важливу роль у забезпеченні доступності, продуктивності, стійкості та відновленні даних для організацій будь-якого розміру. Розуміння концепцій і методів,

пов'язаних з реплікацією, є необхідним для створення стійких до відмов і масштабованих систем обробки даних [10].

Основні принципи організації розподілених баз даних

Найголовніше правило, що враховується у створенні розподілених БД, таким чином, щоб для користувача вони здавались, як нерозподілені БД.

К. Дейт запропонував деякі правила для підкріплення основного принципу організації розподілених БД:

- незалежність або автономність вузлів у розподіленій системі;
- не повинно бути в системі вузлів, без яких функціонування системи стає неможливим;
- користувач повинен мати доступ до бази даних з кожного вузла (виконується умова незалежності від розміщення вузла);
- доступ до інформації здійснюється незалежно від засобів фрагментації;
- незалежність від реплікації (користувач не має засобів доступу до конкретної копії даних та не виконує оновлення копій в базі даних);
- необхідність можливості роботи розподілених запитів;
- необхідність підтримки керування розподіленими транзакціями;
- незалежність від типу СКБД;
- повинна виконуватись на різних апаратних платформах [1][8];
- незалежність програмна та мережна;
- здатність системи чи процесу працювати без перебоїв.

Перелік інтерфейсів, що застосовуються для доступу до БД:

– API (Application Programming Interface) - інтерфейс прикладного програмування;

– SQL/CLI, SQL/Call Level Interface - інтерфейс рівня викликів;

– ODBC (Open Database Connectivity) відкритий інтерфейс для підключення до БД;

– JDBC (Java Database Connectivity) інтерфейс прикладного програмування для мови Java;

– OLE DB (Object Linking and Embedding for DataBase) об'єктно-орієнтований інтерфейс;

– ADO (ActiveX Data Objects) високорівневий об'єктно-орієнтований інтерфейс;

– ADO.NET (ActiveX Data Objects.NET) високорівневий об'єктно-орієнтований інтерфейс для доступу на платформі .NET.

Методологія проектування розподілених БД

Методологія проектування розподілених БД включає шість основних етапів [8].

1. Розробити проект для глобальних відносин.

2. Провести дослідження топології системи. Наприклад, визначити, чи повинно бути передбачено застосування БД в кожному підрозділі компанії, в конкретному вузлі. У першому варіанті може виявитися найбільш доцільною фрагментація відношень за номером підрозділу. А в інших варіантах найбільш вигідне рішення може передбачати фрагментацію відношень з урахуванням того, чи відносяться дані до певного міста чи регіону.

3. Визначити та проаналізувати найбільш важливі транзакції в системі та вирішити, за яких умов може підійти горизонтальна або вертикальна фрагментація.

4. Розглянути відношення , які не слід фрагментувати; копії таких відношень потрібно розподіляти всіма вузлами. На глобальній ER-діаграмі прибрать відношення, які не повинні бути фрагментовані, а також зв’язки, в яких беруть участь виконувані на них транзакції.

5. Перевірити відношення, що відносяться до сторони «один» зв’язку «один до багатьох», та організувати більш прийнятну схему фрагментації для відношень, враховуючи топології системи. Якщо

розглядать відношення на стороні «багато» зв’язку «один до багатьох», то є можливість, що вони є підходять для похідної фрагментації.

6. В залежності від результатів п’ятого етепу органызувать визначення необхідності проведення вертикальної додаткової або змішаної фрагментації (або перевірити, чи є такі транзакції, що потребують доступу до ряду атрибутів відношення).

Таким чином, при роботі з розподіленими БД необхідно мати подання про розподілену обробку даних, що означає термін «Розподілена база даних», які СКБД називають розподіленими. Також необхідно розрізняти розподілені й паралельні системи керування базами даних. Керування паралельною обробкою буде корисним при роботі з БД а знання механізму транзакцій та їх особливостей роботи допоможе вберегти базу даних від невірних результатів роботи та залишати її цілісною й в узгодженому стані. На цей час жодна з розподілених БД не відповідає всім наведеним правилам, однак правила Дейта повністю описують розподілені бази даних і відіграють важливу роль при їх розробці. Реплікація БД відіграє важливу роль у забезпеченні доступності, продуктивності, стійкості та відновленні даних. Розуміння концепцій і методів, пов'язаних з реплікацією баз даних, є необхідним для створення стійких до відмов і масштабованих систем обробки даних. Вивчаючи методологію проектування розподілених БД можна досягнути неабияких успіхів при роботі у цій сфері.

Контрольні запитання:

1. Що означає термін «Розподілена обробка»?
2. Що означає термін «Розподілена БД»?
3. Що означає термін «Розподілена СКБД»?
4. Які системи називають Гомогенними?
5. Які системи називають Гетерогенними?

6. Який комп’ютер називають сервером?

7. Що називають клієнтом при роботі з БД?

8. Що називають транзакцією?

9. Які властивості повинна мати транзакція, щоб механізм транзакцій забезпечував цілісність даних й ізольованість користувачів?

10. Яку процедуру називають серіалізація транзакцій?

11. Для яких цілей застосовують механізм блокувань при роботі з БД?

11. Яким чином визначається порядок у виконанні операцій транзакцій?

12. З яких компонентів можуть складатись розподілені СКБД?

13. Які системи називають мультибазовими системами?

14. Які розподілені системи називають зосередженими?

15. Які розподілені системи називають розосередженими?

16. Які основні правила організації розподілених БД?

17. Які існують інтерфейси, що застосовуються для доступу до БД?

18. Що означає реплікація БД?

10. SQL

Історія мови SQL та її можливості

Історія SQL починається з 70-х років XX століття, коли в дослідницькій лабораторії IBM у штаті Каліфорнія було розроблено першу версію цієї мови. Назва **SQL**, тобто структурована мова запитів, є абревіатурою від **Structured Query Language**, походить від «sequel», що є первісною назвою. Спершу ця мова з'явилась на світ в реляційній СКБД DB2, що виготовила IBM. Відрізняючись від аналогічних мов третього покоління (COBOL, C), що виникли в той період часу, мова SQL не є процедурною [16][17].

Особливість реляційних СКБД (RDBMS) полягає у тому, що вони надають множинно-орієнтовану мову маніпулювання БД, тобто результатом дії мовного оператора є таблиця, яка містить множину даних. Більшість сучасних RDBMS використовують саме мову SQL.

SQL працює в таких СКБД: MySQL, SQL Server [18], Oracle, MS Access, Sybase, Informix, Postgres та інших системах баз даних.

Американський інститут національних стандартів (American National Standards Institute, тобто ANSI) та Міжнародна організація стандартів (International Standards Organization - ISO) займаються описом і підтримкою стандартів цієї мови. Всі сьогочасні СКБД підтримують певний стандарт, проте є й відхилення, які конкретно від ситуації, що склалась, специфікуються в документації програмного продукту. До того ж, існують багато систем, що дають можливість використовувати мову запитів SQL у середовищі програмування.

SQL пропонується у випадку потреби операцій з даними в БД, аналогічних створенню, видалення, читанню та оновлення даних. Ось деякі з **особливостей** БД SQL:

- Гнучкість і масштабованість. БД SQL є дуже гнучкими й можуть бути масштабовані відповідно до різних користувацьких потреб.

- Комплексний інструмент розробки додатків. БД SQL можна використовувати для розробки різноманітних додатків, від простих систем введення даних до складних корпоративних додатків.

- Розширена підтримка транзакцій. БД SQL забезпечують розширену підтримку транзакцій, що гарантує те, що дані можуть оброблятися послідовним і надійним способом.

- Висока продуктивність. БД SQL розроблено для забезпечення високої продуктивності навіть для складних запитів [13].

- Висока доступність. БД SQL створені для високої доступності навіть у разі збою.

Загалом БД SQL є потужним і універсальним інструментом для керування даними. Їх легко опанувати та використовувати, вони мають у своєму арсеналі обширний ряд функцій для ідеальної роботи різноманітних застосувань.

Бази даних SQL є популярним вибором для зберігання та керування даними [14].

SQL поділяється на групи команд:

- Мова для визначення даних (DDL або Data Definition Language): DDL використовується для створення, видалення а також зміни об’єктів бази даних, таких як таблиці, подання, індекси, збережені процедури та БД.

- Мова маніпулюваня даними (DML або Data Manipulation Language): DML використовується і для вставки, і для оновлення та також видалення даних із таблиць.

- Мова для запитів даних (DQL або Data Query Language): DQL використовується у випадку отримання даних із таблиць.

- Мова для керування транзакціями (TCL або Transaction Control Language): TCL використовується для керування транзакціями, які є набором операцій, які розглядаються як єдина одиниця роботи.

- Мова керування привілеями (DCL або Data Control Language) - Команди управління даними дозволяють управляти доступом до інформації, що знаходиться всередині БД. Як правило, використовується для створення об'єктів, пов'язаних з доступом до даних, а також служать для контролю над розподілом привілеїв між користувачами [12].

Вміння працювати з SQL допомагають користувачам БД підтримувати, створювати та отримувати інформацію з РБД. Стає можливим за допомогою SQL отримувати доступ до певної необхідної інформації, оновлювати БД, маніпулювати (додавати, видаляти, змінювати) даними та ін.

Основні відомості про сервер БД MySQLServer

та панель керування phpMyAdmin

MySQL – виступає в якості вільної системи керування реляційними базами даних.

Ця відома система з відкритим кодом з’явилась, як щось альтернативне комерційним системам. Вона розроблена фірмою «TcX» з ціллю для підвищення швидкодії опрацювання великих БД. З початку існування MySQL була подібною до mSQL, але плином часу вона поступово змінювалась. Тепер MySQL відноситься до найпоширеніших СКБД. Оскільки вона має сильну підтримку з боку різних мов програмування, то часто застосовується саме для програмування динамічних web сторінок. Використовується модель «клієнт-сервер».

MySQL - компактний багатонитковий сервер БД з гарними характеристиками щодо швидкодії, простоти застосування та стійкості.

Компіляція вихідних кодів можлива на різних платформах. Крім того MySQL підходить як для невеликих застосувань, так і для

застосувань середнього розміру. В UNIX-системах можна виявити можливості серверу найкращим чином бо вони підтримують багатонитковість. А це підвищує продукційність системи [19].

Активне використання MySQL в веб-програмуванні зумовило її актуальність, а інтуїтивно зрозумілий інтерфейс в сукупності з широкою функціональністю й підтримкою більш 60 мов (зокрема українською) забезпечило їй популярність серед веб-розробників.

Для користування без комерційної мети продукт надається безкоштовно.

Особливі характеристики сервера MySQL:

- характеризується високою швидкодією;
- прості дії встановлення та при використанні;
- можливість у таблицях кількості рядків досягати 50 млн.;
- підтримувати необмежену кількість користувачів, які одночасно мають працювати з БД;
- має високий рівень безпеки, за рахунок вбудованих інструментів безпеки;

Особливості роботи **сервера MySQL:**

Структура клієнт-серверної системи проста. Клієнт підключається через мережу до сервера. Він надсилає запит через інтерфейс користувача (GUI). Якщо інструкції зрозумілі серверу, то він видає інформацію.

У середовищі MySQL відбуваються такі процеси:

- створюється БД MySQL і визначаються відношення;
- клієнти надсилають запити, за допомогою команд SQL;
- сервер відповідає на запити клієнтам.

Серед популярних інтерфейсів є наступні: SequelPro, MySQL WorkBench, DBVisualizer та Navicat DB Admin Tool. Багато інтерфейсів сумісні з популярними ОС.

Застосування OpenServer

OpenServer – це портативний серверний стек, що дозволяє легко запускати та тестувати веб-додатки на Windows. Він включає такі компоненти, як Apache, PHP, MySQL та інші. Ось основні кроки для налаштування OpenServer:

Завантаження та інсталяція:

Необхідно перейти на офіційний сайт OpenServer та завантажити останню версію програми. Встановлення відбувається, якщо слідувати інструкціям та підказкам на екрані.

Запуск OpenServer:

Запуск OpenServer можна організувать через ярлик на робочому столі або через меню "Пуск".

При появі індикатору в треї OpenServer, викликати контекстне меню й обрати "Запустити". З моменту коли прапорець загорівся зеленим кольором, локальний сервер на даному ПК запущений.

Налаштування параметрів сервера:

Клікніть мишею, для вибору контекстного меню, на індикаторі OpenServer в треї та оберіть "Налаштування".

У вікні налаштувань можна змінити версії PHP, MySQL, Apache, а також налаштувати автозапуск сервісів, кодування та інше.

Налаштування доменів:

Якщо у вікні налаштувань перейти у розділ "Домени", то можна додавати, видаляти або змінювати домени (локальні сайти), які ви планується тестувати.

Керування базами даних:

Можливо застосувати phpMyAdmin або інший менеджер БД для створення, імпорту, експорту або зміни баз даних.

Доступ до phpMyAdmin можна отримати через меню OpenServer у треї.

Робота з файлами проектів:

Файли проектів зберігаються в теці "domains" в кореневій директорії OpenServer. Щоб легко переключатись між різними проектами, необхідно додавати нові папки для кожного проекту в цю тецю.

Перевірка роботи сервера:

Необхідно відкрити браузер і ввести назву вашого локального домену, наприклад "myproject.localhost", для перевірки, чи все працює коректно.

Після встановлення OpenServer, виконується автоматична авторизація системи, так як нам не потрібний наразі ніякий захист нашого сайту від злому.

Користувач за замовчуванням «root», пароль у Вас не вказано. Натискаємо вперед і заходимо в phpMyAdmin.

Необхідно також виконати деякі налаштування, щоб постійно не вписувати пароль.

Відкриваємо конфігураційний файл, що зображено на Рисунку 10.01.

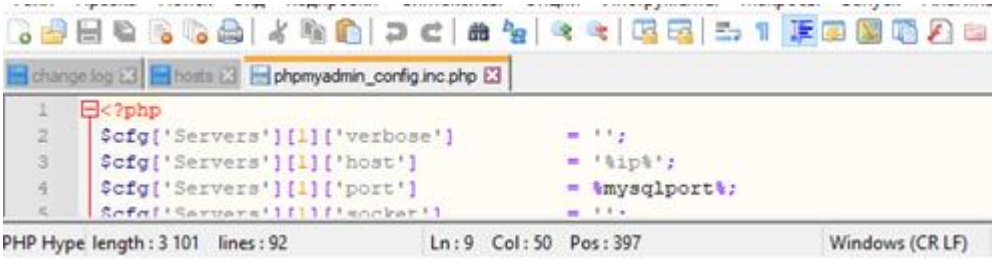


Рисунок 10.01 – Конфігураційний файл

Необхідно змінити [auth_type] = ‘cookie’ на [config]

та додати два налаштування:

[‘controuser’] = ‘root’;

[‘controlpassword’] = ‘ ’

Та дозволити root користувачу все робити, всі права на сервері

[‘allowNoPasswordRoor’] = true;

```
$cfg['Servers'][1]['auth_type']       = 'config';
$cfg['Servers'][1]['controuser']       = 'root';
$cfg['Servers'][1]['controlpassword']  = '';
$cfg['Servers'][1]['AllowRoot']       = true;
$cfg['Servers'][1]['nopassword']      = true;
$cfg['Servers'][1]['AllowNoPassword'] = true;
```

Перезапустивши сервер тепер не буде запитування логіну й будуть в наявності повні права.

Далі ми бачимо наші вкладки. Це наш повноцінний сервер. І далі можна створювати нову БД чи відкривати вже існуючу та виконувати інші операції.

10.1 Застосування DDL у роботі з БД

В SQL є підгрупа команд для зміни структури БД - **DDL** (Data Definition Language) або мова для визначення даних. Оператори DDL дають можливість виконувати наступне [6]:

- Створювати нові БД;
- Визначати структури для нових таблиць й створювати таблиці;
- Видаляти таблиці;
- Змінити для таблиці визначення;
- Визначати представлення даних;
- Забезпечувати умови безпеки в БД;
- Створювати індекси у таблицях;
- Визначати та керувати розташуванням даних.

Три команди SQL, що складають DDL:

- **CREATE** – дає можливість визначати та створювати об’єкт БД та інші БД;
- **DROP** - дозволяє видаляти існуючі об’єкти в БД;
- **ALTER** - дозволяє виконувати зміни визначень для об’єктів бази даних.

Щоб створити базу даних можна застосувать оператор **CREATE DATABASE**.

Синтаксис наступний:

CREATE DATABASE [IF NOT EXISTS] db_name1 [CHARACTER SET charset] [COLLATE collation];

db_name1 - ім'я, яке буде присвоєно створюваній БД;

IF NOT EXISTS - якщо не вказати цей параметр, то при спробі створення БД з вже існуючим ім'ям, виникне помилка;

CHARACTER SET та COLLATE - використовуються для завдання стандартного кодування таблиці й порядку сортування.

У випадку якщо для створення таблиці ці параметри не беруться, то порядок сортування й кодування в новій таблиці визначаються зі значень, що зазначені для всієї БД. Якщо заданий параметр CHARACTER SET, але не заданий параметр COLLATE, в цьому випадку використовується порядок сортування стандартний. Якщо заданий параметр COLLATE, а не заданий CHARACTER SET, в такому випадку кодування залежить від першої частини імені порядку сортування в COLLATE.

Кодування, що задається в CHARACTER SET, має підтримуватись сервером (latin1 або sjis), і порядок сортування має бути для поточного кодування допустимим.

Наступний **приклад** створює БД "my_db1":

CREATE DATABASE `my_db1`

або

CREATE DATABASE `my_db1` CHARACTER SET utf8 COLLATE utf8_general_ci;

Для того, щоб подивитися налаштування вже існуючої БД необхідно виконати оператор SHOW CREATE DATABASE.

При створенні нової БД в MySQL слід дотримуватися **деяких правил** щодо імені БД.

- Максимальна довжина імені в символах не повинна перевищувати 64;

- Можна застосовувати будь-які символи, які допускаються в імені каталогу, за винятком / (слешів) і . (крапок).

- Але не дозволено використовувати наступні символи ASCII (0) і ASCII (255).

Створення БД за допомогою PhpMyAdmin:

Для створення БД вибирають вкладку Databases і в формі Create new database вказують потрібну назву БД, а також її кодування.

На рисунку 10.1 зображено створення БД з назвою “my_db”:

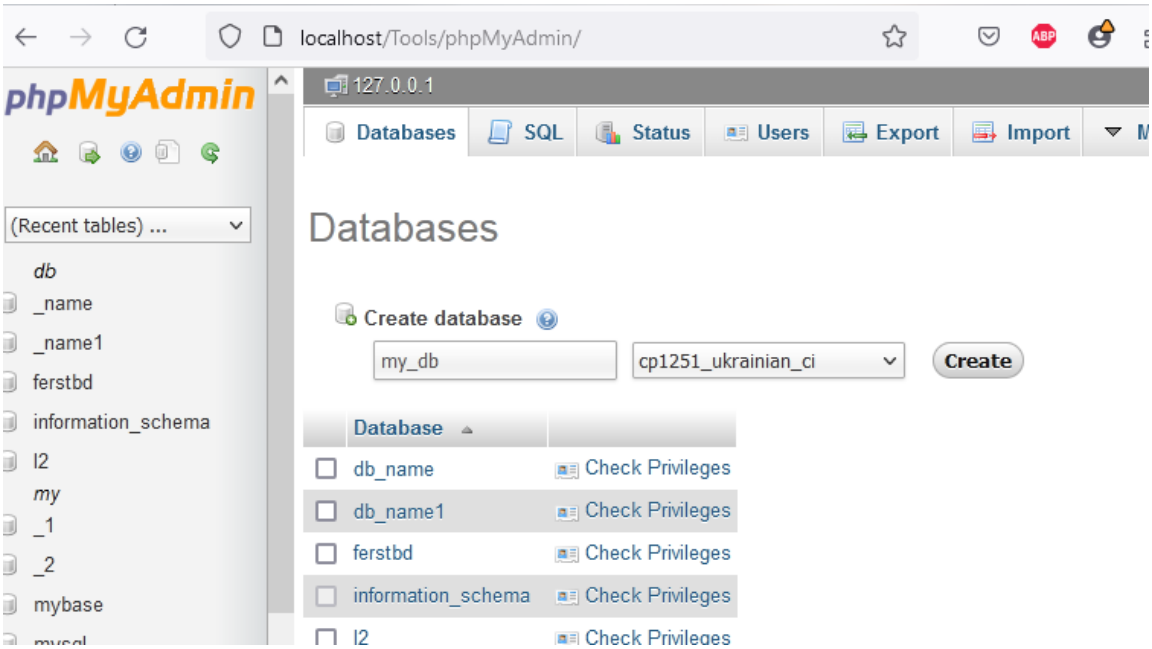


Рисунок 10.1 - Створення нової БД за допомогою phpMyAdmin

Створення таблиці в БД виконується командою **CREATE TABLE**. Синтаксис наступний:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name1
[(create_definition,...)]
[table_options] [select_statement]

tbl_name1 – вказується ім'я таблиці, яку буде створено в поточній БД. Починаючи з MySQL 3.22 введено можливість явно вказати БД, в якій буде створена нова таблиця, за допомогою синтаксису db_name.tbl_name1.

TEMPORARY - це параметр, що використовується для створення tbl_name1, тобто таблиці, що є тимчасовою протягом поточного сценарію. По закінченню роботи цього сценарію створена таблиця видаляється. Вказана можливість з'явилася в 3.23 версії MySQL. В MySQL 4.0.2 для створення тимчасових таблиць також потрібні привілеї створення тимчасових таблиць.

IF NOT EXISTS - якщо цей параметр зазначено та проводиться спроба створити таблицю, що вже існує у поточній БД, то таблиця створена не буде й повідомлення про помилку не з'явиться. В іншому випадку таблиця також створена не буде, але команда викличе помилку. Зауважується, що порівнюються тільки імена таблиць при створенні. Внутрішні структури не порівнюються.

create_definition - визначає внутрішню структуру таблиці, що створюється (назви, типи полів, ключі (якщо є), індекси тощо).

Можливі синтаксиси **create_definition:**

Col1_name type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT] [PRIMARY KEY] [reference_definition]

PRIMARY KEY (index_col_name1,...)

KEY [index_name] (index_col_name1,...)

INDEX [index_name] (index_col_name1,...)

UNIQUE [INDEX] [index_name] (index_col_name1,...)

FULLTEXT [INDEX] [index_name] (index_col_name,...)

[CONSTRAINT symbol] FOREIGN KEY [index_name] (index_col_name1,...) [reference_definition]

CHECK (expr)

Col1_name - позначає ім'я стовпця в таблиці, що сворюється;

type - задає тип даних для певного стовпця.

Можливі значення параметра type:

- **TINYINT[(length)] [UNSIGNED] [ZEROFILL]**

Означає малесеньке ціле число. Діапазон значень зі знаком від -128 до 127, а діапазон значень без знаку від 0 до 255.

- **SMALLINT[(length)] [UNSIGNED] [ZEROFILL]**

Означає маленьке число ціле. Діапазон значень зі знаком від -32768 до 32767, а діапазон значень без знаку від 0 до 65535.

- **MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]**

Означає середнього розміру число. Діапазон значення зі знаком від -8388608 до 8388607, а діапазон без знаку від 0 до 16777215.

- **INT[(length)] [UNSIGNED] [ZEROFILL]**

Означає нормального розміру ціле число. Діапазон значень зі знаком від -2147483648 до 2147483647, а діапазон без знаку від 0 до 4294967295.

- **INTEGER[(length)] [UNSIGNED] [ZEROFILL]**

Означає те саме що і для INT

- **BIGINT[(length)] [UNSIGNED] [ZEROFILL]**

Означає ціле число великого розміру. Діапазон зі знаком від -9223372036854775808 до +9223372036854775807, а діапазон без знаку від 0 до 18446744073709551615.

- **DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]**

Означає, з плаваючою крапкою подвійної точності нормального розміру число. Допустимий діапазон значень: від -1,7976931348623157 E +308 до -2,2250738585072014 E-308, 0, і від 2, 2250738585072014E-308 до +1,7976931348623157 E +308, крім того, у випадку що вказаний атрибут UNSIGNED, то негативні значення недопускаються.

- **REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]**

Означає те саме що і для DOUBLE

- **FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]**

Означає, з плаваючою крапкою мале число звичайної точності. Допустимі значення в діапазоні від -3,402823466 E +38 до -1,175494351 E-38, 0, і від 1,175494351 E-38 до 3,402823466 E +38.

- **DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]**

Означає неупаковане число, а саме, число зберігається у вигляді рядка й при цьому для кожного десяткового знака застосовується один символ, з плаваючою крапкою. Схоже веде себе як колонка CHAR, яка містить цифрове значення.

- **CHAR(length) [BINARY]**

Це означає рядок певної довжини, при зберіганні завжди дописується прогалинами в кінці рядка до потрібного розміру. Діапазон аргументу length встановлено від 0 до 255 символів. Проміжки, що в кінці видаляються, якщо здійснюється виведення значення. У випадку, якщо атрибут чутливості до регістру не вказано BINARY, тоді величини CHAR сортуються й порівнюються, незалежно від регістру, за встано встановленим алфавітом, за замовчуванням.

- **VARCHAR(length) [BINARY]**

Означає рядок змінної довжини

- **DATE**

Означає значення дати. Допустимий інтервал від '1000-01-01 'до '9999-12-31'. Формат, що MySQL виводить значення DATE: 'YYYY-MM-DD'. Є можливість встановлення значень в стовпець DATE, використовуючи й рядки, й числа.

- **TIME**

Означає час. Допустимі значення від '-838:59:59' до '838: 59:59 '. Формат, що MySQL виводить TIME значення 'HH: MM: SS', також допускається встановлення значень в стовпці TIME, використовуючи й рядки, й числа.

- **TIMESTAMP**

Означає часову мітку. Допустимі значення від '1970-01-01 00:00:00 'до деякого значення часу в 2037. Формат виведення значень в MySQL TIMESTAMP: YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD та YYMMDD.

- **DATETIME**

Означає комбінацію дати й часу. Допустимий інтервал від '1000-01-01 00:00:00 'до '9999-12-31 23:59:59', формат виведення значень у MySQL DATETIME: 'YYYY-MM-DD HH: MM: SS', також можливе

встановлення значень в стовпці DATETIME, використовуючи рядки та числа.

- **TINYBLOB**

Стовпець типу BLOB, що має максимальну довжину 255 (2 ^ 8 - 1) символів.

- **BLOB**

Теж стовпець типу BLOB, що має максимальну довжину 65535 (2^16 - 1) символів.

- **MEDIUMBLOB**

Теж стовпець типу BLOB, що має максимальну довжину 16777215 (2^24 - 1) символів.

- **LONGBLOB**

Теж стовпець типу BLOB, але має максимальну довжину 4294967295 (2 ^ 32 - 1) символів. Слід враховувати, що в даний час протокол передачі даних сервер / клієнт і таблиці MyISAM мають обмеження 16 Мб на переданий пакет / рядок таблиці, тому поки не можна використовувати цей тип даних у його повному діапазоні.

- **TINYTEXT**

Стовпець типу TEXT, що має максимальну довжину 255 (2 ^ 8 - 1) символів.

- **TEXT**

Теж стовпець типу TEXT, але має максимальну довжину 65535 (2^16 - 1) символів.

- **MEDIUMTEXT**

Теж стовпець типу TEXT, але має максимальну довжину 16777215 (2^24 - 1) символів.

- **LONGTEXT**

Теж стовпець типу TEXT, але має максимальну довжину 4294967295 (2 ^ 32 - 1) символів. Слід враховувати, що в даний час

протокол передачі даних сервер / клієнт й таблиці MyISAM мають обмеження 16 Мб на передавання пакету / рядку таблиці.

- **ENUM(value1, value2, ...)**

Перерахування. Перераховується тип даних. Тільки одне значення мже мати об'єкт рядка, що вибирається з певного списку величин 'value 1', 'value 2', ..., NULL або значення багу. Максимально список ENUM може утримувати 65535 величин різного виду.

- **SET**(value1,value2,value3,...)

Означає набір. Може містити об'єкт рядка 0 чи більше значень, де кожне значення повинно бути вибраним із заданого списку величин 'value 1', 'value 2', ... Список SET може містити 64 елементи максимум.

[NOT NULL | NULL] - вказує, чи може даний стовпець містити значення NULL чи ні. Якщо не вказано, то за замовчуванням приймається NULL (тобто може містити NULL);

[DEFAULT default_value] - задає значення за замовчуванням для даного стовпця. При вставці нового запису в таблицю командою INSERT якщо значення для поля col_name явно вказано не було, то встановлюється значення default_value;

[AUTO_INCREMENT] - При введенні нового запису в таблицю поле з цим атрибутом автоматично отримає числове значення, більше самого великого значення для цього поля в поточний момент часу на 1. Дана можливість зазвичай використовується для генерування унікальних ідентифікаторів рядків. Стовпець, для якого застосовується атрибут AUTO_INCREMENT, повинен мати цілочисельний тип. Так само цей стовпець повинен бути проіндексований. Відлік послідовності чисел для AUTO_INCREMENT починається з 1 і це можуть бути тільки додатні числа [6].

Приклад 10.1.1 створює таблицю користувачів з 3 полями, де перше поле - унікальний ідентифікатор запису, друге поле - ім'я користувача, а третє поле - його вік:

```
CREATE TABLE `users` (
`id` INT(11) NOT NULL AUTO_INCREMENT,
`name` CHAR(30) NOT NULL,
`age` SMALLINT(6) NOT NULL,
PRIMARY KEY(`id`))
```

Приклад 10.1.2 створює порожню таблицю STUDENTS:

```
… CREATE TABLE STUDENTS
(SNUM INTEGER,
SPRI CHAR (20),
SIMA CHAR (10),
SPB CHAR (15)); …
```

Послідовність вказання полів впливає на порядок розташування їх у таблиці при створенні.

Засіб ALTER TABLE це – команда, що змінює визначення існуючої таблиці, дозволяє додавання полів до існуючої таблиці (ADD), видалення поля (DROP) або змінювати їх (ALTER). Але вона не буде діяти, якщо необхідне перевизначення.

Приклад 10.1.3 [6]:

```
…ALTER TABLE STUDENTS
ADD COURS INTEGER,
SPEC CHAR (10);…
```

Здійснюється додавання до таблиці STUDENTS двох полів для курсу та спеціальності студентів.

Щоб виконати видалення таблиці, можна використати команду DROP TABLE. Відомо, що таблиця наповнена даними не може видалятись. Видаляються таблиці, якщо вони пусті.

Приклад 10.1.4:

```
DROP TABLE STUD;
```

Демонструється видалення пустої таблиці STUD.

Індекс – це корисна річ, що мають всі СКБД – список полів чи груп полів, що впорядковується в таблиці. Таблиця вже повинна бути створена. При створенні індексу у полі, БД запам’ятовує відповідний порядок всіх значень певного поля в області пам’яті. Зауважимо, що імена індексів не можна застосовувати для інших об’єктів БД, крім того, SQL сама вирішує, коли він потрібний для застосування та користується ним автоматично.

Приклад 10.1.5 розглядається створення індексу за полем, яке зберігає прізвище студента:

…CREATE INDEX SPRIIDX ON STUDENTS (SPRI);…

Якщо необхідне створення унікальних індексів беруть ключове слово UNIQUE для команди CREATE INDEX. Фактично такий індекс буде як первинний ключ у таблиці.

Для створення унікальних індексів можна зробити дії:

…CREATE UNIQUE INDEX

SNAMIDX ON STUDENTS (SNAM);…

Ця команда не буде виконуватись, якщо поле SNAM буде містити неунікальні значення [6].

Приклад 10.1.6 щоб видалити індекс, створений раніше за прізвищем студента, можна скористатись командою:

…DROP INDEX SPRIIDX

ON STUDENTS;…

Важливе поняття: обмеження даних – це певна частина визначень таблиці, що обмежує всі значення, що допускаються до введення в поля цієї таблиці. Можливе визначення обмежень або при створенні, або при зміні таблиці. Розглядаються наступні два основних види обмежень. Це обмеження для поля та обмеження для таблиці. Обмеження поля міститься у кінці фрагмента команди, яка оголошує його. Обмеження таблиці знаходиться в кінці оголошення імені таблиці. Такого виду

обмеженя NOT NULL – застосовують, щоб поле не мало порожніх значень.

Є можливість встановити унікальність в якості обмеження стовпця за допомогою ключового слова UNIQUE. А за допомогою обмеження PRIMARE KEY можливо обмежувати таблицю чи окремі стовпці таблиці. Синтаксис та визначення його унікальності такі ж які UNIQUE - первинні ключі не допускають NULL значень, тому перед такими обмеженнями необхідно оголосити NOT NULL.

Приклад 10.1.7:

…CREATE TABLE USP

(UNUM INTEGER NOT NULL PRIMARY KEY,

OCINKA INTEGER,

UDATE DATE,

SNUM INTEGER NOT NULL,

PNUM INTEGER NOT NULL,

UNIQUE (SNUM, PNUM));…

На рисунку 10.2 видно, як створити таблицю з прикладу.

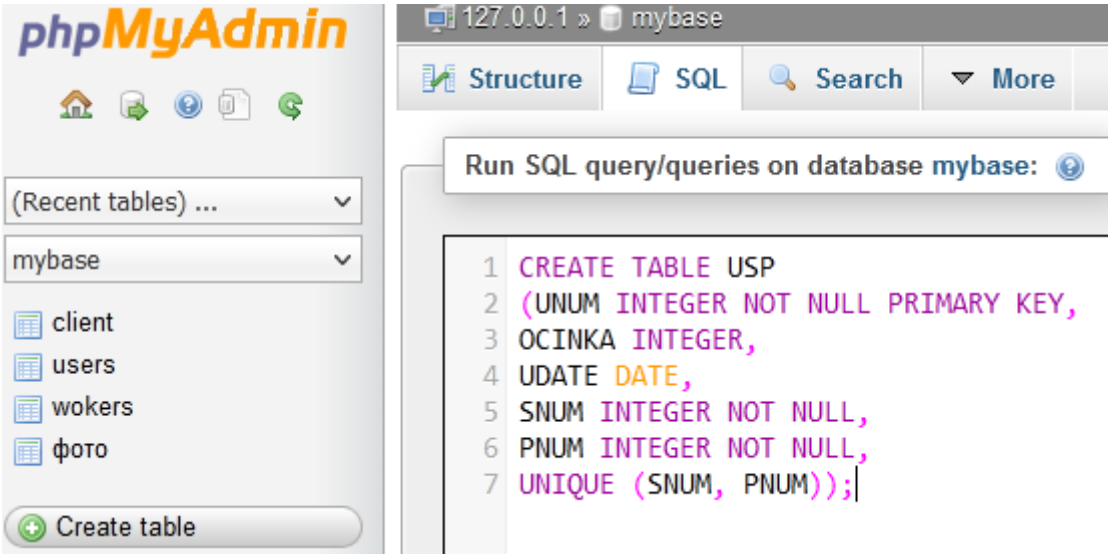


Рисунок 10.2 – Запит на створення таблиці USP

Таблиця, що успішно створюється, з’явиться у списку таблиць БД.

Якщо необхідно перейменувати поле таблиці використовуємо:

ALTER TABLE name_of_table CHANGE old_column_name new_column_name Тип(розмір);

ALTER TABLE STUDENTS CHANGE SPB SPOB VARCHAR(20);

Поле з ім’ям SPB зміниться на SPOB.

Максимальний розмір таблиць в MySQL не має обмежень. Розмір таблиці обмежений її типом. У загальному випадку типу MyISAM обмежений граничним розміром файлу в файловій системі операційної системи. На відміну від MyISAM, в InnoDB є значне обмеження на кількість стовпців, яке можна додати в одну таблицю [19].

Контрольні запитання до частини 10.1:

1. Що представляє собою SQL?
2. Які підгрупи команд складають SQL?
3. Які можливості DDL?
4. На яких трьох командах базується DDL?
5. Що виконує команда …CREATE TABLE…?
6. Що означає обмеження даних?
7. Якою командою SQL треба скористатись для видалення таблиці?
8. Яким чином здійснюється зміна визначення об’єкта БД?
9. За допомогою якої команди SQL можна створити БД?
10. Що виконує команда …ALTER TABLE…?
11. Що виконує команда …DROP TABLE…?
12. Які дії над таблицею треба виконати перед тим як видалити її?
13. Що називають індексом?
14. Яка команда допоможе у створенні таблиці?
15. В якому випадку не можна створити унікальний індекс в таблиці з даними?
16. Що означає обмеження NOT NULL у кінці оголошення імені таблиці?
17. Яким чином можливо перейменувати поле таблиці?

10.2 Застосування DML для роботи з БД

Про запити на зміну за допомогою SQL

Під час роботи з SQL важливо вміти користуватися засобами керування інформацією у таблицях. Запити на зміну використовуються для додавання, вилучення і поновлення записів, а також для збереження результуючого набору записів запиту у вигляді таблиці. Значення можливо вносити, модифікувати та видаляти з таблиць за допомогою трьох команд **DML (Мова маніпулювання даними) - INSERT, DELETE, та UPDATE.**

Введення інформації в БД

В SQL є команда модифікації INSERT, за допомогою якої вносяться записи в таблицю. Але зауважується, що ім'я таблиці, в яку відбувається вставка, повинно бути попередньо визначене, а кожне значення, що вставляється, повинно співпадати з типом даних поля, в яке воно вставляється.

Приклад 10.2.1 - для внесення запису в таблицю з викладачами TEACHERS, можливо скористатись виразом такого виду:

…INSERT INTO TEACHERS

VALUES ('4006', 'Федченко', 'Світлана', 'Григорівна', '2019-09-09');…

Можна вказати стовпці, в які будуть зберігатись введені значення.

Приклад 10.2.2:

…INSERT INTO TEACHERS (TDATE, TFAM, TIMA)

VALUES ('01-09-1999', 'Федченко', 'Світлана');…

Поля, які не вказуються в запиті автоматично встановлюються значеннями за замовчуванням.

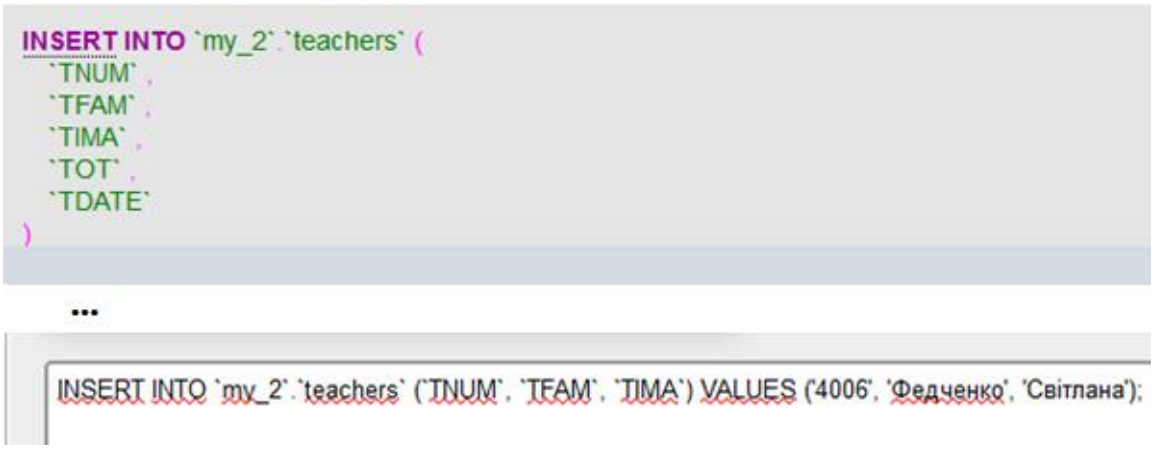


Рисунок 10.3 – Запит на додавання

На рисунку 10.3 отримали запит у режимі SQL, що додає інформацію до таблиці TEACHES з БД MY_2.

Також є можливість командою INSERT отримувати чи вибирати значення з однієї таблиці та поміщати їх в іншу разом з запитом.

Про видалення даних

Видаляти дані за рядками з таблиці виконується командою модифікації DELETE. Зауважується, що команда DELETE може працювати тільки з цілими записами таблиці, а не індивідуальні значення деяких полів.

Приклад 10.2.3. Якщо необхідно видалити весь вміст таблиці STUDENTS спробуйте виконати наступне [6]:

…DELETE FROM ‘STUDENTS’;…

Можливо видаляти тільки дані за виконання умов, тоді застосовують предікат. **Приклад 10.2.4** – ось буде видалення інформації, тільки якщо номер відповідає певному в наступній команді:

DELETE FROM ‘STUDENTS’

WHERE SNUM=3476;

Як предікат, в цьому випадку виступає номер студентського.

Наприклад, на рисунку 10.4 зображена таблиця STUDENTS.

Для видалення всіх даних з таблиці необхідно застосувати SQL-запит, що зображено на рисунку 10.5. У разі потреби, можна

підтвердити виконання дії. Після виконання запиту в повідомленні буде вказано кількість видалених рядків та час виконання операції.

SNUM	SFAM	SIMA	SOTCH
200	Іванов	Іван	*NULL*
201	Петров	Петро	Іванович
202	Сидоров	Семен	Семенович

Рисунок 10.4 – Приклад таблиці STUDENTS

Видалення даних з таблиці STUDENTS.

Рисунок 5 – Запит на видалення рядків з таблиці STUDENTS

Про оновлення даних

Змінювання всіх або будь-яких значень в таблиці реалізується за допомогою команди UPDATE. Там вказується ім’я таблиці, що береться та слово SET, яке визначає оновлення, яке відбудеться для потрібного поля таблиці.

Приклад 10.2.5:

UPDATE ‘ZP’

SET ‘STAVKA’=8000;

Приведе до зміни в таблиці ZP всіх ставок на «8000».

Якщо потрібно змінити тільки єдине значення, можна застосовувати предікат (рисунок 10.6). Ось наприклад:

…UPDATE ‘USP’

SET ‘OCINKA’=5;

WHERE PNUM=2003;…

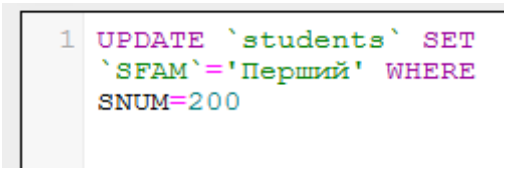


Рисунок 10.6 – Запит з предікатом

При необхідності, за допомогою UPDATE можна здійснити модифікацію даних з кількох полів. У SET можна визначити будь-яку кількість полів, що відокремлені комами.

Але здійснити модифікацію зразу кількох таблиць однією командою UPDATE не дозволяється.

Також можна брати вирази у рядку SET команди UPDATE, розташовуючи їх в списку для того поля, яке потребує змін.

Приклад 10.2.6:

…UPDATE ‘STUDENTS’

SET STIP=STIP*3;…

Командою збільшується значення стипендії в 3 рази.

У команди UPDATE є недоліки. До них відносять неможливість посилатися на таблицю, що вказується в будь-якому підзапиті з команди UPDATE. Наприклад, не можна одною командою виконувати таку дію, як модифікація оцінок студентів, у яких оцінки нижче середньої. При цьому необхідно виконувати один запит (запит на вибірку):

…SELECT AVG (OCINKA)

FROM USP;…

а тоді вже результат цього запиту застосувати для модифікації:

…UPDATE USP

SET OCINKA= OCINKA-1

WHERE OCINKA <4.2;…

Команда UPDATE є одною з ключових команд SQL, як команда , що керує змістом запису.

Контрольні запитання до частини 10.2:

1. Що представляє собою DML?

2. Які команди складають DML?

3. Перелічіть, які є запити на зміну?

4. Якою командою SQL можна створити запит на додавання записів у таблицю?

5. Що виконується командою UPDATE у запитах?

6. Чи можливо за допомогою UPDATE виконувати модифікацію даних з декількох полів?

7. Яка команда SQL застосовується в запитах на вилучення значень у таблиці?

8. Які дії виконуються командою INSERT INTO у запитах SQL?

9. Виконання яких дій стає можливим командою DELETE у запитах SQL?

10. Якою командою SQL можливо змінити значення в таблиці?

11. За допомогою якої команди можливо видалити значення з таблиці?

10.3 Застосування DQL для роботи з БД

Вибірка даних DQL (Data Query Language). Мова запитів DQL найбільш відома користувачам реляційної БД, незважаючи на те, що вона включає одну команду SELECT. Ця команда численними опціями і пропозиціями використовується для формування запитів до реляційної БД.

Синтаксис :

```
SELECT [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT]
[SQL_BUFFER_RESULT]
    [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
[HIGH_PRIORITY]
    [DISTINCT | DISTINCTROW | ALL]
  select_expression,...
  [INTO {OUTFILE | DUMPFILE} 'file_name1' export_options]
  [FROM table_references
    [WHERE where_definition]
    [GROUP BY {unsigned_integer | col_name | formula} [ASC | DESC],
...]
    [HAVING where_definition]
    [ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC],
...]
    [LIMIT [offset,] rows]
    [PROCEDURE procedure_name]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

Приклад 10.3.1:

```
…SELECT SNUM, SFAM, SIMA, SOTCH, STIP
FROM STUDENTS;…
```

SELECT- ключове слово, яке повідомляє БД, що ця команда є запитом.

SNUM, SFAM, SIMA, SOTCH, STIP – це список полів з таблиці, за якими вибираються дані запитом. Такий запит не впливає на інформацію в таблицях, він тільки показує дані на теперішній час.

FROM STUDENTS – ключове слово, яке супроводжується ім'ям таблиці, яку застосовуємо у якості джерела інформації.

Приклад виконання запиту зображено на рисунку 10.7.

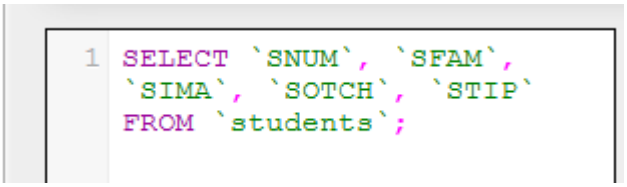


Рисунок 10.7 – Запит на вибірку даних

Результатом такого запиту на вибірку буде таблиця з даними.

Крапка з комою «;»-застосовується у всіх інтерактивних командах SQL для повідомлення БД, що команда готова для виконання.

«*» - необов'язкове скорочення застосовується, якщо необхідно отримати кожне поле таблиці замість переліку всіх полів.

Приклад 10.3.2:

SELECT * FROM STUDENTS;

Така команда призведе до того ж результату, що і попередня команда. Якщо необхідно вивести тільки деякі поля з таблиці, просто необхідно виключити з списку непотрібні поля.

SFAM	SIMA	SOTCH	STIP
Перший	цццц	уууу	100
Струмеленко	Сергій	Петрович	100
Зозуленко	Павел	Романович	1000

Рисунок 10.8 – Результат роботи «*» у запиті

Аргумент DISTINCT застосовують у випадку, якщо є необхідність уникнути дублювання значень у полі.

Наприклад, розглядається таблиця (рисунок 10.8).

Приклад 10.3.3:

…SELECT DISTINCT STIP FROM STUDENTS;…

Цю інструкцію застосовують для отримання списку результатів без дублікатів.

На рисунку 10.9 отримали результат виконання такого запиту.

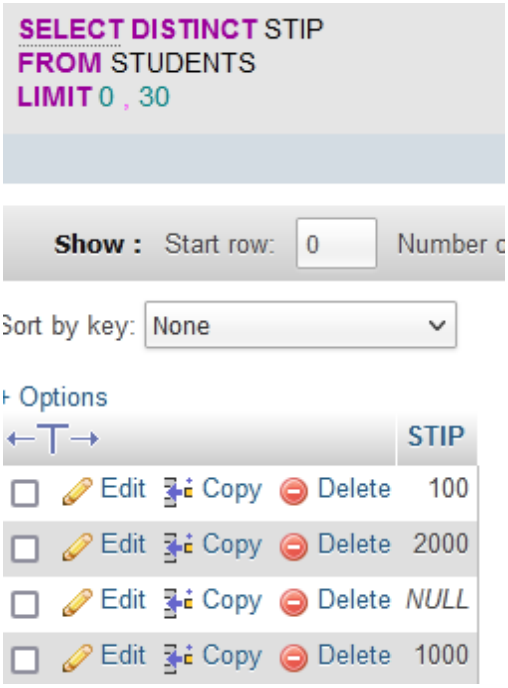


Рисунок 10.9 – Результат запиту на вибірку без дублікатів

Якщо застосувати ALL, то це буде мати протилежний ефект.

WHERE – інструкція команди SELECT, яка дозволяє встановлювати предікати, умова яких можливо буде виконуватись або не виконуватись для будь-якого запису таблиці. Команда отримує тільки ті записи з таблиці, для яких таке твердження буде вірним.

Приклад 10.3.4:

…SELECT SFAM, STIP

FROM STUDENTS

WHERE STIP=45.50;…

виведуться тільки ті прізвища та розмір стипендії студентів, для яких у полі STIP буде значення 45.50.

Пов’язуючи предікати з булевими операторами (AND, OR, NOT) та реляційними операторами (=, >, <, >=, <=, <>), набагато збільшується

можливість вибірки потрібних даних. Можливе також застосування спеціальних операторів IN, BETWEEN, LIKE, IS NULL (наприклад, рисунок 10.10).

```
1 SELECT SFAM, STIP
2 FROM STUDENTS
3 WHERE STIP>=100 AND
  SIMA='Сергій';
4
```

Результат:

SFAM	STIP
Струмеленко	100

Рисунок 10.10 – Результат запиту на вибірку з предікатами з булевими та реляційними операторами

За допомогою запитів можлива узагальнена групова обробка значень полів, що виконується за допомогою агрегатних функцій. Агрегатні функції отримують одиночне значення для всієї групи таблиці. В SQL допускають наступні функції: COUNT, AVG, SUM, MAX, MIN. Щоб знайти суму всієї отриманої стипендії в таблиці про студентів маємо **приклад 10.3.5**:

SELECT SUM(STIP)

FROM STUDENTS;

Функція COUNT застосовується для підрахунку кількості значень у стовпці. **Приклад 10.3.6**:

SELECT COUNT(SNUM)

FROM USP;

Якщо з COUNT застосовувати ALL (використовується за замовчанням) – не можна підрахувати значення NULL, але враховуються дублікати, а COUNT з «*» включає записи з NULL та дублікати.

Приклад запитів з агрегатними функціями та їх результати зображено на рисунку 10.11.

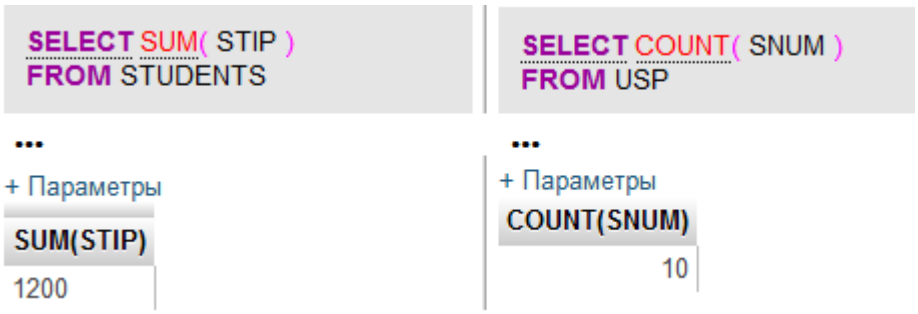


Рисунок 10.11 – Запити на вибірку з застосуванням агрегатних функцій та результати їх виконання

Для впорядкування виведення полів таблиць SQL має команду ORDER BY, що дозволяє сортувати виведення запиту згідно зі значеннями серед кількості полів. Якщо вказати декілька полів, то стовпці виведення впорядковуються один в середині іншого, при цьому треба визначити зростання (ASC) чи спадання (DESC) для кожного стовпця. **Приклад 10.3.7**:

…SELECT *

FROM STUDENTS

ORDER BY SFAM ASC;…

виведе табличку, що містить інформацію про студентів в алфавітному порядку прізвищ.

Однотабличні запити – це запити, які в якості джерела даних використовують тільки одну таблицю.

Бувають ситуації, що виникає необхідність вибору інформації з не одної таблиці. Це працює у запитах, що називають багатотабличними. Один із випадків такого вибору методом об’єднання результатів кількох запитів, що виконуються незалежно.

Об'єднання є механізм, який використовується для об'єднання таблиць всередині оператора. Використовуючи особливий синтаксис, об'єднують дані декількох табличок таким чином, що буде повертатися один результат, і це об'єднання буде "на льоту" пов'язувати потрібні рядки з кожної таблиці.

Об'єднання створюється СКБД в разі потреби й зберігається тільки при виконанні запиту.

Для такого об’єднання, що поєднує результати двох і більше запитів в єдиний набір стовпців та рядків застосовується UNION.

Приклад 10.3.8 - щоб вивести інформацію за учасниками деякого заходу ВНЗ можна застосувати запит:

```
…SELECT SFAM, SIMA, SOTCH
FROM STUDENTS
WHERE NOT(ISNULL(KONF))
UNION
SELECT TFAM, TIMA, TOTCH
FROM TEACHERS
WHERE NOT(ISNULL(KONF));…
```

Можна зауважити, що відсутність «;» після першої інструкції буде означати, що далі йде наступний запит. При необхідності з’єднання двох чи більше стовпців необхідно враховувати, щоб вони були сумісними для об’єднання. Також враховується сумісність типів та для кожного з запитів необхідне включення однакової кількості стовпців, в однаковому порядку. Агрегатні функції заборонено використовувати в в інструкції SELECT запиту на об’єднання. UNION автоматично виключає дублікати рядків з виведення. Дозволяється застосовувати константи та вирази у інструкції SELECT з UNION.

Результатом роботи запиту з прикладу 10.3.8 буде таблиця про учасників конференції, що ґрунтується на двох таблицях, це видно на рисунку 10.12.

SFAM	SIMA	SOTCH
Перший	ццццц	уууууу
Струмеленко	Сергій	Петрович
Терещенко	Анфіса	Петрівна
Степанов	Дмитро	Васильович
Федченко	Світлана	
Іванченко	Олена	Сергіївна
Кочерженко	Вікторія	

Рисунок 10.12 – Результат інструкції SELECT з UNION

Приклад 10.3.9:

…SELECT ‘Студент ‘, SFAM

FROM STUDENT

UNION

SELECT ‘Викладач ‘, TFAM

FROM TEACHER;

Результат запиту з UNION зображено на рисунку 10.13.

Студент	SFAM
Студент	Перший
Студент	Струмеленко
Студент	Зозуленко
Викладач	Терещенко
Викладач	Степанов
Викладач	Федченко
Викладач	Іванченко
Викладач	Кочерженко

Рисунок 10.13 – Результат запиту з UNION

Важливою особливістю запитів SQL є їх властивість визначати зв’язки міжтабличні й отримувати інформацію з них в термінах цих зв’язків. Ці операції називаються об’єднанням. У багатотабличних запитах, таблиці, які подаються у вигляді списку після інструкції FROM, через кому. Предикат запиту може посилатись на будь-який стовпець будь-якої таблиці, та може слугувати для зв’язку між ними. В

цьому випадку потрібно вказувати імена стовпців та таблиць, оскільки це може викликать неоднозначності в багатотабличному запиті. Допускається створювати запити, що об'єднують більше двох таблиць. **Приклад 10.3.10** - необхідно вивести список оцінок, які виставив той чи інший викладач:

…SELECT TEACHERS.TFAM, USP.OCENKA
FROM TEACHERS, PREDMET, USP
WHERE TEACHERS.TNUM=PREDMET.TNUM
AND PREDMET.PNUM=USP.PNUM;…

Такого роду запит може мати такий результат (рисунок 10.14):

```
SELECT TEACHERS.TFAM, USP.OCENKA
FROM TEACHERS, PREDMET, USP
WHERE TEACHERS.TNUM = PREDMET.TNUM
AND PREDMET.PNUM = USP.PNUM
LIMIT 0 , 30
```

…

TFAM	OCENKA
Терещенко	12
Терещенко	0
Терещенко	8
Степанов	9
Степанов	6
Федченко	5
Федченко	9
Федченко	10
Іванченко	7

Рисунок 10.14 – Можливий результат запиту

Є здатність запитів керувати іншими запитами. Це відбувається, якщо розмістити запит всередині іншого предиката, що використовує виведення внутрішнього запиту для визначення та встановлення вірного чи невірного значення предиката. **Приклад 10.3.11** – для отримання даних про студента з прізвищем Перший, якщо його номер невідомий. Спершу номер отримується з таблиці з даними про

студентів, а потім застосовується результат до таблиці про успішність подібним запитом:

…SELECT *

FROM USP

WHERE SNUM=

(SELECT SNUM

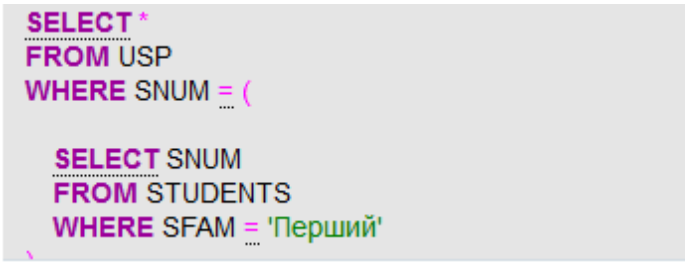
FROM STUDENTS

WHERE SFAM=’Перший’);…

Порядок наступний: SQL спершу аналізує внутрішній запит (його називають **підзапитом чи вкладеним запитом**) всередині речення WHERE. Підзапит повертає одне значення, й тип даних цього поля повинен співпадати з значенням, що порівнюється в предікаті.

Запит з підзапитом з прикладу 10.3.11 дає результат, що зображено на рисунку 10.15.

Спочатку виконується пошук номера студента у таблиці STUDENTS, це 1010, потім в таблиці успішності вибирається все потрібне для цього номера, а саме, на цей момент 3 рядки інформації.



…

UNUM	OCENKA	UDATE	SNUM	PNUM
1	12	2020-01-01	1010	1
3	0	2020-01-02	1010	1
5	8	2020-01-03	1010	1

Рисунок 10.15 – Запит з підзапитом [6]

Корисні спеціальні оператори: EXISTS, ALL, ANY, SOME – так працюють, що в якості аргументів беруть підзапити.

EXISTS - застосовують, для вказівки предикату, щоб виконувати виведення чи не виконувати виведення в підзапиті, при цьому EXISTS дає у результаті чи значення ІСТИНА, чи БРЕХНЯ. Наприклад, вирішується, чи треба отримувати дані з таблички про успішність, якщо в ній присутні незадовільні оцінки. Це досягається таким запитом:

```
…SELECT *
FROM USP
WHERE USP.OCENKA=1
AND EXISTS
(SELECT *
FROM USP
WHERE USP.OCENKA=1);…
```

SOME, ANY та ALL нагадують EXISTS, але відмінності в тому, що застосовуються спільно з операторами реляційними, аналогічно IN в підзапитах. Наприклад:

```
…SELECT *
FROM USP
WHERE OCENKA<>ALL
(SELECT OCENKA
FROM USP
WHERE UDATE=2024-05-10);…
```

Працює підзапит так, що вибере всі оцінки за дату 10.05.2024. Після всього основний запит виведе всі записи з оцінкою, яка не співпадає ні з одною з вибраних.

Схожий результат можна отримати здійснивши таке:

```
…SELECT *
FROM USP
WHERE NOT OCENKA=ANY
(SELECT OCENKA
FROM USP
```

WHERE UDATE=2024-05-10);…

Якщо запросити значення за допомогою команди ALL, не рівні набору значень, то це буде однаковим, що визнати факт відсутності цього значення у наборі.

Про застосування GROUP BY

Для обчислення сумарних значень на основі даних з однієї або з кількох таблиць, можна використати оператор GROUP BY, що має наступний синтаксис:

GROUP BY {column1} [, ...]

Наприклад, наступний запит зв'язує дві таблиці, сортує їх по полю Customeri, для кожного значення Customeri створює один рядок у результуючому наборі даних й обчислює кількість значень поля Orderi для кожного значення Customeri:

SELECT Customers.Customeri, COUNT (Orders.Orderi) FROM Customers INNER JOIN Orders ON Customers.Customeri = Orders.Customeri GROUP BY Customers.Customeri

У наведеному вище прикладі запиту використано в операторі SELECT агрегатну функцію COUNT, що обчислює кількість значень.

Оператор HAVING

Оператор HAVING має призначення, подібне до оператора WHERE, але використовується з агрегатними даними. Наприклад:

SELECT Customers.Customeri, COUNT (Orders.Orderi) FROM Customers INNER JOIN Orders ON Customers.Customeri = Orders.Customeri GROUP BY Customers.Customeri HAVING COUNT(Orders.Orderi) >= 10

Цей запит аналогічний попередньому, але в результуючий набір даних включені тільки ті замовники, які розмістили десять або більше замовлень.

Контрольні запитання до частини 10.3:

1. Що означає DQL?

2. Якою командою в запитах SQL здійснюється вибірка даних?

3. Який значок в SQL-запиті допомагає вивести дані всіх полів з таблиці?

4. Який засіб запобігає дублюванню інформації в SQL-запитах?

5. За допомогою якого засобу у SQL-запитах в результаті зберігається дублювання рядків виведення?

6. Що виконується запитом з командою SELECT...FROM...?

7. Що буде виконуватись в запиті, якщо застосовується команда ORDER BY?

8. Яка інструкція команди SELECT, дозволяє встановлювати предікати?

9. Яка функція застосовується для підрахунку кількості значень у стовпці таблиці в SQL-запиті?

10. Для чого застосовують UNION у запитах?

11. Які запити називають підзапитами?

12. Для чого використовують оператор GROUP BY?

13. Що за запити називають багатотабличними?

14. Чи відрізняється оператор HAVING від WHERE?

10.4 Застосування DCL для роботи з БД

Працюючи з базами даних, важливим моментом є керування доступом до інформації, контроль розподілу привілеїв між користувачами БД.

MySQL - це програмний продукт з відкритим вихідним кодом для керування БД, яке допомагає користувачам зберігати, організовувати й здійснювати доступ до інформації. Воно має безліч варіантів тонкої настройки прав доступу до таблиць і БД для кожного користувача.

Як тільки користувач починає застосовувати MySQL, йому надається ім’я користувача й пароль. Ці початкові облікові дані дають привілеї ‘root-доступу’. Користувач з правами доступу root має повний доступ до всіх БД і таблиць всередині цих баз [6].

DCL (Data Control Language або мова для керування даними) - мова управління привілеями. Команди управління даними дозволяють керувати доступом до інформації, яка складає БД. Як правило, використовується для створення об'єктів, пов'язаних з доступом до даних, а також служать для контролю над розподілом привілеїв між користувачами. Команда **Grant** - додає роль (привілей) користувачу; Команда **Revoke** - команда, яка видаляє роль.

Про синтаксис для команд GRANT і REVOKE:

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)] ...]
  ON {tbl_name | * | *.* | db_name.*}
  TO user_name [IDENTIFIED BY [PASSWORD] 'password']
    [, user_name [IDENTIFIED BY 'password'] ...]
  [REQUIRE
    [{SSL| X509}]
    [CIPHER cipher [AND]]
    [ISSUER issuer [AND]]
    [SUBJECT subject]]
```

[WITH [GRANT OPTION | MAX_QUERIES_PER_HOUR # |
MAX_UPDATES_PER_HOUR # |
MAX_CONNECTIONS_PER_HOUR #]]

REVOKE priv_type [(column_list)] [, priv_type [(column_list)] ...]
ON {tbl_name | * | *.* | db_name.*}
FROM user_name [, user_name ...]

У випадку, якщо потрібні більш жорсткі обмеження, існують способи створення користувачів з особливими наборами прав доступу.

Приклад 10.4.1: спочатку введення нового користувача з консолі MySQL [6]:

CREATE USER 'newuser2'@'localhost' IDENTIFIED BY 'password';

Перевірити користувачів можливо на вкладці Користувачі (або users), як показано на рисунку 10.16.

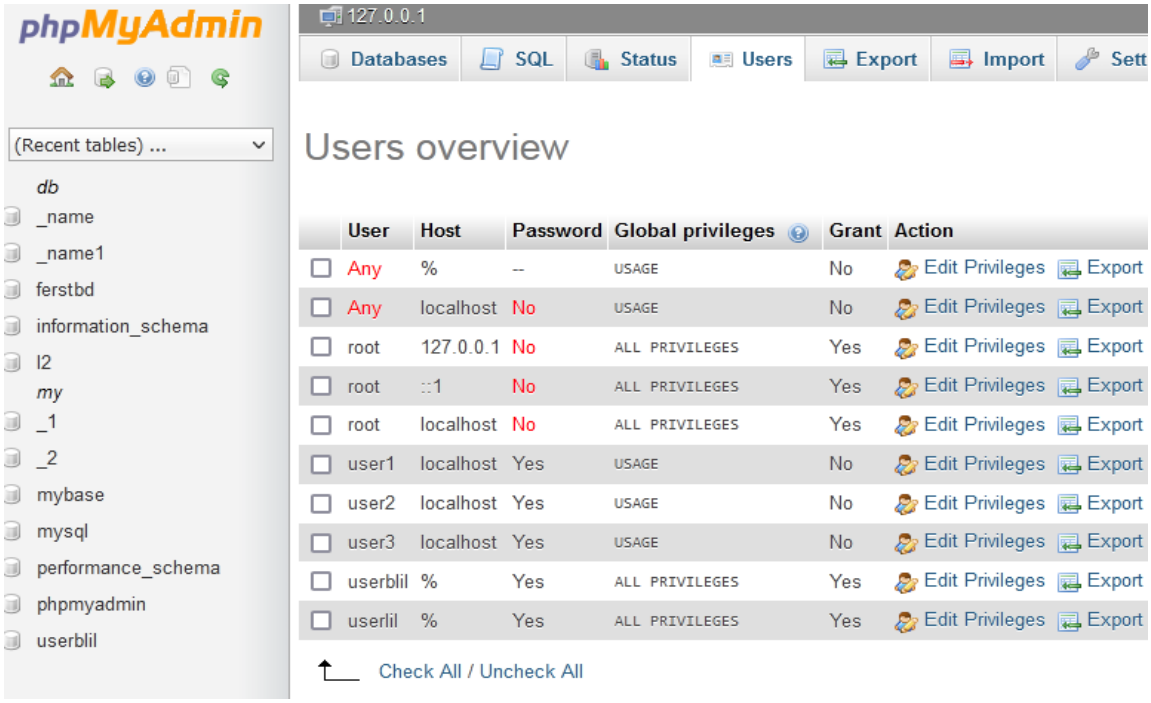


Рисунок 10.16 – Приклад отримання списку користувачів

Потім необхідно надати користувачеві доступ до інформації, яка йому буде потрібна:

GRANT ALL PRIVILEGES ON *.* TO 'newuser2'@'localhost';

Саме ця команда дозволяє користувачеві читати, редагувати, виконувати будь-які дії над усіма базами даних і таблицями. Після завершення налаштування прав доступу нових користувачів, переконайтеся, що ви оновили всі права доступу:FLUSH PRIVILEGES (оновлення всіх прав доступу).

Завдяки командам GRANT й REVOKE дозволяється системним адміністраторам реєструвати та додавати користувачів MySQL; на чотирьох рівнях привілеїв їм надаються права та можливо позбавляти прав.

Про рівні привілеїв

Є **глобальний рівень.** Привілеї глобального рівня зберігаються в табличці mysql.user та стосуються всіх баз даних на зазначеному сервері БД.

Наступний іде **рівень БД.** Цього рівня привілеї стосуються в зазначеній БД всіх табличок. Вони містяться в таких таблицях mysql.db й mysql.host.

Потім іде **рівень таблиці**. Такого рівня привілеї стосуються загалом всіх стовпчиків вказаної таблиці й розміщуються в таблиці з назвою mysql.tables_priv.

І нарешті, **рівень стовпчика.** Такого рівня привілеї містяться в табличці mysql.columns_priv та застосовуються до певних вказаних стовпчиків зазначеної таблиці.

Список деяких з можливих варіантів прав доступу, що надаються користувачам:

ALL PRIVILEGES - це дасть користувачеві MySQL повний доступ до заданої БД (якщо база даних не вказана, то до всіх);

CREATE - дозволяється виконувати створення нових таблиць або баз даних;

DROP - дозволяє видаляти таблиці або бази даних;

DELETE - дозволяється видаляти рядки з таблиць;

INSERT - дозволяється вносити рядки в таблицю;

SELECT - дозволяє використовувати команду Select для читання з баз даних;

UPDATE - дозволить редагувати рядки таблиць;

GRANT OPTION - дозволить призначати або видаляти права доступу для інших користувачів.

Для призначення прав конкретного користувача можна використовувати наступну схему:

GRANT [тип прав] ON [назва бази даних].[Назва таблиці] TO '[ім'я користувача]'@'localhost';

На рисунку 10.17 зображено запит, за допомогою якого, користувачу user1 надаються привілеї повного доступу до всіх таблиць БД my_2.

```
1 GRANT ALL PRIVILEGES ON my_2.* TO
  'user1'@'localhost';
```

Рисунок 10.17 – Запит на повний доступ до всіх таблиць для user1

Перевірити результат запиту можливо на вкладці користувачів (або users), знайшовши користувача необхідно вибрать Edit Previleges (Редагування привілеїв) і вдосконалитись у виконаних діях (рисунок 10.18), також можливо редагувати потрібне. Також можливо вибрать у вкладці Бази даних (DataBases) необхідну БД та вибрати Check privileges для перевірки привілеїв, наприклад як на рисунку 10.19, бачимо список всіх користувачів, що мають доступ до відповідної БД та їх привілеї.

Кожен раз, коли відбувається зміна прав доступу, необхідно застосовувати команду Flush Privileges.

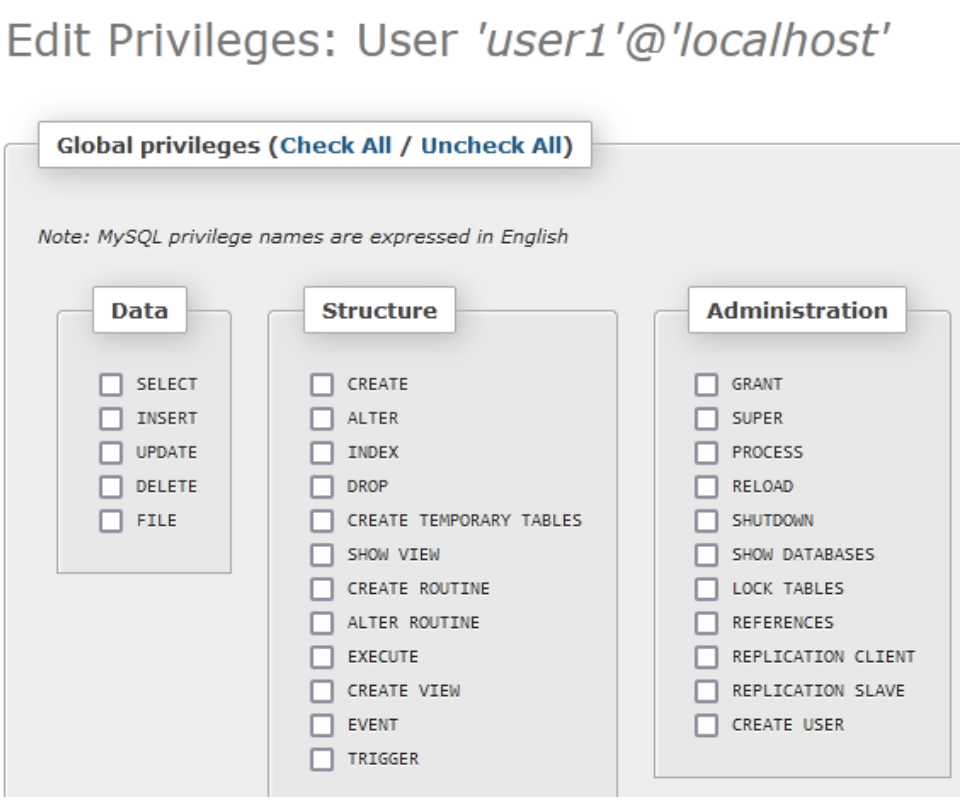


Рисунок 10.18 – Редагування привілеїв для user1

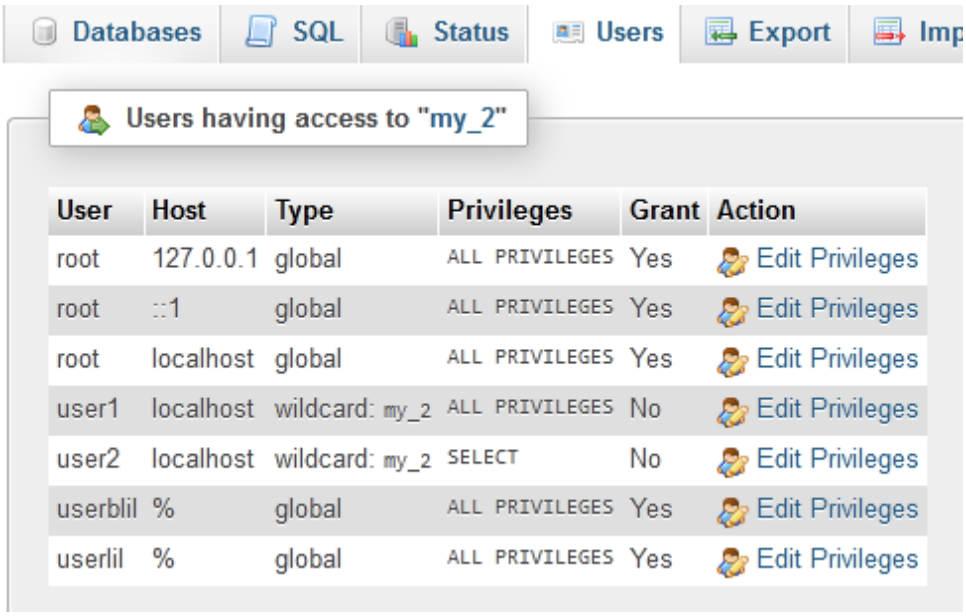


Рисунок 10.19 – Перелік всіх користувачів БД «my_2» та їх привілеїв

Обережно, не варто призначати привілеї ALTER звичайним користувачам. Це надає можливість зруйнувати систему привілеїв шляхом перейменування таблиць вказаному користувачу.

Позбавлення прав доступу ідентично їх призначенням:

REVOKE [тип прав] ON [назва бази даних].[назва таблиці] FROM ‘[им’я користувача]’@‘localhost’;

З використанням команди DROP відбувається видалення користувача:

Приклад 10.4.2:

DROP USER ‘demo’@‘localhost’;

Щоб відобразити список користувачів застосовують запит:

SELECT `user` FROM `mysql` . `user`;

Щоб відобразити привілеї користувача застосовують такий запит:

SHOW GRANTS FOR `username`@`hostname`

Для тестування облікового запису створеного користувача, разлогіньтесь за допомогою команди: quit і залогіниться знову, ввівши в термінаті наступну команду:

mysql -u [ім’я користувача]-p.

Контрольні запитання до частини 10.4:

1. Як визначається мова DCL?
2. Якою SQL командою SQL можна створити нового користувача БД?
3. Якою командою SQL надаються права користувачам БД?
4. Які рівні привілеїв користувачів БД існують в MySQL?
5. Які основні привілеї користувачів БД?
6. Яка з привілей надає користувачу повний доступ до БД?
7. Якою командою позбавляють прав користувачів?
8. Яким чином видаляють користувачів?
9. Які дії виконуються командою FLUSH PRIVILEGES?

10.5 Застосування TCL для роботи з БД

Частина SQL, TCL - мова керування транзакціями (Transaction Control Language.) оператори, що дозволяють контролювати операції транзакцій.

За замовчуванням MySQL працює в режимі autocommit. Це означає, що при виконанні поновлення даних MySQL буде відразу записувати оновлені дані на диск.

В MySQL можна відключити режим autocommit за допомогою наступної команди:

SET AUTOCOMMIT = 0

На рисунку 10.20 відображено, як це виглядає.

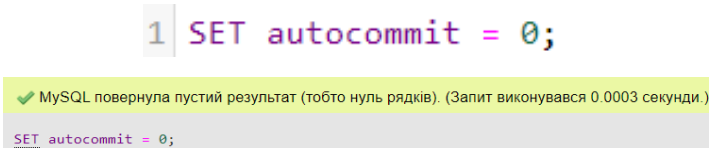


Рисунок 10.20 – відключення режиму AUTOCOMMIT

Нетранзакційні таблиці, наприклад, таблиці типу MyISAM (для кожної таблиці створюється окремий файл) або Memory.

Після цього необхідно застосувати команду COMMIT для запису змін на диск або команду ROLLBACK, яка дозволяє ігнорувати зміни, що відбулись з початку даної транзакції.

Для деяких операторів не можна виконати відкат або відміну за допомогою ROLLBACK. В їх число входять оператори мови визначення даних (Data Definition Language - DDL), які створюють і знищують бази даних, а також створюють, видаляють і змінюють таблиці. До них відносять такі команди:

CREATE INDEX…

DROP INDEX…

CREATE TABLE…

DROP TABLE…

TRUNCATE TABLE…

ALTER TABLE…

RENAME TABLE…

CREATE DATABASE…

DROP DATABASE…

ALTER DATABASE…

Крім того, існує низка операторів, які неявно завершують транзакцію, якби викликали оператора COMMIT:

ALTER TABLE…

BEGIN…

CREATE INDEX…

CREATE TABLE…

CREATE DATABASE…

DROP DATABASE…

DROP INDEX…

DROP TABLE…

DROP DATABASE…

LOAD MASTER DATA…

LOCK TABLES…

RENAME…

SET AUTOCOMMIT=1…

START TRANSACTION…

TRUNCATE TABLE…

Необхідно проектувати свої транзакції таким чином, щоб враховувати ці особливості. Якщо ввести такий оператор на початку транзакції, яка не може бути відмінена, і потім наступний оператор пізніше завершиться аварійно, повний ефект транзакції не може бути скасований оператором ROLLBACK.

Якщо необхідно переключитися з режиму AUTOCOMMIT тільки для виконання однієї послідовності команд, то для цього можна використовувати команду BEGIN або BEGIN WORK:

BEGIN;

SELECT @A: = SUM (salary) FROM table1 WHERE type = 1;

UPDATE table2 SET summmary = @ A WHERE type = 1;

COMMIT;

Відзначимо, що при використанні таблиць, які не підтримують транзакції, зміни будуть записані відразу ж, незалежно від статусу режиму autocommit.

При виконанні команди ROLLBACK після поновлення таблиці, що не підтримує транзакції, користувач отримає помилку (ER_WARNING_NOT_COMPLETE_ROLLBACK) у вигляді попередження. Всі таблиці, що підтримують транзакції, будуть перезаписані, але жодна таблиця, що не підтримує транзакції, не буде змінена.

При виконанні команд BEGIN або SET AUTOCOMMIT = 0 необхідно використовувати бінарний журнал MySQL для резервних копій замість більш старого журналу записи змін. Транзакції зберігаються в двійковому системному журналі як одна порція даних (перед операцією COMMIT), щоб гарантувати, що транзакції, за якими відбувається відкат, не заносяться.

Синтаксис команди SET TRANSACTION:

SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL

{READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE}

Встановлює рівень ізоляції транзакцій.

За замовчуванням рівень ізоляції встановлюється для подальшої (не перший) транзакції. При використанні ключового слова GLOBAL дана команда встановлює рівень ізоляції за замовчуванням глобально для всіх нових з'єднань, створених від цього моменту. Однак для того щоб виконати цю команду, необхідно привілей SUPER. При використанні ключового слова SESSION встановлюється рівень ізоляції

за замовчуванням для всіх майбутніх транзакцій, які виконуються в поточному з'єднанні.

Встановити глобальний рівень ізоляції за замовчуванням для утиліти mysqld можна за допомогою опції --transaction-isolation.

SAVEPOINT ідентифікатор

ROLLBACK TO SAVEPOINT ідентифікатор

Починаючи з версій MySQL 4.0.14 і 4.1.1, innoDB підтримує SQL-оператори SAVEPOINT і ROLLBACK TO SAVEPOINT.

Оператор SAVEPOINT встановлює іменовану точку початку транзакції з ім'ям ідентифікатор. Якщо поточна транзакція вже має точку збереження з таким ім'ям, стара точка видаляється і встановлюється нова.

Приклад застосування точки збереження:

```
…
START TRANSACTION;
SELECT total FROM accounts WHERE user_id = 5;
SAVEPOINT accounts_3;
UPDATE accounts SET total = total - 1000 WHERE user_id = 5;
ROLLBACK TO SAVEPOINT accounts_3;
…
```

Оператор ROLLBACK TO SAVEPOINT відміняє частину транзакції до іменованої точки збереження. Модифікації рядків, які виконувалися поточною транзакцією після цієї точки, скасовуються відкатом, однак InnoDB не знімає блокування рядків, які були встановлені в пам'яті після точки збереження. (Відзначимо, що для знову вставлених рядків інформація про блокування спирається на ідентифікатор транзакції, збережений в рядку, блокування не зберігається в пам'яті окремо. У цьому випадку блокування рядка знімається при скасуванні.) Точки збереження, встановлені в більш пізні моменти, ніж іменована точка, видаляються.

Всі точки збереження транзакцій не враховуються, якщо виконується оператор COMMIT або ROLLBACK без вказівки імені точки збереження.

LOCK TABLES блокує таблиці для поточного потоку сервера. UNLOCK TABLES знімає будь-які блокування, утримувані поточним потоком. Всі таблиці, заблоковані в поточному потоці, неявно розблокуються, коли потік виконує інший оператор LOCK TABLES або коли закривається з'єднання з сервером.

LOCK TABLES не є оператором, безпечним щодо транзакцій, і неявно завершує транзакцію перед спробою блокувати таблиці.

Контрольні запитання до частини 10.5:

1. Що представляє собою TCL?

2. Які команди входять до складу TCL?

3. Яким чином можна відключити режим autocommit?

4. Як почати транзакцію?

5. Що необхідно зробити, щоб підтвердити всі дії транзакції і 6. зберегти зміни у БД?

6. Що необхідно зробити, щоб відмінити всі дії транзакції?

7. Для чого призначена команда COMMIT?

8. Для чого призначена команда ROLLBACK?

9. Для чого призначена команда SET TRANSACTION?

10. Яку дію виконує оператор SAVEPOINT

10.6 Первинні та зовнішні ключі, обмеження, керування зв'язками між таблицями за допомогою SQL

Реляційні таблиці розробляються таким чином, що вся інформація розподіляється по безлічі таблиць, причому для даних кожного типу створюється окрема таблиця. Ці таблиці співвідносяться (зв'язуються) між собою через загальні значення, і таким чином є реляційними (відносними)) в реляційній конструкції [6].

Розподіл даних за таблицями забезпечує їх більш ефективне зберігання, спрощує маніпулювання даними та підвищує масштабованість.

Зовнішні ключі дозволяють встановити зв'язки між таблицями. Зовнішній ключ встановлюється для стовпців з залежної, підлеглої таблиці, і вказує на один із стовпців з головної таблиці. Як правило, зовнішній ключ вказує на первинний ключ з пов'язаної головної таблиці.

Первинні ключі є стовпці, значення яких унікально ідентифікують кожен рядок таблиці. Стовпці, які допускають відсутність значень, не можуть використовуватися в якості унікальних ідентифікаторів.

Було розроблено багато версій мови SQL, перш ніж вона стала настільки повноцінною й потужною. Багато з найбільш ефективних інструментів маніпуляції з даними засновані на таких методах, які забезпечуються за допомогою обмежень.

Реляційні бази даних зберігають дані в багатьох таблицях, кожна з яких містить дані, пов'язані з даними з інших таблиць. Для створення посилань з однієї таблиці на інші використовуються ключі (звідси термін цілісність на рівні посилань).

Розглянемо таблицю, що вміщує імена та адреси клієнтів магазина (таблиця 10.6.1).

Таблиця 10.6.1 – Таблиця Customers (Клієнти)

CustomerID (Ідентифікатор клієнта)	Name	Address	City
1	Юрій Михайлов	5,вул. Садова	м. Київ
2	Михайло Згама	6, вул. Квіткова	м. Харків
3	Валентин Шевченко	7, вул. Вишнева	м. Кропивницький

У таблиці є ім’я – Customers (Клієнти), декілька стовпців з різного роду даними, а також рядки (кортежі) в яких записано відомості про клієнтів.

Таблиця 10.6.2 – Таблиця Orders (Закази)

OrderID (ідентифікатор заказа)	CustomerID (ідентифікатор клієнта)	Amout(Сума)	Date(Дата)
1	3	30.25	02-01-2023
2	1	12.95	15-01-2024
3	2	74.09	15-01-2024
4	3	5.55	01-02-2023

Як правило БД складаються із декількох таблиць, для яких ключі служать сполучними ланками. Так у таблиці 10.6.2 Orders (Закази) розміщено дані про закази, створені клієнтами.

Щоб реляційна база даних працювала належним чином, необхідно упевнитися в тому, що дані в таблиці введені правильно. Наприклад, якщо в таблиці Orders (таблиця 10.6.2) зберігається інформація про замовлення, а в Order Items - його детальний опис, ви повинні бути

впевнені, що всі ідентифікатори замовлень, згадані в таблиці OrderItems, існують і в таблиці Orders. Аналогічно, кожен клієнт, згаданий в таблиці Orders, не повинен бути забутий і в таблиці Customers (таблиця 10.6.1).

Хоча можна проводити відповідні перевірки, перш ніж вводити нові рядки (виконуючи оператор SELECT для іншої таблиці, щоб впевнитися в тому, що потрібні значення правильні), краще уникати такої практики з наступних причин.

Якщо правила, що забезпечують цілісність бази даних, примусово здійснюються на клієнтському рівні, їх доведеться виконувати кожному клієнту (деякі з клієнтів напевно не захочуть цього робити).

Доведеться примусово ввести правила для виконання операцій UPDATE і DELETE. Виконання перевірок на клієнтській стороні - процес, який забирає багато часу. Змусити СКБД виконувати ці перевірки - метод набагато ефективніший.

Обмеження, це правила, які регламентують введення даних в базу даних і маніпуляцію ними.

СКБД примусово забезпечують цілісність на рівні посилань за рахунок обмежень, що накладаються на таблиці бази даних. Більшість обмежень вводиться в визначеннях таблиць (за допомогою операторів CREATE TABLE або ALTER TABLE).

Існує кілька типів обмежень, і кожна СКБД забезпечує свій власний рівень їх підтримки.

Про первинні (Primary) ключі

Первинний ключ визначають, як особливе обмеження, що застосовується для того, щоб вміст стовпців (чи наборів стовпців) був унікальним й ніколи не змінювався. Іншими словами, це стовпець (чи стовпці) таблиці, значення яких однозначно визначають кожен рядок таблиці. Це полегшує безпосереднє маніпулювання окремими рядками і

взаємодію з ними. Без первинних ключів було б дуже важко оновлювати або видаляти певні рядки, не зачіпаючи при цьому інші.

Будь-який з стовпців таблиці можливо призначати на роль первинного ключа, але тільки якщо він задовільняє наступним умовам [6].

- Не допускається однакове значення первинного ключа у декількох записах.

- Наявність значень первинного ключа - неодмінна (не повинно бути дозволено використання значень NULL).

- Модифікація чи оновлення у значеннях первинного ключа не може бути.

- Використання значень первинного ключа повторно ні за яких обставин не може бути. Якщо якийсь рядок видалено з таблиці, його первинний ключ не може бути призначений іншому рядку.

Про зовнішні (Foreign) ключі

Зовнішній ключ визначається, як стовпець однієї таблиці, значення якого збігаються зі значеннями стовпця, що є ключем (первинним) для іншої таблиці. Зовнішні ключі - дуже важлива частина механізму, що забезпечує посилальну цілісність даних. Для кращого розуміння того, що собою представляють зовнішні ключі, розглядається наступний приклад.

Таблиця Orders (таблиця 10.6.2) містить єдиний рядок для кожного замовлення, зафіксованого в БД. Дані, що стосуються клієнта заносяться до таблиці Customers (таблиця 10.6.1). Замовлення, що в таблиці Orders зв'язані з певними рядками в таблиці Customers за рахунок ідентифікатора клієнта. Ідентифікатор клієнта є ключем (первинним) в таблиці Customers. Для кожного клієнта існує унікальний ідентифікатор. Номер замовлення є ключем (первинним) в таблиці Orders; кожне замовлення має свій унікальний номер.

Значення в стовпці таблиці Orders, що містить ідентифікатори клієнтів, не обов'язково унікальні. Якщо клієнт зробив кілька замовлень, то рядків , що містять однаковий ідентифікатор, може бути декілька.

Завдяки зовнішнім ключам, примусово зберігається цілісність посилальних даних, вони можуть виконувати багато інших важливих функцій. Після того, як ключ Foreign визначено, СКБД не дозволить видаляти рядки, що зв'язані з рядками в інших таблицях. Як приклад, ви не зможете видалити інформацію про клієнта, який зробив замовлення. Є спосіб видалити інформацію про такого клієнта, але спершу треба вилучити замовленя, що відносяться до нього (для чого, в свою чергу, потрібно видалити інформацію про предмети цих замовлень). Оскільки в такому випадку потрібне методичне й цілеспрямоване вилучення, то Foreign ключі стануть у пригоді в запобіганні випадкового видалення даних в БД.

У деяких СКБД підтримується можливість, що отримала назву **каскадне видалення**. Якщо така механізм реалізований, можливо видаляти всі пов'язані з цим рядком дані при видаленні його з таблиці. Наприклад, якщо можливо каскадне видалення та ім'я клієнта видаляється з таблиці Customers, всі пов'язані з його замовленням рядки видаляються автоматично.

Додавання та видалення зовнішнього ключа.

Наприклад, необхідно створити дві таблиці, які потім будуть пов’язані. Це виконується за допомогою наступних запитів:

```
CREATE TABLE Customers
(
   Id INT PRIMARY KEY AUTO_INCREMENT,
   Age INT,
   FirstName VARCHAR(20) NOT NULL,
   LastName VARCHAR(20) NOT NULL
```

```
);
CREATE TABLE Orders
(
   Id INT PRIMARY KEY AUTO_INCREMENT,
   CustomerId INT,
   CreatedAt Date
);
```

Для додавання зовнішнього ключа до стовпчика CustomerId таблиці Orders виконується запит:

```
ALTER TABLE Orders
ADD FOREIGN KEY(CustomerId) REFERENCES Customers(Id);
```

Для додавання обмежень можливо вказати для них ім’я, застосовуючи оператор CONSTRAINT, після якого вказується ім’я обмеження:

```
ALTER TABLE Orders
ADD CONSTRAINT orders_customers_fk
FOREIGN KEY(CustomerId) REFERENCES Customers(Id);
```

В цьому випадку обмеження зовнішнього ключа називається orders_customers_fk. За цим іменем можливо видалити це обмеження:

```
ALTER TABLE Orders
DROP FOREIGN KEY orders_customers_fk;
```

Наприклад, для створення зовнішнього ключа PNUM під час зв’язування таблиць, що розглядались раніше PREDMET і USP застосовується запит, що на рисунку 10.21.

```
1 Alter table USP
2 ADD CONSTRAINT USP_Predmet_fk
3 FOREIGN KEY (PNUM) REFERENCES
  Predmet(PNUM);
4
```

Рисунок 10.21 – Приклад застосування запиту для створення зовнішнього ключа

Щоб візуально побачити зв'язок між таблицями можна вибрать Дизайнер з рядку меню (рисунок 10.22)

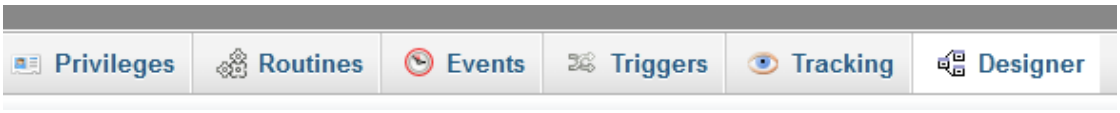


Рисунок 10.22 – Рядок меню з дизайнером в кінці

Результатом може бути схема, наприклад, як на рисунку 10.23.

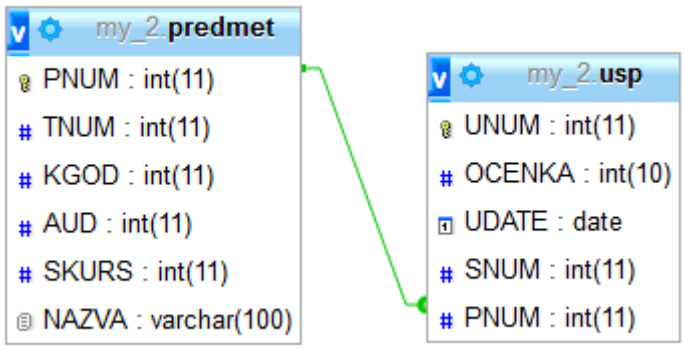


Рисунок 10.23 – Прикдад схеми даних

Щодо загального синтаксису для визначень FOREIGN KEY рівня таблиць: [CONSTRAINT ім'я_обмеження]

FOREIGN KEY (стовпчик1, стовпчик2, ... стовпчикN)

REFERENCES головна_таблиця (стовпчик_головної_таблиці1, стовпчик_головної_таблиці2, ... стовпчик_головної_таблиціN)

[ON DELETE дія]

[ON UPDATE дія]

При створенні обмежень для зовнішнього ключа після FOREIGN KEY вказується стовпець таблиці, що планується як зовнішній ключ.

Слідом за ключовим словом REFERENCES вписується ім'я таблиці, що пов’язана, а слідом в дужках зазначається ім'я стовпця, на який буде вказувати зовнішній ключ. Після висловлення REFERENCES йдуть вирази ON DELETE й ON UPDATE, що вказують на дію при видаленні та оновленні ряду, що з батьківської таблиці відповідно.

Вирази ON DELETE й ON UPDATE

Виразами ON DELETE та, аналогічно, ON UPDATE стає можливим визначати дії, що відбуваються відповідно під час видалення чи зміни рядка, що пов’язаний, з головної таблиці. Є деякі дії, що можливі в цих випадках [6]:

• CASCADE: видаляються або змінюються рядки з залежної таблиці автоматично при видаленні або зміні рядків, що пов'язані, в головній таблиці.

• SET NULL: при видаленні або оновленні рядка, що пов’язаний, з головної таблиці встановлюється для стовпця зовнішнього ключа значення NULL. (В такому випадку стовпчик зовнішнього ключа повинен підтримувати встановлення в NULL)

• RESTRICT: відміняє видалення або змінення рядків в основній таблиці, якщо наявні пов'язані рядки у залежній таблиці.

• NO ACTION: схоже з дією RESTRICT.

• SET DEFAULT: при видаленні рядку, що має зв’язок, з основної таблиці налаштовує для стовпця зовнішнього ключа значення за замовчуванням, що надається за допомогою атрибуту DEFAULT. Незважаючи на те, що ця опція доступна, проте двигун InnoDB не підтримує цей вислів.

Каскадне видалення - дозволяє з видаленням запису в батьківській таблиці автоматично видаляти відповідні йому записи в підлеглих таблицях. Для цього застосовується опція CASCADE [15]:

```
CREATE TABLE Order
(
```

Idn INT PRIMARY KEY AUTO_INCREMENT,

CustomerIdn INT,

CreatedAt Date,

FOREIGN KEY (CustomerIdn) REFERENCES Customers (Idn) ON DELETE CASCADE

);

Наприклад, видалення замовника з таблиці **Замовники** включає видалення всіх його заказів з таблиці **Закази**, а також видалення даних про склад заказів в таблиці **Подробиці заказів**. Тому що в цьому випадку видаляться й не виконані закази, каскадне видалення слід використовувати з обережністю.

Подібним чином працює й інструкція ON UPDATE CASCADE. **Каскадне відновлення -** забезпечує розповсюдження змін в головній таблиці на відповідні записи в підлеглій таблиці. При зміні значення PRIMARY KEY, автоматично зміниться значення, пов'язаного з ним ключа зовнішнього [15]. Однак оскільки первинні ключі дуже рідко змінюються, та й не рекомендується використовувати в якості первинних ключів стовпці зі змінними значеннями, то практично вираз ON UPDATE рідко використовується.

Встановлення в **NULL**

Якщо необхідно оголосити зовнішній ключ з опцією SET NULL, то потрібно, щоб стовпчик зовнішнього ключа мав допускати значення NULL [6]: CREATE TABLE Order

(

Idn INT PRIMARY KEY AUTO_INCREMENT,

CustomerIdn INT,

CreatedAt Date,

FOREIGN KEY (CustomerIdn) REFERENCES Customers (Idn) ON DELETE SET NULL

);

Для огляду прикладу створення БД, застосовуючи MySQL, перейдіть до додатку 1, для огляду інструкцій роботи з БД за допомогою Microsoft Accass, перейдіть до додатку 2.

Ознайомившись з засобами SQL, що дозволяють створювати зв’язки між таблицями, набуваючи навичок у використанні ключів для створення посилань з однієї таблиці на інші, засвоївши правила, які регламентують введення даних в БД і маніпуляцію ними користуючись обмеженнями стає більш реальним завдання навчитись сворювати БД, підтримувати роботу з нею та зберігати її цілісною.

Контрольні запитання до частини 10.6:

1. Що означає поняття первинного ключа?
2. Як визначити зовнішній ключ.
3. Які обмеження ви знаєте?
4. Яким чином можна створити первинний ключ?
5. Якими способами за допомогою SQL визначають зовнішні ключі?
6. Як можливо здійснити видалення зв’язку між таблицями?
7. Для чого використовують опцію CASCADE?
8. Що означає вираз ON UPDATE?
9. Що означає вираз ON DELETE?
10. Що означає поняття каскадне видалення?
11. Що означає поняття каскадне відновлення?

10.7 Типи з’єднань в MySQL

Часто при роботі з БД виникає необхідність у виконанні складних запитів на з’єднання даних з декількох таблиць. Серед інших способів виділяють такий, що виконує з'єднання за допомогою Join.

Join – це операція з'єднання таблиць SQL, яка здатна сполучати дві й більше таблиць в реляційній базі даних, утворюючи нову тимчасову таблицю (з'єднану таблицю).

В SQL згідно з ANSI-стандартом, налічують такі основні варіанти з'єднання: внутрішнє - INNER, зовнішнє – OUTER та перехресне - CROSS. Зовнішнє з'єднання поділяється на ліве - LEFT OUTER, праве - RIGHT OUTER та повне - FULL OUTER. Можливий особливий випадок з'єднання таблиць з собою, що також називається самоз'єднання (або self-join).

Візуально про кожен тип Join можна оглянути на рисунку 10.24

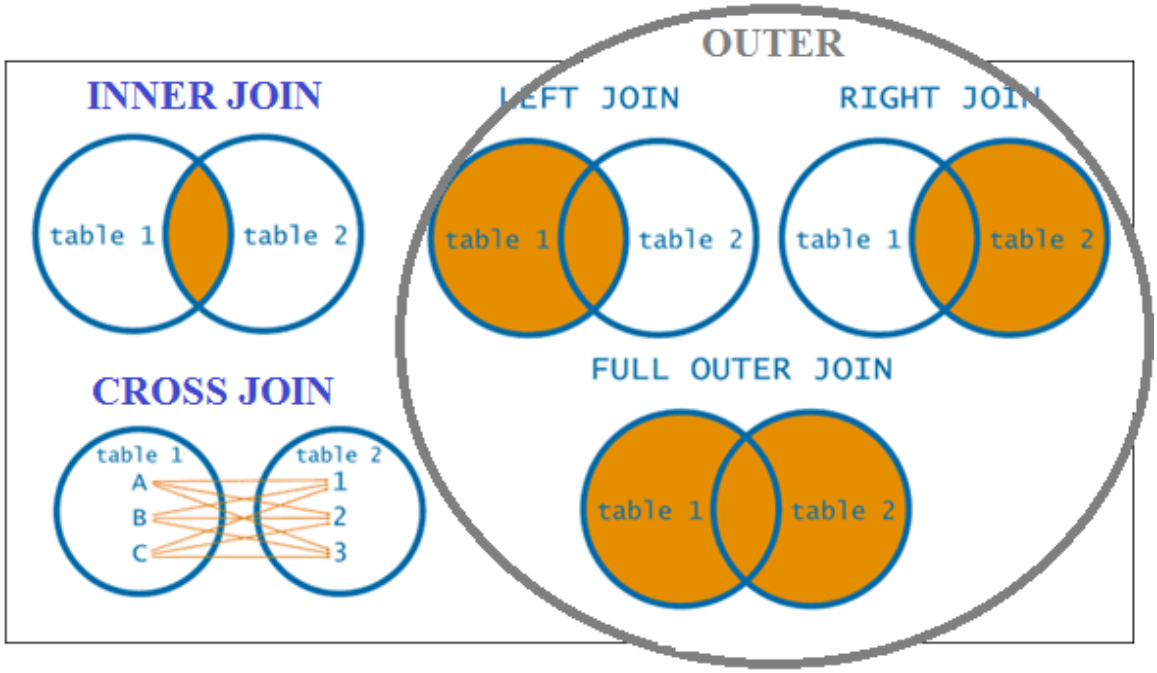


Рисунок 10.24 – Візуальне подання типів з'єднання

При внутрішньому з'єднанні з'єднуються записи двох таблиць (1 і 2) враховуючи предикат з'єднання. Виконується обчислення декартового добутку всіх записів таблиць. Як результат, всі записи

таблиці 1 буде поєднано з кожним із записів таблиці 2. Зауважимо, що після виконаного в таблиці отримаємо лише ті записи, які задовільняють предикат з'єднання.

Результат лівого зовнішнього з'єднання для таблиць 1 і 2 включає всі записи з лівої таблиці (1), навіть якщо умова з'єднання не знаходить відповідностей у правій таблиці (2). Це означає, що якщо порівняння не виявляє відповідних записів у таблиці 2, результат з'єднання все одно поверне рядки, але значення стовпців з таблиці 2 будуть порожніми. Інакше кажучи, ліве зовнішнє з'єднання включає всі записи з лівої таблиці, а також значення стовпців з правої таблиці або NULL, якщо немає відповідності за умовою з'єднання.

Якщо працює праве зовнішнє з'єднання для таблиць 1 і 2, то результат буде містити з правої таблиці - всі записи (2), навіть якщо в умові з'єднання немає збігів з кортежами лівої таблиці (1). Тобто, праве зовнішнє з'єднання повертає всі значення з правої таблиці й додає значення з лівої таблиці або NULL, якщо за предикатом з'єднання немає збігу.

Якщо розглядать повне зовнішнє з'єднання, то при цьому сполучаються результати правого та лівого зовнішніх з'єднань. Отримуємо в результаті таблицю, що містить всі записи з двох таблиць, і позначені NULL-значеннями у випадку відсутності збігів з кожного боку.

Оператор CROSS JOIN, ще називають оператором декартового з'єднання, він по'єднує дві таблиці. Через те, що оператор є симетричним [6], тому порядок вказування таблиць не є принциповим. В результаті отримується таблиця, що має у своєму заголовку об'єднання заголовків таблиць, що по'єднуються. А тіло таблиці результату складається так: кожен рядок одної таблиці по'єднується з рядком іншої таблиці, виконуючи в результаті всі можливі поєднання рядків з двох таблиць.

Наприклад, якщо для виконання роботи застосувати раніше створенні таблиці (orders, customers), що розглядаються на рисунках 10.25 та 10.26, можна за допомогою відповідних запитів виконати вибір необхідної інформації саме в тому вигляді, як потрібно.

←T→	order_id	customer_id	amount	date
	1	3	30.25	2023-01-02
	2	1	12.95	2024-01-15
	3	3	45.00	2020-01-16
	4	1	78.00	2020-01-02
	6	2	25.25	2020-01-07

Рисунок 10.25 – Таблиця orders

Наступна таблиця customers, що на рисунку 10.26 містить інформацію про клієнтів.

←T→	customer_id	name	address	city
	1	Юрій Михайлович	вул.Садова 5	м.Київ
	2	Михайло Згама	вул.Квіткова.6	м.Харків
	3	Валентин Шевченко	вул.Вишнева 7	м.Кропивницький

Рисунок 10.26 – Таблиця customers

Здійснюється пошук заказів з двох таблиць які зробив «Юрій Михайлович» за допомогою наступного запиту.

```
SELECT orders.customer_id, orders.amount, orders.date
FROM customers, orders
WHERE customers.name = 'Юрій Михайлович'
andcustomers.customer_id = orders.customer_id;
```

Результат виконання запиту розглянуто на рисунку 10.27.

customer_id	amount	date
1	12.95	2024-01-15
1	78.00	2020-01-02

Рисунок 10.27 – Результат виконання запиту на вибірку

При роботі з БД додаємо ще 3 таблиці для виконання необхідних запитів. Таблиці розглянуто на рисунках 10.28, 10.29, 10.30.

```
CREATE TABLE books
(
isbnchar(15) NOT NULL PRIMARY KEY,
authorchar(45),
titlechar (100),
pricefloat(4,2)
);
```

←T→	isbn	author	title	price
☐	5-8459-0046-8	Майкл Морган	JAVA2. Для Розробників	34.99
☐	6-8459-0046-8	Кристоф Негус	Linux. Для Розробників	25.99
☐	7-8459-0046-8	Марина Смоліна	Corel. Для Розробників	25.99
☐	8-8459-0046-x	Родерік Сміт	Мережі Linux	49.99

Рисунок 10.28 – Таблиця books

```
CREATE TABLE order_items (
orderidintunsigned NOT NULL,   isbnchar(15) NOT NULL,
quantitytinyintunsigned,
PRIMARY KEY(orderid, isbn)
);
```

←T→	orderid	isbn	quantity

←T→	orderid	isbn	quantity
	1	5-8459-0046-8	2
	2	5-8459-0046-8	1
	3	6-8459-0046-8	1
	3	7-8459-0046-8	1
	4	8-8459-0046-x	3

Рисунок 10.29 – Таблиця order_items

```
CREATE TABLE book_reviews (
isbnchar(13) NOT NULL PRIMARY KEY,
reviewtext);
```

←T→	Isbn	Review
	5-8459-0046-8	Книга Моргана краща по JAVA

Рисунок 10.30 - Таблиця book_reviews

Далі заповнимо таблиці значеннями за допомогою оператору INSERT (по одному запиту) або використовуючи графічний інтерфейс MySql:

```
INSERT INTO books (`isbn`, `author`, `title`, `price`)  VALUES (
("5-8459-0046-8","Майкл Морган","JAVA2. Для Розробників", 34.99),
("6-8459-0046-8","Кристоф Негус","Linux.Для Розробників",25.99),
("7-8459-0046-8","Марина Смоліна","Corel. Для Розробників",25.99),
   ("8-8459-0046-x","РодерікСміт","МережіLinux",49.99)
);
INSERT INTO order_items VALUES (
(1,"5-8459-0046-8",2),
(2,"6-8459-0046-8", 1),
```

(3,"6-8459-0046-8",1),

(3,"7-8459-0046-8", 1),

(4, "8-8459-0046-x",3)

);

…

INSERT INTO `book_reviews` (`isbn`, `review`) VALUES ('5-8459-0046-8', 'Книга Моргана краща по JAVA');

Виконаємо запит, щоб дізнатись хто робив хоча б один заказ з курсу JAVA. Алгоритм дій такий:

1. Для цього потрібно написати, що потрібно отримати в таблиці:

SELECT customers.name

2. Об’єднати усі 4 таблиці, так як заказані книги у нас в таблиці order, а ім’я замовника зв’язане з таблицею custmers.

3. Зроблені замовлення в таблиці Order_items, а опис книг для запиту пошуку по опису відповідно в таблиці books, тому виконаємо запит INNER JOIN замінивши записом через кому:

FROM customers, orders, order_items, books

4. Далі необхідно пройтися по зв’язкам таблиць та задати пошук :

WHERE customers.customer_id = orders.customer_id

AND orders.order_id = order_items.orderid

AND order_items.isbn = books.isbn

andbooks.title like '%Java%';

Загальний вигляд запиту буде таким:

SELECT customers.name

FROM customers, orders, order_items, books

WHERE customers.customer_id = orders.customer_id

AND orders.order_id = order_items.orderid

AND order_items.isbn = books.isbn

And books.title like '%Java%';

А результат роботи запиту надається на рисунку 10.28:

name	title
Валентин Шевченко	JAVA2. Для Розробників
Юрій Михайлович	JAVA2. Для Розробників
Леонід Макаров	Розробка Веб-додатків за допомогою PHP і MySQL та ...

Рисунок 10.28 – Результат роботи запиту

Як видно з рисунку 10.28 в усіх книгах в назві присутня назва «Java».

Далі виконаємо розповсюджений в MySQL тип поєднання даних «LEFT JOIN»

Попередні запити були на пошук кортежів, в яких була відповідність в таблицях. Але інколи буває що потрібно знайти і строчки наприклад які не зробили жодного заказу, або книги які ніхто не заказував. Саме простіше тут виконати лівостороннє з'єднання, при якому виконується пошук рядків за відповідною умовою з'єднання двох таблиць. У випадку, що у вказаній праворуч таблиці немає кортежу, що підходить, то до результату добавляється рядок з нульовими значеннями в відповідних стовпчиках.

Приклад:

…SELECT customers.customer_id, customers.name, orders.order_id

FROM customers LEFT JOIN orders

ONcustomers.customer_id = orders.customer_id;…

Далі віднайдемо тільки тих клієнтів, які нічого не замовили, це перевірка на значення NULL у полі первинного ключа правої таблиці (order_id), бо поля з реальними значеннями не повинні містити значення пусто.

```
SELECT customers.customer_id, customers.name
FROM customers LEFT JOIN orders
USING (customer_id)
WHERE orders.order_id is NULL;…
…
```

Використання псевдонімів

Для цього використовується конструкція AS (alias). Їх можна створити в самому початку запиту, а потім використовувати по необхідності. Часто псевдоніми використовують в якості коротких імен.

Розглянемо запит:

```
SELECT c.name
FROM customersas c, ordersas o, order_items asoi, booksas b
WHERE c.customer_id = o.customer_id
AND o.order_id = oi.orderid
AND oi.isbn = b.isbn
AND b.title LIKE '%Java%';
```

Такий запит більш короткий крім цього псевдоніми можна присвоювати стовпчикам. Псевдоніми таблиць необхідні коли потрібно з’єднати таблицю з самою собою (наприклад для пошуку в цій же таблиці рядків, у яких значення однакові). Так, якщо нам буде необхідно знайти клієнтів, які проживають в одному місті, можна одній і тій самій таблиці присвоїти різні псевдоніми й виконати пошук:

```
SELECT c1.name, c2.name, c1.city
FROM customersas c1, customersas c2
WHERE c1.city = c2.city
AND c1.name != c2.name
```

Функції агрегування MySQL

Нерідко буває, що необхідно при роботі з БД виконати якісь обчислення, наприклад, дізнатися, скільки рядків відноситься до

певного набору або середнє значення якого-небудь стовпця таблиці, скажімо середня сума заказу в грошовому еквіваленті. Для цього в MySQL використовуються функції **агрегування**. Дані функції можна застосовувати як до таблиці в цілому так і до груп даних всередині таблиці.

Функції агрегування, що найбільш часто використовуються на практиці наведено в таблиці 10.7.1.

Таблиця 10.7.1 – Функції агрегування в MySQL

AVG(стовпчик)	Середня величина значень у вказаному стовпці.
COUNT(елементи)	При вказуванні стовпця видається кількість не нульових значень. Якщо пере назвою стовпця помістити оператор DISTINCT, видається тільки кількість різних значень в стовпцю. Якщо вказати COUNT(*) то підрахунок строчок буде проведено незалежно від нульових значень.
MIN(стовпець)	Мінімальне значення у вказаному стовпцю.
MAX(стовпець)	Максимальне значення у вказаному стовпцю.
STD(стовпець)	Стандартне відхилення у вказаному стовпці.
STDDEV(стовпець)	Аналогічно попередньому.
SUM(стовпець)	Сума значень у вказаному стовпці.

Розглянемо декілька прикладів:

Приклад з функцією AVG:

```
USE books;
SELECT AVG(amount) FROM ORDERS;
```

Результат:

AVG(amount)
36.750000

Рисунок 10.29 – Результат з AVG

Для більш детального виду у запиті застосовується конструкція GROUP BY, яка виконується в наступному запиті:

…SELECT customer_id, AVG(amount)

FROM orders

GROUP BY customer_id;

…

Це дозволить отримати середню суму заказу по групам, наприклад за номером клієнта, щоб виявнити хто робить найбільш великі закази. Вказавши разом з конструкцією GROUP BY разом із функцією агрегування, зміниться поведінка функції й тепер ми отримаємо не середню суму всіх заказів, а інформацію за середньою сумою заказів усіх клієнтів (кожним customer_id)

Результат запиту надається на рисунку 10.30:

customer_id	AVG(amount)
1	45.000000
2	25.250000
3	37.500000
4	30.250000

Рисунок 10.30 – Результат запиту з GROUP BY

Об’єднання більше ніж двох таблиць методом JOIN

Дані, що необхідно обробити в БД можуть міститися не тільки в одній, двох, але й в трьох та більше таблицях. У цих випадках для аналізу даних використовуються ланцюжки таблиць, що пов’язані. Може бути така ситуація, що перша табличка міститиме певний кількісний показник, друга таблиця поєднана з першою й третьою зовнішніми ключами - дані перетинаються, але тільки третя таблиця містить умову, в залежності від якої може бути виведений потрібний результат з першої таблиці. За допомогою оператора SQL JOIN можна

об'єднувати велику кількість таблиць в одному запиті. У таких запитах за одним розділом JOIN йде інший, а кожен наступний JOIN приєднує до наступної таблиці таблицю, яка була другою в попередній ланці ланцюжка. Таким чином, щоб об’єднати більше ніж дві таблиці, треба дотримуватись синтаксису:

```
SELECT Назви_стовпців (1..N)
FROM ім’я_таблиці_1 JOIN ім’я_таблиці_2
ON Умова
 JOIN ім’я_таблиці_3
 ON Умова
 ...
        JOIN ім’я_таблиці_M
        ON Умова
```

Реляційні бази даних повинні відповідати вимогам цілісності даних і ненадлишковості, а тому дані можуть розподілятись на деяку кількість таблиць, але JOIN вправно виконує свою роботу й з великою кількістю таблиць. Саме завдяки JOIN РБД мають такий потужний функціонал, що дозволяє не тільки зберігати дані, але й їх, хоча б найпростіший, аналіз за допомогою запитів.

Контрольні питання до частини 10.7:

1. Які типи з’єднань існують в MySQL?
2. На які види з'єднань поділяється зовнішнє об’єднання даних?
3. Що означає застосування CROSS JOIN?
4. Яким чином можна організувать внутрішнє з'єднання даних?
5. Що означає термін самоз’єднання?
6. Що таке псевдоніми (alias)?
7. Які функції агрегування часто розглядаються?
8. Який синтаксис для JOIN при об’єднанні 3 і більше таблиць?

10.8 Застосування SQL при розробці backend частини з написанням frontend

При створені веб-додатків необхідно організовувати і роботу з БД.

За допомогою SQL можливо реалізовувати більшість задач, що стосуються БД.

Frontend - це публічна частина web-додатків (вебсайтів), з якою користувач може безпосередньо взаємодіяти та контактувати. У Frontend входить відображення функціональних завдань, інтерфейсу користувача, що виконуються на стороні клієнта, а також обробка користувальницьких запитів. Backend - це програмно-апаратна частина проекту. Frontend - є клієнтською стороною інтерфейсу користувача до програмно-апаратної частини проекту (бекенду).

Для прикладу розглядається проект, метою якого було створити систему - каталог товарів для магазину.

Проект написаний на мові Golang із застосуванням СКБД PostgreSQL та фреймворком Echo.

Дерево проекту, що отримали в результаті надається нижче:

```
.
├── app
│   ├── handlers
│   │   ├── init_handlers
│   │   │   └── init_handlers.go
│   │   ├── product_handler.go
│   ├── models
│   │   └── product.go
│   ├── repositories
│   │   └── product_repository.go
│   ├── routes
│   │   └── routes.go
│   └── services
│       └── product_service.go
├── cmd
│   └── server
│       └── server.go
├── config
```

```
|   └── config.go
├── config.yaml
├── db
|   └── db.go
├── Front
|   ├── index.html
|   └── js.js
├── go.mod
├── go.sum
├── main.go
├── migrations
|   ├── 1_create_products_table.down.sql
|   └── 1_create_products_table.up.sql
└──────────────────────────────────────
```

Алгоритм дій можна описать наступним чином.

Для початку створюється файл із залежностями для проекту, де будуть знаходитись всі необхідні бібліотеки для створення системи.

```
…
go mod init store
go mod tidy
…
```

Для роботи необхідно визначитись і встановити відповідну СКБД наприклад, як в цьому випадку PostgreSQL. Потім в середовищі Postgre створюється саме база даних, наприклад, із назвою "Store" (Магазин). Застосовується запит SQL на створення БД:

```
…
CREATE DATABASE Store;
…
```

При створенні бекенду необхідно прописать міграції для побудови певних таблиць.

У випадку, наприклад, якщо потрібно зберігати товари в магазині, застосовується запит на створення таблиці в БД:

```
…
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    brand TEXT NOT NULL,
```

```
    model TEXT NOT NULL,
    category_id INT NOT NULL,
    description TEXT NOT NULL,
    price BIGINT NOT NULL,
    sales BIGINT NOT NULL,
    avg_rate FLOAT NOT NULL,
    reviews BIGINT NOT NULL,
    quantity BIGINT NOT NULL
);
...
```

Наступним кроком буде створення файлу конфігурації для під'єднання до існуючої бази даних, для прикладу, створюємо config.yaml та заповнюємо дані для підключення до БД:

```
…
database:
  host: 127.0.0.1
  port: 5432
  user: User database
  password: UserPassword123
name: Store
…
```

На стороні бекенду також потрібно створити файл db.go для роботи з базою даних, тобто підключення до неї та за можливістю застосувати міграції на основі даних, які були прописані в файлі конфігурації config.yaml.

Розглядається приклад на мові програмування Golang:

```
…
var db *sqlx.DB

func init() {
      cfg := config.LoadConfig()

      connectionString := fmt.Sprintf("host=%s port=%d user=%s
password=%s dbname=%s sslmode=disable",
            cfg.DataBase.Host, cfg.DataBase.Port, cfg.DataBase.User,
cfg.DataBase.Password, cfg.DataBase.Name)
```

```
	database, err := sqlx.Connect("postgres", connectionString)
	if err != nil {
		log.Fatalf("[DB] failed to connect to database: %v", err)
	}
	db = database

}
```

…

При розробці бекенду будемо використовувати N-layer архітектуру, так як вона є простою у реалізації.

Для цього створюємо модель товару (файл з назвою product.go) згідно з таблицею, яку було прописано в базі даних.

Розглядається приклад на мові програмування Golang:

…

```
package models
type Product struct {
	ID          int     `json:"id,omitempty" db:"id,omitempty"`
	Brand       string  `json:"brand" db:"brand"`
	Model       string  `json:"model" db:"model"`
	CategoryId  int     `json:"category_id" db:"category_id"`
	Description string  `json:"description" db:"description"`
	Price       int64   `json:"price" db:"price"`
	Sales       int64   `json:"sales" db:"sales"`
	AvgRate     float32 `json:"avg_rate" db:"avg_rate"`
	Reviews     int64   `json:"reviews" db:"reviews"`
	Quantity    int64   `json:"quantity" db:"quantity"`
}
```

…

Далі потрібно створити CRUD репозиторій функцій до моделі продукту (файл з назвою product_repository.go). Наприклад, функція запису товару в базу даних.

Розглядається приклад на мові програмування Golang:

…

```
type IProductRepository interface {
	Create(user *models.Product) error
	GetAll() ([]*models.Product, error)
}
```

```
type ProductRepository struct {
	db *sqlx.DB
}

func NewProductRepository(db *sqlx.DB) *ProductRepository {
	return &ProductRepository{db: db}
}

func (r *ProductRepository) Create(product *models.Product) error {
	_, err := r.db.NamedExec("INSERT INTO (brand, model, category_id,
description, price, sales, avg_rate, reviews, quantity) VALUES (:brand,
:model, :category_id, :description, :price, :sales, :avg_rate, :reviews,
:quantity)", product)
	if err != nil {
		return err
	}
	return nil
}

func (r *ProductRepository) GetAll() ([]*models.Product, error) {
	var products []*models.Product
	err := r.db.Select(&products, "SELECT * FROM products")
	return products, err
}
...
```

Далі згідно з архітектурою проекту, для того щоб забезпечити безпеку роботи програми та сховати реалізацію роботи репозиторію, треба створити сервіс, який містить в собі імплементацію репозиторія (файл з назвою product_service.go).

Подається приклад на мові програмування Golang:

```
...
type IProductService interface {
	CreateProduct(user *models.Product) error
	GetAllProducts() ([]*models.Product, error)
}
type ProductService struct {
	ProductRepository *repositories.ProductRepository
}
```

```
func NewProductService(ProductRepository
*repositories.ProductRepository) *ProductService {
	return &ProductService{ProductRepository: ProductRepository}
}

func (s *ProductService) CreateProduct(product *models.Product) error {
	err := s.ProductRepository.Create(product)
	if err != nil {
		return err
	}
	return nil
}
func (s *ProductService) GetAllProducts() ([]*models.Product, error) {
	products, err := s.ProductRepository.GetAll()
	if err != nil {
		return nil, err
	}
	return products, nil

}
```

…

Далі для того, щоб до розробленої бекенд частини можна було звертатися з зовні, треба прописати хендлер, який використовує наш сервіс (файл з назвою product_handler.go).

Надається приклад на мові програмування Golang:

…

```
type ProductHandler struct {
	ProductService *services.ProductService
}

func NewProductHandler(ProductService *services.ProductService)
*ProductHandler {
	return &ProductHandler{ProductService: ProductService}
}

func (h *ProductHandler) CreateProduct(c echo.Context) error {
	product := &models.Product{}
	if err := c.Bind(product); err != nil {
		return c.JSON(http.StatusBadRequest, "Invalid request")
	}
```

```
	err := h.ProductService.CreateProduct(product)
	if err != nil {
		fmt.Println(err)
		return c.JSON(http.StatusInternalServerError, "Failed to
create product")
	}

	return c.JSON(http.StatusOK, product)
}
func (h *ProductHandler) GetAllProducts(c echo.Context) error {
	products, err := h.ProductService.GetAllProducts()
	if err != nil {
		log.Println(err)
		return c.JSON(http.StatusInternalServerError, "Failed to get
all products")
	}
	return c.JSON(http.StatusOK, products)
}
...
```

При маштабуванні проекту потрібно зібрати всі хендлери в одну структуру для того, щоб було зручно працювати з усіма дескрипторами (файл з назвою init_handlers.go).

Розглядається приклад на мові програмування Golang:

```
...
package init_handlers

import (
	"github.com/jmoiron/sqlx"
	"store/app/handlers"
	"store/app/repositories"
	"store/app/services"
)

type Handlers struct {

	ProductHandler	*handlers.ProductHandler
}

var handler *Handlers
```

```
func InitHandlers(db *sqlx.DB) {


	productRepository := repositories.NewProductRepository(db)
	productService := services.NewProductService(productRepository)
	productHandler := handlers.NewProductHandler(productService)

	handler = &Handlers{
		ProductHandler:	productHandler,
	}

}

func GetHandlers() *Handlers {
	return handler
}
```

Після написання хендлеру необхідно прив’язати його до певного роуту (файл з назвою routes.go).

Це подається в прикладі на мові програмування Golang:

…

```
func InitRoutes(e *echo.Echo, db *sqlx.DB) {
	init_handlers.InitHandlers(db)
	allHandlers := init_handlers.GetHandlers()
	e.POST("/newProduct", allHandlers.ProductHandler.CreateProduct)
	e.GET("/allProducts", allHandlers.ProductHandler.GetAllProducts)
}
```

…

Наступним етапом буде налаштування самого серверу (файл з назвою server.go).

Приклад на мові програмування Golang:

…

```
func main() {
	e := echo.New
	db := database.GetDb()
	routes.InitRoutes(e, db)
	err := e.Start(":8888")
	if err != nil {
```

```
            log.Printf("[SERVER] %s", err)
      }
}
```

…

Після цього необхідно розробити частину фронтенду системи. Для цього застосовуються наступні засоби: Bootstrap для пристойного оформлення та jQuery для можливості комфортно працювати з ajax запитами.

Спочатку створюється html документ, в якому прописується форма для заповнення продукту та при ній дві кнопки для збереження продукту та для отримання всіх існуючих записів продукту у базі даних:

…

```
<!doctype html>
<html lang="en" data-bs-theme="dark">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Project</title>
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.mi
n.css" rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
crossorigin="anonymous">
</head>
<body>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bund
le.min.js" integrity="sha384-
YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
crossorigin="anonymous"></script>
<script src="js.js"></script>
<div class="container text-center">
<div class="row align-items-start mt-5">
<div class="col">
</div>
<div class="col">
<h2>Add new product</h2>
```

```
<div class="form-floating mb-2 mt-3">
<input type="text" class="form-control" id="brand"
placeholder="name@example.com">
<label for="brand">Brand</label>
</div>
<div class="form-floating mb-2">
<input type="text" class="form-control" id="model"
placeholder="Password">
<label for="model">Model</label>
</div>
<div class="form-floating mb-2">
<input type="number" class="form-control" id="categoryId"
placeholder="Password">
<label for="categoryId">Category ID</label>
</div>
<div class="form-floating mb-2">
<input type="text" class="form-control" id="description"
placeholder="Password">
<label for="description">Description</label>
</div>
<div class="form-floating mb-2">
<input type="number" class="form-control" id="price"
placeholder="Password">
<label for="price">Price</label>
</div>
<div class="form-floating mb-2">
<input type="number" class="form-control" id="sales"
placeholder="Password">
<label for="Sales">Sales</label>
</div>
<div class="form-floating mb-2">
<input type="number" class="form-control" id="avgRate"
placeholder="Password">
<label for="avgRate">Avgarage rate</label>
</div>
<div class="form-floating mb-2">
<input type="number" class="form-control" id="reviews"
placeholder="Password">
<label for="reviews">Reviews</label>
</div>
<div class="form-floating mb-2">
```

```
<input type="number" class="form-control" id="quantity"
placeholder="Password">
<label for="quantity">Quantity</label>
</div>
<button onclick="save()" type="button" class="btn btn-outline-
primary">Save</button>
<button onclick="get()" type="button" class="btn btn-outline-
primary">Get all</button>
<hr>
<div id="response"></div>
</div>
<div class="col">
</div>
</div>
</div>

</body>
</html>
```

...

Далі створюється js.js файл з JavaScript, який підключається до html. Та проектується й описується логіка запитів при натисканні на кнопки, що створили.

...

```
var script = document.createElement('script');
script.src = 'https://code.jquery.com/jquery-3.7.1.js';
document.getElementsByTagName('head')[0].appendChild(script);

async function get() {
await $.ajax(
{
type: 'get',
url: 'http://localhost:8888/allProducts',
dataType: 'json',
success: function (res) {
responseDiv = document.getElementById("response")
responseDiv.innerHTML =""
for(let element of res.reverse()){
responseDiv.innerHTML += "ID: " + element.id +"<br>"
responseDiv.innerHTML += "Brand: " + element.brand +"<br>"
responseDiv.innerHTML += "Model: " + element.model +"<br>"
```

```
responseDiv.innerHTML += "Category ID: " + element.category_id +"<br>"
responseDiv.innerHTML += "Description: " + element.description +"<br>"
responseDiv.innerHTML += "Price: " + element.price +"<br>"
responseDiv.innerHTML += "Sales: " + element.sales +"<br>"
responseDiv.innerHTML += "Avg rate: " + element.avg_rate +"<br>"
responseDiv.innerHTML += "Reviews: " + element.reviews +"<br>"
responseDiv.innerHTML += "Quantity: " + element.quantity +"<br><hr>"
}
}
});
}

async function save(){
var data = {
brand : $('#brand').val(),
model: $('#model').val(),
category_id: Number($('#categoryId').val()),
description: $('#description').val(),
price: Number($('#price').val()),
sales: Number($('#sales').val()),
avg_rate: Number($('#avgRate').val()),
reviews: Number($('#reviews').val()),
quantity: Number($('#quantity').val())
}
console.log(data)
await $.ajax({
type: 'post',
url: 'http://localhost:8888/newProduct',
dataType: 'json',
data: JSON.stringify(data),
contentType: 'application/json',
success: function (res){
get()
}
});
}
```

…

В результаті роботи було отримано систему - каталог товарів для магазину. На рисунку 10.31 розглянуто створену форму для додавання товару. Користувач має можливість вводити відповідні дані у форму. В

нижній частині розташовані кнопки. При натисканні на Save відбувається зберігання запису у БД. Якщо користувач вибирає Get All, то відбувається виведення інформації, яка вже внесена в БД, що подається на рисунку 10.32.

Таким чином було розроблено простий наглядний варіант застосування мови SQL у веб-додатках. При необхідності можна систему відредагувати, додати більше можливостей та зробити зручнішою.

Знання та навички SQL необхідні при програмуванні не тільки веб-додатків.

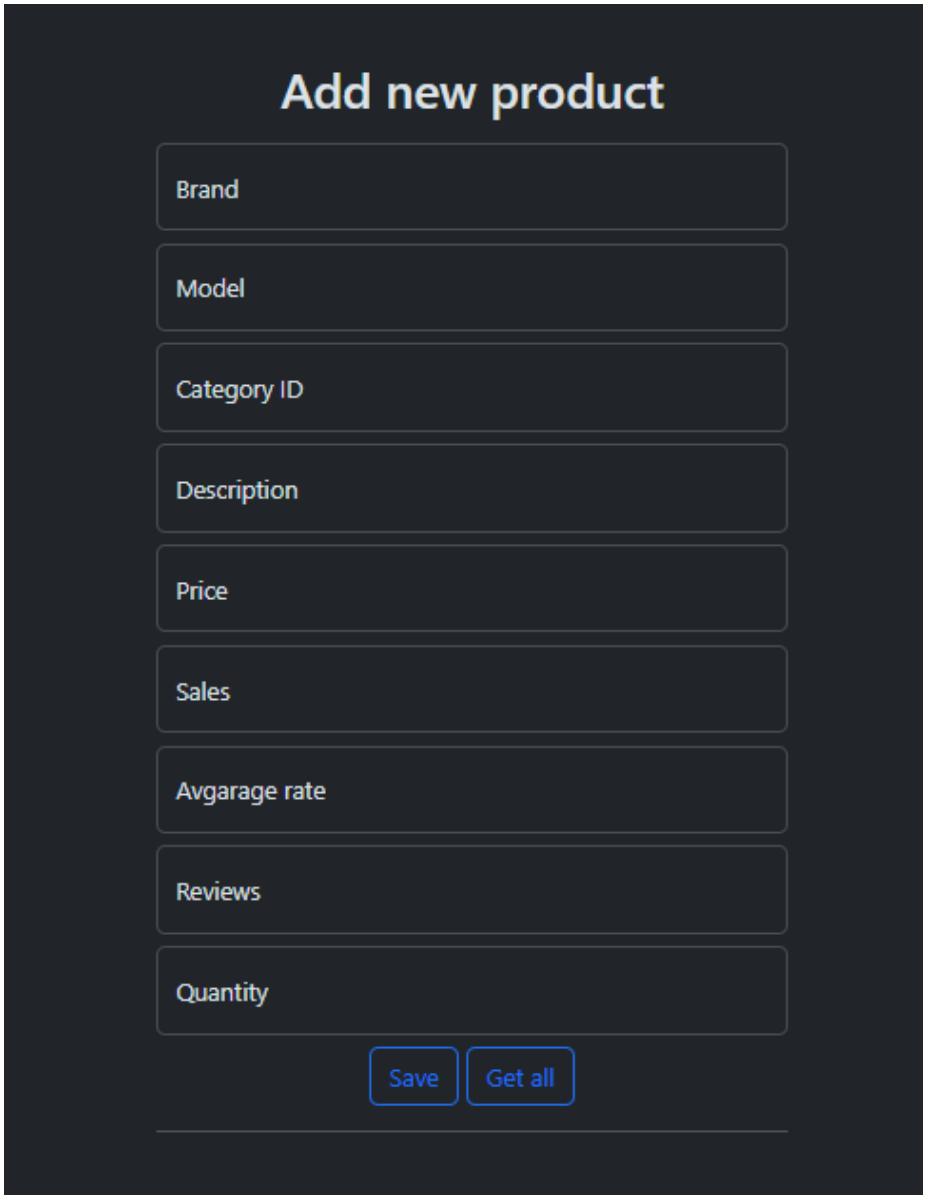


Рисунок 10.30 - Форма додавання продукту до БД

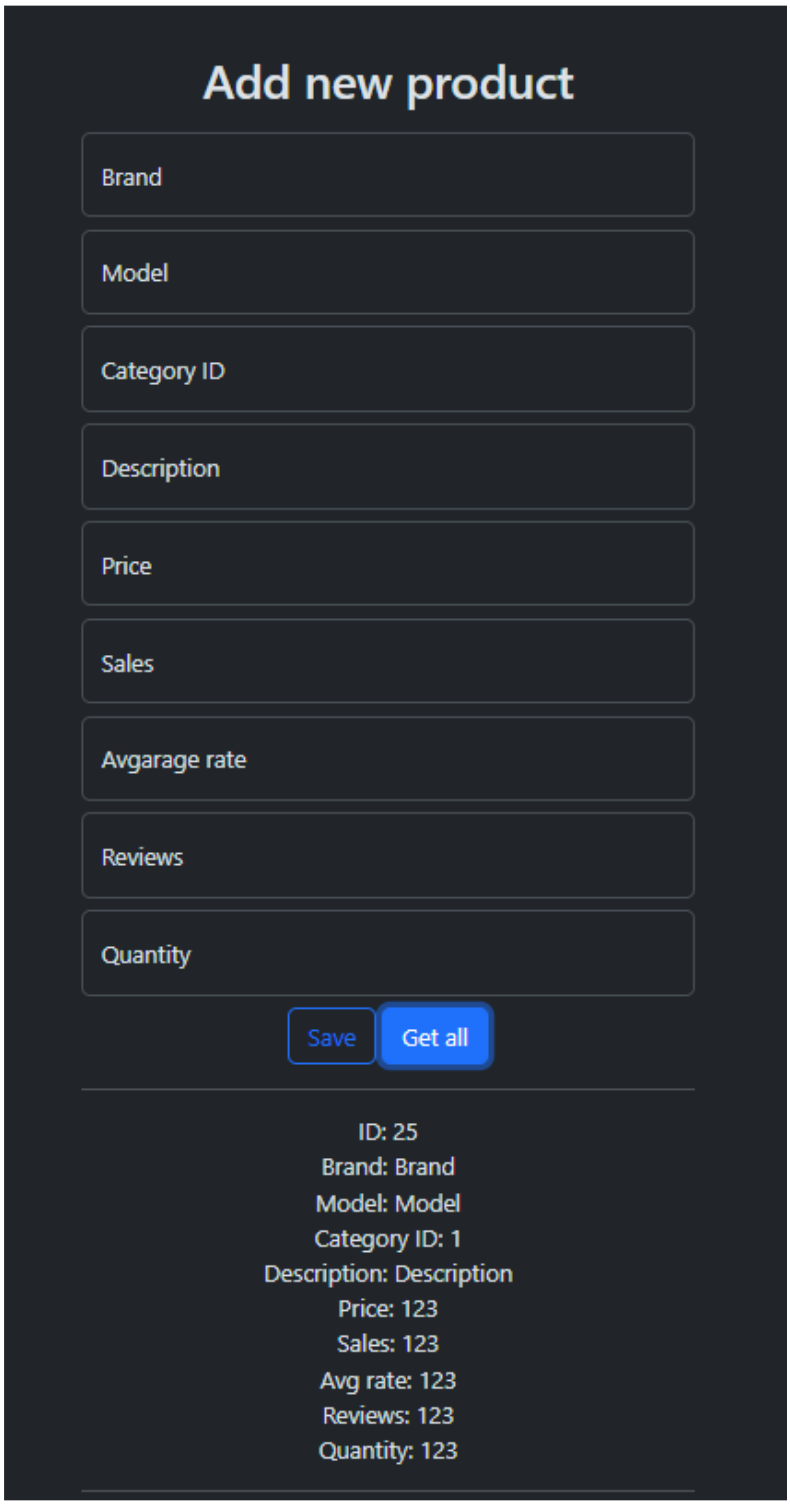


Рисунок 10.31 - Приклад знаходження інформації за наявними продуктами з БД при натисканні кнопки Get all

Підсумовуючи все , що розглядальсь , можна зробити висновки про необхідність взагалі навичок SQL.

Навички SQL допомагають програмістам БД підтримувати, створювати та отримувати інформацію з реляційних баз даних. Завдяки

їм стає можливим отримувати доступ, оновлювати, маніпулювати, вставляти та ефективно змінювати дані. Розглянемо 9 навичок SQL, які корисно мати для кар’єри програміста БД:

1. Вміння роботи з сервером (наприклад, MySQL чи Microsoft SQL).

2. Виконавська майстерність. План виконання - це візуальне подання того, як механізм БД виконує запити на доступ до даних. Важливо володіти достатніми навичками виконання, щоб зрозуміти ефективність запиту та усунути помилки з недостатньою продуктивністю чи нефункціональністю. Цей навик дозволяє покращити та підтримувати продуктивність запитів до БД.

3. Керування БД, управління доступом. Керуючи БД необхідно вміти створювати резервні копії даних, дозволяючи мати альтернативні варіанти, якщо початкова БД буде пошкоджена або втрачена. Є 4 варіанти: повне резервне копіювання, диференційне резервне копіювання, резервне копіювання кінцевого журналу або резервне копіювання журналу транзакцій. Кожен тип резервного копіювання відповідає конкретним потребам, і під час роботи з даними важливо знати, який із них є доречним. Захист БД, керувати доступом і виконувати конфіденційні завдання – необхідні скіли.

4. Навички та знання мов сценаріїв (наприклад, PHP, JavaScript, та Python тощо), які використовуються для розробки веб-сайтів. Хоча вивчення цього напряму не пов’язане з удосконаленням навичок SQL, це є корисним для розуміння, як все функціонує в цілому. Програмісти часто використовують одну мову для створення сайту та SQL для взаємодії та запиту даних на сайті.

5. SQL вміння об'єднувати, агрегувати, групувати. Розуміння пропозицій SQL Joins допомагає поєднання дани з декількох таблиць, полегшуючи аналіз кількох наборів даних, які потребують об’єднання а може й агрегування. Знайомство з пропозиціями Joins допомагає

пришвидшити процес об’єднання даних і виконання запитів порівняно з іншими методами, такими як підзапити.

6. Навички індексування. Індекси в базі даних мають вплив на швидкість виконання запиту. Але некоректна індексація може перешкоджати продуктивності.

7. Навички OLAP. Онлайн-аналітична обробка (OLAP) описує тип БД, яка аналізує дані набагато швидше та більш інноваційним способом, ніж інші бази даних. Це надає можливість для складних обчислень на основі даних.

8. Технічний аналіз даних SQL. SQL має значний вплив на сферу маркетингу та реклами. Ця мова може ефективно обробляти різні типи даних (аналіз, структурування даних, сортування й т.ін.).

9. Управління архітектурою бази даних Windows, Apache, MySQL і PHP (WAMP) - це стек програмного забезпечення, який при спільному використанні може пришвидшити дії, що стосуються створення і керування веб-сайтами [14].

Додаток 1

Приклад варіанту створення простої БД «Книги»

Планується, що БД повинна містити дві таблиці (зв’язані) відповідно до рисунку 1.1. В кожну таблицю необхідно додавать дані, Таблиця Customers – головна, Orders – підлегла.

За допомогою операторів SQL вносяться відповідні дані в кожну таблицю по 5 кортежів в кожну. Виконується зв'язок таблиць. Створюються відповідні запити різних типів для виконання робіт з БД.

Customers (Клієнти)

CustomerID (Ідентифікатор клієнта)	Name	Address	City
1	Юрій Михайлов	5,вул. Садова	м. Київ
2	Михайло Згама	6, вул. Квіткова	м. Харків
3	Валентин Шевченко	7, вул. Вишнева	м. Кропивницький

Orders (Закази)

OrderID (ідентифікатор заказа)	CustomerID (ідентифікатор клієнта)	Amout(Сума)	Date(Дата)
1	3	30.25	02-01-2020
2	1	12.95	15-01-2020
3	2	74.09	15-01-2020
4	3	5.55	01-02-2020

Рисунок 1.1 – Таблиці Customers та Orders для БД «Книги»

В кожному замовленні з таблиці Orders є вказівник на клієнта із таблиці Customers.

Частіше за все, бази даних складаються із деякої кількості таблиць, для яких ключі служать сполучними ланками зв’язків. Так, на рисунку 1.1 показані таблиці, що складають БД. В ній розміщено дані про закази книг, створені клієнтами.

Для реалізації планів роботи з БД завантажується спочатку OpenServer (можливо застосовувать інший сервер). Застосовується PhpMyAdmin.

Після завантаження в PhpMyAdmin вибирається вкладка SQL для виконання SQL запитів та створюється нова БД з назвою «books».

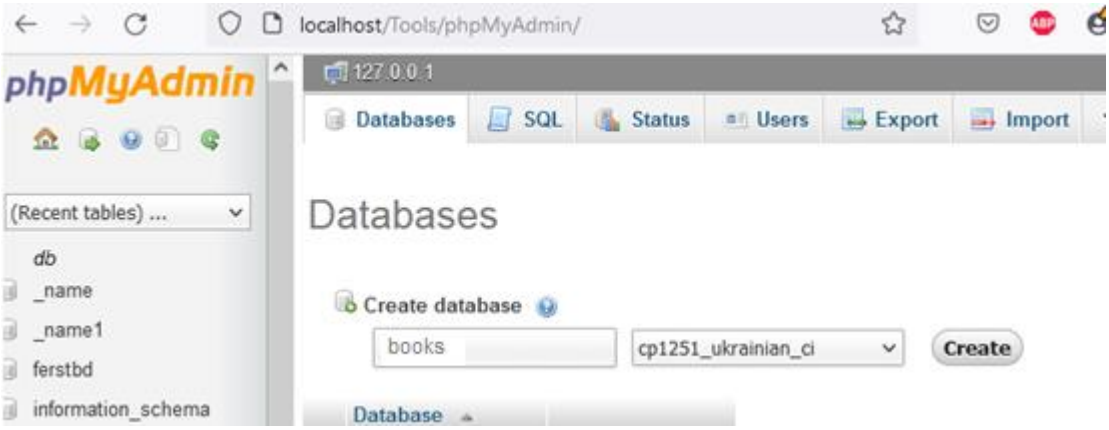


Рисунок 1.2 – Створення нової БД в PhpMyAdmin

За необхідністю виконуються відповідні SQL-запити.

Для видалення непотрібних можливих існуючих таблиць виконуються запити:

DROP TABLE IF EXISTS Customers;

DROP TABLE IF EXISTS Orders;

Та виконується запит для створення відповідної бази даних:

CREATE DATABASE books

Для переходу до відповідної БД застосовується запит:

USE books;

Після створення БД починається наповнення її об’єктами, а саме, деякими таблицями: «Customers» й «Orders» за допомогою мови DDL.

Згідно плану за допомогою запиту DDL у вкладці SQL виконується створення таблиці «Customers» з відповідними полями як показано на рисунку 1.1:

…CREATE TABLE customers(

customer_id INT unsigned NOT NULL AUTO_INCREMENT PRIMARY KEY,

name CHAR(45) NOT NULL,

address CHAR(55) NOT NULL,

city CHAR(25) NOT NULL

);…

Після виконання запиту система повинна повернути поле запиту та час виконання запиту.

Аналогічно також наступним запитом створюємо таблицю «Orders»:

…CREATE TABLE orders(
order_id INT unsigned NOT NULL AUTO_INCREMENT PRIMARY KEY,
customer_id INT unsigned NOT NULL,
amount float(6,2),
date date NOT NULL
);…

Після створення двох таблиць (між ними буде організовано зв'язок), одна з них буде «дочірня» (зберігає дані з батьківської таблиці) а перша буде називатись «батьківська». Це будуть відношення між даними створені за допомогою зовнішніх ключів. Так ми в нашій базі даних створюємо відношення між значеннями в таблиці «Orders» та значеннями в таблиці «Customers».

Зв'язок між таблицями виконується за ідентифікатором клієнта (Customer_id) за допомогою зовнішніх ключів (Foreign key). Синтаксис надається нижче відповідно до документації MySQL [6]:

[CONSTRAINT[*symbol*]] FOREIGN KEY [*index_name*] (*col_namex*,...)
REFERENCES *tbl_name* (*col_name1*,...)
[ON DELETE *reference_option*]
[ON UPDATE *reference_option*]

reference_option:
RESTRICT|CASCADE|SETNULL|NOACTION

Дані до таблиць краще не додавать поки не будуть виконані всі підготовчі дії. Виконуються запити на зміну таблиці для додавання FOREIGN KEY:

…ALTER TABLE orders

ADD FOREIGN KEY(customer_id) REFERENCES

customers(customer_id)

ON DELETE CASCADE;…

Після виконання відповідного запиту перейшовши в розділ Структура (Structure) відповідної таблиці потім – Зв’язки, можна побачити організацію зв’язків міжтабличних та відповідні встановлені обмеження (рисунок 1.3). Також можна здійснити необхідні зміни, додати FOREIGN KEY чи виправити існуючий.

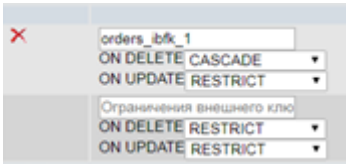


Рисунок 1.3 – Встановлені обмеження

Як результат ми отримали дві пов’язані таблиці. Після того, як виконані всі підготовчі роботи, всі об’єкти необхідні для роботи створені, можна додавать дані до таблиць у відповідній послідовності.

При внесенні даних в таблиці, спочатку повинна заповнюватися батьківська таблиця (клієнти), з причини, що дані в полі Customers_id таблиці Orders будуть залежати від даних батьківської таблиці. Так в таблиці Orders в полі запису Customer_Id буде не коректним і помилковим внесення даних ще не існуючого ID клієнта.

Наступним етапом є заповнення таблиць бази даних інформацією використовуючи DML оператор INSERT, що має наступний вигляд:

INSERT [INTO] таблиця [(стовпець1, стовпець2,….)]

VALUES (значення1, значення2, значення3,…).

Заповнюється таблиця клієнти за прикладом [6], що наведено нижче:

INSERT INTO `customers` (`customer_id`, `name`, `address`, `city`)

VALUES (NULL, 'Михайло Згама', 'вул. Квіткова.6', 'м.Харків');

INSERT INTO `customers` (`customer_id`, `name`, `address`, `city`)
VALUES (NULL, 'Валентин Шевченко', 'вул.Вишнева 7',
'м.Кропивницький');

Після заповнення таблиці «клієнти» приступаємо заповнювати таблицю Orders («закази»)

INSERT INTO `orders` (`order_id`, `customer_id`, `amount`, `date`)
VALUES (NULL, '3', '30', '2020-01-01'), (NULL, '1', '12', '2020-01-07');
INSERT INTO `orders` (`order_id`, `customer_id`, `amount`, `date`)
VALUES (NULL, '3', '45', '2020-01-16'), (NULL, '1', '78', '2020-01-02');

Для перевірки спробуємо додати в таблицю «закази» дані з неіснуючим ідентифікатором клієнта (order_id = 5)

INSERT INTO `orders` (`order_id`, `customer_id`, `amount`, `date`) VALUES (NULL, '5', '45', '2024-01-16').

Звичайно, система видасть повідомлення про помилку, тому що клієнта з таким ідентифікатором, поки що, не існує, й відповідний запит ми зможемо виконати тільки при появі клієнта з Custmer_id = 5.

Такий підхід дозволяє зберегти цілісність даних в таблицях БД. Це добре видно при роботі в графічному інтерфейсі, коли виконується додавання даних, система автоматично пропонує можливі варіанти для заповнення таблиці «Закази» поле «customer_id» рисунок 1.4. Як зображено на рисунку варіантів крім тих що є в таблиці «Клієнти» система не пропонує.

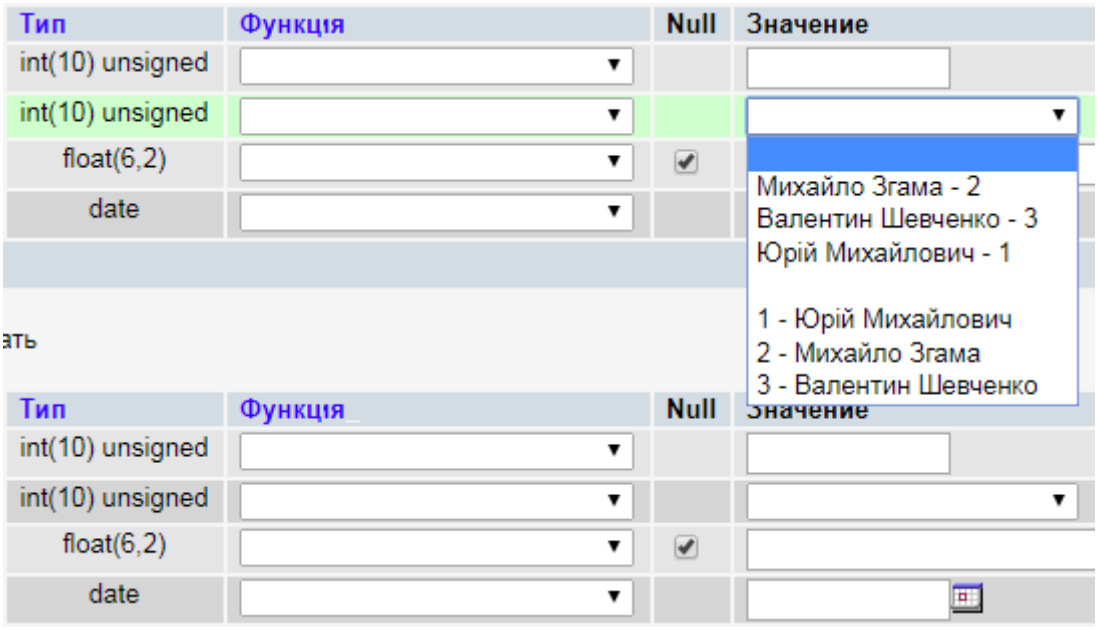


Рисунок 1.4 – Введення даних у таблиці, що пов’язані за допомогою графічного інтерфейсу

Таким чином, було наповнено таблиці необхідними даними. Передбачивши всі можливі варіанти та обмеження проста БД була побудована з виконанням необхідних умов. За необхідності її можна розширять та доповнювань іншими об’єктами та даними.

Додаток 2

Робота з базами даних за допомогою Microsoft Access

Програма Microsoft Access входить до складу пакету Microsoft Office та представляє собою грандіозну систему, яка забезпечує ефективну розробку та супровід баз даних.

Вона дозволяє розв'язувати широке коло завдань користувачів без програмування. Одна з основних переваг СКБД Access полягає в тому, що вона має прості та зручні засоби обробки кількох таблиць в одній базі даних. Таблиця - це місце зберігання даних. Вона є основним об'єктом бази даних.

У Access є зручні та наочні засоби зв'язування таблиць, які зберігаються в одному файлі [15].

Існують переваги зберігання кількох таблиць в одній базі даних. Ось основні переваги: Дані можна організовувати в таблиці залежно від характеру інформації. Наприклад, одна таблиця може містити дані звітів учителів, інша — їхні тижневі розклади, а третя — інформацію про позакласні заходи. Модульне представлення даних у таблицях дозволяє оновлювати та оновлювати окремі таблиці незалежно.

Робота з таблицями

У системі Access як і в інших БД реляційного типу рядок таблиці ототожнюється з терміном "запис", а колонка — з терміном "поле". Кожне поле має ім'я, тип і властивості. При створенні структури таблиці обов'язково вказують імена і типи полів. Властивості полів можуть прийматися за замовчуванням. Одне або кілька полів необхідно визначити як ключові.

В Access використовують такі типи полів: текстове поле, числове поле, поле дата/час**,** логічне поле, поле типу лічильник, поле типу Memo, поле об'єкту OLE.

СКБД Access надає можливість виконувати різноманітні операції з даними, які зберігаються у таблицях. Водночас, одні й ті ж операції можна виконувати за допомогою різних методів.

При створенні нової БД є можливість додавать в неї інші об’єкти (запити, форми, звіти, макроси, модулі) і працювати з ними різними способами. Багато команд головного меню системи Access дублюють ті самі операції, які можуть бути виконані іншими засобами.

Створення таблиць

Створення таблиць можливе у режимі Таблиці та у режимі Конструктора таблиць. На рисунку 2.1 зображено створення таблиці у режимі таблиці.

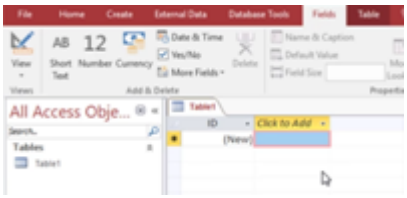


Рисунок 2.1 – Створення Table1 в режимі Таблиці

В базу даних буде додано таблицю з ім’ям Table1, яка відкриється в режимі таблиці.

Режим таблиці преставляє собою бланк (форму) абстрактної таблиці. Потім вона може приймати конкретне наповнення і вміст, в залежності від дій користувача.

В базу даних буде додано таблицю з ім’ям Table1, можна додавать нові поля, вибирать тип полів, наповнювать таблицю інформацією. А також можна перейти в інший режим, режим Конструктора таблиць (або Design View). На рисунку 2.2 зображено Таблицю в режимі Конструктора.

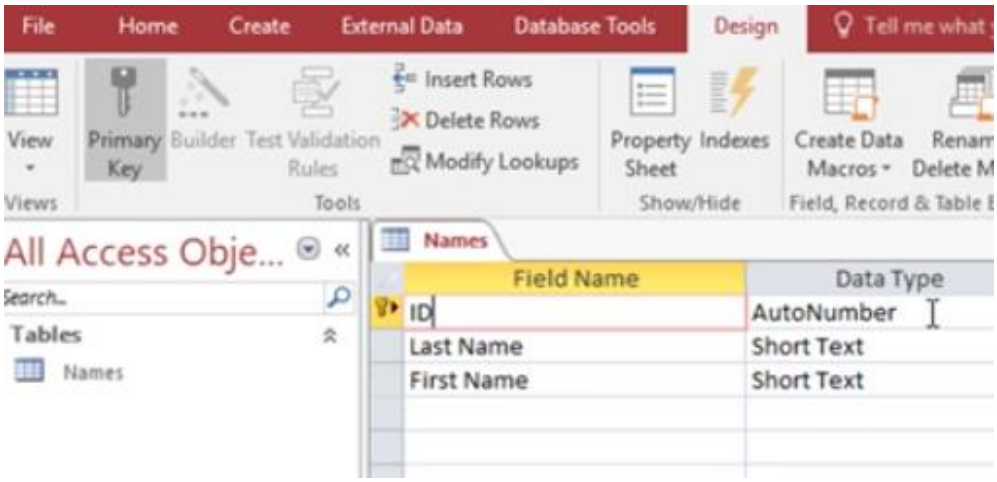


Рисунок 2.2 – Створення та редагування таблиць в режимі Design

В цьому режимі зручно створювати або редагувати структуру таблиці.

Для створення таблиці БД можливо застосовувати **імпортувати список SharePoint** або створити зв’язок з ним [15]. Крім того, можливо створити новый список SharePoint, на основі готового шаблону.

При заповненні таблиці треба пам’ятать, що кожна таблиця повинна вміщувати одне або декілька полів, що однозначно визначають кожен запис в таблиці. Такі поля називають **первинним ключовим полем** таблиці. Якщо для таблиці визначений первинний ключ, то Microsoft Access не допускає дублювання ключа або введення значень Null в ці поля. Після заповнення всієї таблиці необхідно виділити ключове поле. При цьому поряд з іменем цього поля з'явиться зображення ключа.

Зв'язування таблиць БД

Нормалізація схеми БД потрібна для того, щоб організувати дані таким чином, щоб їх редагування виконувалось в одному місці бази даних. Цей процес передбачає необхідність роботи процедури зв'язування таблиць БД [15].

Access створює РБД, які дозволяють об’єднувати інформацію з різних таблиць.

Необхідно встановити взаємозв’язки між таблицями, записи яких логічно пов’язані. Після створення таблиць та визначення ключів, для кожної з них визначаються зв’язки. Потім на основі цих зв’язків стає можливим брати дані з кількох таблиць та відображать їх в одній формі, запиті або звіті.

Існує два основних способи зв’язувати дані: це за допомогою полів підстановки (Lookup Wizard…) та у діалоговому вікні Схема даних.

На рисунку 2.3 видно, як знайти Майстер підстановок з режиму Конструктора таблиць та застосувать до конкретного поля таблиці.

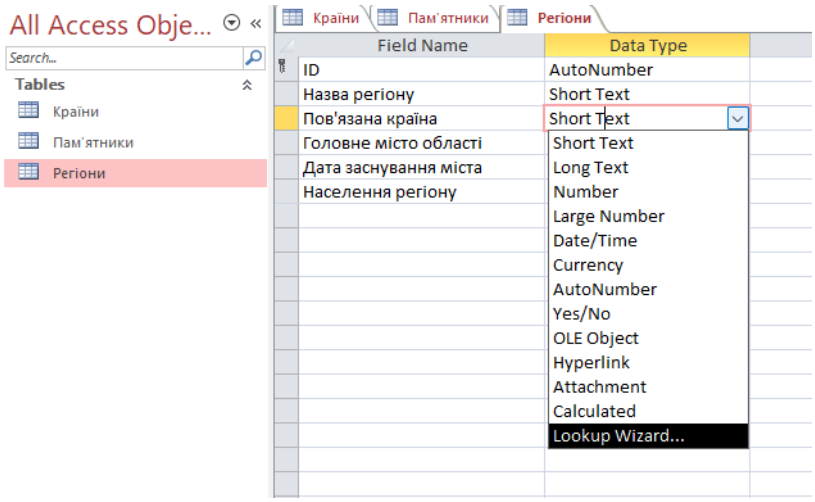


Рисунок 2.3 – Майстер підстановок (Lookup Wizard) для поля Пов’язана країна, таблиці Регіони

Вікно **Схема даних** (Relationships), дозволяє зв’язувати поля та бачити загальну картину відношень між таблицями бази даних.

У відкритому вікні **Схема даних,** якщо є збережений макет схеми даних, цей макет буде виведений на екран. Інакше, потрібно додать таблиці самостійно.

На рисунку 2.4 зображено приклад таблиць із зв’язками у вікні Схема даних БД «Книги».

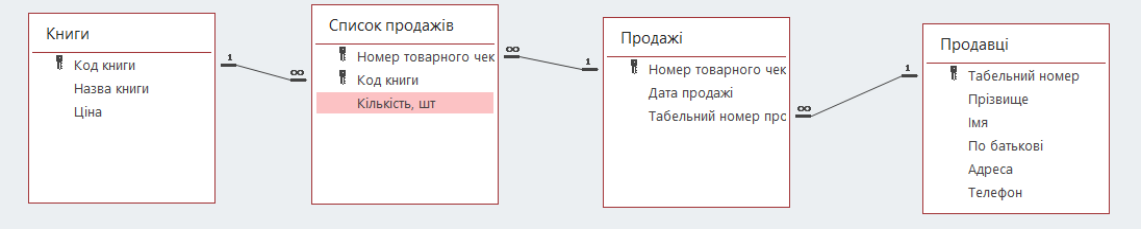


Рисунок 2.4 – Схема даних БД «Книги»

Редагування зв’язку виконується натисканням на лінії зв’язку і викликавши діалогове вікно для зміни, що на рисунку 2.5, де можна встановити цілісність даних, каскадне відновлення та каскадне видалення зв’язків. Також в цьому вікні можливо робити зміни поля, таблиці, що пов’язують та отримати інформацію про тип зв’язку тощо.

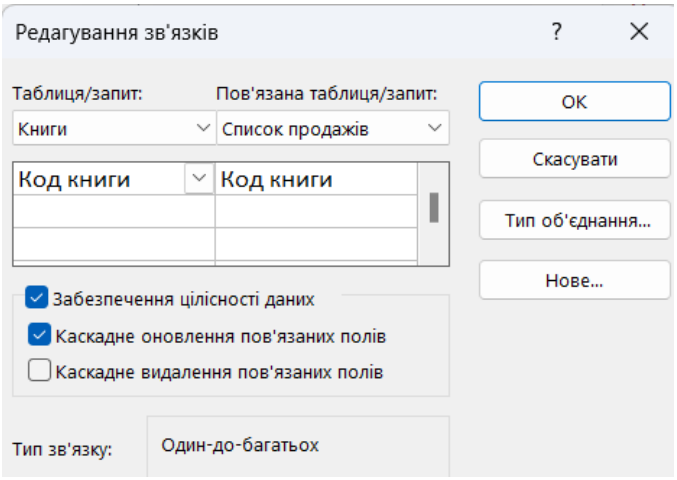


Рисунок 2.5 – Приклад редагування зв’язків у діалоговому вікні редагування

Створення та застосування запитів

Система Access має набагато ширші можливості з пошуку різних відомостей з бази даних окрім пошуку в режимі таблиці.

Запит - це засіб відбору даних з таблиць БД за критеріями визначеними користувачем. У Access застосовують запити різних видів. Запити QBE (Query By Example) це запити за зразком, параметри яких встановлюються у вікні конструктора запитів. SQL-запити – це запити, при створені яких використовуються оператори й функції мови SQL.

Система Access має декілька рішень для створення запитів, можна за допомогою майстрів, в режимі конструктора запитів, а також вибрать спосіб за допомогою SQL інструкцій.

На рисунках 2.6-2.8 видно вікна майстра запитів, де за підказками створюється швидко і просто простий запит.

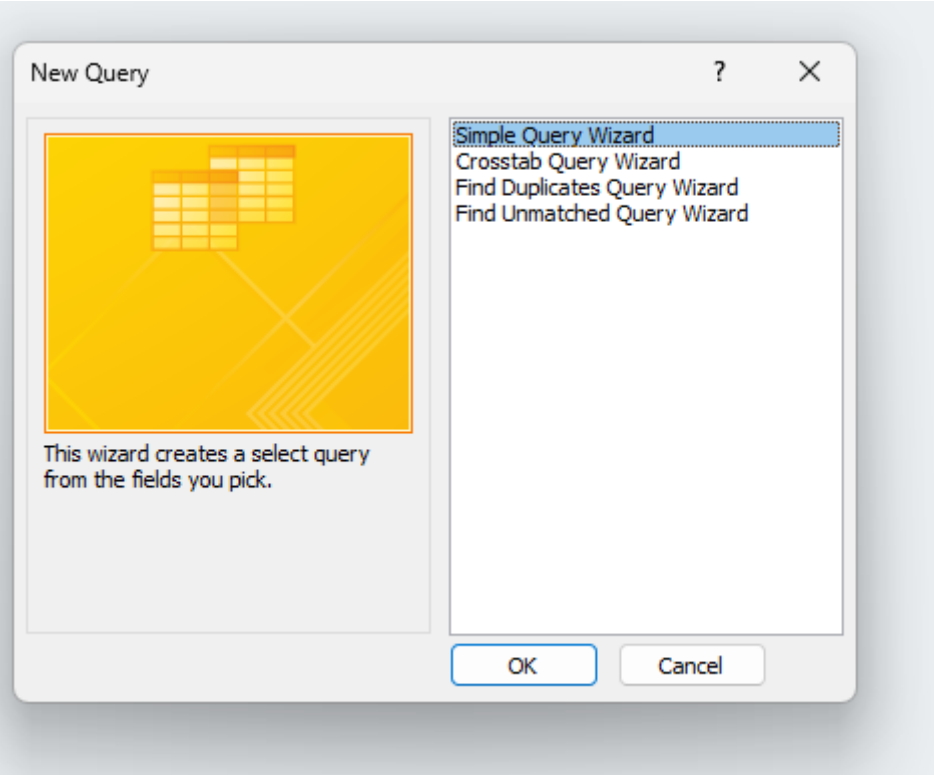


Рисунок 2.6 – Приклад, як будувати простий запит за допомогою майстра, вибір варіанту майстра

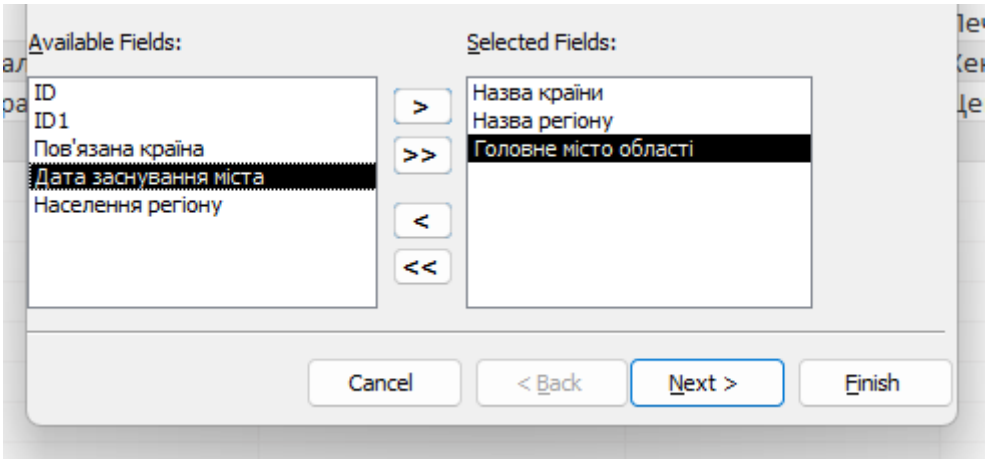


Рисунок 2.7 – Додавання полів у запит в вікні майстра

В залежності від вибраного та натиснутого результат буде змінюватись.

Назва країни	Назва регіону	Населення регіону
США	США	87686
Бразилія	Люксембург	7300000
Канада	Канада	8500000
Франція	Франція	6800000
Люксембург	Угорщина	626000
Швейцарія	Уганда	500000
Польща	Швейцарія	1200000
Уганда	Франція	12300000
Угорщина	Польща	150000
Монголія	Уганда	75000
Турція	Монголія	20000000

Рисунок 2.8 – Приклад результату створеного за допомогою майстра запиту

В режимі конструктора зручно відредагувати створений раніше запит та створити новий (рисунок 2.9).

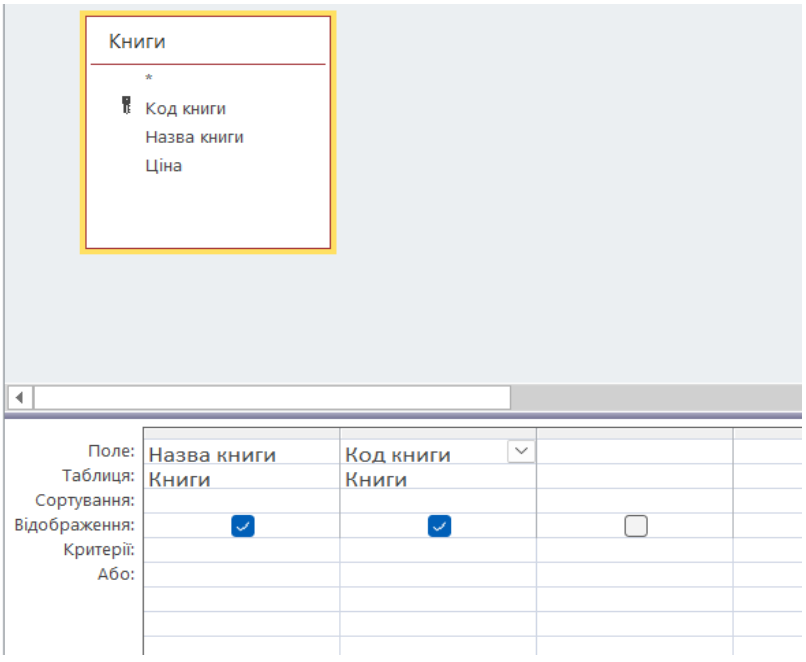


Рисунок 2.10 – Простий запит на вибірку у режимі конструктора

Результатом такого запиту буде тимчасова динамічна таблиця, що буде містити значення двох полів з таблиці Книги.

Можливо за допомогою режиму конструктора запитів встановлювати критерії відбору, наприклад як на рисунку 2.11.

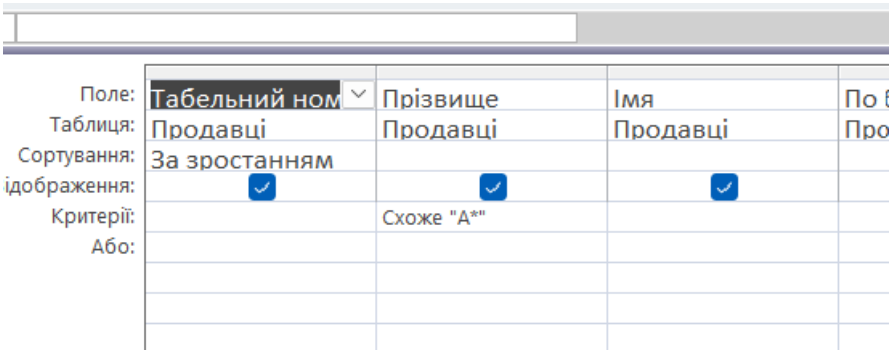


Рисунок 2.11 – Запит з критерієм відбору прізвищ, що починаються з літери А

Результат запиту розглянуто на Рисунку 2.12.

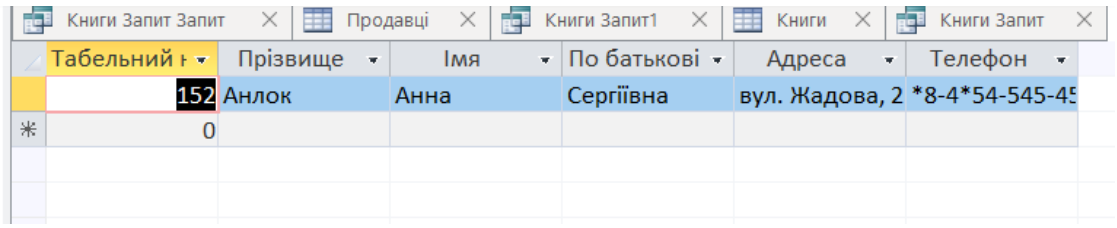


Рисунок 2.12 – Результат запиту з критерієм відбору

У системі Access є також можливість пошуку інформації, використовуючи логічні вирази. Створення цих виразів здійснюється за допомогою програми, яку називають **Побудовувач виразів.** Ця програма створює нову таблицю, зміст якої відповідає заданому виразу.

Сортування даних в запиті виконується , якщо встановити в відповідному рядку потрібний вид сортування (за зростанням чи за спаданням).

Запит можна використовувати для виконання обчислень, розрахунків і підведення результатів, узагальнивши дані з вихідних таблиць. Для цих цілей у Access передбачені статистичні функції SQL (Sum, Avg, Min, Max, Count, First, Last, StDev, Var). Статистичну (агрегатну) функцію задають у рядку Групова операція**,** що з'являється після натискання кнопки з грецькою літерою «сігма», розміщеної на панелі інструментів. За допомогою функції можна опрацювати зміст

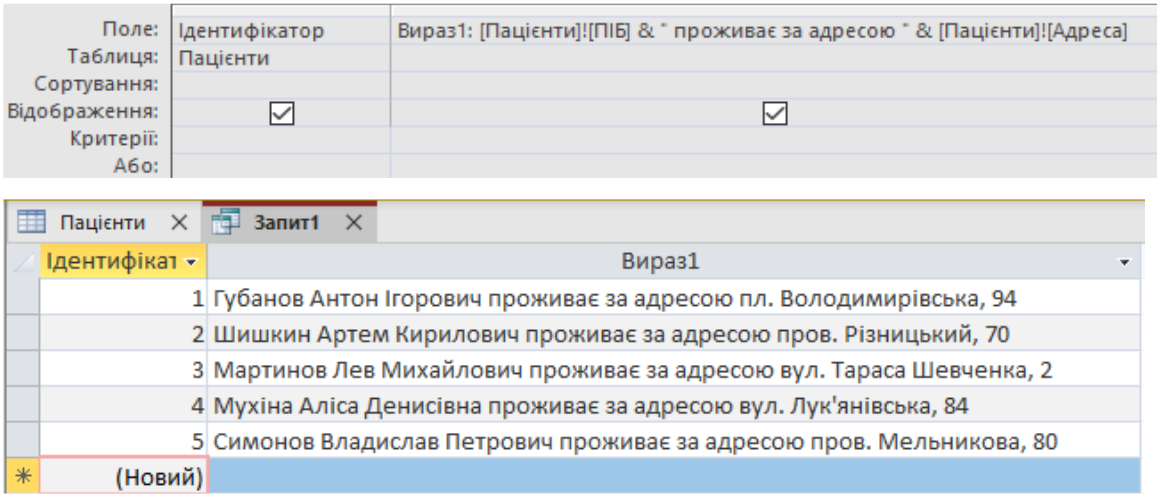
кожного поля запиту. Результат опрацювання з'являється в результуючому наборі записів запиту.

Обчислювальні поля у запитах дозволяють створювати й показувати вирази на основі наявних полів. Вираз представляє собою комбінацію символів-ідентифікаторів, операторів та значень, яка дає певний результат.

Access може утворювати обчислювальні поля з інших полів, буквальних значень та функцій.

Для обчислювальних полів у Access вимагається використовувати певний синтаксис. Наприклад, для комбінації полів Прізвище, Ім'я і кома, синтаксис буде таким: Вираз1: [Прізвище] &"," & [Ім'я].

Синтаксис визначає структуру команди, яка використовується. В обчислювальних полях визначаються стандартні оператори та їх коректне застосування до відповідних полів. Наприклад, на рисунку 2.13 створюється запит з обчисювальним полем.



Поле:	Ідентифікатор	Вираз1: [Пацієнти]![ПІБ] & " проживає за адресою " & [Пацієнти]![Адреса]
Таблиця:	Пацієнти	
Сортування:		
Відображення:	☑	☑
Критерії:		
Або:		

Пацієнти | Запит1

Ідентифікат	Вираз1
1	Губанов Антон Ігорович проживає за адресою пл. Володимирівська, 94
2	Шишкин Артем Кирилович проживає за адресою пров. Різницький, 70
3	Мартинов Лев Михайлович проживає за адресою вул. Тараса Шевченка, 2
4	Мухіна Аліса Денисівна проживає за адресою вул. Лук'янівська, 84
5	Симонов Владислав Петрович проживає за адресою пров. Мельникова, 80
(Новий)	

Рисунок 2.13 – Запит з обчислювальним полем та результат виконання.

Окрім запитів на вибірку є можливість створювати запити інших типів. Наприклад, запити дії (на зміну), на створення таблиці параметричні запити й перехресні запити тощо. Майже всі запити, що створювались методом QBE можливо перетворити у SQL – запити. Для цього потрібно вибрати відповідну опцію у меню Вид (View) запиту, а

саме SQL вид і у полі запиту вводить команди. Діалект SQL Access трохи відрізняється від інших СКБД в деяких моментах, але основні команди не відрізняються.

Побудова інтерфейсу користувача за допомогою форм та звітів

Для зручності та наглядності роботи з даними у БД для користувачів необхідно організовувати інтерфейс. Не всі СКБД мають таку можливість, як Access - можливість створювати інтерфейс за допомогою форм та звітів вбудованими засобами.

Ефективна форма прискорює використання БД, оскільки полегшує пошук. Візуально приваблива форма підвищує зручність й ефективність роботи з інформацією БД, а також допомагає запобігати введенню неприпустимих даних. Можливість створення різних видів форм дозволяє підлаштуватись під різних користувачів.

За допомогою звітів можна представляти користувачу дані на перегляд у зручному вигляді, форматувати та обчислювати дані. Наприклад, можна створити простий звіт, у якому буде подано номери телефонів усіх потрібних контактів БД, або зведений звіт за даними про збут товарів у різних регіонах та за певні часові проміжки у різних формах з додаванням діаграм та графічними матеріалами.

Аналогічно, як і інші об’єкти Access форми і звіти мають засоби швидкої побудови (майстри) та режим конструктора, де можна більш детально відредагувати потрібний об’єкт.

Застосування макросів та набору функцій, що надає Visual Basic for Applications дозволяє покращити роботу з БД та автоматизувати послідовність дій, наприклад відкриття об’єкта, застосовування фільтра, запуск операції експорту та багато інших завдань.

Рекомендовані джерела інформації

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

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

3. Організація баз даних: Методичні вказівки до самостійної роботи студентів за спеціальностями 6.050102/123 «Комп'ютерна інженерія»,125 «Кібербезпека» / уклад. В.В. Сидоренко, Л.В. Константинова -Кропивницький: ЦНТУ, 2017. —88 с.

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

5. 07 - Моделювання даних та маніпулювання даними /Навчальні ресурси СумДУ University online learning ecosystem. СумГУ, 2015 [Електронний ресурс] - Режим доступу: https://elearning.sumdu.edu.ua/free_content/lectured:a8104441b8e00905159c1ff04257b014dd456247/20151109195846/162252/index.html (дата звернення: 20.03.2024).

6. Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни «Організація баз даних» (2 частина) : для студ. денної та заочної форми навч. за спец. : 123 «Комп'ютерна інженерія», 125 «Кібербезпека» / [уклад. : В. В. Босько, Л. В. Константинова] ; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т, каф. кібербезпеки та програмного забезпечення. - Кропивницький : ЦНТУ, 2020. - 60 с. [Електронний ресурс] - Режим доступу: http://dspace.kntu.kr.ua/jspui/handle/123456789/10526 (дата звернення: 7.11.2023).

7. Що таке хешування [Електронний ресурс] - Режим доступу: https://academy.binance.com/uk/articles/what-is-hashing (дата звернення: 7.01.2024).
8. Пєтух А.М., Романюк О.В., Романюк О.Н. Бази даних. Мови запитів, управління транзакціями, розподілена обробка даних, ВНТУ, 2016 [Електронний ресурс] - Режим доступу: https://web.posibnyky.vntu.edu.ua/fitki/11petuh_bazdanyh_movy_zalitiv/index.htm (дата звернення: 12.03.2024).
9. Клієнт-серверна архітектура. QATestLab. 28.05.2020. [Електронний ресурс] - Режим доступу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення: 20.03.2024).
10. Database Replication 101: Everything You Need To Know. January 24th, 2024. [Електронний ресурс] - Режим доступу: https://www.astera.com/type/blog/database-replication-101/ (дата звернення: 20.03.2024).
11. ISO/IEC 9075-1:2023(en). Information technology - Database languages SQL [Електронний ресурс] - Режим доступу: https://www.iso.org/obp/ui/en/#iso:std:76583:en (дата звернення: 25.03.2024).
12. Features of SQL databases / LinkedIn. 27.05.2023. [Електронний ресурс]/ - Режим доступу: https://www.linkedin.com/pulse/features-sql-databases-database-designer-sql-mysql (дата звернення: 24.03.2024).
13. Stephane Faroult with Peter Robson «The Art of SQL». Sebastopol, Calif.: O'Reilly Media Inc. (2006).
14. 10 SQL Skills for Programmers and Developers / Indeed. 11.03.2023 [Електронний ресурс] - Режим доступу: https://www.indeed.com/career-advice/resumes-cover-letters/sql-skills (дата звернення: 25.03.2024).
15. Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни «Бази даних» (1 частина) : для студ. денної та заочної форми навч. за спец. : 123 «Комп'ютерна інженерія», 125

«Кібербезпека» / [уклад. : В. В. Босько, Л. В. Константинова] ; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т, каф. кібербезпеки та програмного забезпечення. - Кропивницький : ЦНТУ, 2020. - 77 с. [Електронний ресурс] - Режим доступу: http://dspace.kntu.kr.ua/jspui/handle/123456789/10516 (дата звернення: 7.11.2023).

16. Paul DuBois, MySQL, 5th Edition, Mar 29, 2013 by Addison-Wesley Professional.

17. SQL Підручник [Електронний ресурс] – Режим доступу: https://w3schoolsua.github.io/sql/index.html#gsc.tab=0 (дата звернення: 7.04.2024).

18. Методичні вказівки до лабораторних робіт з дисципліни «Бази даних і знань» для студентів напряму підготовки «Управління інформаційною безпекою» / [уклад. : Ю. Є. Яремчук, Д. П. Присяжний, І. О. Дьогтєва, О. В. Салієва] ; ВНТУ [Електронний ресурс] - Режим доступу: https://web.posibnyky.vntu.edu.ua/fmib/37yaremchuk_metodvkaz_labrob_bazi_danih_znan_upravlinnya_informacijnoyu_bezpekoyu/07.html (дата звернення: 7.04.2024).

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

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

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

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

23. 97 Things Every Data Engineer Should Know: Collective Wisdom from the Experts. Tobias Macey. O'Reilly, 2021. 264 p.
24. Beginning Spring Data: Data Access and Persistence for Spring Framework 6 and Boot 3 1st ed. Edition. Andres Sacco.Apress, 2022. 439 p.
25. Learning PHP, MySQL & JavaScript. A Step-by-Step Guide to Creating Dynamic Websites. 6th Ed.Robin Nixon.O'Reilly, 2021. 826 p.
26. PHP & MySQL: Novice to Ninja 7th Edition. Tom Butler. SitePoint, 2022. 686 p.
27. PostgreSQL Query Optimization: The Ultimate Guide to Building Efficient Queries. Anna Bailliekova, Henrietta Dombrovskaya, Boris Novikov. Apress, 2021. 344 p.
28. Practical Fraud Prevention. Fraudand AML Analytics for Fintechande Commerce, Using SQL and Python. Gilit Saporta, Shoshana Maraney. O'Reilly, 2022. 394 p.
29. SQL Cookbook. Query Solutionsand Techniques for All SQL Users. 2nd Ed.Anthony Molinaro, RobertdeGraaf. O'Reilly, 2020. 500 p.
30. SQL for Data Analysis. Advanced Techniques for Transforming Data into Insights. 1st Ed. CathyTanimura. O'Reilly, 2021. 350 p.
31. SQL in a Nutshell. A DesktopQuickReference. 4th Edition. Leo Hsu, Regina Obe, Kevin Kline. O'Reilly, 2022. 838 p.
32. SQL PocketGuide: A Guideto SQL Usage. 4th Ed.Alice Zhao. O'Reilly, 2021. 250 p.
33. N. V. Sytnyk and I. S. Zinovieva, “MODERN NoSQL DATABASES FOR TRAINING BACHELORS OF ‘COMPUTER SCIENCE’ SPECIALTY ”, ITLT, vol. 81, no. 1, pp. 255–271, Feb. 2021, doi: 10.33407/itlt.v81i1.3098.
34. Surabhi Gupta, Karthik Ramachandra. Procedural Extensions of SQL: Understanding their usage in the wild. Proceedings of the VLDB Endowment, vol. 14, no. 8, pp. 1378-1391, May2021, doi: 10.14778/3457390.3457402.

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

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

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

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

39. Ashlam A. A., Badii A. and Stahl F. Multi-Phase algorithmic Framework to Prevent SQL Injection Attacks using Improved Machine Learning and Deep learning to Enhance Database security in Real-time. In: 15th International Conference on Securityof Information and Networks (SIN), 11 - 13 November 2022, Sousse, Tunisia, https://doi.org/10.1109/SIN56466.2022.9970504.

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

Список скорочень

БД (DB) – база даних
БнД - банк даних
ІС (IS) - інформаційна система
КБД – персональна база даних користувача
ПО - предметна область
ПЗ – програмне забезпечення
РМД - модель даних реляційного типу
РС – робоча станція
СБД – серверна база даних
СД - словник даних
СКБД - система керування базами даних
ШІ – штучний інтелект
ANSI (American National Standards Institute) - Американський національний інститут стандартів
API (Application Programming Interface) - інтерфейс прикладного програмування
Backend - це програмно-апаратна частина проекту веб-додатку
CASE (Computer Aided Software Engineering)-засоби - це засоби спеціального класу, що покликані вирішувати проблеми автоматизації проектування інформаційних систем
DCL (Data Control Language**) -** мова керування даними (привілеями)
Frontend - це публічна частина web-додатків (вебсайтів), з якою користувач може безпосередньо взаємодіяти та контактувати
TCL (Transaction Control Language) - мова керування транзакціями
DDL (Data Definition Language) - мова визначення даних
DML (Data Manipulation Language) - мова маніпулювання даними
DQL (Data Query Language) – мова вибірки даних
DSD (Data Structure Diagrames) - діаграми структур даних
ERD (Entity Relations Diagram) - діаграми сутність-зв'язок (ER-діаграми) описує модель предметної області
HTML (Hypertext Markup Language) - мова розмітки гіпертексту
HTTP (Hypertext Transfer Prococol) - протокол передачі гіпертексту
ISBL (Information Systems Base Language) – базова мова для інформаційних систем
ISO (International Standards Organization) - міжнародна організація стандартів
OMG (Object Management Group) - група управління об'єктом
OMT (Object Modeling Technique) - метод моделювання об'єктів
OOSE (Object-Oriented Software Engineering) - об'єктно-орієнтоване проектування програмного забезпечення
ODBC (Open Database Connectivity) – сумісність відкритих баз даних.
QBE (Query By Example) – запит за зразком

RAD (Rapid Application Development) - середовище швидкої розробки застосувань
SAD (System Architecture Diagram) - діаграми архітектури
SADT (Structured Analysis and Design Technique) - технологія структурованого аналізу й проектування
SCD (Structure Charts Diagram) - модель бізнес-логіки
SDLC (Systems Development Life Cicle) життєвий цикл інформаційних систем
SQL (Structured Query Language) - структурована мова запитів
SPARC - (Standarts Planning and Requirements Committee) -Комітет планування стандартів і норм
UML (Unified Modeling Language) - уніфікована мова моделювання

Навчальне видання

Босько Віктор Васильович
Константинова Лілія Володимирівна
Поліщук Людмила Іванівна
Конопліцька-Слободенюк Оксана Костянтинівна

БАЗИ ДАНИХ

Навчальний посібник

Українською мовою

Редактор: Константинова Л. В.